New Ways to Garble Circuits

Hanjun Li

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2025

Reading Committee:
Huijia Lin, Chair
Stefano Tessaro, Chair
Shayan Oveis Gharan

Program Authorized to Offer Degree:
Computer Science & Engineering

©Copyright 2025
Hanjun Li

University of Washington

Abstract

New Ways to Garble Circuits

Hanjun Li

Co-Chairs of the Supervisory Committee:
Huijia Lin
Computer Science & Engineering

Stefano Tessaro
Computer Science & Engineering

A garbling scheme transforms a circuit C into a garbled circuit $\widehat{C}$, along with a pair of short keys $(\mathbf{k}_0^{(i)}, \mathbf{k}_1^{(i)})$ for each input bit $\mathbf{x}[i]$, such that the program, garbled program and input keys $(C, \widehat{C}, \{\mathbf{k}_{\mathbf{x}[i]}^{(i)}\})$ can be used to recover the output $\mathbf{z} = C(\mathbf{x})$ while revealing nothing else about the input $\mathbf{x}$.

A main objective in the research of garbling schemes is reducing the size of the garbling material $(\widehat{C}, \{\mathbf{k}_{\mathbf{x}[i]}^{(i)}\})$. On the one hand, theoretical schemes using the heavy tools of attribute-based encryption (ABE) and fully homomorphic encryption (FHE), or indistinguishable obfuscation (iO) can achieve constant size, independent of $|C|$. On the other hand, practically oriented schemes using only symmetric key cryptography all have sizes $\Omega(\lambda \cdot |C|)$.

Motivated by the gap in between, this thesis explores new ways of leveraging light-weight techniques from public-key cryptography to construct communication efficient garbling schemes. In particular, our explorations are centered around two primitives, linearly homomorphic encryption (LHE) and homomorphic secret sharing (HSS).

In Part I, we apply LHE techniques to construct communication efficient garbling schemes that specialize for arithmetic operation gates over a modulus R or bounded integers. We define the (succinctness) rate of such schemes to be the per-gate garbling size normalized by

$\log R$ or the range of bounded integers. Our results include:

- rate-$O(1)$ arithmetic garbling over bounded integers, and
- rate-$O(\lambda_{\mathrm{DCR}})$ mixed garbling over $\mathbb{Z}_R$ and Boolean gates for any modulus R.

In Part II, we apply HSS techniques to construct communication efficient Boolean garbling schemes. Our results lead to a unified framework for garbling arbitrary Boolean gates (as truth tables) with 1-bit per output wire in garbling size. Consequences of this framework include:

- standard Boolean garbling with 1-bit per gate;
- rate-$O(1)$ arithmetic garbling over $\mathbb{Z}_R$ for any modulus R.

All of the mentioned results were achieved for the first time without using FHE or iO.

ACKNOWLEDGMENTS

First and foremost, I am forever grateful to my advisors Huijia Lin and Stefano Tessaro for everything: teaching and guiding me to do research in cryptography, encouraging and inspiring me to aim for a high standard in my work, and supporting and advising my academic career. I am also grateful to Yuval Ishai for hosting me at Technion and for supporting me since then. I am also grateful to Vipul Goyal for supervising my first research experience in cryptography during my Master's study.

I am thankful to the senior students and postdoc researchers from our crypto group: Ashrujit Ghoshal, Ji luo, Tianren Liu, Marshall Ball, Joseph Jaeger, and Binyi Chen, for giving me various advice on research and career choices. I am also thankful to my labmates and friends in our crypto group: Chenzhi Zhu, Marian Dietz, Yao-Ching Hsieh, Champ Chairattana-Apirom, Er-Cheng Tang, and Kameron Shahabi for shared fun conversations and activities.

I am thankful to my thesis committee members: Huijia Lin, Stefano Tessaro, Shayan Oveis Gharan, and Gaku Liu for giving me valuable feedback on my exam talks. I am also thankful to all my great collaborators: Marshall Ball, Amos Beimel, Marian Dietz, Vipul Goyal, Yuval Ishai, Eyal Kushilevitz, Huijia Lin, Tianren Liu, Ji Luo, Sela Navot, Rafail Ostrovsky, Antigoni Polychroniadou, Justin Raizes, Yifan Song,and Stefano Tessaro.

Finally, I am grateful to my wife Yijun, my parents Tiazhou and Lixin, and my parents-in-law, Yuehong and Jiazhong, for supporting and encouraging me throughout my PhD study.

TABLE OF CONTENTS

Page

List of Figures iv

List of Tables vi

Chapter 1: Introduction 1
- 1.1 Our Contributions 4
- 1.2 Related Works 8

Chapter 2: Preliminaries 18
- 2.1 Basic Notations. 18
- 2.2 Definitions for Garbling Schemes. 18
- 2.3 Common Hardness Assumptions 20

Part I: Linearly Homomorphic Encryption (LHE) for Arithmetic Garbling and Label Compression 23

Chapter 3: New Ways to Garble Arithmetic Circuits 24
- 3.1 Introduction 24
- 3.2 Technical Overview 33
- 3.3 Preliminaries 46
- 3.4 Construction of Building Block: LHE 54
- 3.5 Construction of Main Tool: Key Extension for Bounded Integers 66
- 3.6 Construction of Building Block: Linear Seeded Smudger 74
- 3.7 Construction of Main Tool: Key Extension for Modular Arithmetics 79
- 3.8 Construction of Bit Decomposition Gadget for Mixed Computation 91
- 3.9 Garbling Bounded Integer, Modular Arithmetic, and Mixed Computation 108
- 3.10 Concrete Efficiency Analysis 119

Chapter 4: New Ways to Garble Mixed Circuits . . . 132
4.1 Introduction . . . 132
4.2 Technical Overview . . . 137
4.3 Preliminaries . . . 148
4.4 Mixed GC for $\mathbb{Z}_{p^k}$ in Random Oracle Model (ROM) . . . 150
4.5 Mixed GC from Chinese Remainder Theorem in ROM . . . 157
4.6 Mixed GC from DCR-Based LHE in ROM . . . 161

Chapter 5: How to Compress Garble Circuit Labels, Efficiently . . . 178
5.1 Introduction . . . 178
5.2 Technical Overview . . . 187
5.3 Preliminaries . . . 195
5.4 Construction of Main Tool: Batch-Select . . . 202
5.5 Construction of Building Block: LEnc . . . 223
5.6 Construction of Building Block: LHE . . . 236
5.7 Application: Preprocessing Garbling . . . 242
5.8 Adaptive Security and Application to Malicious 2PC . . . 252
5.9 Evaluation . . . 281

Part II: Homomorphic Secret Sharing (HSS) Techniques for Partial and Standard Garbling . . . 293

Chapter 6: Algebraic Homomorphic MAC from Groups Assumptions . . . 294
6.1 Introduction . . . 294
6.2 Technical Overview . . . 300
6.3 Preliminaries . . . 315
6.4 Construction of Main Tool: Algebraic Homomorphic MACs . . . 329
6.5 Application: Succinct Partial Garbling . . . 370
6.6 Application: 1-Key Selective CPRF for Circuits . . . 386

Chapter 7: A Unified Framework for Succinct Garbling from HSS . . . 396
7.1 Introduction . . . 396
7.2 Technical Overview . . . 410
7.3 Preliminaries . . . 420

7.4 aHMAC and HSS as Evaluation Procedures 426
7.5 Succinct Boolean Garbling Schemes 440
7.6 Efficient Arithmetic Garbling Schemes 479
7.7 Concrete Efficiency Analysis 484

Bibliography **503**

LIST OF FIGURES

Figure Number | Page

3.1 AIK arithmetic operation gadgets. 125
3.2 The range of $s'_{\mathrm{res},i}$, conditioning on $s_{1,i}$ and y 126
3.3 The range of $\{s_{1,j}y\}_2 + \{s_{2,j}\}_2$, conditioning on $s_{1,i}$ and $\{y\}_2$ 126
3.4 Setup for bounded integer garbling. 127
3.5 Setup for modular arithmetic garbling. 128
3.6 Setup for mixed bounded integer and Boolean computation garbling. 129
3.7 Formalization of arithmetic circuits. 130
3.8 Garbling sizes in bounded integer, mixed, and modular arithmetic models. . 131

4.1 The naive bit-decomposition gadget. 143
4.2 The naive bit-decompositio gadget, continued. 144
4.3 The naive bit-composition gadget. 152
4.4 The mini bit-composition gadget. 170
4.5 The bit-decomposition gadget in ring $\mathbb{Z}_{p^k}$. 171
4.6 The bit-decomposition gadget in ring $\mathbb{Z}_{p^k}$, continued. 172
4.7 The linear bit-composition gadget over ring $\mathbb{Z}_{p^k}$. 173
4.8 Notation mapping for variants of Damgård-Jurik encryption. 174
4.9 The bit-composition gadget based on vDJ. 174
4.10 The bit-composition gadget based on vDJ, continued. 175
4.11 The bit-decomposition gadget based on $\overline{\mathsf{vDJ}}$ 176
4.12 The bit-decomposition gadget based on $\overline{\mathsf{vDJ}}$, continued. 177

5.1 The T-session 2PC functionality. 269
5.2 Our preprocessing T-session malicious 2PC protocol. 274
5.3 Our preprocessing T-session malicious 2PC protocol, continued. 275
5.4 Sub-protocol AuthGarb, adapted from [WRK17, DILO22]. 287
5.5 Sub-protocol AuthGarb, adapted from [WRK17, DILO22], continued. 288
5.6 Algorithm AuthEval, adapted from [WRK17, DILO22]. 289

5.7 The preprocessing functionality adapted from [WRK17, DILO22]. 290

7.1 2PC for Boolean circuits under Paillier groups, the Init phase. 488
7.2 2PC for Boolean circuits, the Eval, Final phases. 489
7.3 2PC subprotocol for $O(\log \lambda)$-ary Boolean gates. 490
7.4 Leveled 2PC for Boolean circuits under Paillier groups, the Init. 491
7.5 Leveled 2PC for Boolean circuits under Paillier groups, the Eval, Final phases. 492
7.6 Leveled 2PC for Boolean gates. 493
7.7 Leveled 2PC for Boolean gates, continued. 494
7.8 2PC for Boolean circuits under lattices. 495
7.9 Leveled 2PC for Boolean circuits under lattices. 496
7.10 2PC for Boolean circuits under prime-order groups. 497
7.11 Leveled 2PC for Boolean circuits under prime-order groups. 498
7.12 Leveled 2PC for Boolean circuits under prime-order groups, continued. . . . 499
7.13 2PC for arithmetic circuits with large modulus. 500
7.14 2PC for arithmetic gates. 501
7.15 2PC for arithmetic gates, continued. 502

LIST OF TABLES

Table Number Page

3.1 Comparison of arithmetic garbling for bounded integer computation. 27

3.2 Summary of our garbling schemes. 31

3.3 Comparison of garbled circuit size for bounded integer computation. 121

3.4 Comparison of garbled circuit size supporting bit decomposition. 123

3.5 Comparison of arithmetic garbling for modular arithmetic computation. 123

3.6 Comparison of computation costs for bounded integer garbling. 124

4.1 Comparison between our GC and previous works. 136

5.1 Comparison between techniques for input label compression, in terms of *online communication* and *online computation time*. 285

5.2 The amount of offline communication (split into $|\mathsf{ct}_1|$ and $|\mathsf{ct}_2|$). 286

5.3 Concrete online costs (per input bit) of for input length $|\mathbf{x}| \approx 700\text{K}$. 286

5.4 Time cost of the algorithms of batch-select, and sizes of the outputs. 291

5.5 Time cost and number of invocations of batch-select. 292

6.1 Comparison between homomorphic MAC and signature schemes, in terms of *class of functions supported*, size $|\sigma|$ of the tags/signatures, *assumptions*, *black-box* usage of cryptography, and *composability*. 395

7.1 Comparison between Boolean garbling schemes, in terms of *the class of Boolean circuits handled*, *garbled circuit size*, *the cryptographic tool used*, and *assumptions*. 401

7.2 Comparison between *arithmetic* garbling schemes, in terms of *ring supported*, *garbled circuit size*, *the cryptographic tool used*, and *assumptions*. 402

7.3 Concrete sizes for public data pd in our non-leveled schemes, and [LWYY24]. 484

Chapter 1

INTRODUCTION

The notion of a garbling scheme was introduced by Yao in the 1980s [Yao82], and later formalized in the work of [BHR12]. It allows a "garbler" to transform a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$ into a garbled circuit $\widehat{C}$, along with a pair of short keys $\{\mathbf{k}_0^{(i)}, \mathbf{k}_1^{(i)}\}$ for each input bit $\mathbf{x}[i]$. Given the circuit C, the garbled circuit $\widehat{C}$, and the set of input keys $\{\mathbf{k}_{\mathbf{x}[i]}^{(i)}\}$ selected according to some unknown input $\mathbf{x}$, an efficient "evaluator" can recover $C(\mathbf{x})$ while learning nothing else about the input $\mathbf{x}$. An important feature of a garbling scheme is that the input keys are short, growing only with the security parameter λ and not with the size of the circuit C.

Applications of Garbling Schemes. The fact that a garbling scheme can encode a computation C and an input $\mathbf{x}$ separately, and in a decomposable way, implies natural applications for secure computations in the offline-online setting. For a simple example, let Alice represent a computationally weak device sent to collect sensitive data $\mathbf{x}$, which are supposed to be input to an expensive computation C. Alice is too weak to compute $C(\mathbf{x})$ quickly on her own, but also cannot reveal the sensitive data $\mathbf{x}$ for others to help with the computation. A garbling scheme provides a natural solution for this problem: Alice can pre-compute a garbled circuit $\widehat{C}$ in an offline phase, and send it to a helper Bob. After collecting sensitive data $\mathbf{x}$ in the online phase, Alice only needs to perform the simple computation of selecting a set of keys $\{\mathbf{k}_{\mathbf{x}[i]}^{(i)}\}$ according to $\mathbf{x}$, and sends those selected keys to Bob. Bob can help evaluating $C(\mathbf{x})$, while learning nothing else about the sensitive data $\mathbf{x}$.

The above simple solution can be extended to implementing a constant round protocol for general two-party secure computation (2PC), where both Alice and Bob have input data $\mathbf{x}, \mathbf{y}$ sensitive to each other, but wish to jointly compute $C(\mathbf{x}, \mathbf{y})$. In the offline phase, Alice again

computes a garbled circuit $\widehat{C}$ and sends it to Bob. In the online phase, the remaining task is for Alice to send two sets of keys $\{\mathbf{k}^{x,(i)}_{\mathbf{x}[i]}\}$, $\{\mathbf{k}^{y,(i)}_{\mathbf{y}[i]}\}$, each selected according to the sensitive data $\mathbf{x}, \mathbf{y}$. For Alice's own input $\mathbf{x}$, she can directly compute $\{\mathbf{k}^{x,(i)}_{\mathbf{x}[i]}\}$ and send those. For Bob's input $\mathbf{y}$, the two parties can jointly run a much simpler 2PC protocol implementing the "selection" functionality, which reveals to Bob the correct keys $\{\mathbf{k}^{y,(i)}_{\mathbf{y}[i]}\}$ according his input $\mathbf{y}$. Such specialized protocols, known as oblivious transfer (OT), can be efficiently implemented in constant number of rounds, giving an overall constant round 2PC protocol.

The above 2PC solution based on garbling schemes can be further generalized to a constant round multi-party secure computation (MPC) protocol [BMR90], and has been later optimized to just two rounds [GS18b, BL18]. See the survey [App17] for many more applications of garbling schemes.

Approaches to Minimizing Garbling Sizes. A central research direction of garbling schemes is to minimize the garbling sizes, consisting of both the garbled circuit $\widehat{C}$ and the set of selected keys $\{\mathbf{k}^{(i)}_{\mathbf{x}[i]}\}$. This is well motivated by the above applications to 2PC and MPC protocols, where the garbling sizes are communication bottlenecks. Furthermore, in other applications where the same garbling is distributed to multiple parties, minimizing its size has a compounded impact, translating into proportional bandwidth and storage savings across all recipients.

Over the past four decades, great progress has been made on reducing garbling sizes, even reaching the ultimate goal of full succinctness, where the garbling sizes are independent of the circuit size $|C|$. However, as we explain below and in Section 1.2, existing approaches fall into two extremes: (1) extremely fast, but with a garbling size of $O(\lambda \cdot |C|)$ bits, or (2) prohibitively slow, but achieving a garbling size of $(\ell_x + \ell_z) \cdot \mathrm{poly}(\lambda)$, independent of the circuit size $|C|$. This leaves a significant gap for schemes with $o(\lambda)$ bits per gate and are still efficient enough to be implemented and used in practical applications.

- *Fast Non-Succinct Garbling Schemes.* This approach includes Yao's original garbled circuit construction [Yao82], and a long line of works on practical optimizations [BMR90,

NPS99, KS08a, PSSW09, KMR14, GLNP15, ZRE15, RR21]. They achieve the the most practical general-purpose garbling schemes by only relying on fast symmetric-key cryptography, and adopting the (heuristic) random oracle model (ROM) whenever it's helpful for practical efficiency. The current state-of-the-art, due to Rosulek and Roy [RR21], garbles an AND gate using $1.5\lambda + 5$ bits, while XOR and NOT gates are free in the ROM.

However, this approach appears to reach its limit. There are evidence [ZRE15] that suggests the $O(\lambda)$ overhead in garbling sizes is a barrier that requires entirely new constructions.

- *Heavy Fully Succinct Garbling Schemes.* On the other extreme, fully succinct garbling schemes have been shown feasible under standard assumptions. Though theoretically optimal, the concrete garbling sizes are astronomically large, making these constructions practically infeasible. In addition, existing schemes rely either on indistinguishability obfuscation (iO) [KLW15, BCG+18] or on a non-black-box combination of fully homomorphic encryption (FHE) and attribute-based encryption (ABE) [GKP+13b, BGG+14, HLL23], which incur a prohibitively high computational overhead.

Towards Practical Succinct Garbling Schemes. Motivated by the gap in existing approaches to minimizing garbling sizes, this thesis aims to address the following question:

Can we leverage light-weight techniques from public-key cryptography to construct Boolean garbling schemes with $o(\lambda)$ bits per gate?

In particular, the target succinctness is a garbling size below the circuit description length, $|C| \log |C|$, for sufficiently large circuits. This is a natural threshold, since it implies a succinct garbling is less expensive to communicate than the original circuit.

Beyond Boolean Garbling: Arithmetic Garbling Schemes. Arithmetic garbling is a natural generalization to Boolean garbling, first proposed by the work of [AIK11]. In the

generalized setting, the circuits to be garbled consist of addition and multiplications gates over some modulus R, or over the integers bounded by some value B. Boolean garbling can be viewed as a special case of arithmetic garbling over $\mathbb{Z}_2$.

The main goal of arithmetic garbling is to improve over the baseline solution of simply garbling Boolean circuits that emulate arithmetic operations. Using Yao-style garbling, which are the most practical schemes, this baseline achieves garbling sizes $O(\ell \log \ell \lambda \cdot |C|)$, for an ℓ-bit modulus R or when intermediate wire values are integers bounded by 2^ℓ. However, concretely the Boolean circuits for emulating integer arithmetic will be large, making the baseline unsatisfactory in terms of both garbling sizes and computation overheads. Arithmetic garbling schemes aim to garble arithmetic gates in a more direct way, avoiding costly translation to Boolean gates, and with more succinct garbling sizes.

While ideally we can hope for practical succinct arithmetic garbling scheme with sizes below $|C| \log |C|$, existing solutions are still far from this ambitious target. In particular, existing techniques all have per-gate sizes that scales with the bit-length ℓ of the underlying domain. Therefore, we consider a more convenient metric, called (succinctness) rate, that measures the per-gate cost normalized by ℓ. For example, the baseline solution has rate $O(\log \ell \cdot \lambda)$.

The work of [AIK11] constructs an arithmetic garbling over bounded integers with a rate of $\mathrm{poly}(\lambda)$, under the learning with error (LWE) assumption. Two gaps remain: achieving better rate than $\mathrm{poly}(\lambda)$ and supporting modular arithmetic. This thesis aims to address these natural questions:

> *Can we leverage light-weight techniques from public-key cryptography to construct arithmetic garbling schemes with $o(\lambda)$ rate, and supporting modular arithmetic?*

1.1 Our Contributions

In this thesis, we answer the above questions affirmatively by exploring new ways of applying public-key cryptography techniques to constructing various garbling schemes. In particular,

our explorations are centered around two primitives, linearly homomorphic encryption (LHE) and homomorphic secret sharing (HSS). We summarize the main results obtained below, and refer readers to individual chapters for more details and additional results. Unless noted, all highlighted results were obtained for the first time without using the heavy tools of fully homomorphic encryption (FHE) or indistinguishablility obfuscation (iO).

Part I: Applying LHE to Arithmetic Garbling and Label Compression. In Chapter 3, we develop a framework for constructing arithmetic garbling schemes based on suitable LHE schemes. In more detail, the prior work of [AIK11] showed how to reduce the task of constructing an arithmetic garbling scheme to constructing a core key-extension gadget, and our framework further shows how to construct key-extension gadgets from LHE schemes. Instantiating our framework, from LHE based on the learning with errors (LWE) assumption, we obtain

- Rate-$O(\lambda_{\mathrm{LWE}})$ [1] arithmetic garbling over bounded integers;
- Rate-$\widetilde{O}(\lambda_{\mathrm{LWE}})$ arithmetic garbling over $\mathbb{Z}_R$ for any modulus R.

The former re-derives the result of [AIK11] in a more modular way. Then, from LHE based on the decisional composite residuosity (DCR) assumption, we obtain [2]

- Rate-$O(1)$ arithmetic garbling over bounded integers;
- Rate-$O(\lambda_{\mathrm{DCR}})$ arithmetic garbling over $\mathbb{Z}_R$.

In Chapter 4, we focus on constructing a garbling scheme of mixed circuits that combine both arithmetic and Boolean computations. We show how to reduce this task to constructing a pair of bit-decomposition and bit-composition gadgets. Assuming only the random oracle model (ROM) oracle, we obtain a construction of

- Rate-$O(\lambda\ell/\log\ell)$ mixed garbling over $\mathbb{Z}_R$ for any modulus R.

[1] λ_{LWE} is the dimension of an LWE secret, and λ_{DCR} is the bit-length of a DCR secret.

[2] Both results based on DCR assume sufficiently large domain size $\ell = \Omega(\lambda_{\mathrm{DCR}})$.

Then, using LHE based on DCR, we obtain a more optimized construction of

- Rate-$O(\lambda_{\mathrm{DCR}})$ mixed garbling over $\mathbb{Z}_R$ for any modulus R.

In Chapter 5, we focus on minimizing the size of input labels of a Boolean garbling scheme, which in common constructions costs $O(\lambda\ell_x)$ bits for ℓ_x-bits input. Towards this, we relax the setting to allow the garbler prepare a (potentially re-usable) public data that may depend on the secret later used for garbling circuits.

By combining LHE based on the RingLWE assumption with another primitive called lacoinc encryption (LEnc), and assuming ROM, we obtain a concretely efficient solution that compresses labels for ℓ_x-bits input into

- Re-usable public data of $O(\lambda\ell_x \log \ell_x)$ bits;
- Input dependent data of $O(\ell_x) + \mathrm{poly}(\lambda)$ bits.

We note that while the LHE schemes used in the above results have similar syntax and correctness requirements, their security definitions are formalized differently to suit different applications.

Part II: Applying HSS Techniques to (Partial) Garbling. In Chapter 6, we design a new primitive called algebraic homomorphic MAC (aHMAC), which is closely related and inspired by techniques developed from homomorphic secret sharing (HSS) literature [BGI16, BKS19, RS21, OSY21]. The new primitive is inspired by the work of [MORS24], which was the first to identify a connection between garbling schemes and homomorphic secret sharing (HSS) schemes.

As a first application, we consider a generalized notion of partial garbling introduced by [IW14]. It considers Boolean circuits of the form $C(\mathbf{x}, \mathbf{y}) = C_{\mathrm{Priv}}(\mathbf{y}, C_{\mathrm{Pub}}(\mathbf{x}))$, with a public sub-circuit C_{Pub} over the public input $\mathbf{x}$ and a private sub-circuit C_{Priv} over the result of $C_{\mathrm{Pub}}(\mathbf{x})$ and the private input $\mathbf{y}$. Security is relaxed to only guarantee privacy of the input $\mathbf{y}$ from the evaluator. By combining aHMAC and a standard Boolean garbling, we obtain

- Fully succinct partial garbling, with a garbling size of $O(\lambda|C_{\text{Priv}}|)$ bits, independent of the size of the public sub-circuit.

Next, we follow the blueprint in [GKP+13b] for bootstrapping a partially private computation into a fully private one using a homomorphic encryption (HE) scheme. In more detail, we apply partial garbling over the computation defined as follows:

$$C'(\{\mathsf{ct}_i\}, \mathsf{sk}) := \mathsf{Dec}(\mathsf{sk}, \mathsf{HEval}(C, \{\mathsf{ct}_i\})), \text{ where}$$
$$\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{sk}}(\mathbf{x}[i]),\ C_{\text{Priv}} := \mathsf{Dec}(\cdot, \cdot),\ \text{and } C_{\text{Pub}} := \mathsf{HEval}(C, \cdot).$$

Security of partial garbling ensures the HE secret key sk remains private, and security of HE in turn ensures the encrypted inputs $\mathbf{x}$ also remain private. By the full succinctness of our partial garbling, the resulting garbling scheme also has fully succinct sizes independent of the evaluated function f. By instantiating this blueprint with an aHMAC based on groups, and existing HE schemes based on groups, we obtain

- Fully succinct Boolean garbling for bounded-length branching programs; [3]
- Fully succinct Boolean grabling for quadratic polynomials.

In Chapter 7, we develop a framework for constructing Boolean garbling schemes by combining a compatible pair of aHMAC and HSS schemes, under either group or lattice assumptions. The resulting scheme supports arbitrary gates (expressed as truth tables) at the cost of 1 bit per output wire in garbling sizes. By instantiating the framework with standard Boolean gates we obtain

- Rate-1 Boolean garbling for circuits.

By instantiating the framework with truth tables of $\mathbb{Z}_p$ operations, for polynomial sized modulus p, and additionally applying Chinese remainder theorem (CRT) we obtain

- Rate-$O(1)$ arithmetic garbling over $\mathbb{Z}_R$ for any modulus R.

[3]This class includes truth tables, deterministic finite automata (DFAs), and decision trees of arbitrary sizes.

We note that even though the Rate-$O(1)$ arithmetic garbling from Chapter 7 is superior to those from Chapter 3 and 4 in terms of garbling sizes, it is also computationally more expensive due to a more complex construction. In particular, the framework from Chapter 7 requires evaluating a PRF or local PRG inside of HSS, which can be a heavy computation.

Notes on Subsequent Works. In a follow up work [MORS24] to our results from Chapter 3 (published in [BLLL23]), the authors obtained a rate-1 arithmetic garbling scheme over bounded integers based on a circular variant of the DCR assumption. Although only achieving a constant factor improvement over our results, the work of [MORS24] was the first to bring HSS techniques to constructing garbled circuits, and was the inspiration of Part II of this thesis.

In a concurrent work [Hea24], the author obtained a rate-$O(\lambda)$ mixed garbling scheme over $\mathbb{Z}_R$ for any modulus R assuming only ROM. This avoids the DCR assumption compared to our result from Chapter 4 (published in [LL24]). As a trade off, due to the complexity of its construction, the scheme of [Hea24] can be computationally more expensive than ours.

1.2 Related Works

1.2.1 Foundational Works

Yao's Boolean Garbling. The notion of garbled circuits was first introduced by Yao [Yao82], who also provided an elegant construction for garbling Boolean circuits C. The resulting garbling size is $O(\lambda)$ bits per gate, on average, under the minimal assumption that one-way functions (OWF) exist.

The high-level idea is to let the garbler encode the two possible values $0, 1$ on each wire i of C as a pair of random strings $\mathbf{k}_0^{(i)}, \mathbf{k}_1^{(i)} \in \{0,1\}^\lambda$, and provide sufficient decoding information to allow an evaluator recover exactly the strings corresponding to the correct wire values $v^{(i)}$ with respect to some input $\mathbf{x}$.

Garb Samples	Eval Recovers
$\mathbf{k}_0^{(i)}, \mathbf{k}_1^{(i)} \leftarrow \{0,1\}^\lambda$,	$\mathbf{k}_{v^{(i)}}^{(i)}$ for all wires $i \in C$.

As those recovered strings are random, no intermediate wire values are leaked to the evaluator. The evaluator now obtains one string on each output wire, but still doesn't know which wire values they correspond to. In order to reveal those output wire values to the evaluator, the garbler directly provides both strings on each output wire, and which values they correspond to.

This high-level idea is implemented using a symmetric key encryption scheme $\mathsf{Enc}, \mathsf{Dec}$, using the random strings $\mathbf{k}_0^{(i)}, \mathbf{k}_1^{(i)}$ as secret keys. For every gate g with input wires a, b and an output wire c, the garbler provides 4 encryptions of the output keys $\mathbf{k}_0^{(c)}, \mathbf{k}_1^{(c)}$ according to the truth-table of g:

$$\begin{array}{ll} \mathsf{Garb}\text{ Provides} & \mathsf{Eval}\text{ w/ } \mathbf{k}^{(a)}_{v^{(a)}}, \mathbf{k}^{(b)}_{v^{(b)}},\ \text{Decrypts} \\ \mathsf{tb}_g = \begin{cases} \mathsf{Enc}_{\mathbf{k}_0^{(a)}}(\mathsf{Enc}_{\mathbf{k}_0^{(b)}}(\mathbf{k}^{(c)}_{g(0,0)})) \\ \dots \\ \mathsf{Enc}_{\mathbf{k}_1^{(a)}}(\mathsf{Enc}_{\mathbf{k}_1^{(b)}}(\mathbf{k}^{(c)}_{g(1,1)})) \end{cases} & \mathsf{Enc}_{\mathbf{k}^{(a)}_{v^{(a)}}}(\mathsf{Enc}_{\mathbf{k}^{(b)}_{v^{(b)}}}(\mathbf{k}^{(c)}_{v^{(c)}})). \end{array}$$

As illustrated, if the evaluator holds two input keys $\mathbf{k}^{(a)}_{v^{(a)}}, \mathbf{k}^{(b)}_{v^{(b)}}$ corresponding to the correct wire values $v^{(a)}, v^{(b)}$, then it can decrypt exactly one of the entries in tb_g to recover the output key $\mathbf{k}^{(c)}_{v^{(c)}}$ corresponding to $v^{(c)}$. Given one such encrypted table tb_g for every gate $g \in C$, the evaluator starts with a set of input keys $\{\mathbf{k}^{(i)}_{\mathbf{x}[i]}\}$ corresponding to some input $\mathbf{x}$ to C, and proceeds in the topological order to recover a key for every intermediate wire in C, including the output wires.

A final detail is that by observing which entry is decryptable in tb_g, the evaluator may infer the output wire value of g. To prevent this, the entries of tb_g are randomly permuted, and the evaluator tries decrypting all entries and ignores the failed ones.

In summary, in Yao's construction, the garbling $\widehat{C}$ consists of all the per-gate encryptions $\{\mathsf{tb}_g\}_{g \in C}$, and both keys on every output wire i in C, $\mathbf{k}_0^{(i)}, \mathbf{k}_1^{(i)}$. The key function $K^{(i)}$ for the i-th input simply maps a bit $\mathbf{x}[i]$ to $\mathbf{k}^{(i)}_{\mathbf{x}[i]}$. On average, the garbling size is 4 encryptions, i.e. $O(\lambda)$ bits, per gate.

The construction of Yao [Yao82] has a very attractive feature: it only relies on symmetric-key encryptions, which are much more efficient than public-key encryptions. In practice, each encryption is implemented by an AES invocation, which takes roughly tens of *nanoseconds* in a modern CPU (with hardware support for AES). In contrast, common public encryption systems such as RSA or ElGamal can be millions times slower. Consequently, a long line of follow-up research [BMR90, NPS99, KS08a, PSSW09, KMR14, GLNP15, ZRE15, RR21] focuses on reducing the garbling size of Yao's construction while still only relying on symmetric-key encryptions. The state-of-art construction [RR21] without relying on public-key assumptions achieves a garbling size of 1.5λ bits per AND gate, and 0 per XOR gate. However, there are evidence [ZRE15] that suggests existing techniques without relying on public-key encryption have reached their limit. In particular, to overcome the $O(\lambda)$ bits per gate barrier likely require new techniques.

This thesis takes a different direction: we utilize light-weight techniques from public-key encryption schemes to break the $O(\lambda)$ bits per gate barrier. In particular, in Chapter 7 we achieve 1-bit per gate on average.

Arithmetic Garbling. The generalization of garbling schemes to handle arithmetic circuits (over integers bounded by 2^ℓ) was first studied by the work of [AIK11]. A baseline solution exists: for any input arithmetic circuit C over bounded integers, translate it into a Boolean circuit C' emulating C, and then apply Yao's garbling on C'. However, there are two unsatisfactory aspects.

- While the asymptotic overhead (in size) of this translation is just $O(\log \ell)$, the concrete overhead, i.e. the hidden constant, is prohibitively large.
- The baseline solution requires key functions applied to the bit-representations of each input integer, while in some applications the bit-representations of inputs may not be available.

Addressing these two drawbacks, the work of [AIK11] constructed a garbling scheme (1) without translating integer addition and multiplications into Boolean sub-circuits, and (2)

with key functions that directly map each input integer to a single label. The underlying assumption is the hardness of the learning with error (LWE) problem, and the garbling size is $\mathrm{poly}(\lambda) \cdot \ell$ per gate.

The high-level construction of [AIK11] follows Yao's garbling where the garbler encodes all possible wire values in $[2^\ell]$ on each wire i of C into keys $\mathbf{k}_0^{(i)}, \dots \mathbf{k}_{2^\ell-1}^{(i)}$, and provide sufficient decoding information to allow an evaluator recover exactly the strings corresponding to the correct wire values $\mathbf{k}_{v^{(i)}}^{(i)}$.

Garb Samples	Eval Recovers
$\mathbf{k}_0^{(i)}, \dots \mathbf{k}_{2^\ell-1}^{(i)}$,	$\mathbf{k}_{v^{(i)}}^{(i)}$ for all wires $i \in C$.

However, it's now infeasible to sample all 2^ℓ keys per wire as random strings, as the bound 2^ℓ can be exponentially large in λ. The observation in [AIK11] is that the keys don't need to be all random for security. In fact, since the evaluator should learn at most 1 key per wire, it suffices for the 2^ℓ keys to be each marginally random. Therefore, the keys are defined to be

$$\mathbf{k}_v^{(i)} := \mathbf{s}^{(i)} \cdot v + \mathbf{r}^{(i)} \text{ (over } \mathbb{Z}\text{)}, \tag{1.1}$$

where $\mathbf{s}^{(i)}, \mathbf{r}^{(i)}$ are random integer vectors sampled from sufficiently large ranges once per wire.

Following Yao's blueprint, the remaining task is to define suitable encrypted tables tb_g for each gate $g \in C$ (with input wires a, b and an output wire c) that allows an evaluator with two keys $\mathbf{k}_{v^{(a)}}^{(a)}, \mathbf{k}_{v^{(b)}}^{(b)}$ to decrypt $\mathbf{k}_{v^{(c)}}^{(c)}$. However, the method from Yao's construction of encrypting all 2^ℓ possible keys for the output wire separately is again infeasible. The main innovation of [AIK11] is a solution to this task, using an encryption scheme with *additive homomorphism* based on the LWE assumption. The solution of [AIK11] can be summarized as as two pairs algorithms $\mathsf{Add.Garb}, \mathsf{Add.Eval}$, $\mathsf{Mul.Garb}$, $\mathsf{Mul.Eval}$ for handling addition and multiplication gates separately, as illustrated below (for the Mul case).

Garb Provides	Eval w/ $\mathbf{k}_{v^{(a)}}^{(a)}, \mathbf{k}_{v^{(b)}}^{(b)}$, Decrypts
$\mathsf{tb}_g \leftarrow \mathsf{Mul.Garb}(\mathbf{s}^{(a)}, \mathbf{s}^{(b)}, \mathbf{s}^{(c)}, \mathbf{r}^{(a)}, \mathbf{r}^{(b)}, \mathbf{r}^{(c)})$	$\mathbf{k}_{v^{(c)}}^{c} \leftarrow \mathsf{Mul.Eval}(\mathbf{k}_{v^{(a)}}^{(a)}, \mathbf{k}_{v^{(b)}}^{(b)}, \mathsf{tb}_g)$.

The size of tb_g for each gate is $\mathrm{poly}(\lambda) \cdot \ell$ per gate. Therefore, the garbling scheme has a (succinctness) rate of $\mathrm{poly}(\lambda)$. (See Definition 2.3.)

The work of [AIK11] inspires this thesis in two ways. First, it motivates further study of arithmetic garbling schemes. For example, the rate of $\mathrm{poly}(\lambda)$ is a natural target for optimization, and supporting modular instead of bounded integer arithmetics is a natural open question. Indeed, in Chapter 3, we show the rate of $\mathrm{poly}(\lambda)$ can be significantly improved to $O(1)$ for bounded integer arithmetics. And in Chapter 7, we further support modular arithmetics for arbitrary modulus R at rate $O(1)$.

Second, its solution illustrates the potential of using light-weight techniques from public-key encryption schemes (e.g. additively homomorphic encryption) to significantly improve the size of garbling schemes. This thesis continues this approach.

Rate-1 Arithmetic Garbling from HSS Techniques. The work of [MORS24] was a follow-up to our results in Chapter 3 (published in [BLLL23]) on arithmetic garbling over bounded integers. It achieved a constant improvement in rate from $O(1)$ to 1, i.e. a garbling size of $(\ell + \mathrm{poly}(\lambda))$ per gate, and more importantly, was the first to bring techniques from HSS literature to the context of garbling. The underlying assumption was the circular security of Damgård-Jurik (public-key) encryption system.

The first observation of [MORS24] is that the key format of $\mathbf{k}_v^{(i)}$ (Equation 1.1) on each wire i can be further simplified to

$$\mathbf{k}_v^{(i)} := \boldsymbol{\Delta} \cdot v + \mathbf{r}^{(i)} \text{ (over } \mathbb{Z}), \tag{1.2}$$

where the per-wire random vector $\mathbf{s}^{(i)}$ is replaced with a global one $\boldsymbol{\Delta}$. Security only requires the marginal distributions of one key per wire to be random, which is still ensured by the randomness of $\mathbf{r}^{(i)}$. A direct benefit is that now adding two keys $\mathbf{k}_{v^{(a)}}^{(a)} + \mathbf{k}_{v^{(b)}}^{(b)}$ gives another key $\mathbf{k}_{v^{(c)}}^{(c)}$ of the same format:

$$\mathbf{k}_{v^{(c)}}^{(c)} = \mathbf{k}_{v^{(a)}}^{(a)} + \mathbf{k}_{v^{(b)}}^{(b)} = \boldsymbol{\Delta} \cdot \underbrace{(v^{(a)} + v^{(b)})}_{v^{(c)}} + \underbrace{\mathbf{r}^{(a)} + \mathbf{r}^{(b)}}_{\mathbf{r}^{(c)}}.$$

The garbler can directly define $\mathbf{r}^{(c)} := \mathbf{r}^{(a)} + \mathbf{r}^{(b)}$ for every output wire c of addition gates g, and avoid providing any encryptions for those addition gates. The remaining task is to handle multiplication gates g in C as illustrated here.

$$\begin{array}{ll} \mathsf{Garb}\text{ w/ } \mathbf{\Delta}, \mathbf{r}^{(a)}, \mathbf{r}^{(b)} & \mathsf{Eval}\text{ w/ } \mathbf{k}^{(a)}_{v^{(a)}}, \mathbf{k}^{(b)}_{v^{(b)}}, \mathsf{tb}_g \\ \text{Derives } \mathbf{r}^{(c)}, \mathsf{tb}_g, & \text{Derives } \mathbf{k}^{(c)}_{v^{(c)}} \end{array}$$
$$// \text{ s.t. } \mathbf{k}^{(c)}_{v^{(c)}} = \mathbf{\Delta} \cdot v^{(a)} \cdot v^{(b)} + \mathbf{r}^{(c)}.$$

The main innovation of [MORS24] is identifying a connection between the above task and HSS literature. In particular, the values $\mathbf{r}^{(a)}, \mathbf{r}^{(b)}$ held by the garbler and $\mathbf{k}^{(a)}_{v^{(a)}}, \mathbf{k}^{(b)}_{v^{(b)}}$ held by the evaluator form additive shares of $\mathbf{\Delta} \cdot v^{(a)}$ and $\mathbf{\Delta} \cdot v^{(b)}$, due to the key format (Equation 1.2). The desired results $\mathbf{r}^{(c)}, \mathbf{k}^{(c)}_{v^{(c)}}$ should form another additive share of $\mathbf{\Delta} \cdot v^{(a)} \cdot v^{(b)}$.

$$\begin{array}{ll} \text{Garbler, Evaluator} & \text{jointly hold Shares } \langle \mathbf{\Delta} \cdot v^{(a)} \rangle, \langle \mathbf{\Delta} \cdot v^{(b)} \rangle \\ & \text{jointly derive Shares } \langle \mathbf{\Delta} \cdot v^{(a)} \cdot v^{(b)} \rangle. \end{array}$$

Where we use a short-hand $\langle \mathbf{\Delta} \cdot v \rangle$ to denote a pair of additive shares of $\mathbf{\Delta} \cdot v$. In comparison, an HSS scheme enables two parties who jointly hold encryptions of some inputs $v^{(a)}, v^{(b)}$ and additive shares of a global secret $\langle \mathbf{\Delta} \rangle$, to non-interactively derive additive shares of $\langle \mathbf{\Delta} \cdot v^{(a)}, v^{(b)} \rangle$.

$$\begin{array}{ll} \text{HSS Parties } P_0, P_1 & \text{jointly hold } \mathsf{Enc}_{\mathbf{\Delta}}(x), \mathsf{Enc}_{\mathbf{\Delta}}(y), \langle \mathbf{\Delta} \rangle \\ & \text{locally derive Shares } \langle \mathbf{\Delta} \cdot x \cdot y \rangle. \end{array}$$

While the settings don't match exactly, it turns out that techniques from HSS can be adapted to solve the above task in garbling in a more efficient way than the original solution of [AIK11] or of Chapter 3. The garbling size per addition gate is 0 thanks to the special key format, and the garbling size per multiplication gate is $\ell + \text{poly}(\lambda)$ bits using the new HSS based solution. Therefore, the garbling scheme has a rate approaching 1 for large ℓ.

Although the end result achieved by [MORS24] is only a constant factor improvement over [BLLL23], it inspires Part II of this thesis to explore the connection between garbling and HSS techniques further. First, while the technique of [MORS24] depends on the specific

algebraic structures of Damgård-Jurik encryption system, in Part II we develop a unifying framework that covers additional existing HSS constructions under prime-order groups and lattices. Second, while the work of [MORS24] focused only on arithmetic garbling over bounded integers, in Chapter 7 we develop further techniques to achieve succinct Boolean garbling and arithmetic garbling over an arbitrary modulus.

1.2.2 Alternative Approaches

Fully Succinct Garbling from ABE and FHE. The work of [GKP+13b] proposed a theoretical construction of (reusable) garbling schemes by combining the powerful tools of attribute-based encryption (ABE) and fully homomorphic encryption (FHE). When instantiated with the succinct ABE scheme of [BGG+14] based on the LWE assumption, the resulting garbling scheme has a fully succinct size of $(\ell_x + \ell_z) \cdot \mathrm{poly}(\lambda, \mathrm{Depth}(C))$, independent of the size of C. Under the stronger evasive LWE assumption, the resulting scheme [HLL23] has a garbling size of $(\ell_x + \ell_z) \cdot \mathrm{poly}(\lambda)$, independent of the depth of C.

Without delving into the details of an ABE scheme, the first idea in [GKP+13b] is using ABE to implement a garbling scheme over partially private computations, $C(\mathbf{x}, \mathbf{y}) := C_{\mathrm{Priv}}(\mathbf{y}, C_{\mathrm{Pub}}(\mathbf{x}))$, also known as a partial garbling scheme. The evaluator in the garbling scheme learns the public input $\mathbf{x}$ in the clear, together with the garbled circuit and input keys, and can recover the results $\mathbf{z} = C(\mathbf{x}, \mathbf{y})$. Security of the scheme guarantees nothing is leaked about the private input $\mathbf{y}$, beyond the legitimate results $\mathbf{z}$. Succinctness from the ABE scheme translates to a garbling size that's independent of $|C_{\mathrm{Pub}}|$.

The next idea in [GKP+13b] is to bootstrap a partially private computation into a fully private one using an FHE scheme. Compared to a normal encryption scheme with $\mathsf{Enc}, \mathsf{Dec}$ algorithms, an FHE scheme additionally includes an HEval algorithm that allows evaluating any Boolean circuits C over a set of ciphertexts encrypting some input $\mathbf{x}$. The result is

another ciphertext encrypting $C(\mathbf{x})$ that can be decrypted by Dec.

$$\text{For } C : \{0,1\}^{\ell_x} \to \{0,1\}, \quad \mathbf{x} \in \{0,1\}^{\ell_x},$$
$$\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{sk}}(\mathbf{x}[i]), \quad \mathsf{ct}^* \leftarrow \mathsf{HEval}(C, \{\mathsf{ct}_i\}), \quad C(\mathbf{x}) \leftarrow \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}^*).$$

Using an FHE scheme, one can use a partially private computation C' to implement a fully private one C as follows:

$$C'(\{\mathsf{ct}_i\}, \mathsf{sk}) := \mathsf{Dec}(\mathsf{sk}, \mathsf{HEval}(C, \{\mathsf{ct}_i\})), \text{ where}$$
$$\mathsf{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{sk}}(\mathbf{x}[i]),\ C_{\text{Priv}} := \mathsf{Dec}(\cdot,\cdot), \text{ and } C_{\text{Pub}} := \mathsf{HEval}(C, \cdot).$$

Indeed, we have $C'(\{\mathsf{ct}_i\}, \mathsf{sk}) = C(\mathbf{x})$ by the correctness of FHE. Furthermore, by partial privacy one can argue the FHE secret key sk remains private, hence the encrypted inputs $\mathbf{x}$ also remains private by the security of FHE. Therefore, a standard garbling of circuits C can indeed be implemented by a partial garbling of C' as defined above.

Note that this approach of garbling from a combination of ABE and FHE achieves an extremely succinct garbling size, independent of the size of C. However, it is also extremely computationally expensive. On average, garbling or evaluating every gate in C requires evaluating the HEval algorithm as a circuit within the ABE scheme. As both ABE and FHE are already computationally expensive tools, combining them in this way is considered practically infeasible. This thesis takes a different direction: instead of aiming for extreme succinctness, we aim for practically useful schemes with improved succinctness over existing practical schemes.

Fully Succinct Garbling from iO. The work of [BGL+15] proposed another theoretical approach to garbling schemes with extreme succinctness using an indistinguishability obfuscation (iO) scheme, which can be instantiated under a combination of assumptions: LWE, PRG in NC0, and the Decision Linear assumption (DLIN) in symmetric pairing groups [JLS21]. Reducing the assumptions needed to construct iO is still under active research. The computation model considered was Turing machines, instead of circuits, with a bounded running time and space by T. The garbling scheme had a size independent of T.

Similar to the previous approach of combining ABE and FHE, this approach based on iO was also extremely computationally expensive and considered practically infeasible. [4]

1.2.3 *Other Related Works*

Arithmetic Garbling without Public-key Assumptions. The work of [BMR16] constructed an arithmetic garbling scheme over special modulus that are product of distinct primes, relying on only symmetric-key encryptions and assuming the random oracle model (ROM). Such garbling schemes can also be used for implementing an arithmetic garbling over bounded integers, by choosing a sufficiently large special modulus. The garbling size is $O(\ell^2/\log \ell \cdot \lambda)$ per multiplication gate, and 0 per addition gate, for an ℓ-bit special modulus. Therefore, it achieves rate $O(\ell \log \ell \cdot \lambda)$.

Compared with the original work [AIK11] studying arithmetic garbling, the work of [BMR16] had a heavier focus on concrete efficiency, both communication-wise and computation wise. First, to have the best computational efficiency, the authors chose to completely avoid techniques from public-key assumptions, and were willing to rely on the heuristic/ idealized random oracle model. Second, while the asymptotic garbling size of $O(\ell^2/\log \ell \cdot \lambda)$ achieved by [BMR16] was worse than the Boolean baseline $O(\ell \log \ell \cdot \lambda)$ (using state-of-art integer multiplication algorithms), the authors argued that for realistic ranges of ℓ, for example $\ell = 64$, their concrete garbling size still improves over the Boolean baseline.

A later work [Hea24] continued the approach of building arithmetic garbling schemes without public-key assumptions (and relying on ROM). In more detail, the authors constructed arithmetic garbling schemes over any modulus R with size $O(\ell \cdot \lambda)$ per multiplication or addition gate. This work improves over [BMR16] in two ways: (1) supporting arbitrary modulus instead of only special modulus, and (2) achieving rate $O(\lambda)$ instead of $O(\ell \log \ell \cdot \lambda)$. However, the additional techniques introduced by [Hea24] made it significantly more computationally

[4] This is due to currently known construction of iO being expensive. In theory, another approach towards practical succinct garbling is constructing an efficient iO scheme, which is a major open problem in cryptography.

expensive.

This thesis takes a different direction than the above approach, where we are willing to adopt light-weight techniques using public-key assumptions to construct succinct garbling schemes. In particular, in Chapter 3 and Chapter 7, we achieve rate $O(1)$, breaking the $O(\lambda)$ barrier from the above results.

Succinct Garbling from FHE. The recent work of [LWYY25] constructed a Boolean garbling scheme with (amortized) one bit per gate using a specific FHE scheme [GSW13] under the RingLWE assumption. While FHE is still a computationally expensive tool, this result is already a step towards practical efficiency by avoiding the even more expensive combination with ABE from the result of [GKP+13b]. In Part II, we further improve the practical efficiency by relying on the lighter-weight tool of HSS, while keeping the one bit per gate garbling size. Furthermore, we also construct arithmetic garbling over any modulus with rate $O(1)$, which was not considered by [LWYY25].

Chapter 2

PRELIMINARIES

2.1 Basic Notations.

For any positive integer N, let $[N] := \{0, 1, \ldots, N-1\}$, let $\mathbb{Z}_N$ denote the ring of integer modulo N. We use bold letters $\mathbf{x}$ to denote a vector, and write $\mathbf{x}[i]$ to denote its i-th component. We write $\mathbf{x} \otimes \mathbf{y}$ to denote the tensor product between two vectors. We assume modulo operation has lower priority than addition. That is, $a + b \bmod p$ should be interpreted as $(a + b) \bmod p$.

2.2 Definitions for Garbling Schemes.

Definition 2.1 (Boolean Garbling). *A Boolean garbling scheme consists of two efficient algorithms:*

- $\mathsf{Garb}(1^\lambda, C)$ *takes a circuit* $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$, *and outputs a garbling* $\widehat{C}$, *and input key functions* $\{K^{(i)}\}_{i \in [\ell_x]}$, *where each key function* $K^{(i)}$ *maps an input bit* $\mathbf{x}[i]$ *to a label* $L^{(i)} \in \{0,1\}^\lambda$.
- $\mathsf{Eval}(C, \widehat{C}, \{L^{(i)}\}_{i \in [\ell_x]})$ *takes a circuit* C, *a garbling* $\widehat{C}$, *and input labels* $L^{(i)}$ *(corresponding to some input* $\mathbf{x} \in \{0,1\}^{\ell_x}$*). It outputs the evaluation result* $\mathbf{z} \in \{0,1\}^{\ell_z}$.

Correctness: *For every polynomial* $p(\lambda)$, *there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$, *circuits* C *with size* $|C| \le p(\lambda)$, *and inputs* $\mathbf{x} \in \{0,1\}^{\ell_x}$, *the following holds:*

$$\Pr\left[\begin{array}{l} \mathsf{Eval}(C, \widehat{C}, \{L^{(i)}\}) \\ = C(\mathbf{x}) \end{array} \;\middle|\; \begin{array}{r} (\widehat{C}, \{K^{(i)}\}) \leftarrow \mathsf{Garb}(1^\lambda, C), \\ L^{(i)} = K^{(i)}(\mathbf{x}[i]). \end{array}\right] \ge 1 - \mathrm{negl}(\lambda).$$

Security: *There exists an efficient simulator* Sim *such that for every polynomial* $p(\lambda)$

sequence of circuits $\{C_\lambda\}$ *where* $|C_\lambda| \leq p(\lambda)$*, and sequence of inputs* $\{\mathbf{x}_\lambda\}$*, the following holds (suppressing the subscript* λ *for brevity):*

$$\left\{\mathsf{Sim}(1^\lambda, C, C(\mathbf{x}))\right\}_\lambda \approx_c \left\{ \widehat{C}, \{L^{(i)}\}, \;\middle|\; \begin{array}{r} (\widehat{C}, \{K^{(i)}\}) \leftarrow \mathsf{Garb}(1^\lambda, C), \\ L^{(i)} = K^{(i)}(\mathbf{x}[i]). \end{array} \right\}_\lambda$$

Definition 2.2 (Garbling w/ Circuit Privacy)**.** *We say a garbling scheme has* circuit privacy *if there exists an efficient simulator* Sim *satisfying security per Definition 2.1, but instead of taking the circuit description* C *as input only takes the topology of* C*, denoted* $\Phi_{Topo}(C)$*.*

Definition 2.3 (Arithmetic Garbling)**.** *Let* $R(\lambda)$ *be a ring with bit-length bounded by* $|R(\lambda)| \leq 2^{\mathrm{poly}(\lambda)}$*. An arithmetic garbling scheme over* R *consists of two efficient algorithms,* Garb*,* Eval *with the analogous syntax to Definition 2.1 and the following differences:*

- *The input circuit* C *to* $\mathsf{Garb}(1^\lambda, \cdot)$ *consists only of addition (+), subtraction (−), and multiplication (×) gates over* $R(\lambda)$*;*
- *The key functions* $K^{(i)}$ *output by* $\mathsf{Garb}(1^\lambda, C)$ *now is an affine function that maps elements in* $R(\lambda)$ *to a label* $L^{(i)} \in \mathcal{L}(\lambda)$*, where the label space* $\mathcal{L}$ *is a* module *supporting scalar multiplications with elements in* R *and addition between elements in* $\mathcal{L}$*. The bit length of elements in* $\mathcal{L}$ *is a fixed polynomial independent of* C*.*

(Succinctness) Rate: *We define the rate of an arithmetic garbling scheme over* R *to be*

$$\mathrm{rate} := \max_{C,\mathbf{x}} \frac{|\widehat{C}| + |\{L^{(i)}\}|}{|C| \cdot |R|}$$

Definition 2.4 (Arithmetic Garbling over Bounded Integers)**.** *An arithmetic garbling scheme over bounded integers by some value* $B(\lambda) \leq 2^{\mathrm{poly}(\lambda)}$ *consists of two efficient algorithms,* Garb*,* Eval *with the analogous syntax to Definition 2.3 and the following differences:*

- *Correctness and security hold only with respect to* admissible computations $(C, \mathbf{x})$ *where all intermediate wire values in* $C(\mathbf{x})$ *are bounded by* $B(\lambda)$*;*
- *The rate is defined to be* $\mathrm{rate} := \max_{C,\mathbf{x}} (|\widehat{C}| + |\{L^{(i)}\}|)/(|C| \cdot |\log B|)$*.*

Definition 2.5 (Mixed (Arithmetic and Boolan) Garbling)**.** *A mixed garbling scheme over a ring $R(\lambda)$ (resp. over bounded integers by $B(\lambda)$) consists of two efficient algorithms,* Garb*,* Eval *with the analogous syntax to Definition 2.3 (resp. Definition 2.4) and the following differences:*

- *The input circuit C to* $\mathsf{Garb}(1^\lambda, \cdot)$ *consists not only of arithmetic gates $(+, -, \times)$, but also Boolean gates (*AND*,* XOR*, and* NOT*), as well as a special bit-decomposition (*BD*) gate, that takes as input a single arithmetic value $x \in R$ (resp. $x \in \mathbb{Z}_B$), and outputs the bit representation* $\mathsf{bits}(x)$*.*
- *The key functions $\{K^{(i)}\}$ output by* $\mathsf{Garb}(1^\lambda, C)$ *now may have two possibly different label spaces $\mathcal{L}, \mathcal{L}'$ for Boolean and arithmetic inputs respectively.*

2.3 Common Hardness Assumptions

Definition 2.6 (Prime-order Groups)**.** *We consider prime-order groups $\mathcal{G} = \{\mathcal{G}_\lambda\}$ that have efficient instance generation algorithms* Gen*:*

- $\mathsf{Gen}(1^\lambda)$ *outputs a group $G \in \mathcal{G}_\lambda$ with order $p > 2^\lambda$, and and a generator g. The group order p is included in the description of G.*

Definition 2.7 (Paillier Groups)**.** *Paillier groups are defined by the following instance generation algorithm* Gen*.*

- $\mathsf{Gen}(1^\lambda, 1^\zeta)$ *uniformly samples two λ-bit primes p, q such that $p = 2p' + 1$, $q = 2q' + 1$ where p', q' are also primes. It outputs $(N = pq, \zeta)$ as the group description of $G = \mathbb{Z}^*_{N^{\zeta+1}}$.*

Lemma 2.1 (Facts about Paillier Groups [Pai99, DJ01])**.** *Let $G = \mathbb{Z}^*_{N^{\zeta+1}}$ be a Paillier group sampled by* $\mathsf{Gen}(1^\lambda, 1^\zeta)$ *for a polynomial $\zeta(\lambda)$.*

- *G has a subgroup $F = \{(1+N)^x : x \in N^\zeta\}$ where discrete log (i.e., finding x) can be efficiently solved.*

- *G has a subgroup H that's isomorphic to $\mathbb{Z}_N^*$, and $G = F \times H$.*
- *Consider a random element $g \in G$ such that the Jacobi symbol of g mod N is 1. Then $\langle g \rangle$ contains F except with negligible probability. We write $g \leftarrow \mathsf{Samp}(N, \zeta)$ to mean sampling such elements g with Jacobi symbol 1.*

Definition 2.8 (DDH Assumption)**.** *We say DDH holds in Paillier groups if the following holds for every polynomial $\zeta(\lambda)$:*

$$\left\{ \mathsf{pp}, g, g^a, g^b, g^{ab} \;\middle|\; \begin{array}{c} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b \leftarrow [N]. \end{array} \right\}_\lambda$$
$$\approx_c \left\{ \mathsf{pp}, g, g^a, g^b, g^c \;\middle|\; \begin{array}{c} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b, c \leftarrow [N^{\zeta+2}]. \end{array} \right\}_\lambda .$$

We say DDH holds in prime-order groups if the following holds:

$$\left\{ \mathsf{pp}, g, g^a, g^b, g^{ab} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ a, b \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda$$
$$\approx_c \left\{ \mathsf{pp}, g, g^a, g^b, g^c \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ a, b, c \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda .$$

Remark. The DDH assumption in prime-order groups is considered standard. The DDH assumption in Paillier groups is adapted from the formulation by [ADOS22], where the authors formulate a separate "small exponent" assumption stating it's secure to sample the secret exponents in DDH from a smaller, but still sufficiently large, range than the order of g. We directly state the small-exponent variant of DDH in Paillier groups here, as it's required in our applications.

Definition 2.9 (DCR Assumption [Pai99, DJ01])**.** *Let $\zeta = \zeta(\lambda)$ be a polynomial. The assumption DCR_ζ states that the following (computational) indistinguishability holds:*

$$\{\mathsf{pp}, u\} \approx_c \{\mathsf{pp}, v\} \;\middle|\; \begin{array}{c} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ u, u', \leftarrow \mathbb{Z}_{N^{\zeta+1}}^*, \quad v = u'^{N^\zeta} \bmod N^{\zeta+1}. \end{array}$$

Remark. As a consequence the following indistinguishability also holds, which sometimes may be more convenient

$$\{\mathsf{pp}, u\} \approx_c \{\mathsf{pp}, v\} \left| \begin{array}{l} \mathsf{pp} = (N,\zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ \qquad\qquad u \leftarrow \mathsf{QR}_{N^{\zeta+1}}, \quad v \leftarrow \mathsf{HC}_{N^{\zeta+1}}, \end{array} \right.$$

where $\mathsf{QR}_{N^{\zeta+1}} := \{a^2 | a \in \mathbb{Z}^*_{N^{\zeta+1}}\}$ denote the subgroup of quadratic residues, and $\mathsf{HC}_{N^{\zeta+1}} := \{a^{2N^\zeta} | a \in \mathbb{Z}^*_{N^{\zeta+1}}\}$ denote a "hard" subgroup of $\mathbb{Z}^*_{N^{\zeta+1}}$. It is known that $\mathsf{HC}_{N^{\zeta+1}}$ is a cyclic group of size $p'q' \approx \frac{N}{4}$.

Definition 2.10 (LWE Assumption [Reg05])**.** *Let λ be the security parameter, $n = n(\lambda) \leq$ polyλ be a dimension, $q = q(\lambda) \leq 2^{\text{poly}\lambda}$ be a modulus, and $\chi = \chi(\lambda)$ be an error distribution over $\mathbb{Z}$. The assumption $\mathsf{LWE}_{n,q,\chi}$ states that for every polynomial $m = m(\lambda)$, the following (computational) indistinguishability holds:*

$$\{A, A\mathbf{s} + \mathbf{e}\}_\lambda \approx_c \{A, \mathbf{u}\}_\lambda \left| \begin{array}{l} A \leftarrow \mathbb{Z}_q^{m\times n}, \ \mathbf{u} \leftarrow \mathbb{Z}_q^m \\ \mathbf{s} \leftarrow \mathbb{Z}_q^n, \ \mathbf{e} \leftarrow \chi^m \end{array} \right.$$

In this work, we only consider polynomial ring of the form $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where $n(\lambda)$ is a power-of-2.

Definition 2.11 (RingLWE Assumption [LPR10])**.** *Let λ be the security parameter, $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ be a polynomial ring where $n(\lambda)$ is a power-of-2, $q = q(\lambda) \leq 2^{\text{poly}\lambda}$ be a modulus, and $\mathcal{D}_{\mathsf{sk}}(\lambda), \mathcal{D}_{\mathsf{err}}(\lambda)$ be two distributions over $\mathcal{R}$. The assumption $\mathsf{LWE}_{\mathcal{R},q,\mathcal{D}_{\mathsf{sk}},\mathcal{D}_{\mathsf{err}}}$ states that for every polynomial $m = m(\lambda)$, the following (computational) indistinguishability holds:*

$$\{\mathbf{a}, s\cdot\mathbf{a} + \mathbf{e}\}_\lambda \approx_c \{\mathbf{a}, \mathbf{u}\} \left| \begin{array}{l} \mathbf{a}, \mathbf{u} \leftarrow \mathcal{R}_q^m, \\ \quad s \leftarrow \mathcal{D}_{\mathsf{sk}}, \ \mathbf{e} \leftarrow \mathcal{D}_{\mathsf{err}}^m \end{array} \right.$$

Part I

LINEARLY HOMOMORPHIC ENCRYPTION (LHE) FOR ARITHMETIC GARBLING AND LABEL COMPRESSION

Chapter 3

NEW WAYS TO GARBLE ARITHMETIC CIRCUITS

This chapter is adapted from published work [BLLL23].

3.1 Introduction

Garbled circuits, introduced by Yao [Yao82], enable a "Garbler" to efficiently transform a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ into a *garbled circuit* $\widehat{C}$ and a pair of keys $\mathbf{k}_0^i, \mathbf{k}_1^i$ for every input bit. In particular, the input keys are short, of length polynomial in the security parameter only, independent of the complexity of the circuit. An input $x \in \{0,1\}^n$ to the circuit can be encoded by choosing the right keys corresponding to each input bit $\mathbf{L}^x = \{\mathbf{k}_{x_i}^i\}_{i\in[n]}$, referred to as the input *labels*. The garbled circuit and input labels $(\widehat{C}, \mathbf{L}^x)$ together reveal the output of the computation $y = C(x)$, and hide all other information of x. Yao's seminal result [Yao82] constructed garbled circuit using Pseudo-Random Generators (PRGs), which in turn can be based on one-way functions. Since its conception, garbled circuits has found a wide range of applications, and is recognized as one of the most fundamental and useful tools in cryptography.

The arithmetic setting. While there have been remarkable optimizations and analytical improvements in the intervening years, the currently most widely applied approaches to garbling circuits still largely follow Yao's paradigm from the 1980s[1]. Yao's idea involves encrypting the truth tables of gates in the circuit, which becomes inefficient or even infeasible when the truth tables are large. A longstanding open question is designing *arithmetic garbling*,

[1] There have been alternative approaches that rely on strong primitives such as a combination of fully homomorphic encryption and attribute-based encryption [BGG+14, GKP+13a, LLL22], or indistinguishabilty obfuscation [AJS17]. These approaches however are much more complex than Yao's garbling and less employed in applications. See Section 3.1.2 for more discussion.

namely, variants of garbled circuits that apply naturally to arithmetic circuits without "Booleanizing" the computation, meaning bit-decomposing the inputs and intermediate values and garbling the Boolean circuit implementation of arithmetic operations. To achieve arithmetic garbling, fundamentally new techniques different from the mainstream encrypted truth-table methods must be developed.

The work of Applebaum, Ishai, and Kushilevitz (AIK) [AIK11] initiated the study of arithmetic garbling. They first formalized the notion of *Decomposable Affine Randomzed Encoding (DARE)* as follows:

> ARITHMETIC GARBLING (I.E., DARE) is an efficient transformation Garb that converts an arithmetic circuit $C : \mathcal{R}^n \to \mathcal{R}^m$ over a ring $\mathcal{R}$ into a garbled circuit $\widehat{C}$, along with $2n$ key vectors $\mathbf{k}_0^i, \mathbf{k}_1^i \in \mathcal{R}^\ell$, such that $\widehat{C}$ together with the input labels $\mathbf{L}^x = \{\mathbf{L}^i = \mathbf{k}_0^i x_i + \mathbf{k}_1^i\}$ computed over the ring $\mathcal{R}$, reveal $C(x)$ and no additional information about $x \in \mathcal{R}^n$.

The main difference between arithmetic and Boolean garbling is that the input encoding procedure of the former consists of affine functions *over the ring* $\mathcal{R}$, and does not require the bit-representation of the inputs. There are natural information theoretic methods for garbling arithmetic formulas and branching programs over any ring $\mathcal{R}$ [IW14, AIK04]. But garbling general (unbounded depth) arithmetic circuits is significantly more challenging. AIK proposed the first construction supporting *bounded integer computation* – namely computation over integers $\mathcal{R} = \mathbb{Z}$ from a bounded (but possibly exponential) range $[-B, B]$ – based on the Learning With Errors (LWE) assumption. In addition, they presented an alternative construction that generically reduce arithmetic garbled circuits to Yao's Boolean garbled circuits, via a gadget that converts integer inputs into their bit representation using the Chinese Remainder Theorem (CRT). Though general, the CRT-based solution does not satisfy many desiderata of arithmetic garbling, in particular, it still relies on bit-decomposing the inputs and garbling the Boolean circuit implementation of arithmetic operations. So far, the AIK LWE-based construction gives the only known scheme that can garble to general arithmetic circuits without "Booleanizing" them.

3.1.1 Our Results

Despite its importance, little progress were made on arithmetic garbling in the past decade after the work of AIK. In this paper, we revisit this topic and present new ways of arithmetic garbling. Our contributions include *1)* a significantly more efficient arithmetic garbling scheme for bounded integer computations, achieving constant rate, *2)* the first scheme supporting modular arithmetic computation mod p that makes only *black-box* calls to the implementation of arithmetic operations, and *3)* a new way of mixing arithmetic garbling with Boolean garbling. Finally, we diversify the assumptions, showing the Decisional Composite Residuosity (DCR) assumption is also sufficient, in addition to LWE.

Part 1: Constant-Rate Garbling Scheme for Bounded Arithmetic. To highlight our efficiency improvement for bounded integer garbling, we define the *rate* of a garbling scheme to be the maximal ratio between the bit-length of the produced garbled circuit $|\widehat{C}|$ and input encoding, and the bit-length of the tableau of the computation in the clear $|C|\ell$ (i.e., the bit length of merely writing down all the input and intermediate computation values). Let ℓ be bit length of wire values. For a B-bounded integer computation, $\ell = \lceil \log(2B+1) \rceil$.

$$\text{rate } = \max_{C,\mathbf{x}} \frac{|\widehat{C}| + |\mathbf{L}^{\mathbf{x}}|}{|C|\ell}$$

For example, the rate of Yao's garbling for Boolean circuits is $\frac{O((|C'|+|\mathbf{x}|)k_{\text{SKE}})}{|C'|\times(\ell=1)} = O(k_{\text{SKE}})$, where k_{SKE} is the key length of the symmetric key encryption (or PRF) used. For arithmetic garbling, the Boolean baseline of applying Yao's garbling on the Boolean circuit implementation of the arithmetic circuit achieves a rate of $O(\log \ell \cdot k_{\text{SKE}})$, when implementing integer addition/multiplication using the most asymptotically efficient algorithms of complexity $O(\ell \log \ell)$ [HVDH21][2]. The CRT-based construction by AIK reduces arithmetic garbling to Yao's Boolean garbling and achieves the same asymptotic rate $O(\log \ell \cdot k_{\text{SKE}})$ when the circuit size is sufficient large. However, the size of the input labels is $O(n\ell^6 k_{\text{SKE}})$ where n is the number of input elements, which is prohibitive even for relatively small range, say 10-bit,

[2] Note that this approach is entirely impractical for any reasonable length input due to the astronomical constants involved in fast multiplication.

Garbling Scheme	Assumption	Rate	Input Label Size
Boolean Baseline	OWFs	$O(k_{\mathrm{SKE}} \log \ell)$	$O(n\ell k_{\mathrm{SKE}})$
AIK - CRT-based [AIK11]	OWF	$O(k_{\mathrm{SKE}} \log \ell)$	$O(n\ell^6 k_{\mathrm{SKE}})$
AIK - LWE-based [AIK11]	LWE	$O(k_{\mathrm{LWE}})$	$\tilde{O}(n\ell k_{\mathrm{LWE}})$
This work	DCR	$O(1 + \frac{k_{\mathrm{DCR}}}{\ell})$	$O(n(k_{\mathrm{DCR}} + \ell))$

Table 3.1: Comparison of arithmetic garbling for bounded integer computation.

integer computation. The AIK LWE-based construction, on the other hand, has a larger rate of $O(k_{\mathrm{LWE}})$ where k_{LWE} is the LWE dimension, which must be larger than $\ell^{1+\epsilon}$ for some constant $\epsilon \in (0, 1)$. See table 3.1 for a summary.

We show that arithmetic garbling can actually be significantly more efficient than the Boolean baseline. Based on the Decisional Composite Residuosity (DCR) assumption over Paillier groups $\mathbb{Z}^*_{N^{r+1}}$ for $N = pq$ with primes p, q and integer $r \geq 1$ [Pai99, DJ01], we present a scheme producing garbled circuits of size $|\widehat{C}| = O(|C|(\ell + k_{\mathrm{DCR}}))$, and input label of size $|\mathbf{L}^x| = O(n(\ell + k_{\mathrm{DCR}}))$, where $k_{\mathrm{DCR}} = \log N$ is the bit-length of the modulus N. As such, the rate is just a constant $O(1)$ when the integer values are sufficiently large, namely $\ell = \Omega(k_{\mathrm{DCR}})$. To the best of our knowledge, this is the first garbling scheme for general unbounded depth circuits (in any model of computation) that achieve a constant rate, without relying on the strong primitive of iO (see Section 3.1.2 for a more detailed comparison).

Theorem 3.1 (Informal, Arithmetic Garbling for Bounded Integer Computation)**.** *Assume the DCR assumption over $\mathbb{Z}^*_{N^{r+1}}$ for $N = pq$ with primes p, q and r a sufficiently large positive integer. Let $k_{\mathrm{DCR}} = \lfloor \log N \rceil$, $B \in \mathbb{N}$, and $\ell = \lceil (\log 2B + 1) \rceil$. There is an arithmetic garbling scheme for B-bounded integer computation, where the size of the garbled circuit is $|\widehat{C}| = O(|C|(\ell + k_{\mathrm{DCR}}))$ (i.e., rate $O(1 + \frac{k_{\mathrm{DCR}}}{\ell})$), and the length of input label is $O(n(\ell + k_{\mathrm{DCR}}))$ bits.*

Part 2: Arithmetic Garbled Circuit over $\mathbb{Z}_p$. Beyond bounded integer computations, can we support other important models of arithmetic computation? We consider *modular arithmetic computation* over a finite ring $\mathcal{R} = \mathbb{Z}_p$ (where p is not necessarily a prime), which arises naturally in applications, in particular, in cryptosystems.

It turns out that the AIK CRT-based garbling scheme can be adapted to support $\mathbb{Z}_p$-computation[3]. However, as mentioned above, this solution does not satisfy many desiderata of arithmetic garbling, in particular, it makes non-black-box use of the Boolean circuit implementation of arithmetic operations. Though integer multiplication and mod-p reduction are basic operations, there are actually many different algorithms (such as, Karasuba, Tom-Cook, Schönhage–Strassen, Barrett Reduction, Montgomery reduction to name a few), software implementation, and even hardware implementation. It is preferable to avoid applying cryptography to these algorithms/implementation, and have a modular design that can reap the benefits of any software/hardware optimization.

We present an arithmetic garbling scheme for $\mathbb{Z}_p$-computations, which makes only black-box call to the implementation of arithmetic operations.

Theorem 3.2 (Informal, Arithmetic Garbling Scheme for Modular Computation)**.** *Let $p \in \mathbb{N}$ and $\ell = \lceil \log p \rceil$. There are arithmetic garbling schemes for computation over $\mathbb{Z}_p$ that make only black-box use of implementation of arithmetic operations over $\mathbb{Z}_p$, as described below.*

- *Assume DCR. The size of the garbled circuit is $O(|C|(\ell + k_{\mathrm{DCR}})k_{\mathrm{DCR}})$ and the length of input labels is $O(n\ell k_{\mathrm{DCR}})$ bits (i.e., rate $O(k_{\mathrm{DCR}} + \frac{k_{\mathrm{DCR}}^2}{\ell})$).*
- *Assume LWE with dimension k_{LWE}, modulus q, and noise distribution χ that is $\mathrm{poly}(k_{\mathrm{LWE}})$-bounded, such that $\log q = O(\ell) + \omega(\log k_{\mathrm{LWE}})$. The size of the garbled circuit is $|C| \cdot \ell \cdot \tilde{O}(k_{\mathrm{LWE}})$ and the length of input labels is $\tilde{O}(n\ell k_{\mathrm{LWE}})$ bits (i.e., rate $\tilde{O}(k_{\mathrm{LWE}})$).*

[3]This scheme reduces to Yao's garbling by first decomposing the input elements into a bit representation using CRT. As such, this approach works as long as the inputs are integers from a bounded range and the computation can be implemented using Boolean circuits.

We note that being black-box in the implementation of arithmetic operations, is different from being black-box in the ring. The latter has stringent conditions so that a construction that is black-box in the ring can automatically be applied to any ring. Unfortunately, Applebaum, Avron, and Brzuska [AAB15] showed that such garbling is impossible for general circuits. Nevertheless, being black-box in the implementation of arithmetic operations already provides some of the benefits of a modular design. The garbler does not need to choose which algorithm/implementation of arithmetic operations to use, and evaluation can work with any algorithm/implementation.

Part 3: Mixing Bounded Integer and Boolean Computation. Many natural computational tasks mix arithmetic and Boolean computation. For example, a simple neural network component is a (fixed-point) linear functions fed into a ReLU activation functions, where $\mathsf{ReLU}(z) = \max(0, z)$ is much more efficient using (partially) Boolean computation. Even natural arithmetic computational tasks can benefit from (partial) Boolean computation. Take the example of fast exponentiation: given (x, y) one can efficiently compute x^y if one has access to the bits of y, $y_\ell, \ldots, y_0$ using the fact that $x^y = x^{\sum_{i=0}^{\ell} y_i 2^i} = \prod_{i:y_i=1} x^{2^i}$.

This motivates us to consider the following mixed model of computation, represented by a circuit consisting of three types of gates: 1) arithmetic operation gates $+/-/\times : \mathcal{R}^2 \to \mathcal{R}$, 2) Boolean function gates, $g : \{0,1\}^r \to \{0,1\}^{r'}$, where g is implemented using a Boolean circuit, and 3) the bit decomposition gate, $\mathsf{bits} : \mathcal{R} \to \{0,1\}^\ell$, that maps a ring element to its bit representation. Naturally, a Boolean function gate can only take input from the bit decomposition gate or other Boolean function gates (otherwise, there is no restriction on how gates are connected).

We gave a construction for mixed bounded integer and Boolean computation. Our scheme naturally uses Yao's garbled circuit to garble the Boolean function gates, and arithmetic garbled circuit (from Theorem 3.1 or AIK) to garble the arithmetic operation gates over bounded integers. Finally, we design a new gadget for bit decomposition, based on either DCR or LWE.

The bit decomposition gadget is an arithmetic garbling scheme for functions of form $\mathsf{BD}_{\{\mathbf{c}_j,\mathbf{d}_j\}}$ that maps an integer $x \in [-B, B]$ to ℓ labels, where the j'th label is $\mathbf{c}_j\mathsf{bits}(x)_j + \mathbf{d}_j$. This means given the garbled circuit $\widehat{\mathsf{BD}}$ and input label $\mathbf{a}x + \mathbf{d}$, the output labels are revealed and nothing else.

Our scheme puts together the above three components in a modular and black-box way. In terms of efficiency, the size of the garbled circuit naturally depends on the number of gates of each type. More specifically, garbling the Boolean computation gates incurs an rate of $O(k_{\text{SKE}})$ inherited from Yao's garbled circuit, whereas the arithmetic operation gates can be garbled with close to constant rate if using our DCR-based scheme in Theorem 3.1. Our bit decomposition gadget produces a garbled circuit of size $O(\ell^2 \cdot k_{\text{DCR}})$ for sufficiently large integers $\ell = \Omega(k_{\text{DCR}})$ if based on DCR, and of size $\ell^2 \cdot \tilde{O}(k_{\text{LWE}})$ if based on LWE, where k_{LWE} is the LWE dimension. Recall that the AIK CRT-based scheme also relies on performing bit decomposition, however, at a much larger cost of $O(\ell^6 k_{\text{SKE}})$.

Theorem 3.3 (Informal, Arithmetic Garbling Schemes for Mixed Computation)**.** *Let $B \in \mathbb{N}$ and $\ell = \lceil \log 2B + 1 \rceil$. There are arithmetic garbling schemes for mixed B-bounded integer and Boolean computation as described below.*

- *Assume DCR. The size of the garbled circuit is $O(s_b k_{\text{DCR}} + m_a(\ell + k_{\text{DCR}}) + m_b(\ell + k_{\text{DCR}})^2 \cdot k_{\text{DCR}})$, where s_b is the total circuit size of all Boolean function gates, m_a the number of arithmetic operation gates, and m_b the number of bit-decomposition gates. The length of input label is $O(n(\ell + k_{\text{DCR}}))$ bits.*
- *Assume LWE with dimension k_{LWE}, modulus q, and noise distribution χ that is* $\text{poly}(k_{\text{LWE}})$*-bounded, such that $\log q = O(\ell) + \omega(\log k_{\text{LWE}})$. The size of the garbled circuit is $s_b O(\lambda) + m_a \cdot \ell \cdot \tilde{O}(k_{\text{LWE}}) + m_b \cdot \ell^2 \cdot \tilde{O}(k_{\text{LWE}})$. The length of input label is $O(n\ell k_{\text{LWE}})$ bits, where ϵ is a fixed constant.*

Potential for Concrete Efficiency Improvement. The primary goal of this work is designing new arithmetic garbling with good asymptotic efficiency. Though we do not

Computation	Assumption	Rate	Input Label Size
Bounded Arithmetic	DCR	$O(1 + k_{\text{DCR}}/\ell)$	$O(n(k_{\text{DCR}} + \ell))$
Mod p	DCR	$O(k_{\text{DCR}} + k_{\text{DCR}}^2/\ell)$	$O(nk_{\text{DCR}}\ell))$
Mod p	LWE	$\tilde{O}(k_{\text{LWE}})$	$\tilde{O}(n\ell k_{\text{LWE}})$
Mixed	DCR	$O((\ell + k_{\text{DCR}})k_{\text{DCR}})$*	$O(n(\ell + k_{\text{DCR}}))$
Mixed	LWE	$\tilde{O}(\ell k_{\text{LWE}})$*	$O(n\ell k_{\text{LWE}})$

*Rate of Mixed Computation Schemes depends on relative frequency of gate types. Numbers here conservatively assume all gates are the most expensive type.

Table 3.2: Summary of our garbling schemes.

focus on optimizing concrete efficiency, our DCR-based schemes do show potential towards practical garbling. Our concrete analysis demonstrates that when the input domains are large, $\ell \sim k_{\text{DCR}} = 4096$ bits, the size of garbled circuits produced by our constant-rate bounded integer garbling scheme is significantly smaller than that of the Boolean baseline using the state-of-the-art Boolean garbling scheme of [RR21] – the garbling size of addition is ∼100× smaller, and the size of multiplication is ∼500× smaller. See Section 3.10.

3.1.2 Related Works

We briefly survey approaches to garbling Boolean circuits that achieve good rate.

AIK showed that their LWE-based scheme when applied to constant-degree polynomials represented as a sum of monomials has constant-rate. The work of [AJS17] yields a garbling scheme with size $O(|C|)+\text{poly}(\lambda)$ and input size $O(n+m+\text{poly}(\lambda))$, assuming subexponentially secure indistinguishability obfuscation and rerandomizable encryption.

The work of [BGG+14, GKP+13a] presents a $O(|C| + \text{poly}(\lambda, d))$-size garbling of Boolean circuits, with input labels of size $O(nm\text{poly}(\lambda, d))$ where d is the circuit depth, n is the input length, m is the output length, and λ the security parameter. One significant advantage

of their scheme is that the circuit description is given in the clear. We analyze the sizes of garbled circuits and input labels when using their scheme to garble a B-bounded integer computation (C, x) of depth d, in particular, spelling out the exponent in the poly term. For simplicity of notation, we set the input length n, output length m, wire-value bit length $\log B = \ell$, and the size of a FHE bit encryption all to $O(k)$.

$$\text{[BGG}^+\text{14, GKP}^+\text{13a]:} \qquad |\tilde{C}| + |\mathbf{L}^{\mathbf{x}}| > |C| + \tilde{O}(k^3 d^6 + k^6 d^4)$$

$$\text{Our DCR-based scheme:} \qquad |\tilde{C}| + |\mathbf{L}^{\mathbf{x}}| = O(|C|k)$$

In comparison, the garbling of [BGG+14, GKP+13a] has smaller size when k and d are sufficiently small comparing with $|C|$, achieving even sub-constant rate $O(|C|/k)$. However, our garbled circuits are smaller when k and d are larger, achieving a constant rate for all k and d. The term $\tilde{O}(k^3 d^6 + k^6 d^4)$ associated with [BGG+14, GKP+13a] is prohibitive, even for small k, d such as 100, whereas the complexity of our scheme does not have such large exponents. Our scheme is also simpler than [BGG+14, GKP+13a], which combines ABE, FHE, and Yao's garbled circuit in an intricate way.

The works of [BMR16, BCM+19] generalized FreeXOR [KS08b], a technique that allows one to garble XOR gates at zero cost, to general arithmetic setting. They present a scheme for bounded integer computation where addition is for free. They also present a gadget (similar to our bit decomposition gadget) that converts integers to a primorial-mixed-radix representation, which has similar advantage as a Boolean representation (e.g. cheap comparisons). Leveraging free addition, they show that their scheme has concrete performance benefit for certain bounded arithmetic computations, in comparison to directly applying Boolean garbling to arithmetic circuits. However, their construction is not arithmetic; in particular, the input encoding requires a "bit representation" of the inputs.

Finally, the work of [AIKW13] describes a method for generically shortening the length of input labels to $|\mathbf{L}^{\mathbf{x}}| = n\ell + o(n\ell)$ – that is, rate-1 input labels. However, the transformation does not preserve decomposability, which is a property that each input element x_i is encoded separately $\mathbf{L}^i(x_i)$. Many applications of garbling rely on decomposability, e.g., in 2PC, the

party holding x_i can use OT/OLE to obtain $\mathbf{L}^i(x_i)$. The encoding of our schemes, AIK, and Yao's garbled circuits all satisfy decomposability, and our DCR-based bounded integer garbling has the shortest input encoding (see Table 3.3).

Organization. In Section 7.2, we give an overview of our techniques. In Section 3.3, we define three models of computations, bounded integer, modular arithmetic, and mixed computation, our garbling scheme, and the key extension, arithmetic computation and bit decomposition gadgets. In Section 3.4, we introduce a linearly homomorphic encryption scheme (LHE) as a tool for constructing the gadgets. In Section 3.5, 3.7, and 3.8, we construct key extension gadgets in the bounded integer and the modular arithmetic models, and a bit decomposition gadget in the mixed model respectively. We construct the overall garbling schemes for all three models in Section 3.9. In Section 3.10, we compare the concrete efficiency of our scheme with the scheme of [BMR16] and the Boolean baseline using [RR21].

3.2 Technical Overview

We start with reviewing the modular design paradigm of AIK, which is the basis of our approach.

As an arithmetic analog of Yao's Boolean garbled circuits, the AIK garbling shares a similar high-level structure. Like Yao's scheme, AIK's scheme associates each wire value, x_i, with a wire label, $\mathbf{L}^i$, (which hides/encrypts the wire value).[4] Also like Yao's scheme, the Garbler generates "garbled tables" that enable an evaluator holding a wire label for each input wire to a gate in the circuit to derive the corresponding output wire label. However, unlike in Yao's scheme, the tables do not directly correspond to encryptions of the output wire labels under all possible input label pairs.

Instead, AIK builds bounded arithmetic garbled circuits in two steps: (1) they construct an information-theoretically secure garbling scheme for low depth arithmetic circuits over a ring $\mathcal{R}$ (via black-box use of $\mathcal{R}$), (2) they then construct a key extension gadget for bounded

[4] In Yao's scheme, these labels may be chosen independently and uniformly at random. In the arithmetic setting, this is infeasible as the domain may be exponentially large.

arithmetic computation that allows them to efficiently circumvent the depth restriction (the key extension gadget makes non-black-box use of $\mathcal{R}$, but the overall garbling scheme makes use of the key extension gadget in a black-box way).

To begin, let us recall how AIK construct (1) the information-theoretic scheme. This scheme does away with garbled gate information entirely, at the expense of long input labels whose structure depends explicitly on the circuit being garbled. In particular, for every wire of the circuit, the Garbler generates two keys $\mathbf{k}_0^i, \mathbf{k}_1^i$ which are vectors in $\mathcal{R}$. During evaluation, for every wire, the evaluator should obtain a label $\mathbf{L}^i = \mathbf{k}_0^i x_i + \mathbf{k}_1^i$ corresponding to the correct value of the wire as follows:

- *Input Labels:* For each input wire, its label is given to the evaluator.
- *Garbled Gate:* For every gate $x_i = g(x_{j_1}, x_{j_2})$, the invariant is that given the labels $\mathbf{L}^{j_1}, \mathbf{L}^{j_2}$ corresponding to inputs x_{j_1} and x_{j_2}, the evaluator can learn a label $\mathbf{L}^i$ corresponding to the output x_i for each output wire, and no other information. This is achieved using the *arithmetic computation gadget* described in AIK, which are essentially information theoretically secure DARE (Decomposable Affine Randomized Encoding) for functions $f_{+,\mathbf{k}_0^i,\mathbf{k}_1^i}(x_{j_1}, x_{j_2}) = \mathbf{k}_0^i(x_{j_1} + x_{j_2}) + \mathbf{k}_1^i$ and $f_{\times,\mathbf{k}_0^i,\mathbf{k}_1^i}(x_{j_1}, x_{j_2}) = \mathbf{k}_0^i(x_{j_1} \times x_{j_2}) + \mathbf{k}_1^i$. They are summarized in Figure 3.1.[5]

  Remark: Having separate gadgets for addition and multiplication leaks the type of gate. There also exists an universal garbling gadget for arithmetic operation, which hides the gate operation, so that only the topology of the circuit is revealed.
- *Outputs:* For each output wire, the evaluator learns $\mathbf{L}^i = \mathbf{k}_0^i x_i + \mathbf{k}_1^i$, which reveals the output x_i by setting $\mathbf{k}_0^i = 1$ and $\mathbf{k}_1^i = 0$.

The above paradigm gives an information-theoretic arithmetic garbling scheme, however, only for logarithmic depth circuits. Its major issue is that the key-length increases

[5]Note that while the evaluator can efficiently evaluate the garbled circuit from the bottom-up (inputs to outputs), the garbler (as described here) proceeds from the top-down: generating labels for the output wires and then recursively generating increasingly complex keys for the wire layers below.

exponentially in the depth of the circuit, because 1) the key-length of the input wires of a multiplication gate is twice the key-length of its output wire, and 2) the key-length of input wires of any gate grows linearly with the fan-out that gate. On the flip side, this scheme has constant overhead for constant depth circuits.

To go beyond low-depth circuits, AIK introduced a *key-extension gadget* — a DARE for functions $f_{\mathrm{KE},\mathbf{c},\mathbf{d}}(x) = \mathbf{c} \cdot x + \mathbf{d}$. It ensures that given the input label $\mathbf{a} \cdot x + \mathbf{b}$ and garbled table, the evaluator can obtain a new *longer* label $\mathbf{c} \cdot x + \mathbf{d}$, and no other information. Now to support arbitrary depth circuit, AIK uses the arithmetic operation gadgets to handle the computation gates, and whenever the key length $|\mathbf{c}|, |\mathbf{d}|$ becomes too long, it uses the key-extension gadget to shrink the key length down $|\mathbf{a}|, |\mathbf{b}| < |\mathbf{c}|, |\mathbf{d}|$.

It may seem counter-intuitive that a key "extension" gadget would be used to "shrink" keys, so let us discuss how this works in slightly more detail. First, recall that the information-theoretic DARE gadgets described in Figure 3.1 derive (possibly longer) labels for the inputs to a gate from the output labels corresponding to that gate. Next, we break each wire i into two sub wires: the part that comes *out* of the preceding gate, i^{out}, and the part that goes *into* the next gate, i^{in} (for higher fan-out there will other i^{in} wires). By breaking up all wires in this manner, we can garble gates in parallel (as opposed to from the top-down) by independently and locally (a) sampling the (short) labels $\mathbf{L}^{i^{\mathrm{out}}}$, and (b) locally applying the gadgets from Figure 3.1 to derive (long) input labels $\mathbf{L}^{j_1^{\mathrm{in}}}, \mathbf{L}^{j_2^{\mathrm{in}}}$. At this point each wire value is now associated with two labels: a short output label and long input label(s). The key extension gadget allows the evaluator to derive the long input portion(s) from the short input label portion (using some extra information: the gabled table).

Therefore, this paradigm reduces the problem of constructing constant-overhead arithmetic garbling for bounded integer computation (Theorem 3.1) and arithmetic garbling for modular computation (Theorem 3.2) to the problem of designing (efficient) key-extension gadgets for the respective model of computation.

Abstract Key-Extension Gadget. Instead of describing AIK's gadget, we will instead introduce an abstract approach to constructing key-extension gadgets (that also captures

AIK's key-extension gadget). Instantiating this approach has encounter significant technical barriers (discussed at length below), but we believe the high level paradigm is nonetheless instructive.

Recall that to construct a key-extension gadget the garbler knows *short* keys $(\mathbf{a}, \mathbf{b})$ corresponding to *short* wire labels of the form $\mathbf{S}_x = \mathbf{a} \cdot x + \mathbf{b}$ as well as *long* keys $(\mathbf{c}, \mathbf{d})$ corresponding to *long* wire labels of the form $\mathbf{L}_x = \mathbf{c} \cdot x + \mathbf{d}$. The garbler's task is to output some succinct information, tb, so that an evaluator holding a short wire label $\mathbf{S}_x$ can derive the long wire label corresponding to the same value $\mathbf{L}_x$ without learning anything about the other wire labels $\mathbf{L}_y$ (for $y \neq x$).

As a warm up, observe that the Yao's approach can be adapted to give an efficient key extension gadget for small domains. In particular for the boolean case of $x \in \{0, 1\}$, the garbler can simply set tb to consist of two (one-time symmetric key) encryptions: $\mathsf{Enc}_{\mathbf{b}}(\mathbf{d}) = \mathsf{Enc}_{\mathbf{S}_0}(\mathbf{L}_0)$ and $\mathsf{Enc}_{\mathbf{a}+\mathbf{b}}(\mathbf{c} + \mathbf{d}) = \mathsf{Enc}_{\mathbf{S}_1}(\mathbf{L}_1)$ (randomly permuted). Using tb the evaluator can simply decrypt the relevant ciphertext (using the short label as a key) to derive the long label corresponding to the same value. Semantic security implies that the evaluator learns nothing about the other label.

Unfortunately, it is not clear how to extend Yao's approach to large arithmetic domains (with succinct garbled tables). Instead, it seems we need a stronger arithmetic properties from the encryption scheme. In particular, assume we have an encryption scheme, $(\mathsf{Enc}, \mathsf{Dec})$, which is *linearly homomorphic in both the key and message space*: there are operations $\boxplus, \boxtimes$ such that $x \boxtimes \mathsf{Enc}_{\mathbf{a}}(\mathbf{c}) \boxplus \mathsf{Enc}_{\mathbf{b}}(\mathbf{d}) = \mathsf{Enc}_{\mathbf{a}x+\mathbf{b}}(\mathbf{c}x + \mathbf{d})$.

Given such an encryption scheme, consider the case that the wire value x is *public* (we will relax this assumption momentarily). Then note that given a garbled table, tb, comprised of just two cipher texts $\mathsf{Enc}_{\mathbf{a}}(\mathbf{c})$ and $\mathsf{Enc}_{\mathbf{b}}(\mathbf{d})$ the evaluator can use $x, \mathbf{S}_x$ to derive a long label $\mathbf{L}_x$ by homomorphically evaluating $x \boxtimes \mathsf{Enc}_{\mathbf{a}}(\mathbf{c}) \boxplus \mathsf{Enc}_{\mathbf{b}}(\mathbf{d}) = \mathsf{Enc}_{\mathbf{a}x+\mathbf{b}}(\mathbf{c}x + \mathbf{d}) = \mathsf{Enc}_{\mathbf{S}_x}(\mathbf{L}_x)$ and decrypting. We need to additionally show that the evaluator learns nothing about the other output labels. In more detail, observe that we can simulate the view of evaluator holding $\mathbf{S}_x, \mathbf{L}_x, x$ which is comprised of 3 cipher texts: (1) $\mathsf{Enc}_{\mathbf{a}}(\mathbf{c})$, (2) $\mathsf{Enc}_{\mathbf{b}}(\mathbf{d})$, and (3)

$\mathsf{Enc}_{\mathbf{S}_x}(\mathbf{L}_x)$. First, note that given $\mathbf{L}_x, x$, one can derive ciphertext (2) from ciphertexts (1) and (3) (and x) by simply homorphically computing $\mathsf{Enc}_{\mathbf{S}_x}\mathbf{L}_x) \boxminus \mathsf{Enc}_{\mathbf{a}}(\mathbf{c}) \boxtimes x = \mathsf{Enc}_{\mathbf{b}}(\mathbf{d})$. Armed with this observation we can invoke semantic security and simply simulate (given $\mathbf{S}_x, \mathbf{L}_x, x$) by encrypting (3) honestly, replacing (1) with a random encryption, and homomorphically evaluating (2) from the other two ciphertexts.

There are two issues with this approach: the first (which we have already mentioned) is that the wire label is public, the second (and more subtle issue) is that we are implicitly assuming that encryption scheme has a key space that is identical to the message space which is in fact the ring $\mathcal{R}$ we wish to compute over. We will describe a generic approach to dealing with the first issue here, but leave the second issue to the specific settings and implementations below.

We observe that one can effectively assume the wire label is public without loss of generality. The idea is that instead of extending the wire value x directly, we will mask x with a random value, r, that is known to the garbler to get $x' = x + r$. Note that x' can be safely output by the garbled circuit while statistically hiding x. Then we can use our key extension gadget to extend x'. Then once we have a long label $\mathbf{L}_{x'}$ we can easily use another gadget to remove r (known to the garbler).[6]

Key Extension Gadget for Bounded Integer Computation. Our first key extension gadget relies on the Paillier extension of the Paillier encryption [Pai99, DJ01]. This gadget is very efficient: the input label only consists of $O(1)$ ring elements and the table size is proportional to the output label size.

We use a *one-time secure* version of the Paillier encryption. To generate the public parameters, sample two large safe primes and let N be the product of the two safe primes. Choose a small integer $\zeta \geq 1$, and the ciphertexts are vectors modulo $N^{\zeta+1}$. The group $\mathbb{Z}^*_{N^{\zeta+1}}$ contains a hard subgroup of unknown order (i.e., the $2N^{\zeta}$'th residue subgroup, the order of which is hard to compute given N) and an easy subgroup of order N^{ζ} generated

[6]Similar ideas are found in the well-known "half-gates" construction [ZRE15] of Zahur, Rosulek, and Evans for garbling boolean circuits comprised of XOR and AND gates.

by $1+N$, in which discrete logarithm is easy. The public parameters are $(N, \zeta, \mathbf{g})$, where $\mathbf{g} = (g_1, g_2, \ldots, g_\psi)$ are randomly-sampled generators of the "hard" subgroup. The one-time use key s is an integer sampled uniformly from $\{0, \ldots, N\}$. The encryption algorithm takes a message vector $\mathbf{m} \in \mathbb{Z}_{N^\zeta}^{\psi}$ of dimension at most ψ as the input message, and generates a ciphertext as follows:

$$\mathsf{Enc}(s, \mathbf{m}) = \mathbf{g}^s \cdot (1+N)^{\mathbf{m}} = (g_1^s \cdot (1+N)^{m_1}, \ldots, g_\psi^s \cdot (1+N)^{m_\psi}) \ .$$

The Decisional Composite Residuosity (DCR) assumption implies that the ciphertext is pseudorandom. Indeed, the secret key can only be used once; in fact, the encryption algorithm is deterministic.

For our application, the following properties of the Paillier encryption are important:

- *Small Keys:* the secret key s is an integer upper bounded by N which is much smaller than the message space modulus N^ζ.
- *Linear Homomorphism:* for any keys $s_1, s_2 \in \mathbb{Z}$ and messages $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}_{N^\zeta}^{\psi}$,

$$\mathsf{Enc}(s_1, \mathbf{m}_1) \cdot \mathsf{Enc}(s_2, \mathbf{m}_2) = \mathsf{Enc}(\underbrace{s_1 + s_2}_{\text{over } \mathbb{Z}}, \underbrace{\mathbf{m}_1 + \mathbf{m}_2}_{\text{over } \mathbb{Z}_{N^\zeta}}).$$

  In particular, given ciphertexts $\mathsf{Enc}(s_1, \mathbf{c}), \mathsf{Enc}(s_2, \mathbf{d})$ and x, one can homomorphically compute $\mathsf{Enc}(s_1 x + s_2, \mathbf{c}x + \mathbf{d})$.
- *Integer Keys:* To decrypt the output ciphertext produced by the homomorphic evaluation, we need the key $s_1 x + s_2$. Importantly, since the order of the hard group is unknown, we can only hope to use the key $s_1 x + s_2$ computed over $\mathbb{Z}$.

The above observations immediately suggest a naïve construction of key extension gadget: Let $\mathsf{Enc}(s_1, \mathbf{c}), \mathsf{Enc}(s_2, \mathbf{d})$ be the garbled table, and $(x, s_1 x + s_2)$ computed over $\mathbb{Z}$ be the input label. Decryption gives $\mathbf{c} \cdot x + \mathbf{d} \bmod N^\zeta$ as desired. However, such a naïve construction faces two problems:

- *Input label over* $\mathbb{Z}$. The output label is in ring $\mathbb{Z}_{N^\zeta}$. We will set $N^\zeta \gg B$ to be sufficiently large so that a B-bounded computation can be "embeded" in computation modulo

N^ζ. As such, arithmetic operations can be garbled using AIK arithmetic operation gadgets in Figure 3.1 with modulus N^ζ. However, a problem is that to decrypt Paillier encryption, the input label $s_1x + s_2$ must be computed over $\mathbb{Z}$. To close the gap, we crucially rely on the fact that in bounded integer computation, every wire value x is bounded. We can also sample s_1, s_2 from a bounded range so that $s_1x + s_2 < N^\zeta$. Therefore, the input label can be $(x, s_1x + s_2) \bmod N^\zeta = (x, s_1x + s_2)$ over $\mathbb{Z}$.

- *Leakage.* In the naïve construction, x is revealed. To hide x, we replace x by $y = x + r$, a one-time pad of x. Let $(y, s_1y + s_2)$ be the input label, let $\mathsf{Enc}(s_1, \mathbf{c}), \mathsf{Enc}(s_2, \mathbf{d} - r\mathbf{c})$ be the table. The evaluator homomorphically computes $\mathsf{Enc}(s_1y + s_2, \mathbf{c}y + \mathbf{d} - r\mathbf{c})$, then decrypts $\mathbf{c}y + \mathbf{d} - r\mathbf{c} = \mathbf{c}x + \mathbf{d}$.

For clarity, we sketch how this works. Say the wire value x is guaranteed to be bounded by $-B \leq x \leq B$. Sample $r \leftarrow \{-B', \ldots, B'\}$ for some $B' \gg B$, thus $r + x$ statistically hides x. Sample $s_1 \leftarrow \{0, \ldots, N\}$. Sample $s_2 \leftarrow \{0, \ldots, B''\}$ for some $B'' \gg NB'$, so that $s_1(r + x) + s_2$ statistically hides $s_1(r + x)$, which in turn preserves semantic security for encryptions under s_1.[7] Choose ζ so that $N^\zeta > 2B''$. Overall, the gadget consists of the following:

$$\begin{aligned} \text{Input Key: } & \mathbf{a} = (1, s_1) \quad \mathbf{b} = (r, s_1r + s_2) \\ \text{Input Label: } & \mathbf{L}^{\text{in}} = (r + x, s_1(r + x) + s_2) \\ \text{Garbled Table: } & \mathsf{Enc}(s_1, \mathbf{c}) \quad \mathsf{Enc}(s_2, \mathbf{d} - r\mathbf{c}) \ . \end{aligned}$$

We observe that the garbled table has "constant-rate", which is the key leading to constant-rate garbled circuit. More precisely, the size of the above garbled table is $|\mathbf{c}|(\zeta + 1)\log N$. When the integer bound B is sufficiently large, it suffices to set the modulus N to be a constant times longer than B, i.e., $\log N = O(\log B)$. In addition, the dimension of the output key $|\mathbf{c}|$ is proportional to the fan out k of the wire with value x. Therefore, the garbled

[7] We do not need protect s_2 because the corresponding ciphertext can be simulated using the ciphertext encrypted under s_1 and the output label $\mathbf{c}x + \mathbf{d}$.

table has size $|\mathbf{c}|(\zeta+1)\log N = O(k \log B)$, incurring a constant overhead. See Section 3.5 for more details.

Key Extension Gadget for Modulo-p Computation. There are two barriers when we try to extend the previous key extension gadget to the modulo-p computation setting.

- *Arbitrary Message Ring $\mathbb{Z}_p$.* In the Paillier encryption, the message is a vector over ring $\mathbb{Z}_{N^\zeta}$. It supports linear homomorphic evaluation modulo N^ζ, where N is the product of two randomly sampled primes. But we need to perform computation modulo p, where p is an arbitrary integer specified by the given arithmetic circuit.
- *The Input Label over $\mathbb{Z}$.* The AIK arithmetic operations gadgets now uses keys and labels over $\mathbb{Z}_p$. However, as discussed above, to decrypt Paillier encryption, we need the input label $s_1 y + s_2$ to be computed over $\mathbb{Z}$, where y now equals to $(r+x) \bmod p$). In the previous setting, we get around this problem easily because the wire value x is bounded, and hence computing $s_1 y + s_2$ modulo N^ζ is the same as computing it over the integers. Now, the wire value x could be an arbitrary element in $\mathbb{Z}_p$, certainly $s_1 y + s_2 \bmod p$ is very different from $s_1 y + s_2$ over $\mathbb{Z}$. We need a new technique to recover the latter.

To overcome the first barrier, we construct another encryption scheme on top of Paillier encryption, such that the message space is over $\mathbb{Z}_p$. The new encryption scheme is defined as

$$\overline{\mathsf{Enc}}(s, \mathbf{m}) = \mathsf{Enc}(s, \lfloor \mathbf{m} \cdot \tfrac{N^\zeta}{p} \rceil), \qquad \overline{\mathsf{Dec}}(s, \mathbf{c}) = \lfloor \mathsf{Dec}(s, \mathbf{c}) \cdot \tfrac{p}{N^\zeta} \rceil .$$

The new scheme satisfies a weaker form of linear homomorphism. Notice that for any $m_1, m_2 \in \mathbb{Z}_p$,

$$\lfloor m_1 \cdot \tfrac{N^\zeta}{p} \rceil + \lfloor m_2 \cdot \tfrac{N^\zeta}{p} \rceil = \lfloor (m_1 + m_2) \cdot \tfrac{N^\zeta}{p} \rceil + e$$

for some $e \in \{-1, 0, 1\}$. Therefore, for any $s_1, s_2 \in \mathbb{Z}$ and $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}_p^\psi$,

$$\overline{\mathsf{Enc}}(s_1, \mathbf{m}_1) \cdot \overline{\mathsf{Enc}}(s_2, \mathbf{m}_2) = \overline{\mathsf{Enc}}(\underbrace{s_1 + s_2}_{\text{over } \mathbb{Z}}, \underbrace{\mathbf{m}_1 + \mathbf{m}_2}_{\text{modulo } p}) \cdot (1+N)^{\mathbf{e}}$$

for some $\mathbf{e} \in \{-1,0,1\}^{\psi}$, and it can be correctly decrypted to $\mathbf{m}_1 + \mathbf{m}_2$ given key $s_1 + s_2$, by simply decrypting according to Paillier and rounding the result to the nearest multiple of N^{ζ}/p. The homomorphic evaluation can be extended to any linear function $f(x_1, \ldots, x_\ell) = c_1 x_1 + \cdots + c_\ell x_\ell$. For any $s_1, \ldots, s_\ell \in \mathbb{Z}$ and $\mathbf{m}_1, \ldots, \mathbf{m}_\ell \in \mathbb{Z}_p^{\psi}$,

$$\mathsf{Dec}\left(f(s_1, \ldots, s_\ell), \prod_{i=1}^{\ell} \overline{\mathsf{Enc}}(s_i, \mathbf{m}_i)^{c_i}\right) = f(\mathbf{m}_1, \ldots, \mathbf{m}_\ell)$$

as long as $|f|_1 = \sum_i |c_i| \ll \frac{N^{\zeta}}{p}$. Otherwise, if the magnitude of the coefficients are large, then the accumulation of the rounding error may break correctness.

In the main body, we also present an alternative construction of linear homomorphic encryption scheme based on the LWE assumption.

Now, using such a linear homomorphic encryption scheme (whose message space is over $\mathbb{Z}_p$), we construct our key extension gadget: Sample random $r \in \mathbb{Z}_p$ and let $y = x + r \bmod p$ be the the one-time pad of x. Sample random $\mathbf{s}_1 \in \{0,1\}^{\ell}$, $\mathbf{s}_2 \in \{0, \ldots, \lfloor p/2 \rfloor\}^{\ell}$. We set the input label as

$$\mathbf{L}^{\text{in}} = (y, \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2) \mod p\,.$$

Also define

$$\mathbf{s}_{\text{res}} = \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2 \mod p\,,$$
$$\mathbf{s}'_{\text{res}} = \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2 \text{ (over } \mathbb{Z})\,.$$

Then $\mathbf{L}^{\text{in}} = (y, \mathbf{s}_{\text{res}})$ and $\mathbf{s}_{\text{res}} = \mathbf{s}'_{\text{res}} \bmod p$.

Our key observation is that, given $\mathbf{L}^{\text{in}} = (y, \mathbf{s}_{\text{res}})$, one can recover $\mathbf{s}'_{\text{res}}$.

Let $s_{\text{res},i}$ (resp. $s'_{\text{res},i}, s_{1,i}, s_{2,i}$) denote the i-th coordinate of $\mathbf{s}_{\text{res}}$ (resp. $\mathbf{s}'_{\text{res}}, \mathbf{s}_1, \mathbf{s}_2$). Then

$$s'_{\text{res},i} = s_{1,i} y + (1 - s_{1,i}) \cdot \lfloor p/2 \rfloor + s_{2,i} = \begin{cases} y + s_{2,i}, & \text{if } s_{1,i} = 1\,, \\ \lfloor p/2 \rfloor + s_{2,i}, & \text{if } s_{1,i} = 0\,. \end{cases} \tag{3.1}$$

As illustrated by Figure 3.2,

- In case $y < p/2$, we have $0 \leq s'_{\text{res},i} < p$, thus $s'_{\text{res},i} = s_{\text{res},i}$.
- In case $y > p/2$, we have $\lfloor p/2 \rfloor \leq s'_{\text{res},i} < \lfloor p/2 \rfloor + p$, thus $s'_{\text{res},i}$ can also be recovered from $s_{\text{res},i}$.

Therefore,

$$s'_{\text{res},i} = \begin{cases} s_{\text{res},i} + p, & \text{if } y > p/2 \text{ and } s_{\text{res},i} < \lfloor p/2 \rfloor, \\ s_{\text{res},i}, & \text{otherwise.} \end{cases}$$

Since the evaluator can recover $\mathbf{s}'_{\text{res}} = \mathbf{s}_1 y + (1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2$, if the table consists of

$$\text{“}\mathsf{Enc}(\mathbf{s}_1, \mathbf{c})\text{” and “}\mathsf{Enc}((1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2, \mathbf{d} - r\mathbf{c}),\text{”}$$

then the evaluator can homomorphically compute $\mathsf{Enc}(\mathbf{s}'_{\text{res}}, \mathbf{c}x + \mathbf{d})$ and decrypt it to get $\mathbf{c}x + \mathbf{d}$.

To formalize this idea, there are a few problems we have to overcome.

Problem 1: Format Mismatch. In the linear homomorphic encryption scheme, the key should be an integer sampled from a large interval. While $\mathbf{s}_1$ is a vector consisting of 0's and 1's. To close the gap, we introduce a linear function $\mathsf{Lin} : \mathbb{Z}^\ell \to \mathbb{Z}$ to compress the length and to increase the magnitude. For example, if we let $\mathsf{Lin}(s_1, s_2, \ldots, s_\ell) = s_1 + 2s_2 + 2^2 s_3 + 2^3 s_4 + \ldots$, then $\mathsf{Lin}(\mathbf{s}_1)$ is the uniform distribution over $\{0, \ldots, 2^\ell - 1\}$ since $\mathbf{s}_1$ is sampled uniformly from $\{0,1\}^\ell$.

Let the table be

$$\mathsf{Enc}\big(\mathsf{Lin}(\mathbf{s}_1), \mathbf{c}\big), \quad \mathsf{Enc}\big(\mathsf{Lin}((1 - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{s}_2), \mathbf{d} - r\mathbf{c}\big).$$

The evaluator homomorphically computes $\mathsf{Enc}(\mathsf{Lin}(\mathbf{s}'_{\text{res}}), \mathbf{c}x + \mathbf{d})$ and decrypts it to get $\mathbf{c}x + \mathbf{d}$.

After the introduction of Lin, the construction satisfies the correctness requirement. From now on, we will focus on the privacy issues.

Problem 2: the Leakage of $\mathbf{s}_1$. As shown by Equation (3.1) and illustrated in Figure 3.2,

$$s_{1,i} = 1 \implies s'_{\text{res},i} \text{ is uniform in } [y, y + p/2),$$
$$s_{1,i} = 0 \implies s'_{\text{res},i} \text{ is uniform in } [p/2, p).$$

Therefore, $s_{1,i}$ is hidden only if $s'_{\text{res},i} \in [y, y+p/2) \cap [p/2, p)$. Otherwise, when $s'_{\text{res},i} \notin [y, y+p/2) \cap [p/2, p)$, the value of $s_{1,i}$ is leaked by $s'_{\text{res},i}$. For example, in the most extreme case when $y = 0$, the value of $\mathbf{s}_1$ is completely leaked by $\mathbf{s}'_{\text{res}}$.

We will later discuss how to repair the construction when y is close to zero. For now, let us assume $y \in (p/4, 3p/4)$. Under such assumption, for each i, there is a $\geq 50\%$ chance that $s_{1,i}$ is not revealed by $\mathbf{s}'_{\text{res}}$.

For privacy of the encryption scheme, we require that $\mathsf{Lin}(\mathbf{s}_1)$ is "sufficiently random" conditioning on $\mathbf{s}'_{\text{res}}$. In Section 3.6, we construct a (seeded) linear function Lin, such that with overwhelming probability, $\mathsf{Lin}(\mathbf{s}_1)$ *smudges*[8] the uniform distribution over $\{0, \ldots, N\}$.

As analyzed in Section 3.6, let $\mathsf{Lin}(s_1, s_2, \ldots, s_\ell) = \sum_i c_i s_i$, where the coefficients $c_1, \ldots, c_\ell$ are i.i.d. sampled from $\{0, \ldots, N\}$. Then as long as $\ell \geq \log N$, $\mathsf{Lin}(\mathbf{s}_1)$ will smudge the uniform distribution over $\{0, \ldots, N\}$ even if about half of the coordinates of $\mathbf{s}_1 \in \{0,1\}^\ell$ are revealed. Here Lin is essentially a randomness extractor that is linear over $\mathbb{Z}$.

Problem 3: the "Bad" Values of y. So far, we have constructed a key extension gadget that works well when the one-time pad $y = x + r \bmod p$ is in $(p/4, 3p/4)$, but it has serious privacy issue if $y \in [0, p/4) \cup (3p/4, p)$.

To close the leakage, we repeat the gadget one more time. This time use a different one-time pad $\tilde{y} = x + r + \lfloor p/2 \rfloor \bmod p$. Note that, y lies in the "bad" region $[0, p/4) \cup (3p/4, p)$ if and only if $\tilde{y}$ is in the "good" region $(p/4, 3p/4)$.

In greater detail, sample random $r \in \mathbb{Z}_p$ and let

$$y = x + r \bmod p, \qquad \tilde{y} = y + r + \lfloor p/2 \rfloor \bmod p\,.$$

[8]Formally, $\mathsf{Lin}(\mathbf{s}_1)$ smudges the uniform distribution over $\{0, \ldots, N\}$ if $\mathsf{Lin}(\mathbf{s}_1)$ and $\mathsf{Lin}(\mathbf{s}_1) + u$ are statistically indistinguishable, where u is sampled from $\{0, \ldots, N\}$.

Sample random $\mathbf{s}_1, \tilde{\mathbf{s}}_1 \in \{0,1\}^\ell$, $\mathbf{s}_2, \tilde{\mathbf{s}}_2 \in \{0,\ldots,\lfloor p/2 \rfloor\}^\ell$, and let

$$\mathbf{s}_{\text{res}} = \mathbf{s}_1 y + (1-\mathbf{s}_1)\cdot \lfloor p/2 \rfloor + \mathbf{s}_2 \mod p\,,$$

$$\tilde{\mathbf{s}}_{\text{res}} = \tilde{\mathbf{s}}_1 \tilde{y} + (1-\tilde{\mathbf{s}}_1)\cdot \lfloor p/2 \rfloor + \tilde{\mathbf{s}}_2 \mod p\,,$$

$$\mathbf{s}'_{\text{res}} = \mathbf{s}_1 y + (1-\mathbf{s}_1)\cdot \lfloor p/2 \rfloor + \mathbf{s}_2 \text{ (over } \mathbb{Z})\,,$$

$$\tilde{\mathbf{s}}'_{\text{res}} = \tilde{\mathbf{s}}_1 \tilde{y} + (1-\tilde{\mathbf{s}}_1)\cdot \lfloor p/2 \rfloor + \tilde{\mathbf{s}}_2 \text{ (over } \mathbb{Z})\,.$$

Set $\mathbf{L}^{\text{in}} = (y, \mathbf{s}_{\text{res}}, \tilde{\mathbf{s}}_{\text{res}})$ as the input label. Let $(\mathbf{c}_1, \mathbf{d}_1), (\mathbf{c}_2, \mathbf{d}_2)$ be additive sharings of $(\mathbf{c}, \mathbf{d})$. The table consists of

$$\mathsf{Enc}\big(\mathsf{Lin}(\mathbf{s}_1), \mathbf{c}_1\big), \quad \mathsf{Enc}\big(\mathsf{Lin}((1-\mathbf{s}_1)\cdot\lfloor p/2 \rfloor + \mathbf{s}_2), \mathbf{d}_1 - r\mathbf{c}_1\big)\,,$$

$$\mathsf{Enc}\big(\mathsf{Lin}(\tilde{\mathbf{s}}_1), \mathbf{c}_2\big), \quad \mathsf{Enc}\big(\mathsf{Lin}((1-\tilde{\mathbf{s}}_1)\cdot\lfloor p/2 \rfloor + \tilde{\mathbf{s}}_2), \mathbf{d}_2 - r\mathbf{c}_2\big)\,.$$

Given the table and input label, the evaluator homomorphically evaluates $\mathsf{Enc}(\mathsf{Lin}(\mathbf{s}'_{\text{res}}), \mathbf{c}_1 x + \mathbf{d}_1)$, $\mathsf{Enc}(\mathsf{Lin}(\tilde{\mathbf{s}}'_{\text{res}}), \mathbf{c}_2 x + \mathbf{d}_2)$. The evaluator recovers $\mathbf{s}'_{\text{res}}, \tilde{\mathbf{s}}'_{\text{res}}$ from the input label, and decrypts both ciphertexts to get $\mathbf{c}_1 x + \mathbf{d}_1, \mathbf{c}_2 x + \mathbf{d}_2$. In the end, output $\mathbf{L}^{\text{out}} = \mathbf{c}x + \mathbf{d} = (\mathbf{c}_1 x + \mathbf{d}_1) + (\mathbf{c}_2 x + \mathbf{d}_2) \mod p$.

Bit Decomposition Gadget. Besides purely arithmetic computation, we also consider a computation model that combines Boolean operations and arithmetic operation. Garbling such mixed computation is enabled by the *bit decomposition gadget* — a DARE for functions $f_{\text{BD},\{\mathbf{c}_j,\mathbf{d}_j\}}(x) = \{\mathbf{c}_j \mathsf{bits}(x)_j + \mathbf{d}_j\}$. It ensures that given $\mathbf{a}x + \mathbf{b}$ and the garbled table, the evaluator can get a label $\mathbf{c}_j \mathsf{bits}(x)_j + \mathbf{d}_j$ for every bit in the bit representation of x.

Notice that, in order to build the bit decomposition gadget, it suffices to design the *truncation gadget*. Let $\lfloor x \rfloor_{2^j} := \lfloor x/2^j \rfloor$ denotes the integer quotient of x divided by 2^j. This operation truncates j least significant bits. The truncation gadget is a DARE for functions $f_{\text{TC},\mathbf{c},\mathbf{d}}(x) = \mathbf{c}\cdot\lfloor x \rfloor_2 + \mathbf{d}$. Given a label of x and the garbled table, the evaluator can get a label for the truncated value $\lfloor x \rfloor_2$. Once we have the truncation gadget, the evaluator can use the truncation gadgets j times to get a label for the truncated value $\lfloor x \rfloor_{2^j}$ for every j. Thus the evaluator can compute a label of the j-th bit of x via

$$\underbrace{\mathbf{c}\lfloor x \rfloor_{2^{j-1}} + \mathbf{d}_1}_{\text{a label of } \lfloor x \rfloor_{2^{j-1}}} - 2\cdot(\underbrace{\mathbf{c}\lfloor x \rfloor_{2^j} + \mathbf{d}_2}_{\text{a label of } \lfloor x \rfloor_{2^j}}) = \underbrace{\mathbf{c}\cdot\mathsf{bits}(x)_j + (\mathbf{d}_1 - 2\mathbf{d}_2)}_{\text{a label of the } j\text{-th bit of } x}\,.$$

Now the task has been reduced to designing the truncation gadget. Our construction of the truncation gadget is inspired by the techniques used in the key extension gadgets. The first idea is to sample random r from a sufficiently large range, and to consider the one-time pad $y = x + r$. Instead of generating the labels of $\mathsf{bits}(x)_j$, we construct an (imperfect) bit decomposition gadget that generates the labels of each $\mathsf{bits}(y)_j$. Once evaluator has the labels of every bit of y, it can compute the labels of every bit of x, as long as we additionally give the evaluator a Yao's Boolean garbled circuit, with r hard-coded inside. Thus correspondingly, it suffices to construct an (imperfect) truncation gadget that allows the evaluator to get $\mathbf{c}\lfloor y \rfloor_2 + \mathbf{d}$.

Inspired by our key extension gadget for modulo-p computation, the gadget table of the (imperfect) truncation gadget looks like

$$\mathsf{Enc}(\mathsf{Lin}(\mathbf{s}_1), \mathbf{c}), \quad \mathsf{Enc}(\mathsf{Lin}(\lfloor \mathbf{s}_2 \rfloor_2), \mathbf{d}).$$

The evaluator can homomorphically evaluate $\mathsf{Enc}(\mathsf{Lin}(\mathbf{s}_1 \lfloor y \rfloor_2 + \lfloor \mathbf{s}_2 \rfloor_2), \mathbf{c}\lfloor y \rfloor_2 + \mathbf{d})$.

The input label of the truncation gadget is

$$(y, \mathbf{s}_1 y + \mathbf{s}_2), \text{ which equals } (x + r, \mathbf{s}_1 x + (\mathbf{s}_1 r + \mathbf{s}_2)).$$

If the evaluator can recover $\mathbf{s}_1 \lfloor y \rfloor_2 + \lfloor \mathbf{s}_2 \rfloor_2$ from the input label, it can decrypts $\mathbf{c}\lfloor y \rfloor_2 + \mathbf{d}$ using the key $\mathsf{Lin}(\mathbf{s}_1 \lfloor y \rfloor_2 + \lfloor \mathbf{s}_2 \rfloor_2)$.

To enable the recovery, $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{Z}^\ell$ are sampled from carefully chosen distributions. $\mathbf{s}_1$ is sampled uniformly from $\{0,1\}^\ell$. $\mathbf{s}_2$ is sampled conditioning on $\mathbf{s}_1$: for each $i \leq \ell$,

$$s_{2,i} = \begin{cases} \text{a random integer in } [0, B_{\text{smdg}}), & \text{if } s_{1,i} = 1 \\ \text{a random } \textit{odd} \text{ integer in } [0, B_{\text{smdg}}), & \text{if } s_{1,i} = 0 \end{cases}$$

where B_{smdg} is a sufficiently large bound. We sample $\mathbf{s}_1, \mathbf{s}_2$ in such a way to ensure that $s_{1,j} \lfloor y \rfloor_2 + \lfloor s_{2,j} \rfloor_2$ can be recovered from $(y, s_{1,j} y + s_{2,j})$.

Given $(y, s_{1,j} y + s_{2,j})$, the evaluator can compute $\lfloor s_{1,j} y + s_{2,j} \rfloor_2$, which is very close to the

target value. In particular,

$$s_{1,j}\lfloor y\rceil_2 + \lfloor s_{2,j}\rceil_2 = \begin{cases} \lfloor s_{1,j}y + s_{2,j}\rceil_2 - 1, & \text{if both } s_{1,j}y \text{ and } s_{2,j} \text{ are odd} \\ \lfloor s_{1,j}y + s_{2,j}\rceil_2, & \text{otherwise} \end{cases}$$

The evaluator can offset the error if it can tell whether both $s_{1,j}y$ and $s_{2,j}$ are odd. We claim:

$$\text{both } s_{1,j}y \text{ and } s_{2,j} \text{ are odd} \iff y \text{ is odd and } s_{1,j}y + s_{2,j} \text{ is even,} \tag{3.2}$$

By this, the evaluator can recover $\mathbf{s}_1\lfloor y\rceil_2 + \lfloor \mathbf{s}_2\rceil_2$.

The claim (3.2) can be proved by enumerating the possible parities of $s_{1,j}, y, s_{2,j}$. We also provide a visualized proof of this claim. Let $\{z\}_2 := z - 2\lfloor z\rceil_2$ denote the remainder of z divided by 2. The rounding error occurs if and only if $\{s_{1,j}y\}_2 + \{s_{2,j}\}_2 = 2$. As shown by Figure 3.3, when y is even, there is no rounding error; when y is odd, the rounding error occurs only if $s_{1,j}y + s_{2,j}$ is even.

Figure 3.3 also shows that $s_{1,j}$ is not always hidden by $s_{1,j}y + s_{2,j}$. For privacy, we require that $\mathsf{Lin}(\mathbf{s}_1)$ is "sufficiently random" even conditioning on the leakage. Such (seeded) linear function Lin is constructed in Section 3.6.

3.3 Preliminaries

In this chapter, we will construct three garbling schemes:

- Arithmetic garbling over bounded integers (Definition 2.4) (for any bound $B(\lambda) \leq 2^{\text{poly}(\lambda)}$);
- Arithmetic garbling (Definition 2.3) over $\mathbb{Z}_p$ (for any modulus $p = p(\lambda) \leq 2^{\text{poly}(\lambda)}$);
- Mixed garbling (Definition 2.5) between Boolean and arithmetics over bounded integers;

Towards a more modular presentation, we make a minor syntactical change to explicitly require a Setup algorithm for each garbling scheme, that takes a security parameter 1^λ and generates public parameters pp (specifying e.g. the label space $\mathcal{L}$) for the scheme. As we will see later, our constructions are composed of smaller gadgets that each handles a type of gate in the circuit. All gadget algorithms for a garbling scheme have access to the corresponding public parameter pp of the scheme.

3.3.1 Notations for Labels in Garbling Schemes.

Similar to prior schemes [AIK11], when garbling a circuit C, our Garb algorithm assigns two keys $z_1, z_2 \in \mathcal{L}$ to each wire of the circuit C. If this wire has a value x, then the evaluator should obtain a label $L = z_1 x + z_2$ computed over $\mathcal{L}$.

The technical core of this work is a key-extension gadget that can translate a short (in terms of dimension) label for some x to a long target label for the same value x. For a simpler presentation, we abuse the notation to denote a label space as $\mathcal{L}^\ell$ (instead of $\mathcal{L}$) in order to emphasize the dimension ℓ of such labels. We therefore write the keys $\mathbf{z}_1, \mathbf{z}_2 \in \mathcal{L}^\ell$ and labels $\mathbf{L} = \mathbf{z}_1 x + \mathbf{z}_2 \in \mathcal{L}^\ell$ as vectors.

3.3.2 Definition of Garbling Gadgets

Our garbling scheme garbles a circuit in a gate-by-gate fashion. To handle different types of gates, we introduce different garbling gadgets. In addition to the arithmetic computation gates $(+, -, \times)$, bit-decomposition gates (BD), and Boolean computation gates ($\mathsf{AND}, \mathsf{XOR}, \mathsf{NOT}$) introduced earlier, we also consider the following key extension gates (KE), which are artificially added to every circuit at garbling time.

Key Extension Gate has one input and one output wire and implements the identity function $f(x) = x$. Inserting this gate anywhere in a circuit does not change the function computed. However, garbling the key extension gate has the following effect during evaluation: Given a short label for the input wire $\mathbf{z}_1^{in} x + \mathbf{z}_2^{in}$ of dimention ℓ, the evaluator can obtain a much longer label for the output wire $\mathbf{z}_1^{out} x + \mathbf{z}_2^{out}$ of some dimension $\ell' > \ell$.

Our key extension gadget for handling the above gate is exactly the "key shrinking" gadget in [AIK11], and is the technical core of this work. We define it formally below. For completeness, we also provide analogous definitions for the arithmetic computation, bit composition, and Boolean computation gadgets.

Gadgets Share Setup of Garbling Scheme. Each gadget is defined with respect to a garbling scheme. The gadget depends on the public parameters pp generated by the Setup algorithm of the garbling scheme, which specifies the label space $\mathcal{L}$ and dimension $\ell \in \mathbb{N}$. Its algorithms all have random access to pp.

Key Extension Gadget. The key extension gadget consists of three algorithms, KeyGen, Garb, and Dec. To handle a key extension gate, we assume each of its output wires is already assigned an output key pair. Their concatenation form a long "target" key pair $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$. At garbling time, the garbler uses $\mathsf{KeyGen}, \mathsf{Garb}$ to generate a short key pair $\mathbf{z}_1^{in}, \mathbf{z}^{in}$ for the input wire, and a garbled table tb. At evaluation time, the evaluator uses Dec on the input label $\mathbf{L}^{in} = \mathbf{z}_1^{in} x + \mathbf{z}_2^{in}$ for some value x, and the garbled table tb to recover the output lable $\mathbf{L}^{out} = \mathbf{z}_1^{out} x + \mathbf{z}_2^{out}$.

We define the gadget formally w.r.t. arithmetic garbling schemes over some ring $R(\lambda)$ and over bounded integers by some $B(\lambda)$ respectively. Our construction of mixed garbling scheme will use the same gadget w.r.t. to arithmetic garbling over bounded integers.

Definition 3.1 (Key Extension Gadget)**.** *A key extension gadget w.r.t. an arithmetic garbling scheme over some ring $R(\lambda)$ (resp. over bounded integers by $B(\lambda)$) consists of three efficient algorithms:*

- $\mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell)$ *takes as inputs a security parameter λ, and the key dimension ℓ specified by* pp. *It samples a key pair $\mathbf{z}_1^{in}, \mathbf{z}_2^{in} \in \mathcal{L}^\ell$.*
- $\mathsf{KE.Garb}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in})$ *takes as inputs a (long) key pair $\mathbf{z}_1^{out}, \mathbf{z}_2^{out} \in \mathcal{L}^{\ell'}$, and a (short) key pair $\mathbf{z}_1^{in}, \mathbf{z}_2^{in} \in \mathcal{L}^\ell$. It outputs a garbled table* tb *(consisting of many ciphertexts in $\mathcal{E}$).*
- $\mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$ *takes as inputs a short label $\mathbf{L}^{in} \in \mathcal{L}^\ell$ and a garbled table* tb*. It outputs a long label $\mathbf{L}^{out} \in \mathcal{L}^{\ell'}$.*

Correctness. *The scheme is* correct *if for all $\lambda \in \mathbb{N}$, $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda)\right)$, $\mathbf{z}_1^{out}, \mathbf{z}_2^{out} \in \mathcal{L}^{\ell'}$*

of dimension $\ell' \in \mathbb{N}$, and $x \in R$ (resp. $x \in \mathbb{Z}_B$), the following holds.

$$\Pr\left[\begin{array}{l}\mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb}) \\ = \mathbf{L}^{out}\end{array}\middle|\begin{array}{r}\mathbf{z}_1^{in}, \mathbf{z}_2^{in} \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell), \\ \mathsf{tb} \leftarrow \mathsf{KE.Garb}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in}), \\ \mathbf{L}^{in} = \mathbf{z}_1^{in} x + \mathbf{z}_2^{in},\ \mathbf{L}^{out} = \mathbf{z}_1^{out} x + \mathbf{z}_2^{out}\end{array}\right] = 1.$$

Security. *A simulator* $\mathsf{KE.Sim}$ *for the scheme has the following syntax.*

- $\mathsf{KE.Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$ *takes as inputs a security parameter λ, public parameters* pp*, and a long label $\mathbf{L}^{out} \in \mathcal{L}^{\ell'}$. It outputs a simulated short label $\widetilde{\mathbf{L}}^{in} \in \mathcal{L}^{\ell}$ and a simulated garbled table* $\widetilde{\mathsf{tb}}$*.*

The scheme is secure *if there exists an efficient simulator* $\mathsf{KE.Sim}$ *such that for all polynomial $\ell' = \ell'(\lambda)$, sequence of key pairs $\{\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out}\}_\lambda$ where $\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out} \in \mathcal{L}^{\ell'}$, and sequence of inputs $\{x_\lambda\}_\lambda$ where $x_\lambda \in R(\lambda)$ (resp. $x_\lambda \in \mathbb{Z}_{B(\lambda)}$), the following indistinguishability holds. (We suppress the index λ below.)*

$$\begin{array}{l}\left\{\mathsf{pp}, \mathsf{KE.Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})\right\} \\ \approx_c \left\{\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb}\right\}.\end{array}\left|\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda),\ \mathbf{z}_1^{in}, \mathbf{z}_2^{in} \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell), \\ \mathsf{tb} \leftarrow \mathsf{KE.Garb}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in}), \\ \mathbf{L}^{in} = \mathbf{z}_1^{in} x + \mathbf{z}_2^{in},\ \mathbf{L}^{out} = \mathbf{z}_1^{out} x + \mathbf{z}_2^{out}\end{array}\right.$$

Arithmetic Computation Gadget. Let $g : R \times R \to R$ denote a general arithmetic computation gate that's either addition, subtraction, or multiplication over R. It has two input wires x and y, and one output wire w. The garbling algorithm of the gadget for g on input a pair of keys $(\mathbf{z}_1^{out}, \mathbf{z}_2^{out})$ for the output wire w, produces two pairs of (potentially longer) keys, $(\mathbf{z}_1^x, \mathbf{z}_2^x)$ and $(\mathbf{z}_1^y, \mathbf{z}_2^y)$, for the two input wires respectively, together with a (possibly empty) garbled table tb. At evaluation time, given labels for the input wires x, y and the garbled table, the evaluator should obtain the corresponding label for w, and nothing else.

We define the gadget formally w.r.t. arithmetic garbling schemes over $R(\lambda)$. Our constructions of arithmetic garbling over bounded integers and of mixed garbling will use the same gadget w.r.t. arithmetic garbling over $R(\lambda)$.

Definition 3.2 (Arithmetic Computation Gadget)**.** *An arithmetic computation gadget for an arithmetic gate g over $R(\lambda)$ consists of two efficient algorithms.*

- $\mathsf{ACmp.Garb}^{\mathsf{pp}}(1^\lambda, \mathbf{z}_1^{out}, \mathbf{z}_2^{out})$ *takes as inputs a security parameter λ, and a key pair $\mathbf{z}_1^{out}, \mathbf{z}_2^{out} \in \mathcal{L}^{\ell'}$ of dimension ℓ'. It outputs two key pairs $\mathbf{z}_1^x, \mathbf{z}_2^x, \mathbf{z}_1^y, \mathbf{z}_2^y \in \mathcal{L}^{\ell''}$ of (possibly greater) dimension ℓ'', and a garbled table* tb *(which may be empty, as we will see in the construction).*
- $\mathsf{ACmp.Dec}^{\mathsf{pp}}(\mathbf{L}^x, \mathbf{L}^y, \mathsf{tb})$ *takes as inputs two labels $\mathbf{L}^x, \mathbf{L}^y \in \mathcal{L}^{\ell''}$ and a garbled table* tb*. It outputs a label $\mathbf{L}^{out} \in \mathcal{L}^{\ell'}$.*

Correctness. *The scheme is* correct *if for all $\lambda \in \mathbb{N}$, $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda)\right)$, $\mathbf{z}_1^{out}, \mathbf{z}_2^{out} \in \mathcal{L}^{\ell'}$ of dimension $\ell' \in \mathbb{N}$, and $x, y \in R$, the following holds.*

$$\Pr\left[\begin{array}{l|c} \mathsf{ACmp.Dec}^{\mathsf{pp}}(\mathbf{L}^x, \mathbf{L}^y, \mathsf{tb}) & \mathbf{z}_1^x, \mathbf{z}_2^x, \mathbf{z}_1^y, \mathbf{z}_2^y, \mathsf{tb} \leftarrow \mathsf{ACmp.Garb}^{\mathsf{pp}}(1^\lambda, \mathbf{z}_1^{out}, \mathbf{z}_2^{out}), \\ = \mathbf{L}^{out} & \mathbf{L}^x = \mathbf{z}_1^x x + \mathbf{z}_2^x,\ \mathbf{L}^y = \mathbf{z}_1^y y + \mathbf{z}_2^y, \\ & \mathbf{L}^{out} = \mathbf{z}_1^{out} g(x,y) + \mathbf{z}_2^{out} \end{array}\right] = 1.$$

Security. *A simulator* $\mathsf{ACmp.Sim}$ *for the arithmetic computation gadget is an efficient algorithm with the following syntax.*

- $\mathsf{ACmp.Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$ *takes as inputs a security parameter λ, public parameters* pp*, and a label $\mathbf{L}^{out} \in \mathcal{L}^{\ell'}$. It outputs two simulated labels $\widetilde{\mathbf{L}}^x, \widetilde{\mathbf{L}}^y \in \mathcal{L}^{\ell''}$ and a simulated garbled table $\widetilde{\mathsf{tb}}$.*

The scheme is secure *if there exists a simulator* $\mathsf{ACmp.Sim}$ *such that for all polynomial $\ell' = \ell'(\lambda)$, sequence of key pairs $\{\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out}\}_\lambda$ where $\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out} \in \mathcal{L}^{\ell'}$, and sequence of inputs $\{x_\lambda, y_\lambda\}_\lambda$ where $x_\lambda, y_\lambda \in R$, the following indistinguishability holds. (For more concise notations, the index λ is suppressed below.)*

$$\begin{array}{r|c} & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ \left\{\mathsf{pp}, \mathsf{ACmp.Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})\right\} & \mathbf{z}_1^x, \mathbf{z}_2^x, \mathbf{z}_1^y, \mathbf{z}_2^y, \mathsf{tb} \leftarrow \mathsf{ACmp.Garb}^{\mathsf{pp}}(1^\lambda, \mathbf{z}_1^{out}, \mathbf{z}_2^{out}), \\ \approx_c \{\mathsf{pp}, \mathbf{L}^x, \mathbf{L}^y, \mathsf{tb}\} & \mathbf{L}^x = \mathbf{z}_1^x x + \mathbf{z}_2^x,\ \mathbf{L}^y = \mathbf{z}_1^y y + \mathbf{z}_2^y, \\ & \mathbf{L}^{out} = \mathbf{z}_1^{out} g(x,y) + \mathbf{z}_2^{out} \end{array}.$$

Bit Decomposition Gadget. The garbling algorithm of the bit-decomposition gadget takes as input d key pairs $\{(\mathbf{z}_1^i, \mathbf{z}_2^i)\}_{i\in[d]}$ each associated with one of the d output wires, produces a key pair $(\mathbf{z}_1^{in}, \mathbf{z}_2^{in})$ for the input wire, together with a garbled table. At evaluation time, the evaluator given a label for input x and the garbled table, obtains one label for each output bit $\mathsf{bits}(x)_i$ for $i \in [d]$, and learn nothing else.

We define the gadget formally w.r.t. arithmetic garbling schemes over bounded integers.

Definition 3.3 (Bit Decomposition Gadget)**.** *Let $B = B(\lambda) \leq 2^{\mathrm{poly}(\lambda)}$ be a bound, and $d = \lceil \log(2B+1) \rceil$ be the bit length of values in $\mathbb{Z}_B$. A bit decomposition gadget consists of two efficient algorithms.*

- $\mathsf{BD.Garb}^{\mathsf{pp}}(1^\lambda, 1^\ell, \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]})$ *takes as inputs a security parameter λ, the key dimension ℓ specified by* pp*, and r key pairs $\{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]} \in \mathcal{L}^{\ell'}$ of dimension ℓ'. It outputs a key pair $\mathbf{z}_1^{in}, \mathbf{z}_2^{in} \in \mathcal{L}^\ell$, and a garbled table* tb*.*
- $\mathsf{BD.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$ *takes as inputs a label $\mathbf{L}^{in} \in \mathcal{L}^\ell$ and a garbled table* tb*. It outputs d labels $\{\mathbf{L}^i\}_{i\in[d]} \in \mathcal{L}^{\ell'}$.*

Correctness. *The scheme is* correct *if for all $\lambda \in \mathbb{N}$, $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda)\right)$, $\{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]} \in \mathcal{L}^{\ell'}$ of dimension $\ell' \in \mathbb{N}$, and $x \in \mathbb{Z}_B$, the following holds.*

$$\Pr\left[\begin{array}{l} \mathsf{BD.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb}) \\ = \{\mathbf{L}^i\}_{i\in[d]} \end{array} \middle| \begin{array}{c} \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \mathsf{tb} \leftarrow \mathsf{BD.Garb}^{\mathsf{pp}}(1^\lambda, 1^\ell, \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]}), \\ \mathbf{L}^{in} = \mathbf{z}_1^{in} x + \mathbf{z}_2^{in}, \\ \mathbf{L}^i = \mathbf{z}_1^i \mathsf{bits}(x)_i + \mathbf{z}_2^i \ (\textit{over } \mathcal{L}) \end{array}\right] = 1.$$

Security. *A simulator* $\mathsf{BD.Sim}$ *for the bit decomposition gadget is an efficient algorithm with the following syntax.*

- $\mathsf{BD.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^i\}_{i\in[d]})$ *takes as inputs a security parameter λ, public parameters* pp*, and d labels $\mathbf{L}^i \in \mathcal{L}^{\ell'}$. It outputs a simulated label $\widetilde{\mathbf{L}}^{in} \in \mathcal{L}^\ell$ and a simulated garbled table $\widetilde{\mathsf{tb}}$.*

The scheme is secure *if there exists a simulator* BD.Sim *such that for all polynomial* $\ell' = \ell'(\lambda)$, *sequence of* d *key pairs,* $\{\{\mathbf{z}^i_{1,\lambda}, \mathbf{z}^i_{2,\lambda}\}_{i\in[d]}\}_\lambda$ *where* $\mathbf{z}^i_{1,\lambda}, \mathbf{z}^i_{2,\lambda} \in \mathcal{L}^{\ell'}$, *and sequence of inputs* $\{x_\lambda\}_\lambda$ *where* $x_\lambda \in \mathbb{Z}_B$, *the following indistinguishability holds. (For more concise notations, the index* λ *is suppressed below.)*

$$\begin{array}{c|c} & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ \left\{\mathsf{pp}, \mathsf{BD.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^i\}_{i\in[d]})\right\} & \mathbf{z}^{in}_1, \mathbf{z}^{in}_2, \mathsf{tb} \leftarrow \mathsf{BD.Garb}^{\mathsf{pp}}(1^\lambda, \{\mathbf{z}^i_1, \mathbf{z}^i_2\}_{i\in[d]}), \\ \approx_c \left\{\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb}\right\} & \mathbf{L}^{in} = \mathbf{z}^{in}_1 x + \mathbf{z}^{in}_2, \\ & \mathbf{L}^i = \mathbf{z}^i_1 \mathsf{bits}(x)_i + \mathbf{z}^i_2 \end{array}.$$

Boolean Computation Gadget. Let $g_B : \{0,1\}^{d_1} \to \{0,1\}^{d_2}$ a general function that can be implemented by a polynomial size Boolean circuit. Gate g_B has d_1 input wires $x_1, \ldots, x_{d_1}$ and d_2 output wires $y_1, \ldots, y_{d_2}$. The garbling algorithm of the gadget g_B takes input d_2 key pair $\{\mathbf{z}^{out,i}_1, \mathbf{z}^{out,i}_2\}_{i\in[d_2]}$ for the output wire, produces d_1 key pairs $\{(\mathbf{z}^{in,i}_1, \mathbf{z}^{in,i}_2)\}_{i\in[d_1]}$, one for each input wire, and a garbled table. At evaluation time, given one label for each input bit x_i, the evaluator learns the corresponding label for each output bit y_i and nothing else.

We define the gadget formally w.r.t. arithmetic garbling schemes over bounded integers.

Definition 3.4 (Boolean Computation Gadget)**.** *Let* $B = B(\lambda) \leq 2^{\mathrm{poly}(\lambda)}$ *be a bound, and* $d = \lceil \log(2B+1) \rceil$ *be the bit length of values in* $\mathbb{Z}_B$*. Let* $g_B : \{0,1\}^{d_1} \to \{0,1\}^{d_2}$ *be a Boolean computation gate where* $d_1 = d_1(\lambda), d_2 = d_2(\lambda)$ *are polynomial bounded. A Boolean computation gadget consists of two efficient algorithms.*

- $\mathsf{BCmp.Garb}^{\mathsf{pp}}(1^\lambda, 1^\ell, \{\mathbf{z}^{out,i}_1, \mathbf{z}^{out,i}_2\}_{i\in[d_2]})$ *takes as inputs a security parameter* λ, *the key dimension* ℓ *specified by* pp, *and* d_2 *key pairs* $\{\mathbf{z}^{out,i}_1, \mathbf{z}^{out,i}_2\}_{i\in[d_2]}$, *where* $\mathbf{z}^{out,i}_1, \mathbf{z}^{out,i}_2 \in \mathcal{L}^{\ell'}$ *of dimension* ℓ'. *It outputs* d_1 *key pairs* $\{\mathbf{z}^{in,i}_1, \mathbf{z}^{in,i}_2\}_{i\in[d_1]}$, *where* $\mathbf{z}^{in,i}_1, \mathbf{z}^{in,i}_2 \in \mathcal{L}^{\ell}$, *and a garbled table* tb.
- $\mathsf{BCmp.Dec}^{\mathsf{pp}}(\{\mathbf{L}^{in,i}\}_{i\in[d_1]}, \mathsf{tb})$ *takes as inputs* d_1 *labels* $\{\mathbf{L}^{in,i}\}_{i\in[d_1]}$ *where* $\mathbf{L}^{in,i} \in \mathcal{L}^{\ell}$ *and a garbled table* tb*. It outputs* d_2 *labels* $\{\mathbf{L}^{out,i}\}_{i\in[d_2]}$ *where* $\mathbf{L}^{out,i} \in \mathcal{L}^{\ell'}$.

Correctness. *The scheme is* correct *if for all* $\lambda \in \mathbb{N}$, $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda)\right)$, $\{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_{i\in[d_2]}$ *where* $\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i} \in \mathcal{L}^{\ell'}$ *of dimension* $\ell' \in \mathbb{N}$*, and* $\mathbf{x} = (x_1, \ldots, x_{d_1}) \in \{0,1\}^{d_1}$*, the following holds.*

$$\Pr\left[\begin{array}{l|l} & \{\mathbf{z}_1^{in,i}, \mathbf{z}_2^{in,i}\}_{i\in[d_1]}, \mathsf{tb} \\ & \quad \leftarrow \mathsf{BCmp.Garb}^{\mathsf{pp}}(1^\lambda, 1^\ell, \{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_{i\in[d_2]}), \\ \mathsf{BCmp.Dec}^{\mathsf{pp}}(\{\mathbf{L}^{in,i}\}_{i\in[d_1]}, \mathsf{tb}) & \mathbf{y} = (y_1, \ldots, y_{d_2}) = g_B(\mathbf{x}), \\ = \{\mathbf{L}^{out,i}\}_{i\in[d_2]} & \mathbf{L}^{in,i} = \mathbf{z}_1^{in,i} x_i + \mathbf{z}_2^{in,i}, \\ & \mathbf{L}^{out,i} = \mathbf{z}_1^{out,i} y_i + \mathbf{z}_2^{out,i} \text{ (over } \mathcal{L}) \end{array}\right] = 1.$$

Security. *A simulator* BCmp.Sim *for the Boolean computation gadget is an efficient algorithm with the following syntax.*

- $\mathsf{BCmp.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^{out,i}\}_{i\in[d_2]})$ *takes as input a security parameter* λ*, public parameters* pp*, and* d_2 *output-wire labels* $\{\mathbf{L}^{out,i}\}_{i\in[d_2]}$ *where* $\mathbf{L}^{out,i} \in \mathcal{L}^{\ell'}$*. It outputs* d_1 *simulated labels* $\{\widetilde{\mathbf{L}}^{in,i}\}_{i\in[d_1]}$ *where* $\mathbf{L}^{in,i} \in \mathcal{L}^{\ell}$ *and a simulated garbled table* $\widetilde{\mathsf{tb}}$.

The scheme is secure *if there exists a simulator* BCmp.Sim *such that for all polynomial* $\ell' = \ell'(\lambda)$*, sequence of key pairs* $\{\{\mathbf{z}_{1,\lambda}^{out,i}, \mathbf{z}_{2,\lambda}^{out,i}\}_{i\in[d_2]}\}_\lambda$ *where* $\mathbf{z}_{1,\lambda}^{out,i}, \mathbf{z}_{2,\lambda}^{out,i} \in \mathcal{L}^{\ell'}$*, and sequence of inputs* $\{\mathbf{x}_\lambda\}_\lambda$ *where* $\mathbf{x}_\lambda \in \{0,1\}^{d_1}$*, the following indistinguishability holds. (For more concise notations, the index* λ *is suppressed below.)*

$$\begin{array}{l|l} & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ & \{\mathbf{z}_1^{in,i}, \mathbf{z}_2^{in,i}\}_i, \mathsf{tb} \\ \left\{\mathsf{pp}, \mathsf{BCmp.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^{out,i}\}_i)\right\} & \quad \leftarrow \mathsf{BCmp.Garb}^{\mathsf{pp}}(1^\lambda, 1^\ell, \{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_i), \\ \approx_c \left\{\mathsf{pp}, \{\mathbf{L}^{in,i}\}_i, \mathsf{tb}\right\}. & \mathbf{y} = (y_1, \ldots, y_{d_2}) = g_B(\mathbf{x}), \\ & \mathbf{L}^{in,i} = \mathbf{z}_1^{in,i} x_i + \mathbf{z}_2^{in,i}, \\ & \mathbf{L}^{out,i} = \mathbf{z}_1^{out,i} y_i + \mathbf{z}_2^{out,i} \end{array}$$

3.4 Construction of Building Block: LHE

3.4.1 Definition of Basic LHE

We first define a very simple base scheme that creates noisy ciphertexts. Decryption doesn't try to remove the noise, and simply recovers the encrypted message with noise. The scheme allows evaluating linear functions homomorphically over ciphertexts, which increases the level/magnitude of noise in the ciphertexts.

The base scheme can be instantiated under either the learning with error (LWE) assumption or the decisional composite residuosity (DCR) assumption (Construction 2, Construction 3). We will then implement another scheme on top of a base instantiation that's tailored to the needs of our application.

Definition 3.5 (Noisy Linearly Homomorphic Encryption)**.** *A noisy linearly homomorphic encryption scheme consists of five efficient algorithms, and is associated with two exponentially bounded functions in the security parameter λ, $B_e(\lambda), B_s(\lambda) \leq 2^{\text{poly}(\lambda)}$.*

- $\mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param})$ *takes as inputs a security parameter λ, an upper bound $\Psi \in \mathbb{N}$ on the dimensions of message vectors to be encrypted, and additional parameters* param*. It outputs public parameters* pp*, which defines a key space $\mathcal{S} = \mathbb{Z}^{\ell_s}$, a ciphertext space $\mathcal{E}$, and a message modulus P, that satisfy certain properties specified by* param*.*

  By default, the rest of the algorithms have random access to pp*, and receive as inputs* $1^\lambda, \mathsf{param}$ *in addition to other inputs, i.e., we use the simplified notation* $\mathsf{X}(x_1, x_2, \ldots)$ *to mean* $\mathsf{X}^{\mathsf{pp}}(1^\lambda, \mathsf{param}, x_1, x_2, \ldots)$*.*
- $\mathsf{KeyGen}(1^{\ell_s})$ *takes the key dimension ℓ_s (specified by* pp*) as input, and outputs a key $s \in \mathbb{Z}^{\ell_s}$, satisfying that $|s|_\infty < B_s(\lambda)$.*
- $\mathsf{Enc}(s, \mathbf{m})$ *takes as inputs a key $s \in \mathbb{Z}^{\ell_s}$, and a message vector $\mathbf{m} \in \mathbb{Z}^\psi$ of dimension $\psi \leq \Psi$. It outputs a ciphertext* $\mathsf{ct} \in \mathcal{E}$*.*
- $\mathsf{Dec}(s, \mathsf{ct})$ *takes as inputs a key $s \in \mathbb{Z}^{\ell_s}$, and a ciphertext* $\mathsf{ct} \in \mathcal{E}$*. It outputs a (noisy) message vector $\mathbf{m}' \in \mathbb{Z}_P^\psi$ of dimension $\psi \leq \Psi$, or the symbol $\perp$ in case of a decryption error.*

- $\mathsf{Eval}(f, \{\mathsf{ct}_i\})$ *takes as input a* linear *function* f *specified by* d *integer coefficients i.e.,* $f(x_1, \ldots, x_d) = \sum_{i \in [d]} a_i x_i$, *and* d *ciphertexts* $\{\mathsf{ct}_i\}_{i \in [d]}$. *It outputs an evaluated ciphertext* $\mathsf{ct}_f \in \mathcal{E}$.

  (If ct_i *encrypts a message vector* $\mathbf{m}_i \in \mathbb{Z}_P^{\psi}$ *of dimension* $\psi \leq \Psi$, *under a key* s_i, ct_f *should encrypt the vector* $\mathbf{m}_f = f(\mathbf{m}_1, \ldots, \mathbf{m}_d)$, *evaluated coordinate-wise over* $\mathbb{Z}_P$, *under the key* $s_f = f(s_1, \ldots, s_d)$, *evaluated over* $\mathbb{Z}$.)

Correctness w.r.t. B_e**.** *The scheme is* correct *if for all* $\lambda, \Psi \in \mathbb{N}$, param, $\mathsf{pp} \in \mathrm{Supp}\left(\mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param})\right)$, $s \in \mathbb{Z}^{\ell_s}$, $\mathbf{m} \in \mathbb{Z}^{\psi}$ *where* $\psi \leq \Psi$, *it holds that*

$$\Pr\left[\|\mathbf{e}\|_\infty \leq B_e \;\middle|\; \begin{array}{l} \mathsf{ct} \leftarrow \mathsf{Enc}(s, \mathbf{m}),\ \mathbf{m}' = \mathsf{Dec}(s, \mathsf{ct}), \\ \mathbf{e} = \mathbf{m}' - \mathbf{m} \mod P \end{array}\right] = 1,$$

where we calculate the infinity norm $\|\cdot\|_\infty$ *of* $\mathbf{e} \in \mathbb{Z}_P^{\psi}$ *by identifying it as an integer vector over* $[-P/2, P/2]^{\psi}$.

One-time Security. *The scheme is (one-time)* secure *if for all polynomial* $\Psi = \Psi(\lambda)$, *sequence of parameters* $\{\mathsf{param}_\lambda\}_\lambda$ *each of bit length* $|\mathsf{param}_\lambda| \leq \mathrm{poly}(\lambda)$, *and sequence of integer message vectors* $\{\mathbf{m}_{1,\lambda}, \mathbf{m}_{2,\lambda}\}_\lambda$ *where* $\mathbf{m}_{1,\lambda}, \mathbf{m}_{2,\lambda} \in \mathbb{Z}^{\psi_\lambda}$ *of dimension* $\psi_\lambda \leq \Psi(\lambda)$, $\|\mathbf{m}_{i,\lambda}\|_\infty \leq 2^{\mathrm{poly}(\lambda)}$, *the following indistinguishability holds. (We surpress the index* λ *below.)*

$$\{\mathsf{ct}_1, \mathsf{pp}\} \approx_c \{\mathsf{ct}_2, \mathsf{pp}\}\ .\ \left|\ \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param}), \\ s \leftarrow \mathsf{KeyGen}(1^{\ell_s}),\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(s, \mathbf{m}_i), \end{array}\right.$$

Below we define two additional properties satisfied by our base instantiations under either the LWE or the DCR assumption.

Definition 3.6 (Linear Homomorphism)**.** *A LHE scheme (per Definition 5.11) has* linear homomorphism *if for all linear function* f *specified by* d *integer coefficients, for all* $\lambda, \Psi \in \mathbb{N}$, param, $\mathsf{pp} \in \mathrm{Supp}\left(\mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param})\right)$, $s_i \in \mathbb{Z}^{\ell_s}$ *for each* $i \in [d]$, *and* $\mathsf{ct}_i \in \mathcal{E}$ *for each* $i \in [d]$, *such that,* $\mathsf{Dec}(s_i, \mathsf{ct}_i)$ *outputs* $\mathbf{m}_i \neq \perp$ *and all* $\mathbf{m}_i$ *have the same dimension* $\psi \leq \Psi$,

then the following holds:

$$\Pr\left[\begin{array}{l|l}\mathsf{Dec}(s_f, \mathsf{ct}_f) & \mathbf{m}_i = \mathsf{Dec}(s_i, \mathsf{ct}_i),\ \mathsf{ct}_f \leftarrow \mathsf{Eval}(f, \{\mathsf{ct}_i\}), \\ = f(\{\mathbf{m}_i\}) \mod P & s_f = f(\{s_i\})\ (\textit{over}\ \mathbb{Z})\end{array}\right] = 1.$$

Definition 3.7 (Statistical Closeness)**.** *A LHE scheme (per Definition 5.11) has* statistical closeness *if for all* $\lambda, \Psi \in \mathbb{N}$*,* param, $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param})\right)$*,* $s \in \mathbb{Z}^{\ell_s}$*, and any two distributions* D_1, D_2 *of ciphertexts over* $\mathcal{E}$ *such that, for all* $i \in \{1, 2\}$*,* $\Pr\left[\mathsf{Dec}(s, \mathsf{ct}_i) \neq \perp \mid \mathsf{ct}_i \leftarrow D_i\right] = 1$*, the following holds:*

$$\Delta_{\mathsf{SD}}\left(\mathsf{ct}_1, \mathsf{ct}_2\right) = \Delta_{\mathsf{SD}}\left(\mathsf{Dec}(s, \mathsf{ct}_1), \mathsf{Dec}(s, \mathsf{ct}_2)\right) \mid \mathsf{ct}_i \leftarrow D_i,\ .$$

3.4.2 A Construction of Special-Purpose LHE

We next construct a special-purpose LHE scheme $\overline{\mathsf{lhe}}$ using a basic LHE scheme lhe defined in the previous subsection as a black-box. The special-purpose LHE $\overline{\mathsf{lhe}}$ is taylored to the needs of our garbling construction in the following ways:

- Arbitrary Message Space: Note that the message space $\mathbb{Z}_P$ of the basic LHE scheme is specified by the public parameter pp during setup time, and may not match domain, e.g. $\mathbb{Z}_p$ of the computation to be garbled. (For example, in the DCR instantiation, $P = N^r$, where N is a randomly sampled RSA modulus). In $\overline{\mathsf{lhe}}$, the setup algorithm $\overline{\mathsf{Setup}}$ takes an arbitrary modulus p as an additional parameter, and sets up public parameters $\overline{\mathsf{pp}}$ with exactly $\mathbb{Z}_p$ as the message space. In the construction, $\overline{\mathsf{Setup}}$ invokes the basic setup algorithm Setup and makes sure that the basic modulus P is sufficiently large. $\overline{\mathsf{lhe}}$ then embeds the actual message space $\mathbb{Z}_p$ in $\mathbb{Z}_P$.
- Exact Decryption: lhe decryption produces noisy message, where the noise is bounded by B_e, while $\overline{\mathsf{lhe}}$ decryption produces the *exact* message.
- Special-Purpose Linear Homomorphism, and Noise Smudging: Relying on the linear homomorphism and statistical closeness of lhe (Definition 3.6, Definition 3.7), we show in Lemma 3.2 and Lemma 3.1 that $\overline{\mathsf{lhe}}$ satisfies properties tailored for our

construction of garbling schemes. Roughly speaking, it allows evaluating simple linear functions, e.g., $f(x_1, x_2) = yx_1 + x_2$, and $f'(x_{res}, x_1) = x_{res} - yx_1$, and we can smudge the noise in a noisy ciphertext by homomorphically adding an encryption of the smudging noise.

Construction 1 (LHE for $\mathbb{Z}_p$)**.** We construct the special purpose scheme $\overline{\mathsf{lhe}}$ on top of a basic scheme lhe (instantiated under either LWE in Construction 2 or DCR in Construction 3,) below. Let $B_e = B_e(\lambda) \leq 2^{\mathrm{poly}(\lambda)}$ be the fixed noise bound for lhe guaranteed by its correctness (Definition 5.11).

- $\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max})$ takes as input an arbitrary message modulus $p \in \mathbb{N}$, and an upper bound $B_{\max} \in \mathbb{N}$ on noise levels in ciphertexts, and proceeds in the following steps:
  - Set $B_{\mathrm{msg}} = 2p \cdot \max(B_{\max}, B_e)$, and run $\mathsf{Setup}(1^\lambda, 1^\Psi, \mathsf{param} = B_{\mathrm{msg}})$ to obtain pp, which specifies a key dimension ℓ_s, a ciphertext space $\mathcal{E}$, and a message modulus P.

    Our instantiations of lhe guarantee that $P \geq B_{\mathrm{msg}}$, and ℓ_s is polynomial in λ and $\log B_{\mathrm{msg}}$, independent of the maximal message dimension Ψ.
  - Set a scaling factor $\Delta = \lfloor P/p \rfloor$, and output $\overline{\mathsf{pp}} = (\mathsf{pp}, \Delta)$, which specifies a key space $\mathcal{S} = \mathbb{Z}^{\ell_s}$, a ciphertext space $\mathcal{E}$, and a message modulus p.

  Note: *By our setting of B_{msg}, and the guarantee that $P \geq B_{\mathrm{msg}}$, we have*

$$\Delta \geq \lfloor B_{\mathrm{msg}}/p \rfloor = 2\max(B_{\max}, B_e) \ . \tag{3.3}$$

- $\overline{\mathsf{KeyGen}}(1^{\ell_s})$ directly runs $s \leftarrow \mathsf{KeyGen}(1^{\ell_s})$, and outputs s.
- $\overline{\mathsf{Enc}}(s, \mathbf{m})$ takes as input a secret key s and a message vector $\mathbf{m} \in \mathbb{Z}^\psi$. It computes $\mathbf{m}' = (\mathbf{m} \bmod p) \cdot \Delta \in \mathbb{Z}_P^\psi$, and outputs $\mathsf{ct} \leftarrow \mathsf{Enc}(s, \mathbf{m}')$.

  Note: *The* one-time security *of $\overline{\mathsf{lhe}}$ follows directly from that of lhe.*
- $\overline{\mathsf{Dec}}(s, \mathsf{ct})$ first runs $\mathbf{m}' = \mathsf{Dec}(s, \mathsf{ct})$ to recover $\mathbf{m}' \in \mathbb{Z}_P^\psi$, and then computes $\mathbf{m}_p = \lfloor \mathbf{m}'/\Delta \rceil$ to recover $\mathbf{m}_p \in \mathbb{Z}_p^\psi$. It outputs $\mathbf{m}_p$.

Note: *By the correctness of* lhe*, we have*

$$\mathsf{Dec}(s, \overline{\mathsf{Enc}}(s, \mathbf{m})) = \mathsf{Dec}(s, \mathsf{Enc}(s, \mathbf{m}_p \cdot \Delta)) = \mathbf{m}_p \cdot \Delta + \mathbf{e} \in \mathbb{Z}_P^{\psi},$$

for some noise vector $\mathbf{e}$ *such that* $\|\mathbf{e}\|_\infty \leq B_e$*. As noted in Equation* (3.3)*, we have* $\Delta \geq 2B_e$*. Hence rounding by* Δ *recovers the correct message* $\mathbf{m}_p \in \mathbb{Z}_p^{\psi}$ exactly*, i.e., the construction has* correctness *with noise bound* $\overline{B}_e = 0$*.*

- $\overline{\mathsf{Eval}}(f, \{\mathsf{ct}_i\})$ directly runs $\mathsf{ct}_f \leftarrow \mathsf{Eval}(f, \{\mathsf{ct}_i\})$ and outputs ct_f.

  Note: $\overline{\mathsf{Eval}} \equiv \mathsf{Eval}$*, hence it can operate on ciphertexts of both* $\overline{\mathsf{lhe}}$ *and* lhe*.*

Next, relying on the linear homomorphism of lhe, we show that $\overline{\mathsf{lhe}}$ satisfies linear homomorphism w.r.t. linear functions of the form $f(x_1, x_2) = yx_1 + x_2$, which suffices for our garbling constructions.

Lemma 3.1 shows how to "smudge" the noises in an *evaluated* $\overline{\mathsf{lhe}}$ ciphertext ct_R by homomorphically adding a fresh lhe encryption ct_e of a smudging noise vector $\mathbf{e}$ to it, as $\mathsf{ct}'_R = \overline{\mathsf{Eval}}(+, \mathsf{ct}_R, \mathsf{ct}_e)$. As long as the smudging noise is large enough, the result ct'_R is statistically close to homomorphically adding ct_e to a *fresh* $\overline{\mathsf{lhe}}$ ciphertext ct_2, as $\mathsf{ct}'_2 = \overline{\mathsf{Eval}}(+, \mathsf{ct}_2, \mathsf{ct}_e)$.

Lemma 3.2 shows how to set the noise upper bound $B_{\max}$ during $\overline{\mathsf{Setup}}$ so that evaluated $\overline{\mathsf{lhe}}$ ciphertexts can still be decrypted exactly.

Lemma 3.1 (noise smudging)**.** *Suppose the underlying LHE scheme* lhe *in Construction 1 satisfies linear homomorphism (Definition 3.6) and statistical closeness (Definition 3.7). For all* $\lambda, \Psi, p, B_{\max}, \alpha_1 \in \mathbb{N}$*, set the smudging noise level to*

$$\alpha_2 = \lambda^{\omega(1)} \max(p, B_e, \alpha_1)^2 \ .$$

For any $\mathsf{pp} \in \mathsf{Supp}\left(\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max})\right)$*,* $s_1, s_2 \in \mathbb{Z}^{\ell_s}$*,* $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}^{\psi}$ *where* $\psi \leq \Psi$*, and function* $f(x_{res}, x_1) = x_{res} - yx_1$*, where* $|y| < p$*, the following two ciphertexts are statistically close, i.e,* $\Delta_{\mathsf{SD}}\left(\mathsf{ct}'_2, \mathsf{ct}'_R\right) \leq \mathrm{negl}\lambda$*.*

Sampling ct'_2:

- *generate fresh ciphertext* $\mathsf{ct}_2 \leftarrow \overline{\mathsf{Enc}}(s_2, \mathbf{m}_2)$.
- *sample noise* $\mathbf{e} \leftarrow [-\alpha_2, \alpha_2]^{\psi}$*, and encrypt it using key* $\mathbf{0}$*,* $\mathsf{ct}_e \leftarrow \mathsf{Enc}(0, \mathbf{e})$.
- *smudge noise in* ct_2 *via* $\mathsf{ct}'_2 \leftarrow \overline{\mathsf{Eval}}(+, \mathsf{ct}_2, \mathsf{ct}_e)$.

SAMPLING ct'_R:

- *generate fresh ciphertext* $\mathsf{ct}_1 \leftarrow \overline{\mathsf{Enc}}(s_1, \mathbf{m}_1)$.
- *sample noise* $\mathbf{e}_1 \leftarrow [-\alpha_1, \alpha_1]^{\psi}$*, and encrypt it using key 0,* $\mathsf{ct}_{e,1} \leftarrow \mathsf{Enc}(0, \mathbf{e}_1)$.
- *generate additionally noisy ciphertext* $\mathsf{ct}'_1 \leftarrow \overline{\mathsf{Eval}}(+, \mathsf{ct}_1, \mathsf{ct}_{e,1})$.
- *generate fresh ciphertext* $\mathsf{ct}_{res} \leftarrow \overline{\mathsf{Enc}}(s_{res}, \mathbf{m}_{res})$*, where* $s_{res} = ys_1 + s_2$*, and* $\mathbf{m}_{res} = y\mathbf{m}_1 + \mathbf{m}_2 \mod p$.
- *homomophically evaluate* $f(x_{res}, x_1) = x_{res} - yx_1$ *to obtain* $\mathsf{ct}_R \leftarrow \overline{\mathsf{Eval}}(f, \mathsf{ct}_{res}, \mathsf{ct}'_1)$.
- *smudge noise in* ct_R *via* $\mathsf{ct}'_R \leftarrow \overline{\mathsf{Eval}}(+, \mathsf{ct}_R, \mathsf{ct}_e)$*, using the same* ct_e *as above.*

THE SIMPLER CASE: *The statistical closeness also holds when* $\alpha_1 = 0$ *and* ct'_R *is generated using* ct_1 *directly, instead of* ct'_1.

Lemma 3.1. First, by the correctness of lhe, we have

$$\begin{aligned} \mathsf{Dec}(0, \mathsf{ct}_{e,i}) &= \mathbf{e}_i + \mathbf{e}_{e,i} \mod P, \\ \mathsf{Dec}(0, \mathsf{ct}_e) &= \mathbf{e} + \mathbf{e}_e \mod P, \\ \text{for } i = 1,2, \quad \mathsf{Dec}(s_i, \mathsf{ct}_i) &= \mathsf{Dec}(s_i, \mathsf{Enc}(s_i, \mathbf{m}_{i,p}\Delta)) \\ &= \mathbf{m}_{i,p}\Delta + \mathbf{e}_{m,i} \mod P, \\ \mathsf{Dec}(s_{res}, \mathsf{ct}_{res}) &= \mathsf{Dec}(s_{res}, \mathsf{Enc}(s_{res}, \mathbf{m}_{res}\Delta)) \\ &= \mathbf{m}_{res}\Delta + \mathbf{e}_{res} \mod P, \end{aligned}$$

for some $\mathbf{e}_{res}, \mathbf{e}_{e,i}, \mathbf{e}_{m,i} \in [-B_e, B_e]^{\psi}$, and $\mathbf{m}_{i,p} = \mathbf{m}_i \mod p$. Next, by the linear homomorphism of lhe, for $i = 1, 2$ we have

$$\begin{aligned} \mathsf{Dec}(s_i, \mathsf{ct}'_i) &= \mathsf{Dec}(s_i + 0, \mathsf{Eval}(+, \mathsf{ct}_i, \mathsf{ct}_{e,i})) \\ \text{(Definition 3.6)} &= \mathsf{Dec}(s_i, \mathsf{ct}_i) + \mathsf{Dec}(0, \mathsf{ct}_{e,i}) \\ &= \mathbf{m}_{i,p}\Delta + \underbrace{\mathbf{e}_i + \mathbf{e}_{m,i} + \mathbf{e}_{e,i}}_{=\mathbf{e}_{L,i}} \mod P, \end{aligned}$$

where $\|\mathbf{e}_{L,i}\|_\infty \leq \alpha_i + 2B_e$, and

$$\begin{aligned}\mathsf{Dec}(s_2, \mathsf{ct}_R) &= \mathsf{Dec}(f(s_{res}, s_1), \mathsf{Eval}(f, \mathsf{ct}_{res}, \mathsf{ct}'_1)) \\ \text{(Definition 3.6)} &= f(\mathsf{Dec}(s_{res}, \mathsf{ct}_{res}), \mathsf{Dec}(s_1, \mathsf{ct}'_1)) \\ &= (y\mathbf{m}_{1,p} + \mathbf{m}_{2,p})_p\Delta + \mathbf{e}_{res} - y(\mathbf{m}_{1,p}\Delta + \mathbf{e}_{L,1}) \mod P.\end{aligned}$$

By Claim 3.1, we have

$$(y\mathbf{m}_{1,p} + \mathbf{m}_{2,p})_p\Delta = (y\mathbf{m}_{1,p} + \mathbf{m}_{2,p})\Delta - \mathbf{e}_\epsilon \mod P,$$

where $\|\mathbf{e}_\epsilon\|_\infty \leq p^2 + p$. Hence

$$\begin{aligned}\mathsf{Dec}(s_2, \mathsf{ct}_R) &= (y\mathbf{m}_{1,p} + \mathbf{m}_{2,p})_p\Delta + \mathbf{e}_{res} - y\mathbf{m}_{1,p}\Delta - y\mathbf{e}_{L,1} \\ \text{(Claim 3.1)} &= (\cancel{y\mathbf{m}_{1,p}} + \mathbf{m}_{2,p})\Delta - \mathbf{e}_\epsilon + \mathbf{e}_{res} - \cancel{y\mathbf{m}_{1,p}\Delta} - y\mathbf{e}_{L,1} \\ &= \mathbf{m}_{2,p}\Delta \underbrace{-\mathbf{e}_\epsilon + \mathbf{e}_{res} - y\mathbf{e}_{L,1}}_{=\mathbf{e}_R} \mod P,\end{aligned}$$

where $\|\mathbf{e}_R\|_\infty \leq p^2 + p + B_e + p(\alpha_1 + 2B_e) = O(\max(p, B_e, \alpha_1)^2)$. Finally, we have

$$\begin{aligned}\mathsf{Dec}(s_2, \mathsf{ct}'_R) &= \mathsf{Dec}(s_2 + 0, \mathsf{Eval}(+, \mathsf{ct}_R, \mathsf{ct}_e)) \\ \text{(Claim 3.1)} &= \mathsf{Dec}(s_2, \mathsf{ct}_R) + \mathsf{Dec}(0, \mathsf{ct}_e) \\ &= \mathbf{m}_{2,p}\Delta + \underbrace{\mathbf{e}_R + \mathbf{e} + \mathbf{e}_e}_{\mathbf{e}'_R} \mod P.\end{aligned}$$

By the setting of α_2, we have $\Delta_{\mathsf{SD}}(\mathbf{e}_{L,2}, \mathbf{e}_2) \leq \text{negl}\lambda$, and $\Delta_{\mathsf{SD}}(\mathbf{e}'_R, \mathbf{e}_2) \leq \text{negl}\lambda$. Hence $\Delta_{\mathsf{SD}}(\mathbf{e}_{L,2}, \mathbf{e}'_R) \leq \text{negl}\lambda$. By the statistical closeness (Definition 3.7) of lhe, we conclude

$$\begin{aligned}&\Delta_{\mathsf{SD}}(\mathsf{ct}'_2, \mathsf{ct}'_R) \\ \text{(Definition 3.7)} &= \Delta_{\mathsf{SD}}(\mathsf{Dec}(s_2, \mathsf{ct}'_2), \mathsf{Dec}(s_2, \mathsf{ct}'_R)) \\ &\leq \Delta_{\mathsf{SD}}(\mathbf{e}_{L,2}, \mathbf{e}'_R) \leq \text{negl}\lambda.\end{aligned}$$

Similar analysis shows that the above also holds when ct'_R is generated using ct_1 (instead of ct'_1) directly. $\square$

Lemma 3.2 (homomorphic evaluation)**.** *Suppose the underlying LHE scheme* lhe *in Construction 1 satisfies linear homomorphism (Definition 3.6). For all* $\lambda, \Psi, p, \alpha_1, \alpha_2 \in \mathbb{N}$*, if the maximal noise level is set sufficient large*

$$B_{\max} \geq p(p+1+\alpha+2B_e) \ , \quad \alpha = \max(\alpha_1, \alpha_2)$$

then for all $\overline{\mathsf{pp}} \in \mathsf{Supp}\left(\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max})\right)$*,* $s_1, s_2 \in \mathbb{Z}^{\ell_s}$*,* $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}^\psi$ *where* $\psi \leq \Psi$*, homomorphic evaluation of functions of form* $f(x_1, x_2) = yx_1 + x_2$ *where* $|y| < p$ *on additionally noisy ciphertexts yields correct decryption:*

$$\Pr\left[\begin{array}{l|l} \overline{\mathsf{Dec}}(ys_1+s_2, \mathsf{ct}_{res}) & \forall i \in \{1,2\},\ \mathbf{e}_i, \leftarrow [-\alpha_i, \alpha_i]^\psi,\ \mathsf{ct}_{e,i} \leftarrow \mathsf{Enc}(0, \mathbf{e}_i), \\ & \mathsf{ct}_i \leftarrow \overline{\mathsf{Enc}}(s_i, \mathbf{m}_i),\ \mathsf{ct}'_i \leftarrow \overline{\mathsf{Eval}}(+, \mathsf{ct}_i, \mathsf{ct}_{e,i}) \\ \quad = y\mathbf{m}_1 + \mathbf{m}_2 \bmod p & \mathsf{ct}_{res} \leftarrow \overline{\mathsf{Eval}}(f, \mathsf{ct}'_1, \mathsf{ct}'_2) \end{array}\right] = 1.$$

The simpler case: *The above also holds when* $\alpha_1 = 0$ *and* ct'_{res} *is generated using* ct_1*, instead of* ct'_1.

Lemma 3.2. First, by the correctness of lhe, we have

$$\begin{aligned} \mathsf{Dec}(0, \mathsf{ct}_{e,i}) &= \mathbf{e}_i + \mathbf{e}_{e,i} \mod P, \\ \text{for } i = 1,2, \quad \mathsf{Dec}(s_i, \mathsf{ct}_i) &= \mathsf{Dec}(s_i, \mathsf{Enc}(s_i, \mathbf{m}_{i,p}\Delta)) \\ &= \mathbf{m}_{i,p}\Delta + \mathbf{e}_{m,i} \mod P, \end{aligned}$$

for some $\mathbf{e}_{e,1}, \mathbf{e}_{e,2}, \mathbf{e}_{m,1}, \mathbf{e}_{m,2} \in [-B_e, B_e]^\psi$, and $\mathbf{m}_{i,p} = \mathbf{m}_i \mod p$. Next, by the linear homomorphism (Definition 3.6) of lhe, for $i = 1, 2$ we have

$$\begin{aligned} \mathsf{Dec}(s_i, \mathsf{ct}'_i) &= \mathsf{Dec}(s_i + 0, \mathsf{Eval}(+, \mathsf{ct}_i, \mathsf{ct}_{e,i})) \\ {}_{\text{(Definition 3.6)}} &= \mathsf{Dec}(s_i, \mathsf{ct}_i) + \mathsf{Dec}(0, \mathsf{ct}_{e,i}) \\ &= \mathbf{m}_{i,p}\Delta + \mathbf{e}_i + \mathbf{e}_i + \mathbf{e}_{e,i} \mod P, \end{aligned}$$

and

$$\begin{aligned}
\mathsf{Dec}(ys_1 + s_2, \mathsf{ct}_{res}) &= \mathsf{Dec}(f(s_1, s_2), \mathsf{Eval}(f, \mathsf{ct}'_1, \mathsf{ct}'_2)) \\
\text{(Definition 3.6)} &= f(\mathsf{Dec}(s_1 \mathsf{ct}'_1), \mathsf{Dec}(s_2, \mathsf{ct}'_2)) \\
&= y(\mathbf{m}_{1,p}\Delta + \mathbf{e}_{m,1} + \mathbf{e}_1 + \mathbf{e}_{e,1}) \\
&\quad + \mathbf{m}_{2,p}\Delta + \mathbf{e}_{m,2} + \mathbf{e}_2 + \mathbf{e}_{e,2} \\
&= (y\mathbf{m}_{1,p} + \mathbf{m}_{2,p})\Delta + \mathbf{e}_f \mod P,
\end{aligned}$$

where $\mathbf{e}_f = y(\mathbf{e}_1 + \mathbf{e}_{m,1} + \mathbf{e}_{e,1}) + \mathbf{e}_2 + \mathbf{e}_{m,2} + \mathbf{e}_{e,2}$ over $\mathbb{Z}$. Since $|y| < p$, $\|\mathbf{e}_i\|_\infty \leq \alpha$, and $\|\mathbf{e}_{e,i}\|_\infty, \|\mathbf{e}_{m,i}\|_\infty \leq B_e$, we can bound

$$\|\mathbf{e}_f\|_\infty \leq p(\alpha + 2B_e).$$

We now analyse the term $(y\mathbf{m}_{1,p} + \mathbf{m}_{2,p})\Delta \mod P$ using the following claim.

Claim 3.1. *Let $P, p \in \mathbb{N}$ be two modulus, where $P > p$, and let $\Delta = \lfloor P/p \rfloor$. Let f be any linear function with d integer coefficients, and $\{\mathbf{m}_i\}_{i\in[d]} \in \mathbb{Z}^\psi$ of dimension $\psi \in \mathbb{N}$. Let $\mathbf{m}_f = f(\{\mathbf{m}_i\})$ evaluated coordinate-wise over $\mathbb{Z}$. (For more concise notations, we use the short hand $(x)_p$ to mean $x \mod p$ in the following derivations.) It holds that*

$$\mathbf{m}_f \cdot \Delta = (\mathbf{m}_f)_p \cdot \Delta + \mathbf{e}_\epsilon \mod P,$$

for some error vector $\mathbf{e}_\epsilon \in \mathbb{Z}^\psi$ with magnitude $\|\mathbf{e}_\epsilon\|_\infty \leq \|f(\{\mathbf{m}_i\})\|_\infty + p$.

Proof. We first write $\mathbf{m}_f = (\mathbf{m}_f)_p + p\mathbf{k}$ where $\mathbf{k} = \lfloor \mathbf{m}_f / p \rfloor$. By the setting of $\Delta = \lfloor P/p \rfloor = P/p - \epsilon$, $0 \leq \epsilon < 1$, we have

$$\begin{aligned}
\mathbf{m}_f\Delta = ((\mathbf{m}_f)_p + p\mathbf{k})\Delta &= (\mathbf{m}_f)_p\Delta + p\mathbf{k} \cdot (P/p - \epsilon) \\
&= (\mathbf{m}_f)_p\Delta + \cancel{\mathbf{k}P} \underbrace{-\epsilon p\mathbf{k}}_{\mathbf{e}_\epsilon} \mod P.
\end{aligned}$$

It remains to verify that

$$\|\mathbf{e}_\epsilon\|_\infty \leq p\|\mathbf{k}\|_\infty = p\|\lfloor \mathbf{m}_f/p \rfloor\|_\infty \leq p(\|\mathbf{m}_f\|_\infty/p + 1) \leq \|\mathbf{m}_f\|_\infty + p. \qquad \square$$

Using Claim 3.1, we have

$$\begin{aligned}\mathsf{Dec}(ys_1 + s_2, \mathsf{ct}_{res}) &= (y\mathbf{m}_{1,p} + \mathbf{m}_{2,p})\Delta + \mathbf{e}_f \\ &= (y\mathbf{m}_{1,p} + \mathbf{m}_{2,p})_p\Delta + \underbrace{\mathbf{e}_\epsilon + \mathbf{e}_f}_{\mathsf{noise}} \mod P,\end{aligned}$$

where $\|\mathbf{e}_\epsilon\|_\infty \leq \|y\mathbf{m}_{1,p} + \mathbf{m}_{2,p}\|_\infty + p \leq p^2 + p$. Hence

$$\|\mathsf{noise}\|_\infty \leq p^2 + p + p(\alpha + 2B_e) = p(p + 1 + \alpha + 2B_e).$$

As noted in $\overline{\mathsf{Setup}}$, our setting ensures $\Delta \geq 2B_{\max} \geq 2\|\mathsf{noise}\|_\infty$. Hence $\overline{\mathsf{Dec}}$ of ct_{res} succeeds by rounding off Δ. Similar analysis shows that decryption of ct'_{res} also succeeds. □

3.4.3 Instantiation Based on LWE

We first construct a noisy linearly homomorphic encryption (LHE) scheme under the learning with errors (LWE) assumption, which we state below.

Construction 2. This construction relies on learning with errors (LWE) assumption (Definition 2.10) in which the error distribution is bounded in $[-B_e, B_e]$ for some $B_e(\lambda) \leq q/2^{\sqrt{\lambda}}$.

- $\mathsf{Setup}(1^\lambda, 1^\Psi, B_{\text{msg}})$ chooses the key dimension $\ell_s = O(\lambda)$, chooses message modulus P such that $P \geq B_{\text{msg}}$. Sample a random matrix $A \leftarrow \mathbb{Z}_P^{\Psi \times \ell_s}$.

  It outputs $\mathsf{pp} = (P, A)$. The key space is $\mathbb{Z}^{\ell_s}$. The message modulus is P. The ciphertext space is $\mathbb{Z}^{\leq \Psi}$.
- KeyGen samples a random $\mathbf{s} \leftarrow \mathbb{Z}_P^{\ell_s}$.
- $\mathsf{Enc}(\mathbf{s}, \mathbf{m})$, for message vector of length $\psi \leq \Psi$, samples vector $\mathbf{e} \leftarrow \chi^\psi$ from the error distribution χ, and outputs ciphertext

$$\mathsf{ct} := A_{1:\psi}\mathbf{s} + \mathbf{e} + \mathbf{m} \quad (\text{over } \mathbb{Z}_P)$$

  where $A_{1:\psi}$ denotes the first ψ rows of A.
- $\mathsf{Dec}(\mathbf{s}, \mathsf{ct})$, for ciphertext $\mathsf{ct} = (c_1, \ldots, c_\psi) \in \mathcal{E}$, outputs $\mathbf{m}' = \mathsf{ct} - A_{1:\psi}\mathbf{s}$.

- $\mathsf{Eval}(f, \{\mathsf{ct}_i\})$ takes as input a linear function $f(x_1, \ldots, x_d) = \sum_{i\in[d]} a_i x_i$, and d ciphertexts $\{\mathsf{ct}_i\}_{i\in[d]}$. It outputs $\mathsf{ct}_f = \sum_i a_i \mathsf{ct}_i$.

Correctness. The decrypted message is

$$\mathbf{m}' = \mathsf{Dec}(\mathbf{s}, \mathsf{Enc}(\mathbf{s}, \mathbf{m})) = \mathsf{Dec}(\mathbf{s}, A_{1:\psi}\mathbf{s} + \mathbf{e} + \mathbf{m}) = \mathbf{e} + \mathbf{m},$$

where $\mathbf{e} \leftarrow \chi^{\psi}$ is sampled by the encryption algorithm. Therefore,

$$\|\mathbf{m}' - \mathbf{m}\|_{\infty} = \|\mathbf{e}\|_{\infty} \leq B_e.$$

Linear Homomorphism. It follows directly form that fact that Dec is a linear function.

One-time Security. Put Construction 2 in the definition of one-time security, we need to show that for any $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}^{\psi}$

$$\{A, A_{1:\psi}\mathbf{s} + \mathbf{e} + \mathbf{m}_1\} \approx_c \{A, A_{1:\psi}\mathbf{s} + \mathbf{e} + \mathbf{m}_2\} \left| \begin{array}{c} A \leftarrow \mathbb{Z}_P^{\Psi\times\ell_s}, \\ \mathbf{s} \leftarrow \mathbb{Z}^{\ell_s},\ \mathbf{e} \leftarrow \chi^{\psi} \end{array} \right.$$

By the LWE assumption,

$$\begin{array}{l} \{A, A_{1:\psi}\mathbf{s} + \mathbf{e} + \mathbf{m}_1\} \approx_c \{A, \mathbf{u} + \mathbf{m}_1\} \\ \approx_s \{A, \mathbf{u} + \mathbf{m}_2\} \approx_c \{A, A_{1:\psi}\mathbf{s} + \mathbf{e} + \mathbf{m}_2\} \end{array} \left| \begin{array}{c} A \leftarrow \mathbb{Z}_P^{\Psi\times\ell_s},\ \mathbf{u} \leftarrow \mathbb{Z}_P^{\psi}, \\ \mathbf{s} \leftarrow \mathbb{Z}^{\ell_s},\ \mathbf{e} \leftarrow \chi^{\psi} \end{array} \right.$$

3.4.4 Instantiation Based on Paillier

We next construct a LHE scheme under the decisional composite residuosity (DCR) assumption (Definition 2.9), which we state below.

Construction 3. This construction relies on the decisional composite residuosity (DCR) assumption.

- $\mathsf{Setup}(1^{\lambda}, 1^{\Psi}, B_{\text{msg}})$ samples two safe primes $p, q \leftarrow \mathsf{SP}(1^{\lambda})$. Let $N := pq$. Choose ζ as the minimum integer that $N^{\zeta} \geq B_{\text{msg}}$ and let $P := N^{\zeta}$. Sample Ψ random generators

$\tau_1, \ldots, \tau_\Psi \leftarrow \mathsf{HC}_{N^{\zeta+1}}$. That is, it samples a random $a \leftarrow \mathbb{Z}^*_{N^{\zeta+1}}$ and sets $g = a^{2N^\zeta}$, then samples $t_1, \ldots, t_\Psi \leftarrow [N \cdot 2^\lambda]$ and sets $\tau_1 = g^{t_1}, \ldots, \tau_\Psi = g^{t_\Psi}$.

It outputs $\mathsf{pp} = (N, \zeta, \tau_1, \ldots, \tau_\Psi)$. The key space is $\mathbb{Z}$ (i.e. $\ell_s = 1$). The message modulus is P. The ciphertext space is $(\mathbb{Z}^*_{N^{\zeta+1}})^{\leq \Psi}$.

- KeyGen samples a random $s \leftarrow [N/4]$.
- $\mathsf{Enc}(s, \mathbf{m})$, for message vector $\mathbf{m} = (m_1, \ldots, m_\psi) \in \mathbb{Z}^\psi$ of dimension $\psi \leq \Psi$, outputs a ciphertext

$$\mathsf{ct} := (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{2\mathbf{m}} = \left(\tau_1^s \cdot (1+N)^{2m_1}, \ldots, \tau_\psi^s \cdot (1+N)^{2m_\psi}\right).$$

- $\mathsf{Dec}(s, \mathsf{ct})$, for ciphertext $\mathsf{ct} = (c_1, \ldots, c_\psi) \in \mathcal{E}$, for each $i \in [\psi]$, computes $\tau_i^{-s} \cdot c_i$ and decodes $m'_i \in [N^\zeta]$ that $(1+N)^{m'_i} = \tau_i^{-s} \cdot c_i$. ($m'_i$ can be efficiently decoded from $(1+N)^{m'_i}$, as shown in [DJ01].) It outputs $\mathbf{m}' = (m'_1, \ldots, m'_\psi)$.
- $\mathsf{Eval}(f, \{\mathsf{ct}_i\})$ takes as input a linear function $f(x_1, \ldots, x_d) = \sum_{i \in [d]} a_i x_i$, and d ciphertexts $\{\mathsf{ct}_i\}_{i \in [d]}$. It outputs $\mathsf{ct}_f = \prod_i \mathsf{ct}_i^{a_i}$.

Correctness. The correctness of the decryption is shown in [DJ01].

Linear Homomorphism. To show the correctness of the homomorphic evaluation: Assume ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_d$ are the encryption of $\mathbf{m}_1, \ldots, \mathbf{m}_d$ using keys $s_1, \ldots, s_d$ respectively. Then for any linear function $f(x_1, \ldots, x_d) = \sum_{i \in [d]} a_i x_i$,

$$\begin{aligned}
\mathsf{Eval}(f, \{\mathsf{ct}_i\}) = \prod_i \mathsf{ct}_i^{a_i} &= \prod_i \Big((\tau_1, \ldots, \tau_\psi)^{s_i} \cdot (1+N)^{2\mathbf{m}_i} \Big)^{a_i} \\
&= (\tau_1, \ldots, \tau_\psi)^{\sum_i a_i s_i} \cdot (1+N)^{2 \sum_i a_i \mathbf{m}_i} \\
&= (\tau_1, \ldots, \tau_\psi)^{f(s_1, \ldots, s_d)} \cdot (1+N)^{2 f(\mathbf{m}_1, \ldots, \mathbf{m}_d)},
\end{aligned}$$

which is the encryption of $f(\mathbf{m}_1, \ldots, \mathbf{m}_d)$ under key $f(s_1, \ldots, s_d)$.

One-time Security. Put Construction 3 in the definition of one-time security, we need to

show that for any $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{Z}^\psi$

$$\{\tau_1, \ldots, \tau_\psi, \mathsf{ct}_1\} \approx_c \{\tau_1, \ldots, \tau_\psi, \mathsf{ct}_2\} \,.\, \left| \begin{array}{c} p, q \leftarrow \mathsf{SP}(\lambda),\ N = p \cdot q, \\ \tau_1, \ldots, \tau_\psi \leftarrow \mathsf{HC}_{N^{\zeta+1}}, \\ s \leftarrow [N/4], \\ \mathsf{ct}_i = (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{2\mathbf{m}_i} \end{array} \right.$$

As shown by [HO12], DCR implies the Extended-DDH assumption

$$\left\{ \begin{array}{c} \tau_1, \ldots, \tau_\psi \\ (\tau_1, \ldots, \tau_\psi)^s \end{array} \right\} \approx_c \left\{ \begin{array}{c} \tau_1, \ldots, \tau_\psi \\ (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{\mathbf{r}} \end{array} \right\} \left| \begin{array}{c} p, q \leftarrow \mathsf{SP}(\lambda),\ N = p \cdot q, \\ \tau_1, \ldots, \tau_\psi \leftarrow \mathsf{HC}_{N^{\zeta+1}} \\ \mathbf{r} \leftarrow [N^\zeta]^\psi \\ s \leftarrow [N/4] \end{array} \right.$$

Since $(\tau_1, \ldots, \tau_\psi)^s$ is computationally indistinguishable from $(\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{\mathbf{r}}$, we can consider a hybrid world where $(\tau_1, \ldots, \tau_\psi)^s$ is replaced by $(\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{\mathbf{r}}$. In such hybrid world, ct_1 and ct_2 are perfectly indistinguishable, as they are one-time padded by $\mathbf{r}$.

$$\begin{array}{l} \{\tau_1, \ldots, \tau_\psi, \mathsf{ct}_1\} \\ = \{\tau_1, \ldots, \tau_\psi, (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{2\mathbf{m}_1}\} \\ \approx_c \{\tau_1, \ldots, \tau_\psi, (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{\mathbf{r}+2\mathbf{m}_1}\} \\ \approx_s \{\tau_1, \ldots, \tau_\psi, (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{\mathbf{r}+2\mathbf{m}_2}\} \\ \approx_c \{\tau_1, \ldots, \tau_\psi, (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{2\mathbf{m}_2}\} \\ = \{\tau_1, \ldots, \tau_\psi, \mathsf{ct}_2\}. \end{array} \left| \begin{array}{c} p, q \leftarrow \mathsf{SP}(\lambda),\ N = p \cdot q, \\ \tau_1, \ldots, \tau_\psi \leftarrow \mathsf{HC}_{N^{\zeta+1}}, \\ \mathbf{r} \leftarrow [N^{\zeta+1}]^\psi, \\ s \leftarrow [N/4], \\ \mathsf{ct}_i = (\tau_1, \ldots, \tau_\psi)^s \cdot (1+N)^{2\mathbf{m}_i} \end{array} \right.$$

3.5 Construction of Main Tool: Key Extension for Bounded Integers

In this section, we construct the key-extension gadget for B-bounded integer computation. Our starting point is the following observation: A B-bounded computation can be "embedded" in modulo-p computation as long as $p > 2B$:

$$(C, x) \text{ is } B\text{-bounded} \quad \wedge \quad p > 2B \quad \Longrightarrow \quad C(x) \text{ over } \mathbb{Z} = C(x) \bmod p.$$

Therefore, we can directly use the (information theoretic) arithmetic operation gadget for ring $\mathbb{Z}_p$ from AIK (recalled in Section 3.9.1). What remains is to design a key-extension gadget for $\mathbb{Z}_p$, i.e., a mechanism that enables expanding a short label $\mathbf{a}x + \mathbf{b} \bmod p$ to an arbitrarily long label $\mathbf{c}x + \mathbf{d} \bmod p$.

As shown in this section, the fact that every intermediate values x is bounded tremendously simplifies the key extension gadget, especially if it is compared with the key extension gadget for modular computation in Section 3.7.

3.5.1 The Setup Algorithm

Our key extension gadget for bounded integer uses the special-purpose LHE scheme $\overline{\mathsf{lhe}}$ in Construction 1. The parameters of the LHE scheme is setup once by the Setup algorithm of the entire garbling scheme, as shown in Figure 3.4, and is shared by all invocation of gadgets when garbling an arithmetic circuit. The label space of the garbling scheme is $\mathcal{L} = \mathbb{Z}_p$.

We emphasize that the Setup algorithm depends only on the security parameter and the integer bound B. It's independent of any parameters (e.g., maximal size, fan-out, depth) of the circuit to be garbled later. As such, the public parameter pp is generated once and re-used for garbling many poly-sized circuits.

3.5.2 Length-Doubling Key Extension

We present the construction in two steps:

Step 1: Length-doubling. In Construction 4, we present a basic *length-doubling* key extension gadget, that is, at evaluation time, given a label $\mathbf{z}_1^{in}x + \mathbf{z}_2^{in}$ of dimension ℓ produces a label $\mathbf{z}_1^{out}x + \mathbf{z}_2^{out}$ of dimension 2ℓ. This construction already contains our main idea.

Step 2: Arbitrary Expansion. Next, we present a generic transformation 1 in Section 3.5.3 that converts a *length-doubling* key extension gadget, to a full-fledged key extension gadget that produces an output-wire label of arbitrary polynomial dimension $\ell' > \ell$.

At a high-level, the transformation recursively calls the *length-doubling* key extension gadget in a tree fashion till the desired output-wire label dimension ℓ' is reached.

Construction 4 (length-doubling key extension for bounded integers)**.** The algorithms below uses the parameters, p, α, $B_{\max}$, Ψ, specified in Setup (Figure 3.4), and have random access to the public parameters pp, which contains the public parameter $\overline{\mathsf{pp}}$ of the LHE scheme $\overline{\mathsf{lhe}}$ and the key dimension ℓ.

- $\mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell)$: Generate a $\overline{\mathsf{lhe}}$ secret key $\mathbf{s}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s})$, which is an integer vector in $\mathbb{Z}^{\ell_s}$ with $\|\mathbf{s}_1\|_\infty \leq B_s$. Output input-wire keys $\mathbf{z}_1, \mathbf{z}_2$:

$$\mathbf{z}_1^{in} = (\mathbf{s}_1, 1), \quad \mathbf{z}_2^{in} = (r\mathbf{s}_1 + \mathbf{s}_2, r) \quad (\text{over } \mathbb{Z}),$$

  where $r \leftarrow [-B_{\text{smdg}}, B_{\text{smdg}}]$ and $\mathbf{s}_2 \leftarrow [-B'_{\text{smdg}}, B'_{\text{smdg}}]^{\ell_s}$, with $B_{\text{smdg}} = \lambda^{\omega(1)}B$ and $B'_{\text{smdg}} = \lambda^{\omega(1)}B_{\text{smdg}}B_s < p/4$ (the inequality can be satisfied because the message modulus p is set sufficient large; see Equation (3.4)).

  Note: *We make a few observations: i) the input-wire keys are p bounded, that is, they belong to the label/key space $\mathbf{z}_1^{in}, \mathbf{z}_1^{in} \in \mathbb{Z}_p$ as the definition requires, and ii) a label for x equals*

$$\begin{aligned} \mathbf{L}^{in} = \mathbf{z}_1^{in}x + \mathbf{z}_2^{in} &= (\mathbf{s}_1(x+r) + \mathbf{s}_2,\ x+r) \bmod p \\ &= (\underbrace{\mathbf{s}_1(x+r) + \mathbf{s}_2}_{\mathbf{s}_{res}},\ \underbrace{x+r}_{y}) \ \textit{over } \mathbb{Z} \end{aligned}$$

  The last equality holds because the magnitude of entries of $\mathbf{s}_{res}$ and y do not exceed $p/2$. The fact that the labels are effectively computed over the integers is crucial for decoding later, and this crucially relies on the fact that values x are B-bounded and that p can be set sufficiently larger than B.

- $\mathsf{KE.Garb}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in})$: First recover $\mathbf{s}_1, \mathbf{s}_2, r$ from the input-wire keys. Then encrypt $\mathbf{z}_1^{out}$ and $\mathbf{z}'^{out}_2 = \mathbf{z}_2^{out} - r\mathbf{z}_1^{out}$ using $\overline{\mathsf{lhe}}$ under keys $\mathbf{s}_1, \mathbf{s}_2$ respectively. This is

possible because $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$ has dimension $2\ell \leq \Psi$, as set in Figure 3.4, and any integer vector of dimension ℓ_s, e.g. $\mathbf{s}_2$, can be used as a secret key for $\overline{\mathsf{lhe}}$.

$$\mathsf{ct}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1, \mathbf{z}_1^{out}), \quad \mathsf{ct}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_2, \mathbf{z'}_2^{out}).$$

Finally, add a smudging noise of magnitude α (set in Equation (3.5)) to ct_2 to obtain ct_2', and output garbled table $\mathsf{tb} = (\mathsf{ct}_1, \mathsf{ct}_2')$.

$$\mathbf{e} \leftarrow [-\alpha, \alpha]^{\ell'} \;, \quad \mathsf{ct}_e \leftarrow \mathsf{lhe}.\mathsf{Enc}(0, \mathbf{e}) \;, \quad \mathsf{ct}_2' \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_2, \mathsf{ct}_e) \;.$$

- $\mathsf{KE.Dec}^{\mathsf{PP}}(\mathbf{L}^{in}, \mathsf{tb} = (\mathsf{ct}_1, \mathsf{ct}_2'))$ Treat $\mathbf{L}^{in}$ as an integer vector and parse it as $\mathbf{L}^{in} = (\mathbf{s}_{res}, y)$, where $\mathbf{s}_{res} \in \mathbb{Z}^{\ell_s}$, $y \in \mathbb{Z}$. Homomorphically evaluate the linear function $f(x_1, x_2) = yx_1 + x_2$ over ct_1 and $\mathsf{ct'}_2$, decrypt the output ciphertext to obtain $\mathbf{m}_{res}$, and output $\mathbf{L}^{out} = \mathbf{m}_{res}$ as the output-wire label:

$$\mathsf{ct}_{res}' \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f, \mathsf{ct}_1, \mathsf{ct'}_2) \;, \quad \mathbf{L}^{out} = \mathbf{m}_{res} = \overline{\mathsf{lhe}}.\overline{\mathsf{Dec}}(s_{res}, \mathsf{ct}_{res}') \;.$$

Correctness. We show that the above scheme is *correct*, which requires that given a correctly generated input-wire label $\mathbf{L}^{in} = \mathbf{z}_1^{in} x + \mathbf{z}_2^{in} \pmod p$ and garbled table tb, the decoding algorithm $\mathsf{KE.Dec}$ recovers the correct output-wire label $\mathbf{L}^{out} = \mathbf{z}_1^{out} x + \mathbf{z}_2^{out} \pmod p$. As we analyzed above $\mathbf{L}^{in} = (\mathbf{s}_{res}, y)$ where $\mathbf{s}_{res} = \mathbf{s}_1 y + \mathbf{s}_2$ and $y = x + r$ are computed over the integers. By construction, $\mathsf{KE.Dec}$ uses $\mathbf{s}_{res}$ as the secret key to decrypt the $\overline{\mathsf{lhe}}$ ciphertext ct_{res}', where ct_{res}' is the output ciphertext obtained by homomorphically evaluating $f(x_1, x_2) = yx_1 + x_2$ over ct_1 and ct_2 encrypting $\mathbf{z}_1^{out}$ and $\mathbf{z'}_2^{out}$ respectively. By the special-purpose linear homomorphism of $\overline{\mathsf{lhe}}$, namely Lemma 3.2 (the simpler case), ct_{res}' can be decrypted using secret key $f(\mathbf{s}_1, \mathbf{s}_2) = \mathbf{s}_1 y + \mathbf{s}_2$ computed over the integers, which is exactly $\mathbf{s}_{res}$. Therefore,

$$\begin{aligned} \mathbf{m}_{res} &= \overline{\mathsf{lhe}}.\overline{\mathsf{Dec}}(\mathbf{s}_{res} = (\mathbf{s}_1 y + \mathbf{s}_2), \mathsf{ct}_{res}') = (y\mathbf{z}_1^{out} + \mathbf{z'}_2^{out}) \bmod p \\ &= (\underbrace{(x + r)}_{=y} \mathbf{z}_1^{out} + \underbrace{\mathbf{z}_2^{out} - r\mathbf{z}_1^{out}}_{\mathbf{z'}_2^{out}}) \bmod p = \mathbf{z}_1^{out} x + \mathbf{z}_2^{out} \bmod p = \mathbf{L}^{out} \;. \end{aligned}$$

In order to invoke Lemma 3.2, we still need to verify that the prerequisite $B_{\max} \geq p(p + 1 + \alpha + 2B_e)$ is indeed satisfied. This is the case as set by Setup in Figure 3.4.

Lemma 3.3. *Construction 4 is* secure *per Definition 3.1.*

Lemma 3.3. We construct a simulator $\mathsf{KE.Sim}$, that on input a security parameter λ, public parameters $\mathsf{pp} = (\overline{\mathsf{lhe}}.\overline{\mathsf{pp}}, \alpha)$ generated by Setup in Figure 3.4, and an arbitrary output-wire label $\mathbf{L}^{out} \in \mathbb{Z}_p^{2\ell}$ of dimension 2ℓ, simulates the input-wire label $\widetilde{\mathbf{L}}^{in}$ and the garbled table $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{ct}}_1, \widetilde{\mathsf{ct}}_2')$.

- $\mathsf{KE.Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$: Simulate the input-wire label $\widetilde{\mathbf{L}}^{in} = (\widetilde{\mathbf{s}}_{res}, \widetilde{y})$, by sampling $\widetilde{\mathbf{s}}_{res}$ and $\widetilde{y}$ as sufficiently large random integer values.

  $$\widetilde{y} \leftarrow [-B_{\text{smdg}}, B_{\text{smdg}}], \quad \widetilde{\mathbf{s}}_{res} \leftarrow [-B'_{\text{smdg}}, B'_{\text{smdg}}]^{\ell_s}, \quad \widetilde{\mathbf{L}}^{in} = (\widetilde{\mathbf{s}}_{res}, \widetilde{y}),$$

  where $B_{\text{smdg}} = \lambda^{\omega(1)} B$, $B'_{\text{smdg}} = \lambda^{\omega(1)} B_{\text{smdg}} B_s$ are set to the same values as in $\mathsf{KE.KeyGen}$. Next, simulate the garbled table $\mathsf{tb} = (\widetilde{\mathsf{ct}}_1, \widetilde{\mathsf{ct}}_2')$ by generating the former $\widetilde{\mathsf{ct}}_1$ as a fresh encryption of $\mathbf{0}$, that is,

  $$\mathbf{s}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s}) \ , \quad \widetilde{\mathsf{ct}}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1, \mathbf{0}) \ , \ (\mathbf{0} \in \mathbb{Z}^{\ell'}) \ ,$$

  and sampling $\widetilde{\mathsf{ct}}_2'$ via homomorphic evaluation of $\overline{\mathsf{lhe}}$, subject to the constraint that decoding must produce the correct output-wire label $\mathbf{L}^{out}$. More specifically, consider the function $f_R(x_{res}, x_1) = x_{res} - yx_1$, and generate $\widetilde{\mathsf{ct}}_2$ as follows.

  $$\widetilde{\mathsf{ct}}_{res} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\widetilde{\mathbf{s}}_{res}, \mathbf{L}^{out}) \ , \quad \widetilde{\mathsf{ct}}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f_R, \widetilde{\mathsf{ct}}_{res} \ , \widetilde{\mathsf{ct}}_1).$$

  Finally, smudge the noise in $\widetilde{\mathsf{ct}}_2$ to produce $\widetilde{\mathsf{ct}}_2'$ as follows.

  $$\mathbf{e} \leftarrow [-\alpha, \alpha]^{\ell'} \ , \quad \mathsf{ct}_e \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}) \ , \quad \widetilde{\mathsf{ct}}_2' \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2, \mathsf{ct}_e).$$

We now argue that $\mathsf{KE.Sim}$ described above satisfies the security requirement. Consider any sequences $\{\mathbf{z}^{out}_{1,\lambda}, \mathbf{z}^{out}_{2,\lambda}\}_\lambda$ where $\mathbf{z}^{out}_{1,\lambda}, \mathbf{z}^{out}_{2,\lambda} \in \mathcal{L}^{2\ell}$, and $\{x_\lambda\}_\lambda$ where $x_\lambda \in \mathcal{I}$. We define four hybrids, $\mathsf{Hyb}_1, \ldots, \mathsf{Hyb}_4$, where the first hybrid is exactly the real-world distribution, and the last hybrid is exactly the simulated distribution using $\mathsf{KE.Sim}$, and show their indistinguishability. (In the following, we suppress the subscript λ.)

- Hyb_1: This hybrid generates $\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb} = (\mathsf{ct}_1, \mathsf{ct}_2')$ honestly using the algorithms Setup, $\mathsf{KE.KeyGen}$, and $\mathsf{KE.Garb}$. More concretely, the variables are sampled as follows:

  – Generate $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
  – Sample a $\overline{\mathsf{lhe}}$ secret key $\mathbf{s}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s})$, a random integer scalar $r \leftarrow [-B_{\text{smdg}}, B_{\text{smdg}}]$, and a random integer vector $\mathbf{s}_2 \leftarrow [-B'_{\text{smdg}}, B'_{\text{smdg}}]^{\ell_s}$.
  – The input label for x is $\mathbf{L}^{in} = (\mathbf{s}_{res}, y)$ where $y = x + r, \mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2$.
  – ct_1 is a fresh encryption of $\mathbf{z}_1^{out}$ under secret key $\mathbf{s}_1$, $\mathsf{ct}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(s_1, \mathbf{z}_1^{out})$.
  – ct_2' is an additionally noisy encryption of $(\mathbf{z}_2^{out} - r\mathbf{z}_1^{out})$ under key $\mathbf{s}_2$. That is, sample $\mathbf{e} \leftarrow [-\alpha, \alpha]^{2\ell}$ and generate

$$\mathsf{ct}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(s_2, (\mathbf{z}_2^{out} - r\mathbf{z}_1^{out})),\ \mathsf{ct}_e \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}),\ \mathsf{ct}_2' \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_2, \mathsf{ct}_e)\ .$$

- Hyb_2: This hybrid proceeds identically as Hyb_1, except that $\widetilde{\mathsf{ct}}_2'$ is generated via homomorphic evaluation of $\overline{\mathsf{lhe}}$, under the constraint that decryption recovers the correct output-wire label $\mathbf{L}^{out} = \mathbf{z}_1^{out}x + \mathbf{z}_2^{out}$. That is, Hyb_2 first generates $\widetilde{\mathsf{ct}}_{res}$ as

$$\mathbf{L}^{out} = x\mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p, \quad \widetilde{\mathsf{ct}}_{res} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_{res}, \mathbf{L}^{out}).$$

  We next compute $\widetilde{\mathsf{ct}}_2$ via homomorphic evaluation of the function $f_R(x_{res}, x_1) = x_{res} - yx_1$:

$$\widetilde{\mathsf{ct}}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f_R, \widetilde{\mathsf{ct}}_{res}, \mathsf{ct}_1),$$

  Finally, smudge $\widetilde{\mathsf{ct}}_2$ with noise $e \leftarrow [-\alpha, \alpha]$ to get $\widetilde{\mathsf{ct}}_2'$

$$\mathsf{ct}_e \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}),\ \widetilde{\mathsf{ct}}_2' \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2, \mathsf{ct}_e)\ .$$

  Note that the only difference between Hyb_1 and Hyb_2 lies in how ct_2' and $\widetilde{\mathsf{ct}}_2'$ are generated. In the former, ct_2' is an additionally noisy ciphertext of $(\mathbf{z}_2^{out} - r\mathbf{z}_1^{out})$ encrypted under secret key $\mathbf{s}_2$. In the latter, $\widetilde{\mathsf{ct}}_2'$ is the output ciphertext produced by homomorphically evaluating f_R on $\mathsf{ct}_{res}, \mathsf{ct}_1$, smudged with additional noise. It is easy to verify that $f_R(\mathbf{s}_{res}, \mathbf{s}_1) = \mathbf{s}_2$ and $f_R(\mathbf{L}^{out}, \mathbf{z}_1^{out}) = (\mathbf{z}_2^{out} - r\mathbf{z}_1^{out})$. Lemma 3.1 (the simpler case) shows that these two ways of generating ciphertexts are statistically close, provided that the the magnitude α of the smudging noises is sufficiently large. This is indeed the case

since $\alpha = \lambda^{\omega(1)} \max(p, B_e)^2$ (Equation (3.5)). Therefore by the lemma, the distributions of ct_2' in Hyb_1 and $\widetilde{\mathsf{ct}}_2'$ in Hyb_2 are statistically close, and so are these two hybrids.

- Hyb_3: This hybrid proceeds identically as Hyb_2, except that instead of computing $\mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2$ and $y = x + r$ over the integers as in Hyb_2, Hyb_3 directly samples $\mathbf{s}_{res}$ and y as follows:

$$\widetilde{y} \leftarrow [-B_{\text{smdg}}, B_{\text{smdg}}], \quad \widetilde{\mathbf{s}}_{res} \leftarrow [-B'_{\text{smdg}}, B'_{\text{smdg}}].$$

  The distributions of $(y, \mathbf{s}_{res})$ in Hyb_2 and $(\widetilde{y}, \widetilde{\mathbf{s}}_{res})$ Hyb_3 are statistically close. This is because in Hyb_2, the magnitude of r, $B_{\text{smdg}} = \lambda^{\omega(1)} B$, is superpolynomially larger than x, which is bounded by B. Hence, r statistically hides x and the distribution of y is statistically close that of $\widetilde{y}$. Similarly, the magnitude of entries of $\mathbf{s}_2$, $B'_{\text{smdg}} = \lambda^{\omega(1)} B_{\text{smdg}} B_s$, is superpolynomially larger than the entries of $y\mathbf{s}_1$, which are bounded by $B_{\text{smdg}} B_s$. Hence $\mathbf{s}_2$ statistically hides $y\mathbf{s}_1$ and $\mathbf{s}_2$ is statistically close to $\widetilde{\mathbf{s}}_2$. Therefore Hyb_2 and Hyb_3 are statistically close.

- Hyb_4: This hybrid proceeds identically as Hyb_3, except that instead of generating ct_1 as a fresh encryption of $\mathbf{z}_1^{out}$ using secret key $\mathbf{s}_1$ as in Hyb_3, ct_1 is now generated as an encryption of the zero vector $\mathbf{0}$ still using secret key $\mathbf{s}_1$. Observe that since in Hyb_3 and Hyb_4, $\mathbf{s}_{res}$ is sampled randomly, the secret key $\mathbf{s}_1$ is not used anywhere else except for generating ct_1. Therefore, by the one-time security of $\overline{\mathsf{lhe}}$ w.r.t. secret key $\mathbf{s}_1$, we have that Hyb_3 and Hyb_4 are computationally indistinguishable.

By a hybrid argument, we have that Hyb_1 and Hyb_4 are computationally indistinguishable. Since Hyb_1 samples $(\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb})$ exactly as in the real-world, and Hyb_4 samples $(\widetilde{\mathsf{pp}}, \widetilde{\mathbf{L}}^{in}, \widetilde{\mathsf{tb}})$ as the simulator $\mathsf{KE.Sim}$ does, we conclude that the simulated distribution is indistinguishable to the real distribution, and Construction 4 is a secure length-doubling key-extension gadget for bounded integer computation. $\square$

3.5.3 Arbitrary Expansion Key Extension

Next we present a generic transformation from length-doubling key expansion, to arbitrary expansion. We note that this transformation applies not only to key-expansion for bounded integer computation, but also for modular arithmetic computation handled in the next Section. This transformation starts with a length-doubling key-extension gadget $(\mathsf{KE.KeyGen}', \mathsf{KE.Garb}', \mathsf{KE.Dec}')$, and produces a new key-extension gadget $(\mathsf{KE.KeyGen}', \mathsf{KE.Garb}', \mathsf{KE.Dec}')$ that can expand the label length from ℓ to an arbitrary polynomial ℓ'. The basic idea is very simple: Keep calling the length-doubling gadget recursively in a tree-fashion, doubling the label-length at each recursive level, till the desired length ℓ' is reached.

Transformation 1 (length-doubling to arbitrary-expansion key extension)**.**

- $\mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell)$: Simply generate a pair of input-wire keys of the length-doubling scheme $(\mathbf{k}_1^\epsilon, \mathbf{k}_2^\epsilon) \leftarrow \mathsf{KE.KeyGen}'^{\mathsf{pp}}(1^\lambda, 1^\ell)$; the keys have dimension ℓ.
- $\mathsf{KE.Garb}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^\epsilon, \mathbf{z}_2^\epsilon)$: The output-wire keys $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$ have dimension ℓ', where $2^{D-1}\ell < \ell' \le 2^D\ell$ for some integer D. Divide each of $\mathbf{z}_1^{out}$ and $\mathbf{z}_2^{out}$ into 2^D chunks of dimension ℓ each: $\mathbf{z}_i^{out} = (\mathbf{z}_i^\gamma)_{\gamma\in\{0,1\}^D}$ (append 0's if $\mathbf{z}_i^{out}$ is shorter than 2^D). Consider a complete binary tree of depth D, every node $\gamma \in \{0,1\}^{\le D}$ in the tree is associated with a pair of keys:

  - The root is associated with $(\mathbf{z}_1^\epsilon, \mathbf{z}_2^\epsilon)$.
  - The γ'th leaf for $\gamma \in \{0,1\}^D$ is associated with $(\mathbf{z}_1^\gamma, \mathbf{z}_2^\gamma)$, a chunk in the output-wire keys.
  - The intermediate node indexed by $\gamma \ne \epsilon \in \{0,1\}^{<D}$ is associated with freshly sampled input-wire keys $(\mathbf{z}_1^\gamma, \mathbf{z}_2^\gamma) \leftarrow \mathsf{KE.KeyGen}'^{\mathsf{pp}}(1^\lambda, 1^\ell)$.

  For every non-leaf node $\gamma \in \{0,1\}^{<D}$, invoke the garbling algorithm of the length-doubling scheme to generate a garbled table

  $$\mathsf{tb}^\gamma \leftarrow \mathsf{KE.Garb}'^{\mathsf{pp}}((\mathbf{z}_1^{\gamma||0}, \mathbf{z}_1^{\gamma||1}), (\mathbf{z}_2^{\gamma||0}, \mathbf{z}_2^{\gamma||1}), \mathbf{z}_1^\gamma, \mathbf{z}_2^\gamma) \ .$$

  Output all the gabled tables generated $\mathsf{tb} = \{\mathsf{tb}^\gamma\}_{\gamma\in\{0,1\}^{\le D}}$.

- $\mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^{\epsilon}, \mathsf{tb})$: For every non-leaf node $\gamma \in \{0,1\}^{<D}$, invoke the decoding algorithm of the length-doubling gadget to expand the label from the root to the leaves.

$$(\mathbf{L}^{\gamma\|0}, \mathbf{L}^{\gamma\|1}) \leftarrow \mathsf{KE.Dec}'^{\mathsf{pp}}(\mathbf{L}^{\gamma}, \mathsf{tb}^{\gamma}) \ .$$

  Output all the labels associated with the leaves $\mathbf{L}^{out} = \{\mathbf{L}^{\gamma}\}_{\gamma \in \{0,1\}^D}$.

The correctness of the above key-extension gadget follows immediately from that of the underlying length-doubling gadget. The security follows as well. We describe the simulator here and omit the full proof. The simulator $\mathsf{KE.Sim}(1^{\lambda}, \mathsf{pp}, \mathbf{L}^{out})$ recursively calls the the simulator $\mathsf{KE.Sim}'$ of the underlying gadget. More specifically,

- $\mathsf{KE.Sim}(1^{\lambda}, \mathsf{pp}, \mathbf{L}^{out})$: For every non-leaf node $\gamma \in \{0,1\}^{<D}$, use the simulator of the length-doubling gadget to recursively simulate the garbled table and input-wire keys associated with node γ, from nodes in layer $D-1$ to the root.

$$\mathbf{L}^{\gamma}, \mathsf{tb}^{\gamma} \leftarrow \mathsf{KE.Sim}'(1^{\lambda}, \mathsf{pp}, (\mathbf{L}^{\gamma\|0}, \mathbf{L}^{\gamma\|1}))$$

3.6 Construction of Building Block: Linear Seeded Smudger

The linear seeded smudger that will be defined in this section is essentially a (linear) randomness extractor. It is well-known how to construct exactor that is linear over a finite field. Our challenge is that we require smudger to be linear over the integer ring $\mathbb{Z}$. Therefore, we define smudger so that

- The extracted randomness does not need to be close to uniform. (There does not exist an uniform distribution of $\mathbb{Z}$ in the first place.) We only require that the extracted randomness can "*smudge*" a given distribution with high probability.

  For a distribution $\mathcal{X}$ over $\{0,1\}^{\ell}$ and an extractor $E : \{0,1\}^{\ell} \to \mathbb{Z}$, let $E(\mathcal{X})$ denote the distribution of the extracted randomness. We say $E(\mathcal{X})$ *smudges* a distribution $\mathcal{D}$, if the two distributions

$$E(\mathcal{X}) \text{ and } E(\mathcal{X}) + \mathcal{D}$$

  are statistically close.

- Instead of considering any high-entropy source, we only require the smudger to work with the so-call *bix-fixing source*.

  For $\ell \in \mathbb{Z}$ and $p \in (0,1]$, a distribution $\mathcal{X}$ over $\{0,1\}^\ell$ is a (ℓ,p)-bix-fixing source, if $p\ell$ bits of it are i.i.d. uniform, and the remaining $(1-p)\ell$ bits are fixed. Note that (ℓ,p)-bix-fixing sources make up a family of distributions.

We introduce the notion of *linear seeded* $(\ell,p,\lambda_1,\lambda_2)$*-smudgers*. We say an seeded extractor is an $(\ell,p,\lambda_1,\lambda_2)$-smudger, if for every (ℓ,p)-bit-fixing source, the extracted randomness smudges any distribution over $\{0,\ldots,2^{\lambda_1}\}$ with an $O(2^{-\lambda_2})$ statistical error. We say an seeded extractor is an (ℓ,p,λ)-smudger, if it is an (ℓ,p,λ,λ)-smudger.

Definition 3.8 (Linear Seeded Smudger)**.** *A linear seeded smugder consists of two efficient algorithms:*

- $\mathsf{Smdg.Gen}(1^\ell,p) \to \mathbf{s}$*. Here (ℓ,p) is the parameter of bit-fixing source,* $\mathsf{Smdg.Gen}$ *samples the seed* $\mathbf{s}$*. As we are considering linear smudger, we assume w.l.o.g. that the seed* $\mathbf{s} \in \mathbb{Z}^\ell$*.*
- $\mathsf{Smdg.Smudge}(\mathbf{x};\mathbf{s}) \to \langle \mathbf{s},\mathbf{x}\rangle$*.*

Smdg *is a* $(\ell,p,\lambda_1,\lambda_2)$-smudger*, if for any (ℓ,p)-bit-fixing source $\mathcal{X}$ and for any distribution $\mathcal{D}$ over $\{0,1,\ldots,2^{\lambda_1}\}$, with probability at least $1-2^{-\lambda_2}$ over the randomness of sampling* $\mathbf{s} \leftarrow \mathsf{Smdg.Gen}(1^\ell,p)$*, the following two distribution (conditioning on* $\mathbf{s}$*)*

$$\mathsf{Smdg.Smudge}(\mathbf{x};\mathbf{s}) \text{ and } \mathsf{Smdg.Smudge}(\mathbf{x};\mathbf{s}) + \varepsilon$$

have statistical distance at most $2^{-\lambda_2}$, where $\mathbf{x},\varepsilon$ are sampled from $\mathcal{X},\mathcal{D}$ respectively.

Smdg *is an (ℓ,p,λ)-smudger is it is an $(\ell,p,\lambda_1,\lambda_2)$-smudger.*

Theorem 3.4. *For any p,λ, there exists $\ell = O(\lambda/p)$ such that there exists linear seeded (ℓ,p,λ)-smudger. Moreover, the coefficients of the smudger is bounded by $O(\lambda \cdot 2^{2\lambda})$.*

Proof. Let $X_1, \ldots, X_\ell \in \{0,1\}$ be i.i.d. uniform. Let $\mathbf{s} = (s_1, \ldots, s_\ell)$ be sampled from $\mathsf{Smdg.Gen}(1^\lambda, 1^\ell, p)$. The theorem says, for any $J \subseteq \{1, \ldots, \ell\}$ of size $p\ell$, with probability at least $1 - 2^{-\lambda}$ over the randomness of sampling $\mathbf{s}$,

$$\left(\sum_J s_j X_j\right) \approx_{2^{-\lambda}} \left(\mathcal{D} + \sum_J s_j X_j\right).$$

We let $s_1, \ldots, s_\ell$ be independently sampled from the uniform distribution of $\{1, \ldots, B\}$, where B will be chosen later. Since $s_1, \ldots, s_\ell$ are i.i.d., we can get rid of J, and it suffices to prove that

$$\left(\sum_{j=1}^{p\ell} s_j X_j\right) \approx_{2^{-\lambda}} \left(\mathcal{D} + \sum_{j=1}^{p\ell} s_j X_j\right)$$

with probability at least $1 - 2^{-\lambda}$ over the randomness of sampling $\mathbf{s}$. Concretely, we let each s_j to be independently sampled from $\{1, \ldots, B\}$.

Since the support of $\mathcal{D}$ is bounded by 2^λ, it suffices to prove

$$\left(\sum_{j=1}^{p\ell} s_j X_j\right) \approx_{2^{-2\lambda}} \left(1 + \sum_{j=1}^{p\ell} s_j X_j\right)$$

with probability at least $1 - 2^{-\lambda}$ over the randomness of sampling $\mathbf{s}$.

Define

$$u_s(x) = \begin{cases} 1, & \text{if } x = s \\ 0, & \text{otherwise.} \end{cases} \qquad g_s = \frac{1}{2}(u_0 + u_s) \qquad h = \frac{1}{2}(u_0 - u_1)$$

Then g_{s_j} is the probability mass function of $s_j X_j$. Let $\circ$ denote the convolution product that $(f \circ g)(x) = \sum_y f(y) g(x - y)$. Then $g_{s_1} \circ \cdots \circ g_{s_{p\ell}}$, $u_1 \circ g_{s_1} \circ \cdots \circ g_{s_{p\ell}}$ are the probability mass functions of $\sum_{j=1}^{p\ell} s_j X_j$, $1 + \sum_{j=1}^{p\ell} s_j X_j$ respectively. The statistical distance between them can be written as

$$\Delta_{\mathrm{SD}}\left(\sum_{j=1}^{p\ell} s_j X_j, 1 + \sum_{j=1}^{p\ell} s_j X_j\right)$$
$$= \frac{1}{2}\left\| g_{s_1} \circ \cdots \circ g_{s_{p\ell}} - u_1 \circ g_{s_1} \circ \cdots \circ g_{s_{p\ell}} \right\|_1 = \left\| h \circ g_{s_1} \circ \cdots \circ g_{s_{p\ell}} \right\|_1.$$

We use Fourier analysis to bound the above L1 distance. Let N be a sufficiently large number such that $N > 1 + p\ell B$. Thus $h, g_{s_1}, \ldots, g_{s_\ell}$ can be viewed as functions from $[N]$ to $\mathbb{R}$, and their convolution product $h \circ g_{s_1} \circ \cdots \circ g_{s_\ell}$ over $\mathbb{Z}$ is the same their convolution product modulo N.

For any $f : [N] \to \mathbb{C}$, its Fourier transform $\hat{f} : [N] \to \mathbb{C}$ is defined as

$$\hat{f}(k) = \sum_{y \in [N]} f(y) e^{-i \frac{k}{N} 2\pi y}.$$

And it is easy to verify that $f(x) = \frac{1}{N} \sum_k \hat{f}(k) \cdot e^{i \frac{k}{N} 2\pi x}$.

We abuse the notation and let $f = h \circ g_{s_1} \circ \cdots \circ g_{s_\ell}$, then

$$\|f\|_1 \leq \frac{1}{N} \sum_k |\hat{f}(k)| \cdot \|x \mapsto e^{i \frac{k}{N} 2\pi x}\|_1 = \sum_k |\hat{f}(k)| = \sum_k |\hat{h}(k)| \cdot |\hat{g}_{s_1}(k)| \cdot \ldots \cdot |\hat{g}_{s_\ell}(k)|.$$

The right-hand side of the inequality can be bounded by considering "small" k's and "large" k's separately.

To bound $|\hat{f}(k)|$ for "small" k (i.e. $k \leq \frac{2N}{B}$ or $k \geq N - \frac{2N}{B}$). It suffices to bound $|\hat{h}(k)|$,

$$|\hat{f}(k)| \leq |\hat{h}(k)| = \sin(\frac{k}{N}\pi) \leq \frac{\min(k, N-k)}{N} \cdot \pi \leq \frac{\pi}{B}.$$

We choose B such that $\frac{4N}{B} \cdot \frac{\pi}{B} \leq \frac{1}{2^{2\lambda+1}}$. Then

$$\sum_{\text{“small”} k} |\hat{f}(k)| \leq \sum_{\text{“small”} k} |\hat{h}(k)| \leq \frac{4N}{B} \cdot \frac{\pi}{B} \leq \frac{1}{2^{2\lambda+1}}. \tag{3.6}$$

To bound $|\hat{f}(k)|$ for "large" k (i.e. $\frac{2N}{B} < k < N - \frac{2N}{B}$). We have

$$\begin{aligned} \hat{g}_s(k) &= \frac{1 + e^{-i \frac{k}{N} 2\pi s}}{2} \\ \implies |\hat{g}_s(k)|^2 &= \frac{1 + e^{-i \frac{k}{N} 2\pi s}}{2} \frac{1 + e^{i \frac{k}{N} 2\pi s}}{2} = \frac{e^{i \frac{k}{N} 2\pi s} + 2 + e^{-i \frac{k}{N} 2\pi s}}{4}. \end{aligned}$$

As s is sampled uniformly from $\{1,\ldots,B\}$,

$$\begin{aligned}
\mathop{\mathbb{E}}_{s\leftarrow\{1,\ldots,B\}}\left[|\hat{g}_s(k)|^2\right] &= \frac{1}{B}\sum_{s=1}^{B}|\hat{g}_s(k)|^2\\
&= \frac{\frac{1}{B}\frac{e^{i\frac{k}{N}2\pi B}-1}{1-e^{-i\frac{k}{N}2\pi}}+2+\frac{1}{B}\frac{e^{-i\frac{k}{N}2\pi B}-1}{1-e^{i\frac{k}{N}2\pi}}}{4}\\
&\le \frac{\frac{1}{B}\frac{2}{|1-e^{-i\frac{k}{N}2\pi}|}+2+\frac{1}{B}\frac{2}{|1-e^{i\frac{k}{N}2\pi}|}}{4}\\
&= \frac{1}{2}+\frac{1}{B}\frac{1}{|1-e^{-i\frac{k}{N}2\pi}|}\\
&= \frac{1}{2}+\frac{1}{B\cdot|\sin(\frac{k}{N}\pi)|}\\
&\le \frac{1}{2}+\frac{1}{B\cdot|\sin(\frac{2}{B}\pi)|}.
\end{aligned}$$

As long as $B\ge 4$, we have $B\cdot|\sin(\frac{2}{B}\pi)|\ge 4$, thus

$$\mathop{\mathbb{E}}_{s\leftarrow\{1,\ldots,B\}}\left[|\hat{g}_s(k)|^2\right]\le\frac{3}{4}.$$

By Chernoff bound,

$$\Pr_{\mathbf{s}}\left[\frac{1}{p\ell}\sum_{j=1}^{p\ell}|\hat{g}_s(k)|^2\le\frac{3}{4}+\delta\right]\le e^{-d_{\mathrm{KL}}(\frac{3}{4}+\delta\|\frac{3}{4})\cdot p\ell}. \tag{3.7}$$

Let $\delta\in(0,1/4)$ be the solution of $-\frac{1}{4}\log(\frac{3}{4}+\delta)=d_{\mathrm{KL}}(\frac{3}{4}+\delta\|\frac{3}{4})=C$, where $C>0$ is a constant.

Let $p\ell=\frac{1}{C}\log(2N2^\lambda)$, then (3.7) says

$$\frac{1}{p\ell}\sum_{j=1}^{p\ell}|\hat{g}_s(k)|^2\le\frac{3}{4}+\delta \tag{3.8}$$

with probability at least $1-\frac{1}{2N2^\lambda}$. By the union bound, with probability at least $1-\frac{1}{2^{\lambda+1}}$, (3.8) holds for all "large" k.

By the inequality of arithmetic and geometric means, (3.8) implies

$$\prod_{j=1}^{p\ell}|\hat{g}_s(k)|^2\le\left(\frac{3}{4}+\delta\right)^{p\ell}=\left(\frac{1}{2N2^\lambda}\right)^4.$$

Therefore, with probability at least $1 - \frac{1}{2^{\lambda+1}}$,

$$\sum_{\text{“large”}\,k} |\hat{f}(k)| \leq \sum_{\text{“large”}\,k} \prod_{j=1}^{p\ell} |\hat{g}_s(k)| \leq \sum_{\text{“large”}\,k} \left(\frac{1}{2N2^{\lambda}}\right)^2 \leq \frac{1}{2^{2\lambda+1}}. \tag{3.9}$$

Finally, by combining (3.6) and (3.9) using the union bound, with probability at least $1 - \frac{1}{2^{\lambda}}$,

$$\begin{aligned}
&\Delta_{\mathrm{SD}}\left(\sum_{j=1}^{p\ell} s_j X_j, 1 + \sum_{j=1}^{p\ell} s_j X_j\right) \\
&\quad \leq \sum_{k} |\hat{f}(k)| = \sum_{\text{“small”}\,k} |\hat{f}(k)| + \sum_{\text{“large”}\,k} |\hat{f}(k)| \leq 2^{-2\lambda}.
\end{aligned}$$

As for the parameters, we require

$$N > 1 + p\ell B, \quad \frac{4N}{B} \cdot \frac{\pi}{B} \leq \frac{1}{2^{2\lambda+1}}, \quad B \geq 4, \quad p\ell = \frac{1}{C}\log(2N2^{\lambda}).$$

So it suffices to let $p\ell = O(\lambda)$, $B = O(\lambda \cdot 2^{2\lambda})$ and $N = O(\lambda^2 \cdot 2^{2\lambda})$. □

3.7 Construction of Main Tool: Key Extension for Modular Arithmetics

In our application, we will use a linear seeded smudger define and constructed in Section 3.6 for smudging LHE keys generated by the $\overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}$ algorithm, which has infinity norm bounded by B_s. To this end, we require a $(\ell, 1/4, \lambda_1, \lambda_2)$-smudger $\mathsf{Smdg} = (\mathsf{Smdg.Gen}, \mathsf{Smdg.Smudge})$, where the (log) smudging range is $\lambda_1 = \log B_s$, the (log) smudging distance is $\lambda_2 = \omega(\log \lambda)$, and the smudging source dimension is $\ell_{\mathrm{smdg}} = O(\lambda_1 + \lambda_2)$. To smudge an LHE key $\mathbf{s}^* \in \mathbb{Z}^{\ell_s}$, we will first generate a long source vector $\mathbf{s} \in \mathbb{Z}^{\ell^*}$, where $\ell^* = \ell_s \ell_{\mathrm{smdg}}$ and ℓ_s seeds $\mathsf{sd}_i \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{\mathrm{smdg}}}, 1/4)$ for $i \in [\ell_s]$. We then write the following shorthand

$$\mathbf{s}^* = \mathsf{Smdg.Smudge}(\mathbf{s}; \{\mathsf{sd}_i\}_{i \in [\ell_s]}) \tag{3.10}$$

to mean each component of $\mathbf{s}^*$ is computed by running $\mathsf{Smdg.Smudge}(\cdot; \mathsf{sd}_i)$ on the corresponding chunk of $\mathbf{s}$ of dimension ℓ_{smdg}.

3.7.1 The Setup Algorithm

Similarly to Section 3.5, we describe the Setup algorithm (Figure 3.5) of the entire garbling scheme that computes appropriate parameters for setting up the special-purpose LHE scheme $\overline{\mathsf{lhe}}$ in Construction 1. The label space of the garbling scheme is $\mathcal{L} = \mathbb{Z}_p$.

3.7.2 Key Extension

When constructing a key extension gadget for modular arithmetic over $\mathbb{Z}_p$, we need to solve a correctness issue. During KE.Dec, the algorithm receives a LHE key $\mathbf{s}_{res} = x\mathbf{s}_1 + \mathbf{s}_2 \mod p$, and needs to further recover $\mathbf{s}'_{res} = x\mathbf{s}_1 + \mathbf{s}_2$ over $\mathbb{Z}$. In the bounded integer computation model, where the input x has magnitude bounded by B, and the LHE keys $\mathbf{s}_1, \mathbf{s}_2$ have magnitude bounded by B_s, we can make $\mathbf{s}'_{res} = \mathbf{s}_{res}$ over $\mathbb{Z}$ by setting $p \gg B \cdot B_s$. However, in the modular arithmetic computation model (over $\mathbb{Z}_p$), the input x can have any value between $[0, p-1]$. It's no longer possible to set p such that $\mathbf{s}'_{res} = \mathbf{s}_{res}$ over $\mathbb{Z}$.

We first construct a key extension gadget for modular arithmetic computation over $\mathbb{Z}_p$ that solves the above issue at the cost of achieving a weaker security. We will then combine two instances of the weaker gadget to achieve full security. Similarly to Section 3.5.2, we first construct the weak and the fully secure key extension schemes under the assumption that the dimension of output-wire keys ℓ' is double the length of the dimension of input-wire keys ℓ, i.e. $\ell' = 2\ell$. Applying Transformation. 1 to the fully secure scheme then removes this restriction.

Construction 5 (length-doubling weak key extension for modular arithmetic)**.** This construction uses two LHE schemes $\overline{\mathsf{lhe}}$, lhe as ingredients, where $\overline{\mathsf{lhe}}$ has a fixed key magnitude bound $B_s = B_s(\lambda) < 2^{\mathrm{poly}(\lambda)}$. This construction additionally uses a $(\ell_{\mathrm{smdg}}, 1/4, \lambda_1, \lambda_2)$-seeded smudger scheme Smdg as guaranteed by Theorem 3.4.

- $\mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell)$: Generate two smudging source vectors $\mathbf{s}_1 \leftarrow \{0,1\}^{\ell^*}$ where $\ell^* = \ell_s \ell_{\mathrm{smdg}}$, and $\mathbf{s}_2$ as

$$\mathbf{r}_2 \leftarrow [0, \lfloor (p-1)/2 \rfloor]^{\ell^*}, \quad \mathbf{s}_2 = (\mathbf{1} - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{r}_2 \mod p. \tag{3.12}$$

Sample a one-time pad $r \leftarrow \mathbb{Z}_p$, and output $\mathbf{z}_1^{in} = (\mathbf{s}_1, 1)$, $\mathbf{z}_2^{in} = (r\mathbf{s}_1 + \mathbf{s}_2, r)$ computed over $\mathbb{Z}_p$.

Note: *For any $x \in \mathcal{I} = \mathbb{Z}_p$, let $y = x + r \mod p$, we have*

$$\mathbf{L}^{in} = x\mathbf{z}_1^{in} + \mathbf{z}_2^{in} = (y\mathbf{s}_1 + \mathbf{s}_2, y) \mod p.$$

We also define a convenient syntax $\mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r)$ to mean the algorithm uses the provided r value in the above description, while still sampling $\mathbf{s}_1, \mathbf{r}_2$ at random.

- $\mathsf{KE.Garb}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in})$: Recover smudging source vectors $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{Z}^{\ell^*}$ and a one-time pad $r \in \mathbb{Z}_p$ from the input-wire keys $\mathbf{z}_1^{in}, \mathbf{z}_2^{in}$. Sample ℓ_s smudging seeds $\mathsf{sd}_i \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{smdg}}, 1/4)$ for $i \in [\ell_s]$, and compute two LHE keys $\mathbf{s}_1^*, \mathbf{s}_2^* \in \mathbb{Z}^{\ell_s}$ from the source vectors.

$$\mathbf{s}_1^* = \mathsf{Smdg.Smudge}(\mathbf{s}_1; \{\mathsf{sd}_i\}), \quad \mathbf{s}_2^* = \mathsf{Smdg.Smudge}(\mathbf{s}_2; \{\mathsf{sd}_i\}).$$

Then encrypt $\mathbf{z}_1^{out}$ and $\mathbf{z}_2'^{out} = \mathbf{z}_2^{out} - r\mathbf{z}_1^{out} \pmod p$ under the LHE keys $\mathbf{s}_1^*, \mathbf{s}_2^*$ to produce ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$.

$$\mathsf{ct}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(s_1^*, \mathbf{z}_1^{out}), \quad \mathsf{ct}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(s_2^*, \mathbf{z'}_2^{out}).$$

Finally, add smudging noises $\mathbf{e}_i \leftarrow [-\alpha_i, \alpha_i]^{\ell^*}$ to $\mathsf{ct}_1, \mathsf{ct}_2$ via homomorphic evaluation to produce $\mathsf{ct}'_1, \mathsf{ct'}_2$.

$$i = 1, 2 \quad \mathsf{ct}_{e,i} \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}_i), \quad \mathsf{ct'}_i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_i, \mathsf{ct}_{e,i}).$$

The smudging noise magnitudes α_1, α_2 are set to $\alpha_1 = \lambda^{\omega(1)} \max(p, B_e)^2$, $\alpha_2 = \lambda^{\omega(1)}\alpha_1^2$, such that $\alpha_2 = \alpha$ as set in Eq. (3.11). Output the garbled table $\mathsf{tb} = (\mathsf{ct}'_1, \mathsf{ct}'_2, \{\mathsf{sd}_i\}_{i \in [\ell_s]})$.

Note: *The smudging seeds $\{\mathsf{sd}_i\}_{i \in [\ell_s]}$ in the above construction can computed using a PRG instead. We can output only the short PRG seed as an optimization.*

- $\mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$: Parse the input-wire label as $\mathbf{L}^{in} = (\mathbf{s}_{res}, y)$ where $\mathbf{s}_{res} \in \mathbb{Z}_p^{\ell^*}$, and $y \in \mathbb{Z}_p$, and $\mathsf{tb} = (\mathsf{ct}'_1, \mathsf{ct}'_2, \{\mathsf{sd}_i\}_{i \in [\ell_s]})$. Treat $\mathbf{s}_{res}, y$ as values over $[0, p-1] \subset \mathbb{Z}$, and compute a smudging source vector $\mathbf{s}'_{res} \in \mathbb{Z}^{\ell^*}$ (with components $s'_{res,i}$) as follows:

$$s'_{res,i} = \begin{cases} s_{res,i} + p & \text{if } y > \lfloor p/2 \rfloor, s_{res,i} < \lfloor p/2 \rfloor \\ s_{res,i} & \text{otherwise.} \end{cases} \tag{3.13}$$

  Then use $\mathbf{s}'_{res}$ to compute a LHE key $\mathbf{s}^*_{res} = \mathsf{Smdg.Smudge}(\mathbf{s}'_{res}; \{\mathsf{sd}_i\})$. Finally, recover $\mathbf{m}_{res}$ by computing homomorphically evaluating the function $f(x_1, x_2) = yx_1 + x_2$ over $\mathsf{ct}'_1, \mathsf{ct}'_2$, and decrypt the output ciphertext using the LHE key $\mathbf{s}^*_{res}$.

$$\mathsf{ct}_{res} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f, \mathsf{ct}'_1, \mathsf{ct}'_2), \quad \mathbf{m}_{res} = \overline{\mathsf{lhe}}.\overline{\mathsf{Dec}}(\mathbf{s}^*_{res}, \mathsf{ct}_{res}),$$

  Output $\mathbf{L}^{out} = \mathbf{m}_{res} \in \mathbb{Z}_p^{\ell'}$.

Correctness. We show that the scheme is *correct*. Similar to the correctness arguments for Construction 4, we will show that

$$\mathbf{s}^*_{res} = y\mathbf{s}^*_1 + \mathbf{s}^*_2 \text{ (over } \mathbb{Z}),$$

and then invoke Lemma 3.2.

We have noted in the construction of $\mathsf{KE.KeyGen}$ that $\mathbf{L}^{in} = (y\mathbf{s}_1 + \mathbf{s}_2, y) \mod p$, where $y = x + r$. That is, $\mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$ parses $\mathbf{L}^{in}$ into $\mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2 \mod p$, and y as values over $[0, p-1]$. Let

$$\mathbf{s}''_{res} = y\mathbf{s}_1 + \mathbf{s}_2 = y\mathbf{s}_1 + (\mathbf{1} - \mathbf{s}_1)\lfloor p/2 \rfloor + \mathbf{r}_2 \text{ (over } \mathbb{Z}).$$

We verify the following facts about $\mathbf{s}''_{res}$ (with components $s''_{res,i}$).

- $\forall i \in [\ell^*]$, either $s_{1,i} = 1$, and $s''_{res,i} = y + r_{2,i}$, or $s_{1,i} = 0$, and $s''_{res,i} = \lfloor p/2 \rfloor + r_{2,i}$. That is,

$$\min(y, \lfloor p/2 \rfloor) + r_{2,i} \le s''_{res,i} \le \max(y, \lfloor p/2 \rfloor) + r_{2,i}.$$

- $\forall i \in [\ell^*]$, if $y \le \lfloor p/2 \rfloor$, then we have

$$s''_{res,i} \le \max(y, \lfloor p/2 \rfloor) + r_{2,i} \le \lfloor p/2 \rfloor + \lfloor (p-1)/2 \rfloor < p.$$

  That is, $s''_{res,i} = s_{res,i}$ over $\mathbb{Z}$.
- $\forall i \in [\ell^*]$, if $y > \lfloor p/2 \rfloor$, then we have

$$s''_{res,i} \ge \min(y, \lfloor p/2 \rfloor) + r_{2,i} \ge \lfloor p/2 \rfloor,$$

  and

$$s''_{res,i} \le \max(y, \lfloor p/2 \rfloor) + r_{2,i} \le (p-1) + \lfloor (p-1)/2 \rfloor < p + \lfloor p/2 \rfloor.$$

  It follows that if $s_{res,i} \ge \lfloor p/2 \rfloor$, then $s''_{res,i} = s_{res,i}$, and if $s_{res,i} < \lfloor p/2 \rfloor$, then $s''_{res,i} = s_{res,i} + p$.

The above facts about $s''_{res,i}$ matches exactly how we compute $s'_{res,i}$ from $s_{res,i}$ in $\mathsf{KE.Dec}$ (Eq. (3.13)). Therefore, we have $\mathbf{s}'_{res} = \mathbf{s}''_{res} = y\mathbf{s}_1 + \mathbf{s}_2$ (over $\mathbb{Z}$). Note that $\mathsf{Smdg.Smudge}(\cdot; \{\mathsf{sd}_i\})$ is a linear function over $\mathbb{Z}$. Hence we have

$$\begin{aligned} \mathbf{s}^*_{res} &= \mathsf{Smdg.Smudge}(\mathbf{s}'_{res}; \{\mathsf{sd}_i\}) \\ \text{(linearity)} &= y \cdot \mathsf{Smdg.Smudge}(\mathbf{s}_1; \{\mathsf{sd}_i\}) + \mathsf{Smdg.Smudge}(\mathbf{s}_2; \{\mathsf{sd}_i\}) \\ &= y\mathbf{s}^*_1 + \mathbf{s}^*_2 \text{ (over } \mathbb{Z}). \end{aligned}$$

Finally, correctness follows from invoking Lemma 3.2.

We now define and prove the weaker security satisfied by Construction 5.

Definition 3.9 (Weak Key Extension Security)**.** *Consider a key extension gadget (per Definition 3.1) w.r.t. an arithmetic garbling scheme over* $\mathbb{Z}_p$*,* $p = p(\lambda) \le 2^{\mathrm{poly}(\lambda)}$*, and with a label space* $\mathcal{L} = \mathbb{Z}_p$*.*

A pair of weak simulators $\mathsf{KE.Sim}', \mathsf{KE.Sim}''$ *for the key extension gadget are two efficient algorithm with the following syntax.*

- $\mathsf{KE.Sim}'(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y)$ *takes the same inputs as* $\mathsf{KE.Sim}$ *in Definition 3.1, and additionally a value* $y \in \mathcal{L} = \mathbb{Z}_p$*. It outputs* $\widetilde{\mathbf{L}}^{in}, \widetilde{\mathsf{tb}}$*, similarly to* $\mathsf{KE.Sim}$*.*

- $\mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y, \mathbf{z}_1^{out})$ *takes the same inputs as* $\mathsf{KE.Sim}'$*, and additionally an output-wire key* $\mathbf{z}_1^{out} \in \mathcal{L}^{\ell'}$*. It outputs* $\widetilde{\mathbf{L}}^{in}, \widetilde{\mathsf{tb}}$*, similarly to* $\mathsf{KE.Sim}$*.*

Let $\delta = \max(1, \lfloor p/5 \rfloor)$*. Define the good region* $\mathsf{GOOD} = [\delta, p - \delta] \subset \mathcal{L} = \mathbb{Z}_p$*. The key extension gadget is* weakly secure *if there exists simulators* $\mathsf{KE.Sim}', \mathsf{KE.Sim}''$ *such that for all sequences* $\{\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out}\}_\lambda$ *where* $\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out} \in \mathcal{L}^{\ell'}$*,* $\ell' \leq 2\ell$*,* $\{x_\lambda, y_\lambda, y'_\lambda\}_\lambda$ *where* $x_\lambda \in \mathcal{I}, y'_\lambda \in \mathcal{L}, y_\lambda \in \mathsf{GOOD}$*, the following indistinguishabilities hold. (For more concise notations, the index* λ *is suppressed below.)*

$$\begin{array}{r|c} \left\{\mathsf{pp}, \mathsf{KE.Sim}'(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y)\right\} & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda),\ r = y - x \mod p, \\ & \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r), \\ \approx_c \left\{\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb}\right\}, & \mathsf{tb} \leftarrow \mathsf{KE.Garb}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in}), \end{array}$$

$$\begin{array}{r|c} \left\{\mathsf{pp}, \mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y', \mathbf{z}_1^{out})\right\} & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), r' = y' - x \mod p \\ & \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r'), \\ \approx_s \left\{\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb}\right\}, & \mathsf{tb} \leftarrow \mathsf{KE.Garb}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in}), \end{array}$$

where $\mathbf{L}^{in} = \mathbf{z}_1^{in} x + \mathbf{z}_2^{in}$*,* $\mathbf{L}^{out} = \mathbf{z}_1^{out} x + \mathbf{z}_2^{out} \mod p$ *in the above.*

Lemma 3.4. *Construction 4 is* weakly secure *per Definition 3.9.*

Lemma 3.4. We construct a simulator $\mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y, \mathbf{z}_1^{out})$ that on input a security parameter λ, public parameters $\mathsf{pp} = \overline{\mathsf{lhe}}.\overline{\mathsf{pp}}, \alpha$ generated by Setup in Figure 3.5, an arbitray output-wire label $\mathbf{L}^{out} \in \mathbb{Z}_p^{2\ell}$ of dimension 2ℓ, the masked input $y \in \mathbb{Z}_p$, and one of the output-wire key $\mathbf{z}_1^{out} \in \mathbb{Z}_p^\ell$, simulates the input-wire label $\widetilde{\mathbf{L}}^{in}$ and the garbled table $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{ct}}'_1, \widetilde{\mathsf{ct}}'_2)$.

- $\mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y, \mathbf{z}_1^{out})$: Follow the honest algorithms $\mathsf{KE.KeyGen}$ and $\mathsf{KE.Garb}$ to compute the smudging source vectors $\mathbf{s}_1, \mathbf{s}_2$, the LHE keys $\mathbf{s}_1^*, \mathbf{s}_2^*$, and the first output ciphertext ct'_1 (encrypting the provided output-wire key $\mathbf{z}_1^{out}$ under the LHE key $\mathbf{s}_1^*$).

  Simulate ct'_2 via homomorphic evaluation of $\overline{\mathsf{lhe}}$, subject to the constraint that decoding must produce the correct output-wire label $\mathbf{L}^{out}$. More specifically, compute the LHE

key $\mathbf{s}_{res}^* = y\mathbf{s}_1^* + \mathbf{s}_2^*$ over $\mathbb{Z}$, and compute

$$\widetilde{\mathsf{ct}}_{res} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_{res}^*, \mathbf{L}^{out}).$$

Then, evaluate the function $f_R(x_{res}, x_1) = x_{res} - yx_1$, over $\widetilde{\mathsf{ct}}_{res}$ and ct_1' to produce $\widetilde{\mathsf{ct}}_2$

$$\widetilde{\mathsf{ct}}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f_R, \widetilde{\mathsf{ct}}_{res}, \mathsf{ct}_1').$$

Finally, smudge the noise in $\widetilde{\mathsf{ct}}_2$ to produce $\widetilde{\mathsf{ct}}_2'$ as follows

$$\mathbf{e}_2 \leftarrow [-\alpha_2, \alpha]^{2\ell}, \quad \mathsf{ct}_{e,2} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(0, \mathbf{e}_2), \quad \widetilde{\mathsf{ct}}_2' \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2, \mathsf{ct}_{e,2}).$$

Computes $\mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2 \pmod p$, and output $\widetilde{\mathbf{L}}^{in} = (\mathbf{s}_{res}, y)$, $\widetilde{\mathsf{tb}} = (\mathsf{ct}_1', \widetilde{\mathsf{ct}}_2')$.

The fact that $\mathsf{KE.Sim}''$ statistically simulates ct_2' follows from Lemma 3.1 in a similar way as argued in the proof of Lemma 3.3, Hyb_2. We omit details here, and conclude that $\mathsf{KE.Sim}''$ described above is secure.

We next construct a simulator $\mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y)$, that on input a security parameter λ, public parameters $\mathsf{pp} = (\overline{\mathsf{lhe}}.\overline{\mathsf{pp}}, \alpha)$ generated by Setup in Figure 3.5, an arbitray output-wire label $\mathbf{L}^{out} \in \mathbb{Z}_p^{2\ell}$ of dimension 2ℓ, and the masked input $y \in \mathbb{Z}_p$, simulates the input-wire label $\widetilde{\mathbf{L}}^{in}$ and the garbled table $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{ct}}_1', \widetilde{\mathsf{ct}}_2')$.

- $\mathsf{KE.Sim}'(1^\lambda, \mathsf{pp}, \mathbf{L}^{out}, y)$: Follow the honest algorithms $\mathsf{KE.KeyGen}$ and $\mathsf{KE.Garb}$ to sample the smudging source vectors $\mathbf{s}_1 \leftarrow \{0,1\}^{\ell^*}$, and $\mathbf{s}_2$ as

$$\mathbf{r}_2 \leftarrow [0, \lfloor (p-1)/2 \rfloor]^{\ell^*}, \quad \mathbf{s}_2 = (\mathbf{1} - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{r}_2 \mod p.$$

Sample smudging seeds $\mathsf{sd}_i \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{smdg}}, 1/4)$ for $i \in [\ell_s]$, and compute the two LHE keys $\mathbf{s}_1^*, \mathbf{s}_2^* \in \mathbb{Z}^{\ell_s}$ from the source vectors.

$$\mathbf{s}_1^* = \mathsf{Smdg.Smudge}(\mathbf{s}_1; \{\mathsf{sd}_i\}), \quad \mathbf{s}_2^* = \mathsf{Smdg.Smudge}(\mathbf{s}_2; \{\mathsf{sd}_i\}).$$

Simulate the ciphertext $\widetilde{\mathsf{ct}}_1$ as the sum of two encrpytions of $\mathbf{0} \in \mathbb{Z}^{2\ell}$, under a fresh LHE key $\mathbf{s} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s})$ and the key $\mathbf{s}_1^*$ respectively.

$$\mathsf{ct}_s \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}, \mathbf{0}), \quad \mathsf{ct}_0 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1^*, \mathbf{0}), \quad \widetilde{\mathsf{ct}}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_s, \mathsf{ct}_0).$$

Then smudge the noise in $\widetilde{\mathsf{ct}}_1$ with a noise vector $\mathbf{e}_1 \leftarrow [-\alpha_1, \alpha_1]^{2\ell}$ to produce $\widetilde{\mathsf{ct}}_1'$ as follows:

$$\mathsf{ct}_{e_1} \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}_1), \quad \widetilde{\mathsf{ct}}_1' \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_1, \mathsf{ct}_{e_1}).$$

Next, compute the vector $\mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2 \mod p$ and follow the honest decryption algorithm $\mathsf{KE.Dec}$ to compute s^*_{res}. Then simulate $\widetilde{\mathsf{ct}}_2'$ in the same way as described in $\mathsf{KE.Sim}''$ above:

$$\widetilde{\mathsf{ct}}_{res} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}^*_{res}, \mathbf{L}^{out}), \quad \widetilde{\mathsf{ct}}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f_R, \widetilde{\mathsf{ct}}_{res}, \mathsf{ct}_1'),$$

where $f_R(x_{res}, x_1) = x_{res} - yx_1$. Finally, smudge the noise in $\widetilde{\mathsf{ct}}_2$ to produce $\widetilde{\mathsf{ct}}_2'$ as follows

$$\mathbf{e}_2 \leftarrow [-\alpha_2, \alpha_2]^{2\ell}, \quad \mathsf{ct}_{e,2} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(0, \mathbf{e}_2), \quad \widetilde{\mathsf{ct}}_2' \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2, \mathsf{ct}_{e,2}).$$

Output $\widetilde{\mathbf{L}}^{in} = (\mathbf{s}_{res}, y)$ and $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{ct}}_1', \widetilde{\mathsf{ct}}_2', \{\mathsf{sd}_i\})$.

We now argue that $\mathsf{KE.Sim}'$ described above satisfies the security requirement. In particular, let $\ell' = \ell'(\lambda)$ be any polynomial, consider any sequences $\{\mathbf{z}^{out}_{1,\lambda}, \mathbf{z}^{out}_{2,\lambda}\}_\lambda$ where $\mathbf{z}^{out}_{1,\lambda}, \mathbf{z}^{out}_{2,\lambda} \in \mathcal{K}^{\ell'}$, and $\{x_\lambda, y_\lambda\}_\lambda$ where $x_\lambda \in \mathcal{I}, y_\lambda \in \mathsf{GOOD}$. We define five hybrids, $\mathsf{Hyb}_1, \ldots, \mathsf{Hyb}_5$, where the first hybrid is exactly the real-world distribution in Definition 3.9, and the last hybrid is exactly the simulated distribution using $\mathsf{KE.Sim}'$. (In the following, we surpress the subscript λ.)

- Hyb_1: compute $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, and compute $\mathbf{L}^{in}$, $\mathsf{tb} = (\mathsf{ct}_1', \mathsf{ct}_2', \{\mathsf{sd}_i\})$ as described in $\mathsf{KE.KeyGen}$, $\mathsf{KE.Garb}$:

$$\mathbf{s}_1 \leftarrow \{0,1\}^{\ell^*}, \quad \mathbf{r}_2 \leftarrow [0, \lfloor (p-1)/2 \rfloor]^{\ell^*}, \mathbf{s}_2 = (\mathbf{1} - \mathbf{s}_1) \cdot \lfloor p/2 \rfloor + \mathbf{r}_2 \mod p,$$

$$i \in [\ell_s], \quad \mathsf{sd}_i \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{\text{smdg}}}, 1/4),$$
$$\mathbf{s}^*_1 = \mathsf{Smdg.Smudge}(\mathbf{s}_1; \{\mathsf{sd}_i\}_i), \quad \mathbf{s}^*_2 = \mathsf{Smdg.Smudge}(\mathbf{s}_2; \{\mathsf{sd}_i\}_i),$$

$$\mathbf{z'}_2^{out} = \mathbf{z}_2^{out} - r\mathbf{z}_2^{out} \mod p, \quad \mathsf{ct}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1^*, \mathbf{z}_1^{out}), \quad \mathsf{ct}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_2^*, \mathbf{z'}_1^{out}),$$

$$i = 1, 2 \quad \mathbf{e}_i \leftarrow [-\alpha_i, \alpha_i]^{\ell'}, \quad \mathsf{ct}_{e,i} \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}_i), \quad \mathsf{ct}'_i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_i, \mathsf{ct}_{e,i}),$$

Let $y = x + r$, $\mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2 \mod p$, and set $\mathbf{L}^{in} = (\mathbf{s}_{res}, y)$.

- Hyb_2: We use $\mathbf{L}^{out} = x\mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p$ to simulate $\widetilde{\mathsf{ct}}'_2$, while keeping everything else unchanged. We first compute $\mathbf{s}_{res}^*$ from $\mathbf{s}_{res}, y$ in the same way as described in KE.Dec. We next compute

$$\widetilde{\mathsf{ct}}_{res} = \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_{res}^*, \mathbf{L}^{out}), \quad \widetilde{\mathsf{ct}}_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f_R, \widetilde{\mathsf{ct}}_{res}, \mathsf{ct}'_1),$$

where $f_R(x_1, x_2) = x_1 - yx_2$. Finally we add a smudging noise to $\widetilde{\mathsf{ct}}_2$ to produce $\widetilde{\mathsf{ct}}'_2$ as follows:

$$\mathbf{e}_2 \leftarrow [-\alpha_2, \alpha_2]^{2\ell}, \quad \mathsf{ct}_{e,2} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(0, \mathbf{e}_2), \quad \widetilde{\mathsf{ct}}'_2 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2, \mathsf{ct}_{e,2}).$$

We have shown in *correctness* that $\mathbf{s}_{res}^* = y\mathbf{s}_1^* + \mathbf{s}_2^*$ over $\mathbb{Z}$. Hence by Lemma 3.1, this hybrid is statistically close to the previous one.

- Hyb_3: We use $\mathbf{s}_1^*$ to smudge a fresh LHE key $\mathbf{s}$, while keeping everything else unchanged. We compute

$$\mathbf{s} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s}), \quad \widetilde{\mathbf{s}}_1^* = \mathbf{s}_1^* + \mathbf{s} \text{ (over } \mathbb{Z}).$$

We first argue that given $\mathbf{s}_{res} = y\mathbf{s}_1 + \mathbf{s}_2 \mod p$, with overwhelming probability, at least $1/4$ of the components of $\mathbf{s}_1$ remains hidden.

Claim 3.2. *Let $\mathbf{s}_1, \mathbf{s}_{res}$ be computed as described in Hyb_1. Let $s_{1,i}, s_{res,i}$ be their components. If $y \le \lfloor p/2 \rfloor$, then for all $v \in [\lfloor p/2 \rfloor, y + \lfloor (p-1)/2 \rfloor]$, we have*

$$\forall i \in [\ell^*], \quad \Pr[s_{1,i} = 1 \mid s_{res,i} = v] = \Pr[s_{1,i} = 0 \mid s_{res,i} = v] = 1/2.$$

If $y > \lfloor p/2 \rfloor$, then for all $v \in [y, p-1]$, we similarly have

$$\forall i \in [\ell^*], \quad \Pr[s_{1,i} = 1 \mid s_{res,i} = v] = \Pr[s_{1,i} = 0 \mid s_{res,i} = v] = 1/2.$$

Claim 3.3. *Let* $\mathbf{s}_1, \mathbf{s}_{res}$ *be computed as described in* Hyb_1*. Then with overwhelming probability, at least* $1/4$ *components of* $\mathbf{s}_{res}$ *have values satisfying the condition in Claim 3.2.*

Now, we can invoke the smudging property of $\mathsf{Smdg.Smudge}$ to argue that

$$\{\widetilde{\mathbf{s}}_1^*, \mathbf{s}_{res}, \{\mathsf{sd}_i\}\} \approx_s \{\mathbf{s}_1^*, \mathbf{s}_{res}, \{\mathsf{sd}_i\}\}.$$

Hence this hybrid is statistically close to the previous one.

- Hyb_4: We simulate $\widetilde{\mathsf{ct}}_1'$, while keeping everything else unchanged. We compute

$$\mathsf{ct}_s \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}, \mathbf{z}_1^{out}), \quad \mathsf{ct}_0 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1^*, \mathbf{0}),$$
$$\widetilde{\mathsf{ct}}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_s, \mathsf{ct}_0).$$

Then smudge the noise in $\widetilde{\mathsf{ct}}_1$ to produce $\widetilde{\mathsf{ct}}_1'$ as follows.

$$\mathbf{e}_1 \leftarrow [-\alpha_1, \alpha_1]^{2\ell}, \quad \mathsf{ct}_{e,1} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(0, \mathbf{e}_1), \quad \widetilde{\mathsf{ct}}_1' \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_1, \mathsf{ct}_{e,1}).$$

By Lemma 3.1, this hybrid is statistically close to the previous one.

- Hyb_5: We simulate $\widetilde{\mathsf{ct}}_s$ as an encryption of $\mathbf{0}$ (instead of $\mathbf{z}_1^{out}$), while keeping everything else unchanged. We compute

$$\widetilde{\mathsf{ct}}_s \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}, \mathbf{0}).$$

Because $\mathbf{s}$ is a fresh LHE key, not used for computing anything else, by the security of $\overline{\mathsf{lhe}}$, this hybrid is computationally indistinguishable from the previous one. □

We now construct a fully secure key extension gadget using Construction 5.

Construction 6 (length-doubling key extension for modular arithmetic)**.** This construction uses the weakly secure key extension scheme KE in Construction 5 as an ingredient.

- $\overline{\mathsf{KE}}.\overline{\mathsf{KeyGen}}^{\mathsf{pp}}(1^\lambda, 1^\ell)$: Sample two correlated random values $r \leftarrow \mathbb{Z}_p$, $r' = r + 2\delta \mod p$, where $\delta = \max(1, \lfloor p/5 \rfloor)$. Then run two instances of the weakly secure scheme, using r, r' respectively (using the convenient syntax defined in Construction 5)

$$\mathbf{z}_{1,1}^{in}, \mathbf{z}_{2,1}^{in} \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r), \quad \mathbf{z}_{1,2}^{in}, \mathbf{z}_{2,2}^{in} \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r').$$

  Concatenate the outputs from the weak schemes, and output them as $\mathbf{z}_1^{in} = (\mathbf{z}_{1,1}^{in}, \mathbf{z}_{1,2}^{in})$, and $\mathbf{z}_2^{in} = (\mathbf{z}_{2,1}^{in}, \mathbf{z}_{2,2}^{in})$.
- $\overline{\mathsf{KE}}.\overline{\mathsf{Garb}}^{\mathsf{pp}}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, \mathbf{z}_1^{in}, \mathbf{z}_2^{in})$: Parse the input-wire keys $\mathbf{z}_1^{in}, \mathbf{z}_2^{in}$ as $\mathbf{z}_1^{in} = (\mathbf{z}_{1,1}^{in}, \mathbf{z}_{1,2}^{in})$, $\mathbf{z}_2^{in} = (\mathbf{z}_{2,1}^{in}, \mathbf{z}_{2,2}^{in})$. Additively share the output-wire keys $\mathbf{z}_1^{in}, \mathbf{z}_2^{in}$ as $\mathbf{z}_{1,1}^{out}, \mathbf{r}_{2,1}^{out} \leftarrow \mathbb{Z}_p^{\ell'}$ and $\mathbf{z}_{1,2}^{out} = \mathbf{z}_1^{out} - \mathbf{z}_{1,1}^{out}$, $\mathbf{z}_{2,2}^{out} = \mathbf{z}_2^{out} - \mathbf{z}_{2,1}^{out} \pmod p$. Then run two instances of the weakly secure scheme using the two shares respectively.

$$\mathsf{tb}_1 \leftarrow \mathsf{KE.Garb}^{\mathsf{pp}}(\mathbf{z}_{1,1}^{out}, \mathbf{z}_{2,1}^{out}, \mathbf{z}_{1,1}^{in}, \mathbf{z}_{2,1}^{in}),$$
$$\mathsf{tb}_2 \leftarrow \mathsf{KE.Garb}^{\mathsf{pp}}(\mathbf{z}_{1,2}^{out}, \mathbf{z}_{2,2}^{out}, \mathbf{z}_{1,2}^{in}, \mathbf{z}_{2,2}^{in}),$$

  Output the garbled table $\mathsf{tb} = (\mathsf{tb}_1, \mathsf{tb}_2)$.
- $\overline{\mathsf{KE}}.\overline{\mathsf{Dec}}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$: Parse the input-wire label $\mathbf{L}^{in}$ and the garbled table tb as a concatenation of two (weak) instances: $\mathbf{L}^{in} = (\mathbf{L}_1^{in}, \mathbf{L}_2^{in})$, $\mathsf{tb} = (\mathsf{tb}_1, \mathsf{tb}_2)$. Run the decoding algorithm from the weak scheme on each of the two instances to recover two output-wire labels $\mathbf{L}_1^{out}, \mathbf{L}_2^{out}$, and output their sum $\mathbf{L}^{out} = \mathbf{L}_1^{out} + \mathbf{L}_2^{out} \mod p$.

$$\mathbf{L}_1^{out} = \mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}_1^{in}, \mathsf{tb}_1), \quad \mathbf{L}_2^{out} = \mathsf{KE.Dec}^{\mathsf{pp}}(\mathbf{L}_2^{in}, \mathsf{tb}_2).$$

Note: *The* correctness *of this construction directly follows from that of* KE*.*

Lemma 3.5. *Construction 6 is* secure *per Definition 3.1.*

Lemma 3.5. We construct a simulator $\overline{\mathsf{KE}}.\overline{\mathsf{Sim}}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$ whose goal is to simulate $\widetilde{\mathbf{L}}^{in}$ and $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{tb}}_1, \widetilde{\mathsf{tb}}_2)$. It first samples $y \leftarrow \mathbb{Z}_p$, and computes $y' = y + 2\delta \mod p$. The following claim shows that at least one of y, y' is in the GOOD region, as defined in Definition 3.9. Without loss of generality, assume $y \in \mathsf{GOOD}$.

Claim 3.4. *For all integer $p > 2$, let $\delta = \max(1, \lfloor p/5 \rfloor)$, and $\mathsf{GOOD} = [\delta, p-\delta] \subset \mathbb{Z}_p$. For all $y \in \mathbb{Z}_p$, $y' = y + 2\delta \mod p$, at least one of y, y' is in GOOD.*

The simulator samples

$$\mathbf{z}_{1,2}^{out} \leftarrow \mathbb{Z}_p^{\ell'}, \quad \mathbf{L}_2^{out} \leftarrow \mathbb{Z}_p^{\ell'},$$

and computes $\mathbf{L}_1^{out} = \mathbf{L}^{out} - \mathbf{L}_2^{out} \mod p$. It runs

$$\widetilde{\mathbf{L}}_1^{in}, \widetilde{\mathsf{tb}}_1 \leftarrow \mathsf{KE.Sim}'(1^\lambda, \mathsf{pp}, \mathbf{L}_1^{out}, y),$$
$$\widetilde{\mathbf{L}}_2^{in}, \widetilde{\mathsf{tb}}_2 \leftarrow \mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}_2^{out}, y', \mathbf{z}_{1,2}^{out}),$$

where $\mathsf{KE.Sim}'$, $\mathsf{KE.Sim}''$ are two weak simulators guaranteed by the security of KE. Finally, it outputs $\widetilde{\mathbf{L}}^{in} = (\widetilde{\mathbf{L}}_1^{in}, \widetilde{\mathbf{L}}_2^{in})$, $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{tb}}_1, \widetilde{\mathsf{tb}}_2)$.

We now argue that $\mathsf{KE.Sim}$ described above satisfies the security requirement. In particular, let $\ell' = \ell'(\lambda)$ be any polynomial, consider any sequences $\{\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out}\}_\lambda$ where $\mathbf{z}_{1,\lambda}^{out}, \mathbf{z}_{2,\lambda}^{out} \in \mathcal{L}^{\ell'}$, and $\{x_\lambda\}_\lambda$ where $x_\lambda \in \mathcal{I}$. We define four hybrids, $\mathsf{Hyb}_1, \ldots, \mathsf{Hyb}_4$, where the first hybrid is exactly the real-world distribution in Definition 3.1, and the last hybrid is exactly the simulated distribution using $\mathsf{KE.Sim}$. (In the following, we surpress the subscript λ.)

- Hyb_1: We first compute $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, and compute $\mathbf{z}_{1,1}^{in}, \mathbf{z}_{2,1}^{in}$, $\mathbf{z}_{1,2}^{in}, \mathbf{z}_{2,2}^{in}$, and $\mathsf{tb}_1, \mathsf{tb}_2$ as described in $\overline{\mathsf{KE}}.\overline{\mathsf{KeyGen}}$, $\overline{\mathsf{KE}}.\overline{\mathsf{Garb}}$:

$$r \leftarrow \mathbb{Z}_p, \quad r' = r + 2\delta,$$

$$\mathbf{z}_{1,1}^{in}, \mathbf{z}_{2,1}^{in} \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r), \quad \mathbf{z}_{1,2}^{in}, \mathbf{z}_{2,2}^{in} \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell; r'),$$

$$\mathbf{z}_{1,2}^{out}, \mathbf{z}_{2,2}^{out} \leftarrow \mathbb{Z}_p^{\ell'}, \quad \mathbf{z}_{1,1}^{out} = \mathbf{z}_1^{out} - \mathbf{z}_{1,2}^{out}, \quad \mathbf{z}_{2,1}^{out} = \mathbf{z}_2^{out} - \mathbf{z}_{2,2}^{out} \mod p,$$

$$i = 1, 2 \quad \mathsf{tb}_i \leftarrow \mathsf{KE.Garb}^{\mathsf{pp}}(\mathbf{z}_{1,i}^{out}, \mathbf{z}_{2,i}^{out}, \mathbf{z}_{1,i}^{in}, \mathbf{z}_{2,i}^{in}).$$

  Let $\mathbf{L}_i^{in} = x\mathbf{z}_{1,i}^{in} + \mathbf{z}_{2,i}^{in} \mod p$ for $i = 1, 2$. The distribution is defined as

$$\mathsf{Hyb}_1 = \{\mathsf{pp}, \mathbf{L}^{in} = (\mathbf{L}_1^{in}, \mathbf{L}_2^{in}), \mathsf{tb} = (\mathsf{tb}_1, \mathsf{tb}_2)\}.$$

- Hyb_2: Let $y = x + r$, $y' = x + r' \mod p$. By Claim 3.4, at least one of y, y' is in GOOD. Without loss of generality, assume $y \in \mathsf{GOOD}$. We use $\mathsf{KE.Sim}''$ to simulate $\widetilde{\mathbf{L}}_2^{in}, \widetilde{\mathsf{tb}}_2$, while keeping everything else unchanged. Let $\mathbf{L}_2^{out} = x\mathbf{z}_{1,2}^{out} + \mathbf{z}_{2,2}^{out} \mod p$. We run

$$\widetilde{\mathbf{L}}_2^{in}, \widetilde{\mathsf{tb}}_2 \leftarrow \mathsf{KE.Sim}''(1^\lambda, \mathsf{pp}, \mathbf{L}_2^{out}, y', \mathbf{z}_{1,2}^{out}).$$

  By the security of $\mathsf{KE.Sim}''$, this hybrid is statistically close to the previous one.

  Claim 3.5. $\mathsf{Hyb}_2 \approx_s \mathsf{Hyb}_1$.

- Hyb_3: We use $\mathsf{KE.Sim}'$ to simulate $\widetilde{\mathbf{L}}_1^{in}, \widetilde{\mathsf{tb}}_1$, while keeping everything else unchanged. Let $\mathbf{L}_1^{out} = \mathbf{L}^{out} - \mathbf{L}_2^{out} \mod p$. We run

$$\widetilde{\mathbf{L}}_1^{in}, \widetilde{\mathsf{tb}}_1 \leftarrow \mathsf{KE.Sim}'(1^\lambda, \mathsf{pp}, \mathbf{L}_1^{out}, y).$$

  Since $y \in \mathsf{GOOD}$, by the security of $\mathsf{KE.Sim}'$, this hybrid is computationally indistinguishable from the previous one.

  Claim 3.6. $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_2$.

- Hyb_4: We simulate y, y' and $\mathbf{L}_2^{out}$ as

$$y \leftarrow \mathbb{Z}_p, \quad y' = y + 2\delta \mod p, \quad \mathbf{L}_2^{out} \leftarrow \mathbb{Z}_p^{\ell'}.$$

  By the randomness of r and $\mathbf{z}_{2,2}^{out}$, this hybrid is identical to the previous one.

  Claim 3.7. $\mathsf{Hyb}_4 \equiv \mathsf{Hyb}_3$. □

3.8 Construction of Bit Decomposition Gadget for Mixed Computation

In this section we construct the bit decomposition gadget for mixed bounded integer and Boolean computation.

3.8.1 The Setup Algorithm

Similarly to Section 3.5, we first describe Setup (in Figure 3.6) of the garbling scheme, which is shared by our gadget constructions in the mixed bounded integer and Boolean computation model. The label space of the garbling scheme is $\mathcal{L} = \mathbb{Z}_p$.

3.8.2 Bit Decomposition

Our observation for constructing the bit decomposition gadget is that it's enough to construct a gadget for truncation (by powers-of-2). Such a gadget has the following simplified syntax:

$$\begin{array}{r|l} \mathsf{TC.Garble}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, i) \to \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \mathsf{tb} & \mathbf{L}_x^{in} = x\mathbf{z}_1^{in} + \mathbf{z}_2^{in}, \\ \mathsf{TC.Dec}(\mathbf{L}_x^{in}, \mathsf{tb}) \to \mathbf{L}_{\lfloor x \rfloor_{2^i}}^{out}. & \mathbf{L}_{\lfloor x \rfloor_{2^i}}^{out} = \lfloor x \rfloor_{2^i} \mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p. \end{array}$$

It says that given an input label $\mathbf{L}_x^{in}$ and the garbled table tb, an evaluator can recover the output label $\mathbf{L}_{\lfloor x \rfloor_{2^i}}^{out}$. Note that for any non-negative integer x, we have $\mathsf{bits}(x)_i = \lfloor x \rfloor_{2^{i-1}} - 2\lfloor x \rfloor_{2^i}$. (For simplicity, we only consider non-negative integer input x in this overview.) Therefore, if we want to obtain an output label $\mathbf{L}_{\mathsf{bits}(x)_i}^{out} = \mathsf{bits}(x)_i \mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p$, it's enough to obtain

$$\begin{array}{l} \mathbf{u} = \lfloor x \rfloor_{2^i} \mathbf{z}_1^{out} + \mathbf{r} \\ \mathbf{v} = \lfloor x \rfloor_{2^{i-1}} \mathbf{z}_1^{out} + 2\mathbf{r} + \mathbf{z}_2^{out} \end{array} \implies \mathbf{L}_{\mathsf{bits}(x)_i}^{out} = \mathbf{v} - 2\mathbf{u} \mod p,$$

where $\mathbf{r}$ can be just a random vector over $\mathbb{Z}_p$. Now, to obtain the labels $\mathbf{u}, \mathbf{v}$ in the above, we just run $\mathsf{TC.Garble}(\mathbf{z}_1^{out}, \mathbf{r}, i)$ and $\mathsf{TC.Garble}(\mathbf{z}_1^{out}, 2\mathbf{r} + \mathbf{z}_2^{out}, i-1)$. Repeating the above for each bit position i gives a bit decomposition gadget.

However, we can only construct a truncation scheme TC' with a weaker correctness and security guarantee. $\mathsf{TC}'.\mathsf{Garb}$ takes additionally an argument r, such that $\mathsf{TC}'.\mathsf{Dec}$ recovers $\mathbf{L}_{\lfloor y \rfloor_{2^i}}^{out}$, where $y = x + r$ over $\mathbb{Z}$.

$$\begin{array}{r|l} \mathsf{TC}'.\mathsf{Garb}(\mathbf{z}_1^{out}, \mathbf{z}_2^{out}, i, r) \to \mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \mathsf{tb} & \mathbf{L}_x^{in} = x\mathbf{z}_1^{in} + \mathbf{z}_2^{in}, \\ \mathsf{TC}'.\mathsf{Dec}(\mathbf{L}_x^{in}, \mathsf{tb}) \to \mathbf{L}_{\lfloor y \rfloor_{2^i}}^{out}. & \mathbf{L}_{\lfloor y \rfloor_{2^i}}^{out} = \lfloor y \rfloor_{2^i} \mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p. \end{array}$$

Security only holds if the additional argument r is set to be a secret random integer that can statistically smudge the input x.

With this imperfect truncation scheme TC', the earlier observation only allows us to obtain $\mathbf{L}_{\mathsf{bits}(y)_i}^{out} = \mathsf{bits}(y)_i \mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p$, where $y = x + r$ over $\mathbb{Z}$, for some secret random non-negative integer r. To construct a true bit decomposition scheme that "removes" the

random value r without hurting security, our idea is to use the labels $\mathbf{L}^{out}_{\mathsf{bits}(y)_i}$ to encode evaluation keys of a Yao's garbled (Boolean) circuit $\widehat{C}^r_{\text{sub}}$, whose input is exactly $\mathsf{bits}(y)$, and has r hard-coded within. To achieve this, we set

$$\begin{aligned} \bar{\mathbf{z}}^i_1 &= \bar{\mathbf{k}}^i_1 - \bar{\mathbf{k}}^i_0 \mod p \\ \bar{\mathbf{z}}^i_2 &= \bar{\mathbf{k}}^i_0 \end{aligned} \implies \begin{aligned} \bar{\mathbf{L}}^i_{\mathsf{bits}(y)_i} &= \mathsf{bits}(y)_i \bar{\mathbf{z}}^i_1 + \bar{\mathbf{z}}^i_2 \\ &= \bar{\mathbf{k}}^i_{\mathsf{bits}(y)_i} \mod p, \end{aligned} \tag{3.16}$$

where $\bar{\mathbf{k}}^i_0, \bar{\mathbf{k}}^i_1$ encodes (as $\mathbb{Z}_p$ vectors) the binary evaluation keys $\mathbf{k}^i_0, \mathbf{k}^i_1$ of a Yao's garbled circuit.

Now, when an evaluator obtains $\{\bar{\mathbf{L}}^i_{\mathsf{bits}(y)_i}\}_i$, it can further use them to evaluate the garbled circuit $\widehat{C}^r_{\text{sub}}$, which we define $C^r_{\text{sub}}(\mathsf{bits}(y))$ to output the desired labels $\{\mathbf{L}^i_{\mathsf{bits}(x)_i}\}_i$. This gives us a correct bit decomposition scheme. By the security of Yao's garbled circuit, $\widehat{C}^r_{\text{sub}}$ hides the random value r, which allows us to prove security.

Below, we follow the above outline to first construct an imperfect truncation scheme TC', with a more convenient "batch" syntax.

Constructing the Imperfect Truncation Scheme. The algorithms below have access to the public parameter pp that Setup algorithm (Figure 3.6) generates, which contains the public parameter $\overline{\mathsf{pp}}$ of the LHE scheme $\overline{\mathsf{lhe}}$ and the dimension ℓ of keys of the input wire $(\mathbf{z}^{in}_1, \mathbf{z}^{in}_2)$. The dimension of output-wire keys $(\{\mathbf{z}^i_1, \mathbf{z}^i_2\})$ is $2\ell_k$, where ℓ_k is the length of an (Boolean) evaluation key in the Boolean garbling scheme Yao.

- $\mathsf{TC}'.\mathsf{Garb}^{\mathsf{pp}}(1^\lambda, \{\mathbf{z}^i_1, \mathbf{z}^i_2\}_{i\in[d]}, r)$: Takes as input a security parameter λ, d output-wire key pairs $\{\mathbf{z}^i_1, \mathbf{z}^i_2\}_{i\in[d]}$ where $\mathbf{z}^i_1, \mathbf{z}^i_1 \in \mathbb{Z}^{2\ell_k}_p$, and a integer r in the range of $[0, 2B_{\text{smdg}}]$, where $B_{\text{smdg}} = \lambda^{\omega(1)}B$. It outputs an input-wire key pair $\mathbf{z}^{in}_1, \mathbf{z}^{in}_2 \in \mathbb{Z}^\ell_p$, and a garbled table tb.
- $\mathsf{TC}'.\mathsf{Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$: Takes as input an input-wire label $\mathbf{L}^{in} \in \mathbb{Z}^\ell_p$ and a garbled table tb. It outputs the d corresponding output-wire labels $\{\mathbf{L}^i\}_{i\in[d]}$, where $\mathbf{L}^i \in \mathbb{Z}^{2\ell_k}_p$.

The correctness of the scheme TC' is described below.

$$\begin{array}{r|l} \mathsf{TC}'.\mathsf{Garb}(1^\lambda, \{\mathbf{z}^i_1, \mathbf{z}^i_2\}_{i\in[d]}, r) \to \mathbf{z}^{in}_1, \mathbf{z}^{in}_2, \mathsf{tb} & \mathbf{L}^{in}_x = x\mathbf{z}^{in}_1 + \mathbf{z}^{in}_2, \\ \mathsf{TC}'.\mathsf{Dec}(\mathbf{L}^{in}_x, \mathsf{tb}) \to \{\mathbf{L}^i\}_{i\in[d]}, & \mathbf{L}^i = \lfloor y \rfloor_{2^{i-1}} \mathbf{z}^i_1 + \mathbf{z}^i_2 \mod p. \end{array} \tag{3.17}$$

where $y = r + x$ over $\mathbb{Z}$.

Construction 7 (imperfect truncation)**.** The algorithm uses a linear seeded $(\ell_{\text{smdg}}, 1/4, \lambda_1, \lambda_2)$-smudger scheme Smudge (Theorem 3.4), used for smudging LHE keys generated by the $\overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}$ algorithm, which has infinity norm bounded by B_s. To this end, we set the (log) smudging range $\lambda_1 = \log B_s$, the (log) smudging distance $\lambda_2 = \omega(\log \lambda)$, and the smudging source dimension $\ell_{\text{smdg}} = O(\lambda_1 + \lambda_2)$.

- $\mathsf{TC'Garb^{pp}}(1^\lambda, \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]}, r)$: Proceeds in two steps.

    1. Prepare d pairs of LHE keys $\{\mathbf{s}_1^{i,*}, \mathbf{s}_2^{i,*}\}_{i\in[d]}$. For $i = 1$, sample $\mathbf{s}_1^{1,*} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s})$, $\mathbf{s}_2^{1,*} \leftarrow [-B'_{\text{smdg}}, B'_{\text{smdg}}]^{\ell_s}$, where $B'_{\text{smdg}} = \lambda^{\omega(1)} B_{\text{smdg}} < p/4$. (The inequality is satisfied because the message modulus p is set sufficiently large; see Eq. (3.14)). Define the input keys $\mathbf{z}_1^{in}, \mathbf{z}_2^{in}$ as

    $$\mathbf{z}_1^{in} = (\mathbf{s}_1^{1,*}, 1), \quad \mathbf{z}_2^{in} = (r\mathbf{s}_1^{1,*} + \mathbf{s}_2^{1,*}, r), \text{ (over } \mathbb{Z}).$$

    For $i = 2, \ldots, d$, compute two source vectors $\mathbf{s}_1^i \leftarrow \{0,1\}^{\ell_s \ell_{\text{smdg}}}$, and $\mathbf{s}_2^i$ as

    $$\begin{aligned} \mathbf{r}_2^i \leftarrow \{0,1\}^{\ell_s \ell_{\text{smdg}}}, \quad & \mathbf{s}_{2,1}^i \leftarrow [-B'_{\text{smdg}}, B'_{\text{smdg}}]^{\ell_s \ell_{\text{smdg}}}, \\ \mathbf{s}_{2,2}^i = \mathbf{1} - \mathbf{s}_1^i + \mathbf{s}_1^i \otimes \mathbf{r}_2^i, \quad & \mathbf{s}_2^i = 2\mathbf{s}_{2,1}^i + \mathbf{s}_{2,2}^i \text{ (over } \mathbb{Z}), \end{aligned} \tag{3.18}$$

    where $\otimes$ means coordinate-wise multiplication, and sample smudging seeds $\mathsf{sd}_j^i \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{\text{smdg}}}, 1/4)$ for $j \in [\ell_s]$. Next, compute LHE keys $\mathbf{s}_1^{i,*}, \mathbf{s}_2^{i,*}$ as (using the short hand in Eq. (3.10))

    $$\mathbf{s}_1^{i,*} \leftarrow \mathsf{Smdg.Smudge}(\mathbf{s}_1^i; \{\mathsf{sd}_j^i\}_j), \quad \mathbf{s}_2^{i,*} \leftarrow \mathsf{Smdg.Smudge}(\lfloor \mathbf{s}_2^i \rfloor_2; \{\mathsf{sd}_j^i\}_j).$$

    2. Compute LHE ciphertexts $\mathsf{ct}_1'^{,i}, \mathsf{ct}_2'^{,i}$ encrypting the vectors $\mathbf{z}_1^i, \mathbf{z}_2^i$ under LHE keys $\mathbf{s}_1^{1,*}, \mathbf{s}_2^{1,*}$, for $i \in [d]$. First, define noise smudging magnitudes $\alpha_1 = \lambda^{\omega(1)} \max(p, B_e)^2$, $\alpha_2 = \lambda^{\omega(1)} \alpha_1^2$ such that $\alpha_2 = \alpha$ as set in Eq. (3.15), and compute

    $$j = 1, 2 \quad \mathsf{ct}_j^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_j^{i,*}, (\mathbf{z}_j^i, \mathbf{s}_j^{i+1})).$$

Deal with the edge case of $i = d$ by setting dummy vectors $\mathbf{s}_1^{d+1} = \mathbf{s}_2^{d+1} = \mathbf{0}$. Next, add a smudging noise of magnitude α_j to ct_j^i to obtain $\mathsf{ct}_j^{\prime,i}$, and define the garbled table $\mathsf{tb} = (\{\mathsf{ct}_1^{\prime,i}, \mathsf{ct}_2^{\prime,i}\}, \{\mathsf{sd}_j^i\})$. Output $\mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \mathsf{tb}$.

$$\mathbf{e}_j^i \leftarrow [-\alpha_j, \alpha_j]^{2\ell_k + \ell_s \ell_{\mathrm{smdg}}}, \quad \mathsf{ct}_{e,j}^i \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}_j), \quad \mathsf{ct}_j^{\prime,i} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_j^i, \mathsf{ct}_{e,j}^i).$$

- $\mathsf{TC}'.\mathsf{Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$: Treat the input-wire label $\mathbf{L}^{in}$ as an integer vector and parse it as $\mathbf{L}^{in} = (\mathbf{s}_{res}^{1,*}, y)$, where $\mathbf{s}_{res}^{1,*} \in \mathbb{Z}^{\ell_s}$, $y \in \mathbb{Z}$. Parse the garbled table tb as $\mathsf{tb} = (\{\mathsf{ct}_1^{\prime,i}, \mathsf{ct}_2^{\prime,i}\}_{i\in[d]}, \{\mathsf{sd}_j^i\}_{j\in[\ell_s]}^{i\in[d]})$. Repeat the following for $i = 1, \ldots, d$: (We use $y_i = \lfloor y \rfloor_{2^i}$ as short hand in the following.)

  1. Homomorphically evaluate the linear function $f(x_1, x_2) = y_{i-1}x_1 + x_2$ over $\mathsf{ct}_1^{\prime,i}, \mathsf{ct}_2^{\prime,i}$, decrypt the output ciphertext to obtain $\mathbf{m}_{res}^i$.

  $$\mathsf{ct}_{res}^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f, \mathsf{ct}_1^{\prime i}, \mathsf{ct}_2^{\prime i}), \quad \mathbf{m}_{res}^i = \overline{\mathsf{lhe}}.\overline{\mathsf{Dec}}(\mathbf{s}_{res}^{i,*}, \mathsf{ct}_{res}^i).$$

  Parse $\mathbf{m}_{res}^i$ as $\mathbf{m}_{res}^i = (\mathbf{L}^i, \mathbf{s}_{res}^{i+1})$, where $\mathbf{L}^i \in \mathbb{Z}_p^{2\ell_k}$, and $\mathbf{s}_{res}^{i+1} \in \mathbb{Z}_p^{\ell_s \ell_{\mathrm{smdg}}}$.

  2. Treat $\mathbf{s}_{res}^{i+1}$ as an integer vector (denote its j-th entry by $s_{res,j}^{i+1}$). Recover the smudging source vector $\mathbf{s}_{res}^{\prime,i+1}$ (denote its j-th entry by $s_{res,j}^{\prime,i+1}$) via

  $$s_{res,j}^{\prime,i+1} = \lfloor s_{res,j}^{i+1} \rfloor_2 - \begin{cases} 1 & \text{if } y_{i-1} = 1, s_{res,j}^{i+1} = 0 \mod 2 \\ 0 & \text{otherwise.} \end{cases} \tag{3.19}$$

  Finally, compute the LHE key $\mathbf{s}_{res}^{i+1,*}$ for the next iteration

  $$\mathbf{s}_{res}^{i+1,*} \leftarrow \mathsf{Smdg.Smudge}(\mathbf{s}_{res}^{\prime,i+1}; \{\mathsf{sd}_j^i\}_j).$$

  After iteration $i = d$, output the recovered labels $\{\mathbf{L}^i\}_{i\in[d]}$.

Correctness. We show that the above scheme is *correct* as specified by Eq. (3.17), which requires that given a correctly generated input-wire label $\mathbf{L}^{in} = x\mathbf{z}_1^{in} + \mathbf{z}_2^{in} \pmod p$ with input $x \in \mathbb{Z}_B$ and a integer r, and the garbled table tb, the decoding algorithm $\mathsf{TC}'.\mathsf{Dec}$

recovers the correct output-wire labels $\mathbf{L}^i = y_{i-1}\mathbf{z}_1^i + \mathbf{z}_2^i \pmod p$, where $y = x + r$ over $\mathbb{Z}$, and $y_{i-1} = \lfloor y \rfloor_{2^i}$.

By construction, $\mathsf{TC}'.\mathsf{Dec}$ uses $\mathbf{s}_{res}^{i,*}$ as the secret key to decrypt the $\overline{\mathsf{lhe}}$ ciphertext ct_{res}^i, which is the result of homomorphically evaluating $f_i(x_1, x_2) = y_{i-1}x_1 + x_2$ over $\mathsf{ct}_1'^{,i}, \mathsf{ct}_2'^{,i}$ respectively. By the special-purpose linear homomorphism of $\overline{\mathsf{lhe}}$ (Lemma 3.2), ct_{res}^i can be decrypted using secret key $f_i(\mathbf{s}_1^{i,*}, \mathbf{s}_2^{i,*})$ computed over $\mathbb{Z}$, i.e., we need to show

$$\forall i \in [d], \quad \mathbf{s}_{res}^{i,*} = f_i(\mathbf{s}_1^{i,*}, \mathbf{s}_2^{i,*}) = y_{i-1}\mathbf{s}_1^{i,*} + \mathbf{s}_2^{i,*} \text{ (over } \mathbb{Z}). \tag{3.20}$$

Note that if Eq. (3.20) holds, then invoking Lemma 3.2 shows correctness:

$$\begin{aligned} \mathbf{m}_{res}^i &= f((\mathbf{z}_1^i, \mathbf{s}_1^{i+1}), (\mathbf{z}_2^i, \mathbf{s}_2^{i+1})) \\ &= (f(\mathbf{z}_1^i, \mathbf{z}_2^i), f(\mathbf{s}_1^{i+1}, \mathbf{s}_2^{i+1})) \\ &= (\underbrace{y_{i-1}\mathbf{z}_1^i + \mathbf{z}_2^i}_{\mathbf{L}^i}, \underbrace{y_{i-1}\mathbf{s}_1^{i+1} + \mathbf{s}_2^{i+1}}_{\mathbf{s}_{res}^{i+1}},) \mod p. \end{aligned} \tag{3.21}$$

We now prove by induction that Eq. (3.20) holds. The base case for $i = 1$ follows directly by construction. For $i > 1$, recall that by construction the LHE keys $\mathbf{s}_1^{i,*}, \mathbf{s}_2^{i,*}, \mathbf{s}_{res}^{i,*}$ are each computed by applying $\mathsf{Smdg}.\mathsf{Smudge}(\cdot; \{\mathsf{sd}_j^i\}_j)$ to the source vectors $\mathbf{s}_1^i$, $\lfloor \mathbf{s}_2^i \rfloor_2, \mathbf{s}_{res}'^{,i}$, respectively. Therefore, it's enough to show

$$\mathbf{s}_{res}'^{,i} = f_i(\mathbf{s}_1^i, \lfloor \mathbf{s}_2^i \rfloor_2) = y_{i-1}\mathbf{s}_1^i + \lfloor \mathbf{s}_2^i \rfloor \text{ (over } \mathbb{Z})$$

instead.

Suppose $\mathbf{s}_{res}^{t,*} = y_{i-1}\mathbf{s}^{t,*} + \mathbf{s}^{t,*}$ holds over $\mathbb{Z}$, for some integer $t \geq 1$. Then Eq. (3.21) implies that

$$\mathbf{s}_{res}^{t+1} = y_{t-1}\mathbf{s}_1^{t+1} + \mathbf{s}_2^{t+1} \bmod p = y_{t-1}\mathbf{s}_1^{t+1} + \mathbf{s}_2^{t+1} \text{ (over } \mathbb{Z}),$$

where the last equality holds because the magnitude of every entry in $\mathbf{s}_{res}^{t+1}$ does not exceed

$p/2$. In the following, we use the short hand $(x)_p$ to mean $x \mod p$. We further derive:

$$\begin{aligned}
\mathbf{s}_{res}^{t+1} &= y_{t-1}\mathbf{s}_1^{t+1} + \mathbf{s}_2^{t+1} \\
&= ((y_{t-1})_2 + 2y_t)\mathbf{s}_1^{t+1} + (\mathbf{s}_2^{t+1})_2 + 2\lfloor \mathbf{s}_2^{t+1} \rfloor_2 \\
&= \underbrace{(y_{t-1})_2\mathbf{s}_1^{t+1} + (\mathbf{s}_2^{t+1})_2}_{=\mathbf{c}} + 2\underbrace{y_t\mathbf{s}_1^{t+1} + \lfloor \mathbf{s}_2^{t+1} \rfloor_2}_{=f_{t+1}(\mathbf{s}_1^{t+1}, \lfloor \mathbf{s}_2^{t+1} \rfloor_2)} \\
\implies \lfloor \mathbf{s}_{res}^{t+1} \rfloor_2 &= f_{t+1}(\mathbf{s}_1^{t+1}, \lfloor \mathbf{s}_2^{t+1} \rfloor_2) + \lfloor \mathbf{c} \rfloor_2.
\end{aligned}$$

Compare with Eq. (3.19), we observe that if each coordinate of $\mathbf{c}$ satisfies

$$c_j = \begin{cases} 1 & \text{if } y_{i-1} = 1, s_{res,j}^{t+1} = 0 \mod 2 \\ 0 & \text{otherwise,} \end{cases} \tag{3.22}$$

then we can conclude that the source vector $\mathbf{s}_{res}^{\prime,t+1}$ computed by the decryption algorithm $\mathsf{TC}'.\mathsf{Dec}$ indeed satisfy $\mathbf{s}_{res}^{\prime,t+1} = \lfloor \mathbf{s}_{res}^{t+1} \rfloor_2 - \lfloor \mathbf{c} \rfloor = f_{t+1}(\mathbf{s}_1^{t+1}, \lfloor \mathbf{s}_2^{t+1} \rfloor_2)$, i.e. Eq. (3.20) holds for $i = t+1$.

It remains to show Eq. (3.22). We expand each coordinate of $\mathbf{c}$:

$$\begin{aligned}
c_j &= (y_{t-1})_2 s_1^{t+1} + (s_2^{t+1})_2 \\
\text{(Eq. 3.18)} &= (y_{t-1})_2 s_{1,j}^{t+1} + 1 - s_{1,j}^{t+1} + s_{1,j}^{t+1} \cdot r_{2,j}^{t+1},
\end{aligned}$$

where the terms $(y_{t-1})_2$, $s_{1,j}^{t+1}$, and $r_{2,j}^{t+1}$ are all binary values. We now directly analyze the values of c_j in all possible cases. If $(y_{t-1})_2 = 0$, then $0 \le e_j \le 1$. If $(y)_2 = 1$, then $c_j = 1 + s_{1,j}^2 \cdot r_{2,j}^2 \le 2$. That is, the only case when $\lfloor c_j \rfloor_2 = 1$ is when $(y)_2 = 1$, and $c_j = 2 \iff (s_{res,j}^{t+1})_2 = 0$. We have shown Eq. (3.22).

Security. We show that the scheme TC' in Construction 7 admits a weaker simulator $\mathsf{TC}'.\mathsf{Sim}$ that besides the output-wire labels $\{\mathbf{L}^i\}$ needs the value $y = x + r$ over $\mathbb{Z}$ to help with the simulation.

- $\mathsf{TC}'.\mathsf{Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^i\}_{i\in[d]}, y)$ takes as inputs a security parameter 1^λ, the public parameters generated by Setup in Figure 3.6, d arbitrary output-wire labels $\mathbf{L}^i \in \mathbb{Z}_p^{2\ell_k}$, and a integer y. It outputs the simulated input-wire label $\widetilde{\mathbf{L}}^{in}$ and garbled table $\widetilde{\mathsf{tb}}$

Lemma 3.6. *There exists a simulator* $\mathsf{TC'.Sim}$ *defined above such that for all sequences* $\{\mathbf{z}^i_{1,\lambda}, \mathbf{z}^i_{2,\lambda}\}_\lambda$ *where* $\mathbf{z}^i_{1,\lambda}, \mathbf{z}^i_{2,\lambda} \in \mathbb{Z}_p^{2\ell_k}$, $\{x_\lambda, r_\lambda\}_\lambda$ *where* $x_\lambda \in \mathbb{Z}_B$ $r_\lambda \in [0, 2B_{\text{smdg}}]$, *the following indistinguishability holds. (For more concise notations, the index* λ *is suppressed below.)*

$$\begin{array}{l|c} & \mathsf{pp} \leftarrow \mathsf{Setup}^{1^\lambda}, \\ \{\mathsf{pp}, \mathsf{TC'.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^i\}_{i\in[d]}, y)\} & \mathbf{z}^{in}_1, \mathbf{z}^{in}_2, \mathsf{tb} \leftarrow \mathsf{TC'.Garb}^{\mathsf{pp}}(1^\lambda, \{\mathbf{z}^i_1, \mathbf{z}^i_2\}_{i\in[d]}, r) \\ \approx_c \{\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb}\}. & \mathbf{L}^i = x\mathbf{z}^{in}_1 + \mathbf{z}^{in}_2 \mod p, \\ & y = x + r \ (\textit{over } \mathbb{Z}) \end{array}$$

Proof. We construct a simulator $\mathsf{TC'.Sim}$ that takes as inputs a security parameter 1^λ, the public parameters generated by Setup in Figure 3.6, d arbitrary output-wire labels $\mathbf{L}^i \in \mathbb{Z}_p^{2\ell_k}$, and an integer y. It simulates the input-wire label $\widetilde{\mathbf{L}}^{in}$ and garbled table $\widetilde{\mathsf{tb}} = (\{\widetilde{\mathsf{ct}}'^{,i}_1, \widetilde{\mathsf{ct}}'^{,i}_2\}_{i\in[d]}, \{\widetilde{\mathsf{sd}}^i_j\}^{i\in[d]}_{j\in[\ell_s]})$.

- $\mathsf{TC'.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^i\}_{i\in[d]}, y)$: Simulate the input-wire label $\widetilde{\mathbf{L}}^{in} = (\widetilde{\mathbf{s}}^{1,*}_{res}, y)$ by sampling $\widetilde{\mathbf{s}}^{1,*}_{res}$ as sufficiently large random integer values, and using the provided argument y directly.

  $$\widetilde{\mathbf{s}}^{1,*}_{res} = [-B'_{\text{smdg}}, B'_{\text{smdg}}]^{\ell_s}, \quad \widetilde{\mathbf{L}}^{in} = (\widetilde{\mathbf{s}}^{1,*}_{res}, y),$$

  where the smudging magnitude $B'_{\text{smdg}} = \lambda^{\omega(1)} B_{\text{smdg}} B_s$ is set to the same value as in $\mathsf{TC'.Garb}$. For $i = 2, \ldots, d$, sample smudging source vectors $\mathbf{s}^i_1 \leftarrow \{0,1\}^{\ell_s \ell_{\text{smdg}}}$ and $\mathbf{s}^i_2 \in \mathbb{Z}^{\ell_s \ell_{\text{smdg}}}$ as described in Eq. (3.18), and compute the source vectors $\mathbf{s}^i_{res}, \mathbf{s}'^{,i}_{res}$ as

  $$\mathbf{s}^i_{res} = y_{i-2}\mathbf{s}^i_1 + \mathbf{s}^i_2, \quad \mathbf{s}'^{,i}_{res} = y_{i-1}\mathbf{s}^i_1 + \lfloor \mathbf{s}^i_2 \rfloor \ (\text{over } \mathbb{Z}),$$

  where we use $y_i = \lfloor y \rfloor_{2^i}$ as a shorthand. Sample smudging seeds $\widetilde{\mathsf{sd}}^i_j \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{\text{smdg}}}, 1/4)$ for $j \in [\ell_s]$, and compute LHE keys $\mathbf{s}^{i,*}_1, \mathbf{s}^{i,*}_{res} \in \mathbb{Z}^{\ell_s}$ as

  $$\mathbf{s}^{i,*}_1 \leftarrow \mathsf{Smdg.Smudge}(\mathbf{s}^i_1; \{\widetilde{\mathsf{sd}}^i_j\}_j), \quad \mathbf{s}^{i,*}_{res} \leftarrow \mathsf{Smdg.Smudge}(\mathbf{s}'^{,i}_{res}; \{\widetilde{\mathsf{sd}}^i_j\}_j).$$

  Next, simulate the ciphertexts $\widetilde{\mathsf{ct}}'^{,i}_1$. The case of $i = 1$ is simpler: We simulate $\widetilde{\mathsf{ct}}^1_1$ as a fresh encryption of $\mathbf{0} \in \mathbb{Z}^{2\ell_k + \ell_s \ell_{\text{smdg}}}$.

  $$\mathbf{s}^{1,*}_1 \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s}), \quad \widetilde{\mathsf{ct}}^1_1 = \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}^{1,*}_1, \mathbf{0}).$$

For the cases of $i = 2, \ldots d$, sample a fresh LHE key $\mathbf{s}^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s})$, and simulate $\widetilde{\mathsf{ct}}_1^i$ by homomorphically adding two encryptions of $\mathbf{0}$, each using the LHE key $\mathbf{s}^i$ and $\mathbf{s}_1^{i,*}$.

$$\mathsf{ct}_s^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}^i, \mathbf{0}), \quad \mathsf{ct}_0^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1^{i,*}, \mathbf{0}), \quad \widetilde{\mathsf{ct}}_1^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_s^i, \mathsf{ct}_0^i).$$

For all $i \in [d]$, smudge the noise in $\widetilde{\mathsf{ct}}_1^i$ to produce $\widetilde{\mathsf{ct}}_1^{\prime,i}$ using an encryption of a fresh noise vector $\mathbf{e}_1^i \leftarrow [-\alpha_1, \alpha_1]^{2\ell_k + \ell_s \ell_{\text{smdg}}}$ under the LHE key 0.

$$\mathsf{ct}_{e,1}^i \leftarrow \mathsf{lhe}.\mathsf{Enc}(0, \mathbf{e}_1^i), \quad \widetilde{\mathsf{ct}}_1^{\prime,i} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_1^i, \mathsf{ct}_{e,1}^i).$$

Finally, simulate the ciphertexts $\widetilde{\mathsf{ct}}_2^{\prime,i}$, for all $i \in [d]$, via homomorphic evaluation of $\overline{\mathsf{lhe}}$, subject to the constraint that the decryption procedure in $\mathsf{TC}'.\mathsf{Dec}$ must produce the correct results $\mathbf{m}_{res}^i = (\mathbf{L}^i, \mathbf{s}_{res}^{i+1})$. More specifically, consider the function $f_R(x_{res}, x_1) = x_{res} - y_{i-1} x_1$, and generate $\widetilde{\mathsf{ct}}_2^i$ as follows.

$$\widetilde{\mathsf{ct}}_{res}^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(s_{res}^{i,*}, \mathbf{m}_{res}^i), \quad \widetilde{\mathsf{ct}}_2^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(f_R, \widetilde{\mathsf{ct}}_{res}^i, \widetilde{\mathsf{ct}}_1^{\prime,i}).$$

Smudge the noise in $\widetilde{\mathsf{ct}}_2^{\prime i}$ similarly as the computation of $\widetilde{\mathsf{ct}}_1^{\prime i}$, using a fresh noise vector $\mathbf{e}_2^i \leftarrow [-\alpha_2, \alpha_2]^{2\ell_k + \ell_s \ell_{\text{smdg}}}$.

$$\mathsf{ct}_{e,2}^i \leftarrow \mathsf{lhe}.\mathsf{Enc}(0, \mathbf{e}_2^i), \quad \widetilde{\mathsf{ct}}_2^{\prime,i} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2^i, \mathsf{ct}_{e,2}^i).$$

We now argue that $\mathsf{TC}'.\mathsf{Sim}$ described above satisfies the security requirement. Consider and sequences $\{\mathbf{z}_{1,\lambda}^i, \mathbf{z}_{2,\lambda}^i\}_\lambda$ where $\mathbf{z}_{1,\lambda}^i, \mathbf{z}_{2,\lambda}^i \in \mathbb{Z}_p^{2\ell_k}$, $\{x_\lambda, r_\lambda\}_\lambda$ where $x_\lambda \in \mathbb{Z}_B$ $r_\lambda \in [0, 2B_{\text{smdg}}]$. We define $3d + 2$ hybrids where the first hybrid is exactly the real-world distribution, and the last hybrid is exactly the simulated distribution using $\mathsf{TC}'.\mathsf{Sim}$, and show their indistinguishability. (In the following, we suppress the subscript λ.)

- Hyb_1: This hybrid generates $\mathsf{pp}, \mathbf{L}^{in}$, $\mathsf{tb} = (\{\mathsf{ct}_1^{\prime,i}, \mathsf{ct}_2^{\prime,i}\}_{i\in[d]}, \{\mathsf{sd}_j^i\}_{j\in[\ell_s]}^{i\in[d]})$ honestly using the algorithms Setup, and $\mathsf{TC}'.\mathsf{Garb}$. More concretely, the variables are sampled as follows:

  - Generate $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.

– For $i = 1$, sample LHE keys $\mathbf{s}_1^{1,*} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(1^{\ell_s})$, $\mathbf{s}_2^{1,*} \leftarrow [-B'_{\text{smdg}}, B'_{\text{smdg}}]^{\ell_s}$. The input-wire label $\mathbf{L}^{in}$ equals

$$y = x + r \text{ (over } \mathbb{Z}), \quad \mathbf{L}^{in} = (y\mathbf{s}_1^{1,*} + \mathbf{s}_2^{1,*}, y) \mod p.$$

– For $i = 2$, first sample smudging source vectors $\mathbf{s}_1^i \leftarrow \{0,1\}^{\ell_s \ell_{\text{smdg}}}$, and $\mathbf{s}_2^i$ as Eq. (3.18). Next sample smudging seeds $\mathsf{sd}_j^i \leftarrow \mathsf{Smdg.Gen}(1^{\ell_{\text{smdg}}}, 1/4)$ for $j \in [\ell_s]$, and compute LHE keys $\mathbf{s}_1^{i,*}, \mathbf{s}_2^{i,*}$ as

$$\mathbf{s}_1^{i,*} \leftarrow \mathsf{Smdg.Smudge}(\mathbf{s}_1^i; \{\mathsf{sd}_j^i\}_j), \quad \mathbf{s}_2^{i,*} \leftarrow \mathsf{Smdg.Smudge}(\lfloor \mathbf{s}_2^i \rfloor_2; \{\mathsf{sd}_j^i\}_j).$$

– Compute the ciphertexts $\mathsf{ct}_1^i, \mathsf{ct}_2^i$ as $\mathsf{ct}_j^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_j^{i,*}, (\mathbf{z}_j^i, \mathbf{s}_j^{i+1}))$ for $j = 1, 2$. Then add smudging noises via homomorphic evaluation to produce $\mathsf{ct}_1'^{,i}, \mathsf{ct}_2'^{,i}$.

$$\mathbf{e}_j^i \leftarrow [-\alpha_j, \alpha_j]^{2\ell_k + \ell_s \ell_{\text{smdg}}}, \quad \mathsf{ct}_{e,j}^i \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}_j), \quad \mathsf{ct}_j'^{,i} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_j^i, \mathsf{ct}_{e,j}^i).$$

- Hyb_2: This hybrid proceeds identically as Hyb_1, except that the ciphertexts $\widetilde{\mathsf{ct}}_2'^{,i}$ are generated via homomorphic evaluation of $\overline{\mathsf{lhe}}$, under the constraint that decryption recovers the correct values $\mathbf{m}_{res}^i = (\mathbf{L}^i, \mathbf{s}_{res}^{i+1})$ where

$$\mathbf{s}_{res}^i = y_{i-2}\mathbf{s}_1^i + \mathbf{s}_2^i \text{ (over } \mathbb{Z}), \quad \mathbf{L}^i = y_{i-1}\mathbf{z}_1^i + \mathbf{z}_2^i \mod p.$$

More specifically, Hyb_2 first computes the LHE key $\mathbf{s}_{res}^{i,*}$ as

$$\mathbf{s}_{res}'^{,i} = y_{i-1}\mathbf{s}_1^i + \lfloor \mathbf{s}_2^i \rfloor \text{ (over } \mathbb{Z}), \quad \mathbf{s}_{res}^{i,*} = \mathsf{Smdg.Smudge}(\mathbf{s}_{res}'^{,i}; \{\widetilde{\mathsf{sd}}_j^i\}_j),$$

and then compute $\widetilde{\mathsf{ct}}_2^i$ as

$$\widetilde{\mathsf{ct}}_{res}^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_{res}^{i,*}, \mathbf{m}_{res}^i), \quad \widetilde{\mathsf{ct}}_2^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(f_R, \widetilde{\mathsf{ct}}_{res}^i, \mathsf{ct}_1'^{,i}),$$

where $f_R(x_{res}, x_1) = x_{res} - y_{i-1}x_1$. Finally, smudge $\widetilde{\mathsf{ct}}_2^i$ with noise $\mathbf{e}_2^i \leftarrow [-\alpha, \alpha]^{2\ell_k + \ell_s \ell_{\text{smdg}}}$ to get $\widetilde{\mathsf{ct}}_2'^{,i}$

$$\mathsf{ct}_{e,2}^i \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}_2^i), \ \widetilde{\mathsf{ct}}_2'^{,i} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_2^i, \mathsf{ct}_{e,2}^i).$$

Note that the only difference between Hyb_1 and Hyb_2 lies in how ct_2' and $\widetilde{\mathsf{ct}}_2'$ are generated. In the former, ct_2' is an additionally noisy ciphertext of $(\mathbf{z}_2^i, \mathbf{s}_2^{i+1})$ encrypted under the

LHE key $\mathbf{s}_2^{i,*}$. In the latter, $\widetilde{\mathsf{ct}}_2'$ is the output ciphertext produced by homomorphically evaluating f_R on $\mathsf{ct}_{res}^i, \mathsf{ct}_1'^{,i}$, smudged with additional noise.

By the linearity of $\mathsf{Smdg.Smudge}(\cdot; \{\mathsf{sd}_j^i\}_j)$, we have $f_R(\mathbf{s}_{res}^{i,*}, \mathbf{s}_1^{i,*}) = \mathbf{s}_2^{i,*}$. It's also easy to verify that $f_R(\mathbf{m}_{res}^i, (\mathbf{z}_1^i, \mathbf{s}_1^{i+1})) = (\mathbf{z}_2^i, \mathbf{s}_2^{i+1})$. Lemma 3.1 shows that these two ways of generating ciphertexts are statistically close, provided that the the magnitude α_2 of the smudging noises is sufficiently large. This is indeed the case since $\alpha_2 = \lambda^{\omega(1)} \max(p, B_e, \alpha_1)^2$ (Equation (3.5)). Therefore by the lemma, the distributions of $\mathsf{ct}_2'^{i}$ in Hyb_1 and $\widetilde{\mathsf{ct}}_2'^{,i}$ in Hyb_2 are statistically close, and so are these two hybrids.

- $\mathsf{Hyb}_{3.1.1}$: This hybrid proceeds identically as Hyb_2, except that instead of computing $\mathbf{s}_{res}^{1,*} = y\mathbf{s}_1^1 + \mathbf{s}_2^i$ over the integers as in Hyb_2, $\mathsf{Hyb}_{3.1.1}$ directly samples

$$\widetilde{\mathbf{s}}_{res}^{1,*} \leftarrow [-B'_{\text{smdg}}, B'_{\text{smdg}}].$$

Similar arguments to those in Hyb_3 of Lemma 3.3 shows that $\mathsf{Hyb}_{3.1.1}$ is statistically close to Hyb_2.

- $\mathsf{Hyb}_{3.1.2} = \mathsf{Hyb}_{3.1.3}$: This hybrid proceeds identically as $\mathsf{Hyb}_{3.1.1}$, except that instead of generating ct_1^1 as a fresh encryption of $(\mathbf{z}_1^1, \mathbf{s}_1^2)$ using the LHE key $\mathbf{s}_1^{1,*}$ as in $\mathsf{Hyb}_{3.1.1}$, $\widetilde{\mathsf{ct}}_1^1$ is now generated as an encryption of the vector $\mathbf{0}$, still using the LHE key $\mathbf{s}_1^{1,*}$. Similar arguments to those in Hyb_4 of Lemma 3.3 shows that $\mathsf{Hyb}_{3.1.2}$ and $\mathsf{Hyb}_{3.1.1}$ are computationally indistinguishable.

- $\mathsf{Hyb}_{3.i.1}$, $i = 2, \ldots, d$: This hybrid proceeds identically as $\mathsf{Hyb}_{3.(i-1).3}$, except that the LHE key $\widetilde{\mathbf{s}}_1^{i,*}$ is generated as the sum of a fresh LHE key $\mathbf{s}^i$ and the original $\mathbf{s}_1^{i,*}$ as computed in $\mathsf{Hyb}_{3.(i-1).3}$.

$$\mathbf{s}^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}(1^{\ell_s}), \quad \widetilde{\mathbf{s}}_1^{i,*} = \mathbf{s}_1^{i,*} + \mathbf{s}^i \text{ (over } \mathbb{Z}),$$

where

$$\mathbf{s}_1^{i,*} = \mathsf{Smdg.Smudge}(\mathbf{s}_1^i; \{\widetilde{\mathsf{sd}}_j^i\}_j).$$

By the smudging property of $\mathsf{Smdg.Smudge}$, if we can show that at least $1/4$ coordinates of $\mathbf{s}_1^i$ remains hidden in $\mathsf{Hyb}_{3.(i-1).3}$, then we can conclude that $\mathsf{Hyb}_{3.i.1}$ and $\mathsf{Hyb}_{3.(i-1).3}$ are statistically close.

Recall that in $\mathsf{Hyb}_{3.i-1.1}$, the ciphertexts $\widetilde{\mathsf{ct}}_1^{\prime,i}, \widetilde{\mathsf{ct}}_1^{\prime,i}$ that originally encrypt vectors that contain $\mathbf{s}_1^i, \mathbf{s}_2^i$ in the honest world, have been simulated using only the vector $\mathbf{s}_{res}^i = y_{i-2}\mathbf{s}_1^i + \mathbf{s}_2^i$ over $\mathbb{Z}$. Therefore, the only values that possibly leak information about $\mathbf{s}_1^i$ are $\mathbf{s}_{res}^i$,y. We show that given those, with overwhelming probability, at least $1/4$ coordinates of $\mathbf{s}_1^i$ remains hidden using the following claims.

Claim 3.8. *Let* $\mathbf{s}_1^i, \mathbf{s}_{res}^i$ *be computed as described in* $\mathsf{Hyb}_{3.(i-1).3}$. *Let* $s_{1,j}^i, s_{res,j}^i$ *be the* j^{th} *entries, we have*

$$\forall j \in [\ell^*], \Pr[s_{1,j}^i = 1 | s_{res,j}^i = 1] = \Pr[s_{1,j}^i = 0 | s_{res,j}^i = 1] = 1/2.$$

Claim 3.9. *let* $\mathbf{s}_1^i, \mathbf{s}_{res}^i$ *be computed as described in* $\mathsf{Hyb}_{3.(i-1).3}$. *With overwhelming probability, at least* $1/4$ *coordinates of* $\mathbf{s}_{res}^i$ *have value* 1.

Now, we can invoke the smudging property of $\mathsf{Smdg.Smudge}$ to conclude that $\mathsf{Hyb}_{3.i.1}$ and $\mathsf{Hyb}_{3.(i-1).3}$ are statistically close.

- $\mathsf{Hyb}_{3.i.2}$, $i = 2, \ldots, d$: This hybrid proceeds identically as $\mathsf{Hyb}_{3.(i-1).3}$, except that instead of generating the ciphertext $\mathsf{ct}_1^{\prime,i}$ as an additional noisy encryption of $(\mathbf{z}_1^i, \mathbf{s}_1^{i+1})$ under the key $\widetilde{\mathbf{s}}_1^{i,*} = \mathbf{s}_1^{i,*} + \mathbf{s}^i$, $\mathsf{Hyb}_{3.i.2}$ computes $\widetilde{\mathsf{ct}}_1^{\prime,i}$ via homomorphic addition of ciphertexts $\mathsf{ct}_s^i, \mathsf{ct}_0^i$, where ct_s^i encrypts the above vector under the fresh LHE key $\mathbf{s}^i$, and ct_0^i encrypts the vector $\mathbf{0}$ under the LHE key $\mathbf{s}_1^{i,*}$.

$$\mathsf{ct}_s^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}^i, (\mathbf{z}_1^i, \mathbf{s}_1^{i+1})), \quad \mathsf{ct}_0^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Enc}}(\mathbf{s}_1^{i,*}, \mathbf{0}), \quad \widetilde{\mathsf{ct}}_1^i \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \mathsf{ct}_s^i, \mathsf{ct}_0^i).$$

Finally, smudge the noise in $\widetilde{\mathsf{ct}}_1^i$ with noise $\mathbf{e}_1^i \leftarrow [-\alpha_1, \alpha_1]^{2\ell_k + \ell_s \ell_{\text{smdg}}}$ to get $\widetilde{\mathsf{ct}}_1^{\prime,i}$.

$$\mathsf{ct}_{e,1}^i \leftarrow \mathsf{lhe.Enc}(0, \mathbf{e}_1^i), \quad \widetilde{\mathsf{ct}}_1^{\prime,i} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Eval}}(+, \widetilde{\mathsf{ct}}_1^i, \mathsf{ct}_{e,1}^i).$$

Similar arguments to those in Hyb_3 of Lemma 3.3 shows that $\mathsf{Hyb}_{3.i.2}$ is statistically close to $\mathsf{Hyb}_{3.i.1}$.

- $\mathsf{Hyb}_{3.i.3}$, $i = 2, \ldots, d$: This hybrid proceeds identically as $\mathsf{Hyb}_{3.i.2}$, except that instead of generating ct_s^i as a fresh encryption of $(\mathbf{z}_1^i, \mathbf{s}_1^{i+1})$ using the LHE key $\mathbf{s}^i$ as in $\mathsf{Hyb}_{3.i.2}$,

$\widetilde{\mathsf{ct}}_s^i$ is now generated as an encryption of the vector $\mathbf{0}$, still using the LHE key $\mathbf{s}^i$. Similar arguments to those in Hyb_4 of Lemma 3.3 shows that $\mathsf{Hyb}_{3.1.2}$ and $\mathsf{Hyb}_{3.1.1}$ are computationally indistinguishable.

By a hybrid argument, we have that Hyb_1 and $\mathsf{Hyb}_{3.d.3}$ are computationally indistinguishable. Since Hyb_1 samples $(\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb})$ exactly as in the real-world, and $\mathsf{Hyb}_{3.d.3}$ samples $(\widetilde{\mathsf{pp}}, \widetilde{\mathbf{L}}^{in}, \widetilde{\mathsf{tb}})$ as the simulator $\mathsf{TC}'.\mathsf{Sim}$ does, we conclude that the simulated distribution is indistinguishable to the real distribution. Hence Lemma 3.6 holds. □

Constructing the Bit Decomposition Gadget. Next, we construct a bit decomposition scheme using the imperfect truncation scheme in Construction 7, and a garbling scheme Yao for Boolean circuits with circuit privacy per Definition 2.1, 2.2.

Construction 8 (bit decomposition)**.** We construct a bit decomposition scheme for the mixed bounded integer and boolean computation model, over the domain $\mathcal{I} = \mathbb{Z}_B$. An element in $\mathbb{Z}_B$ has bit length $d = \lceil \log(2B+1) \rceil$.

- $\mathsf{BD}.\mathsf{Garb}^{\mathsf{pp}}(1^\lambda, 1^\ell, \{\mathbf{z}_1^i, \mathbf{z}_2^i\}_{i\in[d]})$ proceeds in two steps.

  1. Sample a random non-negative integer $r' \leftarrow [0, B_{\text{smdg}}]$, where $B_{\text{smdg}} = \lambda^{\omega(1)}B$, and shift it by B to obtain a random integer $r = B + r'$. Defines the Boolean circuit C^r_{sub} as

$$C^r_{\text{sub}}(\mathsf{bits}(y)) = \{\mathbf{L}^i\}_{i\in[d]}, \quad \left| \begin{array}{l} x = y - r \text{ (over } \mathbb{Z}) \\ \mathbf{L}^i = \mathsf{bits}(x)_i \mathbf{z}_1^i + \mathbf{z}_2^i \mod p \end{array} \right. \tag{3.23}$$

     where C^r_{sub} has the value r and the keys $\mathbf{z}_1^i, \mathbf{z}_2^i$ hardcoded. Let $d' = \lceil \log(B_{\text{smdg}} + 2B + 1) \rceil + 1$ be the bit length of y. Compute the garbled circuit and (Boolean) evaluation keys $\widehat{C}^r_{\text{sub}}, \{\mathbf{k}_0^i, \mathbf{k}_1^i\}_{i\in[d']} \leftarrow \mathsf{Yao}.\mathsf{Garb}(1^\lambda, C^r_{\text{sub}})$.

  2. Define vectors $\bar{\mathbf{z}}_1^i, \bar{\mathbf{z}}_2^i$ to encode the evaluation keys $\mathbf{k}_0^i, \mathbf{k}_1^i$ as described in Eq. (3.16): $\bar{\mathbf{z}}_1^i = \bar{\mathbf{k}}_1^i - \bar{\mathbf{k}}_0^i \mod p$, $\bar{\mathbf{z}}_2^i = \bar{\mathbf{k}}_0^i$, where $\bar{\mathbf{k}}_b^i \in \mathbb{Z}_p^{\ell_k}$ encodes $\mathbf{k}_b^i \in \{0,1\}^{\ell_k}$. Next, sample

random vectors $\{\mathbf{r}^i \leftarrow \mathbb{Z}_p^{\ell_k}\}_{i\in[d']}$, and define vectors $\{\bar{\mathbf{z}}_1^{\prime,i}, \bar{\mathbf{z}}_2^{\prime,i}\}_{i\in[d']}$ to be passed as arguments into $\mathsf{TC}'.\mathsf{Garb}$:

$$\begin{aligned}\bar{\mathbf{z}}_1^{\prime,i} &= (\bar{\mathbf{z}}_1^{i-1}, \quad \bar{\mathbf{z}}_1^{i}) \\ \bar{\mathbf{z}}_2^{\prime,i} &= (\mathbf{r}^{i-1}, \quad 2\mathbf{r}^i + \bar{\mathbf{z}}_2^i) \mod p.\end{aligned} \tag{3.24}$$

To deal with the edge case $i = 1$, set dummy vectors $\mathbf{r}^0 = \bar{\mathbf{z}}_1^0 = \mathbf{0}$. Finally, compute the keys of the input wire $\mathbf{z}_1^{in}, \mathbf{z}_2^{in}$, and the garbled table tb' using $\mathsf{TC}'.\mathsf{Garb}$, and output $\mathbf{z}_1^{in}, \mathbf{z}_2^{in}$, $\mathsf{tb} = (\mathsf{tb}', \widehat{C}_{\text{sub}}^r)$.

$$\mathbf{z}_1^{in}, \mathbf{z}_2^{in}, \mathsf{tb}' \leftarrow \mathsf{TC}'.\mathsf{Garb}^{\mathsf{pp}}(1^\lambda, \{\bar{\mathbf{z}}_1^{\prime,i}, \bar{\mathbf{z}}_2^{\prime,i}\}_{i\in[d']}, r).$$

- $\mathsf{BD}.\mathsf{Dec}^{\mathsf{pp}}(\mathbf{L}^{in}, \mathsf{tb})$: Parse the garbled table tb as $\mathsf{tb} = (\mathsf{tb}', \widehat{C})$, and recover labels $\{\bar{\mathbf{L}}^i\}_{i\in[d']}$ by running $\mathsf{TC}'.\mathsf{Dec}$. $\{\bar{\mathbf{L}}^i\}_{i\in[d']} \leftarrow \mathsf{TC}'.\mathsf{Dec}(\mathbf{L}^{in}, \mathsf{tb}')$. Parse each label $\bar{\mathbf{L}}^i$ as $\bar{\mathbf{L}}^i = (\mathbf{u}^{i-1}, \mathbf{v}^i)$, and recover the encoding $\bar{\mathbf{z}}_{res}^i$ for the Boolean evaluation key $\mathbf{k}^i$ as

$$\bar{\mathbf{z}}_{res}^i = \mathbf{v}^i - 2\mathbf{u}^i \mod p.$$

Finally, use $\{\mathbf{k}^i\}_{i\in[d']}$ to evaluate the garbled circuit $\widehat{C}$ to recover and output $\{\mathbf{L}^i\}_{i\in[d]}$.

$$\{\mathbf{L}^i\}_{i\in[d]} \leftarrow \mathsf{Yao}.\mathsf{Eval}(\widehat{C}, \{\mathbf{k}^i\}_{i\in[d']}).$$

Correctness. By the correctness of the Boolean garbling scheme Yao, the decryption result is correct if the encodings $\bar{\mathbf{z}}_{res}^i$ recovered during $\mathsf{BD}.\mathsf{Dec}$ are indeed the encodings for evaluation keys $\mathbf{k}_{\mathsf{bits}(y)_i}^i$, i.e., $\bar{\mathbf{z}}_{res}^i = \mathbf{k}_{\mathsf{bits}(y)_i}^i$.

By the correctness of $\mathsf{TC}'.\mathsf{Garb}$ and $\mathsf{TC}'.\mathsf{Dec}$ as specified in Eq. (3.17), for all $i \in [d']$, we have

$$\begin{aligned}\bar{\mathbf{L}}^i &= y_{i-1}\bar{\mathbf{z}}_1^{\prime,i} + \bar{\mathbf{z}}_2^{\prime,i} \\ \text{(Eq. 3.24)} &= y_{i-1} \cdot (\bar{\mathbf{z}}_1^{i-1}, \bar{\mathbf{z}}_1^i) + (\mathbf{r}^{i-1}, 2\mathbf{r}^i + \bar{\mathbf{z}}_2^i) \\ &= (\underbrace{y_{i-1}\bar{\mathbf{z}}_1^{i-1} + \mathbf{r}^{i-1}}_{=\mathbf{u}^{i-1}}, \underbrace{y_{i-1}\bar{\mathbf{z}}_1^i + 2\mathbf{r}^i + \bar{\mathbf{z}}_2^i}_{=\mathbf{v}^i}) \mod p \\ \implies \mathbf{u}^i &= y_i\bar{\mathbf{z}}_1^i + \mathbf{r}^i \\ \mathbf{v}^i &= y_{i-1}\bar{\mathbf{z}}_1^i + 2\mathbf{r}^i + \bar{\mathbf{z}}_2^i \mod p,\end{aligned} \tag{3.25}$$

where $y_i = \lfloor y \rfloor_{2^i}$. Recall that in $\mathsf{BD.Garb}$, we set the smudging integer r as $r = B + r'$, where r' is sampled from $[0, B_{\text{smdg}}]$. For any $x \in \mathbb{Z}_B$, we are guaranteed that $y = x + r = (x + B) + r'$ is a non-negative integer. Using the above, we verify that for all $i \in [d']$,

$$\begin{aligned} \bar{\mathbf{z}}^i_{res} = \mathbf{v}^i - 2\mathbf{u}^i &= \underbrace{(y_{i-1} - 2y_i)}_{\mathsf{bits}(y)_i} \bar{\mathbf{z}}^i_1 + \bar{\mathbf{z}}^i_2 \\ (\text{Eq. } 3.16) &= \mathbf{k}^i_{\mathsf{bits}(y)_i} \mod p. \end{aligned} \tag{3.26}$$

Security.

Lemma 3.7. *Construction 8 is* secure *per Definition 3.3.*

Proof. We construct a simulator $\mathsf{BD.Sim}$ that takes as inputs a security parameter λ, the public parameters $\mathsf{pp} = (\overline{\mathsf{lhe}}.\overline{\mathsf{pp}}, \alpha)$ generated by Setup in Figure 3.6, and d arbitray output-wire labels $\mathbf{L}^i \in \mathbb{Z}_p^{\ell'}$, where d is the bit length of B-bounded integers, and the dimension ℓ' of the output-wire labels can be an arbitrary polynomial in λ. It simulates the input wire label $\widetilde{\mathbf{L}}^{in}$, and the garbled table $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{tb}}', \widetilde{C}^r_{\text{sub}})$.

- $\mathsf{BD.Sim}(1^\lambda, \mathsf{pp}, \{\mathbf{L}^i\}_{i\in[d]})$: Simulate the Boolean garbled circuit $\widetilde{C}^r_{\text{sub}}$ and the Boolean evaluation keys $\{\widetilde{\mathbf{k}}^i\}$ by running the simulator $\mathsf{Yao.Sim}$, guaranteed by the security of the scheme Yao.

  $$\{\widetilde{\mathbf{k}}^i\}_{i\in[d']}, \widetilde{C}^r_{\text{sub}} \leftarrow \mathsf{Yao.Sim}(1^\lambda, \Phi_{\text{Topo}}(C^r_{\text{sub}}), \{\mathbf{L}^i\}_{i\in[d]}),$$

  where the topology $\Phi_{\text{Topo}}(C^r_{\text{sub}})$ of the circuit C^r_{sub} can be computed without knowing the hardcoded values in it. Encode each Boolean evaluation key $\widetilde{\mathbf{k}}^i$ as a vector $\widetilde{\mathbf{z}}^i_{res}$ in $\mathbb{Z}_p$. Next simulate intermediate vectors $\{\widetilde{\mathbf{u}}^i\}_{i\in[d']}$ as random $\mathbb{Z}_p$ vectors, and simulate $\{\widetilde{\mathbf{v}}^i\}_{i\in[d']}$ to satisfy the constraint that $\widetilde{\mathbf{z}}^i_{res} = \widetilde{\mathbf{v}}^i - 2\widetilde{\mathbf{u}}^i \pmod p$.

  $$\widetilde{\mathbf{u}}^i \leftarrow \mathbb{Z}_p^{\ell_k}, \quad \widetilde{\mathbf{v}}^i = 2\widetilde{\mathbf{u}}^i + \widetilde{\mathbf{z}}^i_{res} \mod p.$$

  Finally, simulate the output-wire labels $\{\widetilde{\mathbf{L}}^i\}_{i\in[d']}$ that are supposed to be recovered by $\mathsf{TC}'.\mathsf{Dec}$ as concatenations of the intermediate vectors $\{\widetilde{\mathbf{u}}^i, \widetilde{\mathbf{v}}^i\}_{i\in[d']}$.

  $$\widetilde{\mathbf{L}}^i = (\widetilde{\mathbf{u}}^{i-1}, \widetilde{\mathbf{v}}^i).$$

To deal with the edge case $i = 1$, set a dummy vector $\widetilde{\mathbf{u}}^0 = \mathbf{0}$. Finally, simulate the input-wire label $\widetilde{\mathbf{L}}^{in}$ and the garbled table $\widetilde{\mathsf{tb}}'$ by running the simulator sim', guaranteed by the security of the imperfect truncation scheme. Output $\widetilde{\mathbf{L}}^{in}$, $\widetilde{\mathsf{tb}} = (\widetilde{\mathsf{tb}}', \widetilde{C}^r_{\text{sub}})$.

$$\widetilde{y} \leftarrow [0, B_{\text{smdg}}], \quad \widetilde{\mathbf{L}}^{in}, \widetilde{\mathsf{tb}}' \leftarrow \mathsf{TC}'.\mathsf{Sim}(1^\lambda, \mathsf{pp}, \{\widetilde{\mathbf{L}}^i\}_{i\in[d]}, d', \widetilde{y}).$$

We now argue that $\mathsf{BD.Sim}$ described above satisfies the security requirement. Let $\ell' = \ell'(\lambda)$ be any polynomial, consider any sequences $\{\{\mathbf{z}^i_{1,\lambda}, \mathbf{z}^i_{2,\lambda}\}_{i\in[d]}\}_\lambda$ where $\mathbf{z}^i_{1,\lambda}, \mathbf{z}^i_{2\lambda} \in \mathcal{L}^{\ell'}$, and $\{x_\lambda\}_\lambda$ where $x_\lambda \in \mathbb{Z}_B$. We define six hybrids, $\mathsf{Hyb}_1, \ldots, \mathsf{Hyb}_6$, where the first hybrid is exactly the real-world distribution, and the last hybrid is exactly the simulated distribution using $\mathsf{BD.Sim}$, and show their indistinguishability. (In the following, we surpress the subscript λ).

- Hyb_1: This hybrid generates $\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb} = (\mathsf{tb}', \widehat{C}^r_{\text{sub}})$ honestly using the algorithms Setup, and $\mathsf{BD.Garb}$. More concretely, the variables are sampled as follows:

  - Generate $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$.
  - Compute a random integer $r = B + r'$, where $r' \leftarrow [0, B_{\text{smdg}}]$. The smudging magnitude $B_{\text{smdg}} = \lambda^{\omega(1)} B$ is set to the same value as in $\mathsf{BD.Garb}$.
  - Generate the Boolean garbled circuit and Boolean evaluation keys $\widehat{C}^r_{\text{sub}}$, $\{\mathbf{k}^i_0, \mathbf{k}^i_1\}_{i\in[d']}$ $\leftarrow \mathsf{Yao.Garb}(1^\lambda, C^r_{\text{sub}})$, where the Boolean circuit C^r_{sub} is defined as in Eq. (3.23).
  - Compute output-wire keys $\{\bar{\mathbf{z}}'^{,i}_1, \bar{\mathbf{z}}'^{,i}_2\}_{i\in[d']}$ as

    $$\bar{\mathbf{z}}'^{,i}_1 = (\bar{\mathbf{z}}^{i-1}_1, \bar{\mathbf{z}}^i_1), \quad \bar{\mathbf{z}}'^{,i}_2 = (\mathbf{r}^{i-1}, 2\mathbf{r}^i + \bar{\mathbf{z}}^i_2) \mod p,$$

    where $\mathbf{r}^i \leftarrow \mathbb{Z}^{\ell_k}_p$, and the vectors $\bar{\mathbf{z}}^i_1, \bar{\mathbf{z}}^i_1$ are computed from encodings of the evaluation keys $\bar{\mathbf{k}}^i_0, \bar{\mathbf{k}}^i_1$ as in Eq. (3.16).
  - Compute the input-wire keys and the garbled table

    $$\mathbf{z}^{in}_1, \mathbf{z}^{in}_2, \mathsf{tb}' \leftarrow \mathsf{TC}'.\mathsf{Garb}^{\mathsf{pp}}(1^\lambda, \{\bar{\mathbf{z}}'^{,i}_1, \bar{\mathbf{z}}'^{,i}_2\}_{i\in[d']}, d', r),$$

    and define the input label for x as $\mathbf{L}^{in} = x\mathbf{z}^{in}_1 + \mathbf{z}^{in}_2 \mod p$.

- Hyb_2: This hybrid proceeds identically as Hyb_1, except that the input-wire keys and the garbled table $\widetilde{\mathbf{z}}^{in}_1, \widetilde{\mathbf{z}}^{in}_2, \widetilde{\mathsf{tb}}'$ are generated by the simulator sim'.

$$\widetilde{\mathbf{z}}^{in}_1, \widetilde{\mathbf{z}}^{in}_2, \widetilde{\mathsf{tb}}' \leftarrow \mathsf{TC}'.\mathsf{Sim}(1^\lambda, \mathsf{pp}, \{\bar{\mathbf{L}}^i\}_{i\in[d]}, d', y),$$

where the output-wire labels $\bar{\mathbf{L}}^i$ are computed as by

$$y = x + r, y_{i-1} = \lfloor y \rfloor_{2^{i-1}} \text{ (over } \mathbb{Z}), \quad \bar{\mathbf{L}}^i = y_{i-1}\bar{\mathbf{z}}_1'^{,i} + \bar{\mathbf{z}}_2'^{,i} \mod p.$$

Directly by the security of the truncation scheme TC' (Lemma 3.6), Hyb_1 and Hyb_2 are computationally indistinguishable.

- Hyb_3: This hybrid hybrid proceeds identically as Hyb_2, except that the input-wire labels $\{\bar{\mathbf{L}}^i\}_i$ are computed using intermediate vectors $\{\mathbf{u}^i, \mathbf{v}^i\}_i$ as follows:

$$\mathbf{u}^i = y_i\bar{\mathbf{z}}_1^i + \mathbf{r}^i, \quad \mathbf{v}^i = 2\mathbf{u}^i + \bar{\mathbf{z}}_{res}^i \mod p, \quad \bar{\mathbf{L}}^i = (\mathbf{u}^{i-1}, \mathbf{v}^i),$$

where $\bar{\mathbf{z}}_{res}^i \in \mathbb{Z}_p^{\ell_k}$ is the encoding of the evaluation key $\mathbf{k}^i_{\mathsf{bits}(y)_i}$. As shown in Eq. (3.25),3.26, this change is purely syntactic. Hence Hyb_2 and Hyb_3 are identical.

- Hyb_4: This hybrid hybrid proceeds identically as Hyb_3, except that the intermediate vectors $\{\widetilde{\mathbf{u}}^i\}_{i\in[d]}$ are sampled at random. $\widetilde{\mathbf{u}}^i \leftarrow \mathbb{Z}_p^{\ell_k}$. Note that in Hyb_3, the intermediate vector $\mathbf{u}^i = y_i\bar{\mathbf{z}}_1^i + \mathbf{r}^i \pmod p$ was perfectly hidden by the random vector $\mathbf{r}^i$ as a one-time pad. Therefore we conclude that Hyb_3 and Hyb_4 are identical.

- Hyb_5: This hybrid hybrid proceeds identically as Hyb_4, except that it uses the simulator $\mathsf{Yao.Sim}$ to compute the garbled circuit and evaluation keys.

$$\{\widetilde{\mathbf{k}}^i_{\mathsf{bits}(y)_i}\}_{i\in[d']}, \widetilde{C}^r_{\text{sub}} \leftarrow \mathsf{Yao.Sim}(1^\lambda, \{\mathbf{L}^i\}_{i\in[d]}, \Phi_{\text{Topo}}(C^r_{\text{sub}})),$$

where the topology $\Phi_{\text{Topo}}(C^r_{\text{sub}})$ of the circuit C^r_{sub} can be computed without knowing the hardcoded values in it. Directly by the security of the scheme Yao, Hyb_5 and Hyb_4 are computationally indistinguishable.

- Hyb_6: This hybrid proceeds identically as Hyb_4, except that instead of computing computing $r' \leftarrow [0, B_{\text{smdg}}]$, $y = x + B + r'$, it directly samples $\widetilde{y} \leftarrow [0, B_{\text{smdg}}]$. The distributions of y in Hyb_5 and $\widetilde{y}$ in Hyb_6 are statistically close. This is becase $x + B$ is a value between $[0, 2B+1]$ while the range of r', $B_{\text{smdg}} = \lambda^{\omega(1)}B$ is superpolynomially larger than that of $x + B$. Hence r' statistically hides $x + B$. Therefore, Hyb_6 and Hyb_5 are statistically close.

By a hybrid argument, we have that Hyb_1 and Hyb_6 are computationally indistinguishable. Since Hyb_1 samples $(\mathsf{pp}, \mathbf{L}^{in}, \mathsf{tb})$ exactly as in the real-world, and Hyb_6 samples $(\widetilde{\mathsf{pp}}, \widetilde{\mathbf{L}}^{in}, \widetilde{\mathsf{tb}})$ as the simulator $\mathsf{BD.Sim}$ does, we conclude that the simulated distribution is indistinguishable to the real distribution, and Construction 8 is a secure bit decomposition gadget. □

3.9 Garbling Bounded Integer, Modular Arithmetic, and Mixed Computation

In this section, we first give the overall garbling scheme for bounded integer and modular arithmetic computation in Section 3.9.1, and next note the differences for the mixed bounded integer and Boolean computation in Section 3.9.2. We include an asymptotic efficiency analysis in this section. See Section 3.10 for a comparison of our concrete efficiency with prior works.

3.9.1 Bounded Integer and Modular Arithmetic Computation

In this section, we present our arithmetic garbling schemes for bounded integer computation $\mathcal{C}^{\mathrm{BI}}_{\mathbb{Z}_B}$ and for modulo-p computation $\mathcal{C}^{\mathrm{Arith}}_{\mathbb{Z}_p}$.

Arithmetic Computation Gadgets From [AIK11]. Sec. 3.5 (resp. Sec. 3.7) has prepared all the essential gadgets needed for garbling bounded integer computation (resp. modulo-p computation). For completeness, below we also include a description of the arithmetic computation gadgets taken from [AIK11]

Construction 9 (addition modulo p gadget)**.** We describe the "addition modulo p" gadget, with the label space $\mathcal{L} = \mathbb{Z}_p$. This construction does not need access to the public parameters pp generated by Setup.

- $\mathsf{ACmp.Garb}(1^\lambda, \mathbf{z}_1^{out}, \mathbf{z}_2^{out})$ samples a vector $\mathbf{r} \leftarrow \mathbb{Z}_p^{\ell'}$ of the same dimension ℓ' as the output-wire keys $\mathbf{z}_i^{out}$. It sets

$$\mathbf{z}_1^x = \mathbf{z}_1^y = \mathbf{z}_1^{out}, \quad \mathbf{z}_2^x = \mathbf{r}, \quad \mathbf{z}_2^y = \mathbf{z}_2^{out} - \mathbf{r} \mod p,$$

  and outputs $\mathbf{z}_1^x, \mathbf{z}_2^x, \mathbf{z}_1^y, \mathbf{z}_2^y$, and $\mathsf{tb} = \emptyset$.

- $\mathsf{ACmp.Dec}(\mathbf{L}^x, \mathbf{L}^y, \mathsf{tb})$ simply outputs $\mathbf{L}^{out} = \mathbf{L}^x + \mathbf{L}^y \mod p$.

Correctness. For all $x, y \in \mathcal{I} = \mathbb{Z}_p$, we have

$$\mathbf{L}^x + \mathbf{L}^y = (x\mathbf{z}_1^{out} + \mathbf{r}) + (y\mathbf{z}_1^{out} + \mathbf{z}_2^{out} - \mathbf{r}) = (x+y)\mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p.$$

Construction 10 (multiplication modulo p gadget)**.** We describe the "multiplication modulo p" gadget, with $\mathcal{L} = \mathbb{Z}_p$.

- $\mathsf{ACmp.Garb}(1^\lambda, \mathbf{z}_1^{out}, \mathbf{z}_2^{out})$ The output-wire keys $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$ are dimension ℓ' vectors over $\mathbb{Z}_p$. Generate the input-wire keys as:

$$\mathbf{z}_1^x = (\mathbf{z}_1^{out}, r_3\mathbf{z}_1^{out}) \qquad \mathbf{z}_2^x = (\mathbf{r}_1, r_3\mathbf{r}_1 - \mathbf{r}_2 - \mathbf{z}_2^{out}) \qquad \pmod p$$
$$\mathbf{z}_1^y = (1, \mathbf{r}_1) \qquad \mathbf{z}_2^y = (r_3, \mathbf{r}_2) \qquad \pmod p\ .$$

  where $\mathbf{r}_1, \mathbf{r}_2, r_3$ have uniformly random elements from $\mathbb{Z}_p$. Output $(\mathbf{z}_1^x, \mathbf{z}_2^x), (\mathbf{z}_1^y, \mathbf{z}_2^y)$ and $\mathsf{tb} = \emptyset$.

  Note: *Note that the length of the keys for the first input wire x have doubled, and the length of the keys for second input wire y has increased by 1, compared with the length of the output-wire keys.*

- $\mathsf{ACmp.Dec}(\mathbf{L}^x, \mathbf{L}^y, \mathsf{tb})$ parses $\mathbf{L}^x = (\mathbf{L}_1^x, \mathbf{L}_2^x)$, and $\mathbf{L}^y = (L_1^y, \mathbf{L}_2^y)$, where $\mathbf{L}_1^x, \mathbf{L}_2^x, \mathbf{L}_2^y \in \mathbb{Z}_p^{\ell'}$, $L_1^y \in \mathbb{Z}_p$. It outputs

$$\mathbf{L}^{out} = L_1^y \mathbf{L}_1^x - \mathbf{L}_2^x - \mathbf{L}_2^y \mod p\ . \tag{3.27}$$

Correctness. Given correctly generated input-wire labels for x and y,

$$\mathbf{L}^x = x\mathbf{z}_1^x + \mathbf{z}_2^x = (\underbrace{x\mathbf{z}_1^{out} + \mathbf{r}_1}_{\mathbf{L}_1^x}, \underbrace{xr_3\mathbf{z}_1^{out} + r_3\mathbf{r}_1 - \mathbf{r}_2 - \mathbf{z}_2^{out}}_{\mathbf{L}_2^x}) \mod p,$$

$$\mathbf{L}^y = y\mathbf{z}_1^y + \mathbf{z}_2^y = (\underbrace{y + r_3}_{L_1^y}, \underbrace{y\mathbf{r}_1 + \mathbf{r}_2}_{\mathbf{L}_2^y}), \quad \mod p.$$

the decryption procedure (Equation (3.27)) should compute the correct output label $\mathbf{L}^{out} = \mathbf{z}_1^{out}(x+y) + \mathbf{z}_2^{out}$. This is indeed the case as shown below.

$$\begin{aligned}\mathbf{L}^{out} &= (y+r_3)\big(x\mathbf{z}_1^{out} + \mathbf{r}_1\big) - (xr_3\mathbf{z}_1^{out} + r_3\mathbf{r}_1 - \mathbf{r}_2 - \mathbf{z}_2^{out}) - (y\mathbf{r}_1 + \mathbf{r}_2)\\ &= xy\mathbf{z}_1^{out} + \mathbf{z}_2^{out} \mod p.\end{aligned}$$

Lemma 3.8. *Construction 9, and Construction 10, are secure per Definition 3.2.*

Lemma 3.8. We describe two simulators $\mathsf{ACmp}_{+}.\mathsf{Sim}$ and $\mathsf{ACmp}_{\times}.\mathsf{Sim}$ for Construction 9 and Construction 10 respectively.

$\mathsf{ACmp}_{+}.\mathsf{Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$ samples $\widetilde{\mathbf{L}}^x \leftarrow \mathbb{Z}_p^{\ell'}$ of the same dimension ℓ' as $\mathbf{L}^{out}$, and computes $\widetilde{\mathbf{L}}^y = \mathbf{L}^{out} - \widetilde{\mathbf{L}}^x \mod p$. It outputs $\widetilde{\mathbf{L}}^x, \widetilde{\mathbf{L}}^y$ and $\widetilde{\mathsf{tb}} = \emptyset$.

$\mathsf{ACmp}_{\times}.\mathsf{Sim}(1^\lambda, \mathsf{pp}, \mathbf{L}^{out})$ samples $\widetilde{\mathbf{L}}_1^x, \widetilde{\mathbf{L}}_1^y \leftarrow \mathbb{Z}_p^{\ell'}$, and $\widetilde{L}_1^y \leftarrow \mathbb{Z}_p$. It computes

$$\widetilde{\mathbf{L}}_2^x = \widetilde{L}_1^y \widetilde{\mathbf{L}}_1^x - \widetilde{\mathbf{L}}_2^y - \mathbf{L}^{out} \mod p,$$

and outputs $\widetilde{\mathbf{L}}^x = (\widetilde{\mathbf{L}}_1^x, \widetilde{\mathbf{L}}_2^x)$, $\widetilde{\mathbf{L}}^y = (\widetilde{L}_1^y, \widetilde{\mathbf{L}}_2^y)$, and $\widetilde{\mathsf{tb}} = \emptyset$. □

The Overall Garbling Scheme. We next assembles the gadgets following the AIK paradigm. As in the AIK paradigm, every wire is assigned with a pair of keys $(\mathbf{z}_1, \mathbf{z}_2)$, such that the label of this wire is $\mathbf{L} = x\mathbf{z}_1 + \mathbf{z}_2$ if x is the value of this wire.

- For each output wire, the assigned pair of keys is $\mathbf{z}_1 = 1, \mathbf{z}_2 = 0$, so that the output can be read from the label.
- For each gate, the keys of its input wires is generated by the corresponding computation gadget based on the pair of keys of its output wire. We also use the key-extension gadget to keep the key size small.

We formalized the AIK paradigm in Construction 11, which is garbling scheme for garbling bounded integer computation (resp. modulo-p computation) if all the gadgets are instantiated by the ones constructed in Sec. 3.5 (resp. Sec. 3.7).

Construction 11 (Garbling)**.** The garbling schemes make almost black-box use the gadgets defined in previous sections.

Setup The setup algorithm Setup is constructed in Figure 3.4 (resp. Figure 3.5) for garbling bounded integer computation (resp. modulo-p computation). The setup algorithm outputs the public parameter pp, which specifies the message modulus p. The message modulus p equals the computation modulus when garbling modulo-p computation. When garbling bounded integer computation, the message modulus p is a sufficiently large integer.

Garbling $\mathsf{Garb}^{\mathsf{pp}}(1^\lambda, C)$ takes as input the description of a circuit. Let the circuit be represented following the format in Figure 3.7.

The garbling algorithm enumerates all the gates in the inverse topological order (from output gates to input gates). During the enumeration, the algorithm assigns a table and a pair of keys to the gate, also assigns a pair of keys to the gate's incoming wire(s).

For $t = m, \ldots, 1$, the garbling algorithm performs the following steps:

1. If the t-th gate is an output gate, set $\mathbf{z}_1^t = 1, \mathbf{z}_2^t = 0$ (of dimension 1) as the t-th gate's pair of key. Let (i, t) be the only incoming wire to the t-th gate, set $\mathbf{z}_1^{i,t} = 1, \mathbf{z}_2^{i,t} = 0$ as well. Set the table $\mathsf{tb}^t = \varnothing$ and skip the following steps.
2. Say the t-th gate has fan-out k. Wires $(t, j_1), \ldots, (t, j_k)$ start from the t-th gate. The algorithm concatenates all their pairs of keys into a pair of long keys

$$\mathbf{z}_1^{t,\text{long}} = (\mathbf{z}_1^{t,j_1}, \ldots, \mathbf{z}_1^{t,j_k}), \quad \mathbf{z}_2^{t,\text{long}} = (\mathbf{z}_2^{t,j_1}, \ldots, \mathbf{z}_2^{t,j_k}).$$

runs $\mathsf{KE.KeyGen}$ to generate the pair of keys for this gate

$$\mathbf{z}_1^t, \mathbf{z}_1^t \leftarrow \mathsf{KE.KeyGen}^{\mathsf{pp}}(1^\lambda, 1^\ell),$$

and then runs $\mathsf{KE.Garb}$ to generate the garbling table for this gate

$$\mathsf{tb}^t \leftarrow \mathsf{KE.Garb}^{\mathsf{pp}}(\mathbf{z}_1^{t,\text{long}}, \mathbf{z}_2^{t,\text{long}}, \mathbf{z}_1^t, \mathbf{z}_2^t)\,.$$

3. If the t-th gate is a computation gate, it must have two incoming wires (i_1, t) and (i_2, t). Use the appropriate arithmetic garbling scheme to generate the key pairs for the two incoming wires

$$\mathbf{z}_1^{i_1,t}, \mathbf{z}_2^{i_1,t}, \mathbf{z}_1^{i_2,t}, \mathbf{z}_2^{i_2,t}, \leftarrow \mathsf{ACmp}_{g_t}.\mathsf{Garb}(1^\lambda, \mathbf{z}_1^t, \mathbf{z}_2^t).$$

In the end, the garbling algorithm outputs $\widehat{C} = \{\mathsf{tb}^t\}_{t \in [m]}$ as the garbled circuit. W.l.o.g., we assume the t-th gate is the input gate corresponding to the t-th coordinate of the input, for t no greater than the input length. The algorithm also outputs $\mathbf{z}_1^t, \mathbf{z}_1^t$ as the key pairs for the i-the coordinate of the input.

Evaluation $\mathsf{Eval}^{\mathsf{pp}}(C, \widehat{C}, \{\mathbf{L}^i\}_{i \in [n]})$ takes as input labels $\mathbf{L}^1, \ldots, \mathbf{L}^n$ and a garbled circuit $\widehat{C} = \{\mathsf{tb}^t\}_{t \in [m]}$. The decoding algorithm enumerates all the gates in topological order, and computes the label of each gate.

For $t = 1, \ldots, m$, the decoding algorithm performs the following steps:

1. *Compute the label of the t-th gate.*

   If the t-th gate is an input gate, the label $\mathbf{L}^i$ is given.

   If the t-th gate is a computation gate, let $(i_1, t), (i_2, t)$ be its incoming wires. The label of this gate can be recovered by

$$\mathbf{L}^t \leftarrow \mathsf{ACmp}_{g_t}.\mathsf{Dec}^{\mathsf{pp}}(\mathbf{L}^{i_1,t}, \mathbf{L}^{i_1,t})\,.$$

   If the t-th gate is an output gate, let (i, t) be its only incoming wire, set $\mathbf{L}^t = \mathbf{L}^{i,t}$.

2. *Compute the labels of the wires starts from the t-th gate.*

   If the t-th gate is an input gate or a computation gate, it has a table tb^t that allows key extension operation

$$\mathbf{L}^{t,\mathrm{long}} \leftarrow \mathsf{KE}.\mathsf{Dec}^{\mathsf{pp}}(\mathbf{L}^t, \mathsf{tb}^t)\,.$$

Let $(t, j_1), \ldots, (t, j_k)$ be the wires that starts from the t-th gate. The long label $\mathbf{L}^{t,\text{long}}$ can be parsed as the concatenation of labels of the outgoing wires $(\mathbf{L}^{t,j_1}, \ldots, \mathbf{L}^{t,j_k}) = \mathbf{L}^{t,\text{long}}$.

In the end, for each output gate, the decoding algorithm outputs $\mathbf{L}^t$ (which has dimension 1) as the corresponding coordinate of the circuit output.

Note that the garbling algorithm in Construction 11 is highly parallelizable. The garbling algorithm can 1) first generate all key pairs in parallel; 2) then generate all garbling tables in parallel.

Correctness. The correctness follows almost directly from the correctness of key extension gadget KE and arithmetic computation gadgets $\mathsf{ACmp}_+, \mathsf{ACmp}_\times$. The invariant is that the evaluator gets the label $\mathbf{L} = \mathbf{z}_1 x + \mathbf{z}_2$ for every gate and every wire.

- For each input gate, the label is given to the evaluator.
- For each wire, as long as the label of its starting gate is learnt by the evaluator, the correctness of the key extension gadget KE ensures the evaluator gets the label of the wire.
- For each arithmetic computation gate, as long as the label of its two incoming wires are learnt by the evaluator, the correctness of the corresponding arithmetic computation gadgets ACmp ensures the evaluator gets the label of the gate.
- For each output gate, the label equals that of its incoming wire. The evaluator can decodes the output from the label $\mathbf{L}^t = \mathbf{z}_1^t x + \mathbf{z}_2^t$, since $\mathbf{z}_1^t = 1, \mathbf{z}_2^t = 0$.

Privacy. In the real world, the evaluator gets the label $\mathbf{L} = \mathbf{z}_1 x + \mathbf{z}_2$ for every gate and every wire. The evaluator's view can be simulated in the ideal world, give only the output.

- For each output gate, its label and the label of its preceding wire, equal one coordinate of the output.
- For each computation gate or input gate, given the label of its outgoing wires, $\mathsf{KE.Sim}$ simulates the label of the gate, together with the garbling table of the gate.

- For each computation gate, given the label of the gate, $\mathsf{ACmp.Sim}$ simulates the labels of its preceding wires.

The security of such simulation can be proved by a hybrid argument. Consider hybrid worlds

- Hyb_i: For $t \geq i$, the label and table of the i-th gate is generated in the real world. For $t \geq i$, the label of any wire (j, i) is generated in the real world. The rest of the view is simulated.
- Hyb'_i: For $t \geq i$, the label and table of the i-th gate is generated as the real world. For $t > i$, the label of any wire (j, i) is generated in the real world. The rest of the view is simulated.

Hybrids $\mathsf{Hyb}_{i+1}, \mathsf{Hyb}'_i$ are indistinguishable, due to the security of $\mathsf{KE.Sim}$. Hybrids $\mathsf{Hyb}'_i, \mathsf{Hyb}_i$ are indistinguishable, due to the security of $\mathsf{ACmp.Sim}$, if the i-th gate is a computation. Otherwise, hybrids $\mathsf{Hyb}'_i, \mathsf{Hyb}_i$ are identical.

By definition, Hyb_0 is the real world, and Hyb_{m+1} is the ideal world. So the simulated view is indistinguishable from the real view.

Efficiency Analysis. The bottleneck of the garbling scheme is the key extension gadget. The input label is the key extension gadget input label, which is a dimension ℓ vector in $\mathbb{Z}_p$. Every gate has a garbling table, which is also generated by the key extension gadget. The size of the table is proportional to the gate's fan-out. Thus for efficiency analysis, it is fine to assume every gate has constant fan-out.

First look at the length doubling key extension gadgets in Construction 4, Construction 6, that only hanle output-wire keys $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$ of fixed dimension $\ell' = 2\ell$. They both generate a garbled table consisting of constant number of (concretly, 2 and 4) LHE ciphertexts. In our construction (Construction 1), each LHE ciphertext encrypting an ℓ' dimension vector consists of ℓ' elements in $\mathbb{Z}_P$, where P is the message modulus of the underlying LHE instantiation lhe. Therefore, the length doubling key extension gadgets for both bounded integer and modular computation generate garbled tables with $O(\ell')$ elements in $\mathbb{Z}_P$.

Next consider the arbitrary expansion key extension gadgets, obtained through Transformation. 1. To hanle output-wire keys $\mathbf{z}_1^{out}, \mathbf{z}_2^{out}$ of arbitrary dimension ℓ', the transformation divides them into chunks of fixed dimension 2ℓ, and calls the length doubling gadgets recursively in a tree-fashion. Note that at each level of the tree, the total dimension of the output-wire chunks is reduced by half. Therefore, the total size of a garbled table generated by the transformed gadgets is still $O(\ell')$ elements in $\mathbb{Z}_P$.

Finally, in our garbling scheme, assuming each gate has constant fan-out, we have $\ell' = O(\ell)$, where ℓ is the input-wire label dimension. In Construction 1, we set the $P \geq 2p \cdot \max(B_{\max}, B_e)$, where B_e is a fixed bound on LHE decryption error associated with the scheme lhe, and $p, B_{\max}$ are parameters introduced in the Setup algorithms in Figure 3.4 and Figure 3.5 respectively for bounded integer and modular computation. In both cases, we have

$$\log P = O(\omega(\log \lambda) + \log p + \log B_e). \tag{3.28}$$

Concretely, we have:

- *Garbling B-bounded integer computation.* As specified in Figure 3.4, the label space $\mathcal{L} = \mathbb{Z}_p$ has bit length $\log p = \omega(\log \lambda) + \log B + \log B_s$, where B_s is a fixed bound on LHE key magnitude associated with the scheme $\overline{\mathsf{lhe}}$ and lhe. The input-wire label dimension is $\ell = \ell_s + 1 = O(\ell_s)$, where ℓ_s is the LHE key dimension of $\overline{\mathsf{lhe}}$ and lhe.

  Under the DCR instantiation, we have $\ell_s = 1$, $\log B_s = O(\lambda)$, and $\log B_e = 0$. Therefore, the total bit length of $\widehat{C}$ is

$$\begin{aligned} |\widehat{C}| &= O(|C|\ell' \log P) \\ &= O(|C|\ell_s(\omega(\log \lambda) + \log B + \log B_s + \log B_e)) \\ &= O(|C|(\log B + \lambda)). \end{aligned}$$

  And the bit length of input labels is $O(n\ell \log p) = O(n(\log B + \lambda))$.

  Under the LWE instantiation, we have $\ell_s = k$, where k is the LWE dimension, and

$\log B_s = \log B_e = \omega(\log\lambda) = \widetilde{O}(1)$. Therefore, the total bit length of $\widehat{C}$ is

$$|\widehat{C}| = O(|C|\ell_s(\omega(\log\lambda) + \log B + \log B_s + \log B_e))$$
$$= |C| \cdot \widetilde{O}(k) \cdot \log B.$$

And the bit length of input labels is $O(n\ell \log p) = n \cdot \widetilde{O}(k) \cdot \log B$.

- *Garbling modulo-p computation.* As specified in Figure 3.5, the input-wire label dimension is $\ell = O(\ell_s \ell_{\mathrm{smdg}})$, where ℓ_s is the LHE key dimension of $\overline{\mathsf{lhe}}$ and lhe, and ℓ_{smdg} is the smudging source length set to $\ell_{\mathrm{smdg}} = O(\log B_s + \omega(\log\lambda))$ as specified in Section 3.6.

  Under the DCR instantiation, we have $\ell_s = 1$, $\log B_s = O(\lambda)$, and $\log B_e = 0$. Therefore, we have $\ell_{\mathrm{smdg}} = O(\lambda)$, and the total bit length of $\widehat{C}$ is

$$|\widehat{C}| = O(|C|\ell' \log P)$$
$$= O(|C|\ell_s \ell_{\mathrm{smdg}}(\omega(\log\lambda) + \log p + \log B_e)$$
$$= O(|C|\lambda(\omega(\log\lambda) + \log p).$$

  And the bit length of input labels is $O(n\ell \log p) = O(n\lambda \log p)$.

  Under the LWE instantiation, we have $\ell_s = k$, where k is the LWE dimension, and $\log B_s = \log B_e = \omega(\log\lambda)$. Therefore, we have $\ell_{\mathrm{smdg}} = \omega(\log\lambda) = \widetilde{O}(1)$, and the total bit length of $\widehat{C}$ is

$$|\widehat{C}| = O(|C|\ell_s \ell_{\mathrm{smdg}}(\omega(\log\lambda) + \log p + \log B_e)$$
$$= |C| \cdot \widetilde{O}(k) \cdot \log p.$$

  And the bit length of input labels is $O(n\ell \log p) = n \cdot \widetilde{O}(k) \cdot \log p$.

3.9.2 Mixed Bounded Integer and Boolean computation

The garbling scheme for the mixed bounded integer and Boolean computation model follow the same steps as Construction 11 in Section 3.9.1 for the other two simpler computation models. The only difference is that it has three types of gates to handle (instead of just arithmetic computation gates):

- Arithmetic computation gates are handled using the scheme ACmp from Construction 9 in the same way as Construction 11.
- Bit decomposition gates are handled using the scheme BD from Construction 8.
- To handle a Boolean computation gate, $g_B : \{0,1\}^{d_1} \to \{0,1\}^{d_2}$, we need to compute d_1 input-wire keys $\{\mathbf{z}_1^{in,i}, \mathbf{z}_2^{in,i}\}_{i\in[d_1]}$, where $\mathbf{z}_1^{in,i}, \mathbf{z}_2^{in,i} \in \mathbb{Z}_p^{\ell}$ given d_2 output-wire keys $\{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_{i\in[d_1]}$, where $\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i} \in \mathbb{Z}_p^{\ell'}$. We use a Boolean garbling scheme (e.g. Yao's garbling) Yao. First, we define a circuit C (with $\{\mathbf{z}_1^{out,i}, \mathbf{z}_2^{out,i}\}_i$ hardcoded) that on input $\mathbf{x} \in \{0,1\}^{d_1}$ computes:

$$C(\mathbf{x}) = \{\mathbf{L}^{out,i}\}_{i\in[d_2]}. \left| \begin{array}{c} \mathbf{y} = (y_1, \ldots, y_{d_2}) = g_B(\mathbf{x}) \\ \mathbf{L}^{out,i} = y_i \mathbf{z}_1^{out,i} + \mathbf{z}_2^{out,i} \mod p \end{array} \right.$$

  We then compute a garbled circuit and d_1 pair of evaluation keys $\widehat{C}, \{\mathbf{k}_0^i, \mathbf{k}_1^i\}_{i\in[d_1]} \leftarrow \mathsf{Yao.Garb}(1^\lambda, C)$. The input-wire keys are defined as

$$\mathbf{z}_1^{in,i} = \bar{\mathbf{k}}_1^i - \bar{\mathbf{k}}_0^i \mod p, \quad \mathbf{z}_2^{in,i} = \bar{\mathbf{k}}_0^i,$$

  where $\bar{\mathbf{k}}_b^i$ is a $\mathbb{Z}_p$ vector encoding the Boolean evaluation key $\mathbf{k}_b^i$. It's easy to verify that an evaluator with input-wire labels $\mathbf{L}^{in,i} = \mathbf{z}_1^{in,i} x_i + \mathbf{z}_2^{in,i} = \bar{\mathbf{k}}_{x_i}^i \pmod p$ can evaluate the garbled circuit $\widehat{C}$ to obtain the output-wire labels $\{\mathbf{L}^{out,i}\}_i$.

  Note that The above description gives a Boolean computation gadget.

Efficiency Analysis. Similar to the cases of bounded integer and modular computation (analyzed in the end of Section 3.9), the input label is the key extension gadget input label, which is a dimension ℓ vector in $\mathbb{Z}_p$, where the dimension ℓ and the modulus p is specified in the Setup algorithm in Figure 3.6. And the garbled table generated by the key extension gadget consists of $O(\ell')$ elements in $\mathbb{Z}_P$, where ℓ' is the output-wire label dimension, and P is the message modulus of the underlying LHE instantiation lhe. We have $\ell' = O(\ell)$, and $\log P = O(\omega(\log \lambda) + \log p + \log B_e)$.

Different from the cases of bounded integer and modular computation, the bit decomposition gadget and the Boolean computation gadget also generate garbled tables. In bit

deomposition (Construction 8), a garbled table consists of two parts: a specific Yao's garbled circuit and $2d'$ LHE ciphertexts, where $d' = \omega(\log\lambda) + \log B$, is the bit length of an integer y that statistically smudges any B-bounded value. Since the Yao's garbled circuit is not the bottleneck, we ignore it in this efficiency analysis. The $2d'$ LHE ciphertexts each encrypts an $\ell'' = 2\ell_k + \ell_s\ell_{\text{smdg}}$ dimension vector, where $\ell_k = O(\lambda)$ is the bit length of an evaluation key for Yao's garbled circuit. Therefore, in total the garbled table consists of $O(d'\ell'')$ elements in $\mathbb{Z}_P$.

The Boolean computation gadget for a Boolean function g_B basically outputs the Yao's garbled circuit for the circuit representation of g_B as its garbled table. Therefore, we count the total size of all garbled tables generated by Boolean computation gadgets as $s_b\lambda$, where s_b is the total circuit size of all Boolean functions in the circuit.

Let m_a be the number of arithmetic computation gates, and m_b, the number of bit-decomposition gates. We can now calculate concretely:

- *Garbling mixed B-bounded integer and Boolean computation.* As specified in Figure 3.6, the label space $\mathcal{L} = \mathbb{Z}_p$ has bit length $\log p = \omega(\log\lambda) + \log B + \log B_s$, where B_s is a fixed bound on LHE key magnitude associated with the scheme $\overline{\mathsf{lhe}}$ and lhe. The input-wire label dimension is $\ell = O(\ell_s)$, where ℓ_s is the LHE key dimension of $\overline{\mathsf{lhe}}$ and lhe. The smudging source length ℓ_{smdg} is set to $\ell_{\text{smdg}} = O(\log B_s + \omega(\log\lambda))$ as specified in Construction 7.

  Under the DCR instantiation, we have $\ell_s = 1$, $\log B_s = O(\lambda)$, and $\log B_e = 0$. Therefore, $\ell'' = 2\ell_k + \ell_s\ell_{\text{smdg}} = O(\lambda)$, and the total bit length of $\widehat{C}$, is

$$\begin{aligned}
|\widehat{C}| &= O((m_a\ell' + m_b 2d'\ell'')\log P + s_b\lambda) \\
&= O((m_a + m_b(\underbrace{\omega(\log\lambda) + \log B}_{=d'})\underbrace{\lambda}_{=\ell''})(\underbrace{\lambda + \log B}_{=\log P}) + s_b\lambda) \\
&= O(s_b\lambda + m_a(\lambda + \log B) + m_b(\lambda + \log B)^2\lambda).
\end{aligned}$$

  And the bit length of input labels is $O(n\ell\log p) = O(n(\lambda + \log B))$.

Under the LWE instantiation, we have $\ell_s = k$, where k is the LWE dimension, and $\log B_s = \log B_e = \omega(\log \lambda) = \widetilde{O}(1)$. Therefore, $\ell'' = 2\ell_k + \ell_s \ell_{\text{smdg}} = \widetilde{O}(k+\lambda)$, and the total bit length of $\widehat{C}$ is

$$\begin{aligned}|\widehat{C}| = &O((m_a \ell' + m_b 2d'\ell'')\log P + s_b\lambda)\\ =&s_b O(\lambda) + m_a \cdot \log B \cdot \widetilde{O}(k) + m_b(\log B)^2 \widetilde{O}(k+\lambda).\end{aligned}$$

And the bit length of input labels is $O(n\ell \log p) = O(nk \log p)$.

3.10 *Concrete Efficiency Analysis*

In this section, we compare the concrete efficiency of our garbling scheme based on the DCR assumption against the scheme of [BMR16] (BMR), which garbles arithmetic circuits in the bounded integer model with free addition and subtraction, and the baseline solution that first converts arithmetic circuits into Boolean circuits and then runs the Boolean garbling scheme of [RR21] (RR). Note that, the concrete efficiency of our construction is not optimized. The calculations and comparisons in this section are only to demonstrate the potential towards more practical garbling schemes for garbling arithmetic circuits with large domains.

In Section 3.5.2, we presented a length doubling key extension gadget. It is easy to see that we can adapt the gadget to extend the key length arbitrarily, at the cost of increasing the length of public parameters pp proportional to the length of the expanded key. This can be done in practice when we know a bound on the maximal fan-out of the circuits. In Section 3.5.3, we present another method for achieving "arbitrary expansion", which however increases the size of a garbled table tb by at most twice. Below, we analyze the more efficient variant, assuming the maximal fan-out is known a-priori.

Concrete Setting for Comparison. Concretely, we consider the Paillier modulus N to have 4096 bits. For B-bounded integer garbling, we set the bit length $\ell = \log(B)$ to be just slightly below 4096, specifically 3808 bits (the setting is described below), and for mod-p garbling, we similarly consider bit length $\ell = \log p = 3808$ bits. We set the statistical security parameter $\kappa = 80$.

Under the concrete setting, the most efficient Boolean circuit implementation for integer multiplication uses Karatsuba's method. We conservatively count the number of AND gates (as XOR gates in RR is free) in a multiplication circuit as $\ell^{1.58}$, ignoring any hidden constants, and an addition circuit as $\ell \log \ell$. In mod-p garbling, we assume the baseline solution adds a mod-p reduction (also count as $\ell^{1.58}$) after every integer multiplication or addition.

At a high level, the BMR scheme works by decomposing a large B-bounded integer into its Chinese Remainder Theorem (CRT) representation using the smallest distinct primes ($p_1 = 2, p_2 = 3, \ldots, p_k$) whose product exceeds B. Under the concrete setting, the number of primes is $k = 394$.

To complement the following comparisons under our concrete setting, with $\ell = 3808$ bits, we also plot the garbling size comparisons for a larger range, from $\ell = 500$ to $10,000$ bits, in Figure 3.8a, 3.8b, and 3.8c. Details for the comparisons in bounded integer, mixed computation, and modular arithmetic models are described below.

Size of Bounded Integer Garbling. Under standard DCR, our bounded integer garbling significantly improves the garbling size of both addition (∼100×) and multiplication (∼500×) gates over the Boolean baseline using RR, as shown in Table 3.3. BMR supports free addition, but multiplication is more expensive than RR.

The formula for our scheme is derived as follows. In our garbling scheme, the garbled table for each multiplication gate consists of 12 ring elements in $\mathbb{Z}_P$: according to Figure 3.1, the two input wires each have a pair of keys of dimension 4 and 2 (as the label $x_{j_2} + s$ is available before key extension and does not need to be regenerated). In the DCR instantiation, we set $P = N^3$. Because when $\ell = \log B = 3808$, it holds that $N^2 \geq N 2^{2\kappa} B$, which is how large the values encrypted in Paillier encryption are. Note that this is different from how Setup algorithm (Figure 3.4) specifies the modulus P, because Setup is designed to fit both the DCR and the LWE instantiations. The size of garbling an addition gate is calculated the same way as multiplication, except with key dimensions 2 and 1 for the input wires.

Size of Mixed Computation Garbling. The BMR scheme has a gadget for converting a

Scheme	Garbled Table Size	
Ours (per Mult Gate)	$12 \cdot 3 \cdot \log N$	18.0 KB
[RR21] (per Mult Gate)	$1.5 \cdot 128 \cdot \ell^{1.58}$	10.4 MB
[BMR16] (per Mult Gate)	$2 \cdot 128 \cdot \sum_{i=1}^{k}(p_i - 1)$	15.0 MB
Ours (per $+/-$ Gate)	$6 \cdot 3 \cdot \log N$	9.0 KB
[RR21] (per $+/-$ Gate)	$1.5 \cdot 128 \cdot \ell \log \ell$	1.0 MB
[BMR16] (per $+/-$ Gate)	Free	0 b
Ours, Improved (per Mult Gate)	$12 \cdot 2 \cdot \log N$	12.0 KB
Ours, Improved (per $+/-$ Gate)	$6 \cdot 2 \cdot \log N$	6.0 KB

The last two lines assume the stronger small exponent assumption.

Table 3.3: Comparison of garbled circuit size for bounded integer computation.

B-bounded integer into a representation where each "digit" is relative to a distinct prime base $(2, 3, 5, \ldots, p_k)$. This representation has similar advantages as the Boolean representation, such as cheap comparisons.[9]

Under a strengthened DCR assumption, (we call it "strong DCR",) our bit-decomposition gadget gives a $3\times$ improvement over the similar decomposition gadget in BMR, as shown in Table 3.4. The comparison for addition and multiplication gates remains the same as in the bounded integer model (Table 3.3).

We next explain the formula for our scheme, and the motivation for the "strong DCR" assumption. Our bit decomposition gadget requires the input wire to have a much longer key dimension, $O(\log N + \kappa)$, than addition or multiplication, due to the use of the seeded smudger. The hidden constant is also large. As a preliminary attempt at optimizing the key

[9]A followup work [BCM+19] considered optimizations to related gadgets, e.g. for comparison, using computational search and relaxations that only guaranteed approximate correctness. We do not compare to this later work. Note that the naive implementations of their search problem is infeasible for the parameter setting we are considering in this section.

dimension, we strengthen the DCR assumption in two steps.

1. We adopt the "small exponent assumption" from [ADOS22]. At a high level, the DCR assumption says h^x is indistinguishable from a random group element, where h is a generator of the hard group and x is sampled from $[N]$ at random. The small exponent assumption says the indistinguishability holds even if the exponent x is sampled from a small range, $0, \ldots, 2^{\mathsf{lsk}}$, independent of the Paillier modulus N. For concrete number we set $\mathsf{lsk} = 128$. The key dimension becomes $O(\mathsf{lsk} + \kappa)$ under the small exponent assumption.

   We note that, the small exponent assumption also slightly reduces our garbling sizes for bounded integer garbling, as we can now set the modulus $P = N^2 \geq N2^{\mathsf{lsk}}2^{2\kappa}B$, as shown in the last two lines of Table 3.3.

2. We further strength the assumption to get rid of the seeded smudger. At a high level, smudger linearly combines entropic bits (each with 1/2 bit of entropy) into a secure Paillier secret exponent. Instead, we simply concatenate the entropic bits as $x = \sum_{i \in [2\mathsf{lsk}]} x_i 2^i$, which gives an exponent x with lsk bits of entropy. We strengthen DCR to "strong DCR", which assumes that the indistinguishability holds even if x is sampled as above. The key dimension becomes $2 \cdot \mathsf{lsk}$ under the new assumption.

The bit decomposition gadget itself generates a garbled table $\mathsf{tb}_{\mathsf{BD}}$ with $4 \cdot \mathsf{lsk} \cdot (\ell + \kappa)$ ring elements in $\mathbb{Z}_P$. Additionally, the garbled table tb encrypting one such key pair consists of $4 \cdot \mathsf{lsk}$ ring elements in $\mathbb{Z}_P$, which is negligible compared to the above table $\mathsf{tb}_{\mathsf{BD}}$. In the DCR instantiation, we set $P = N^2 \geq N2^{\mathsf{lsk}}2^{2\kappa}B$.

Size of Modular Arithmetic Garbling. Again, under "strong DCR", our garbling has comparable sizes to the Boolean baseline using RR for both addition and multiplication, as shown in Table 3.5. Note that BMR doesn't support mod-p garbling for an arbitrary modulus p, hence is not included for comparison.

The formula for our scheme is derived as follows. In the modular arithmetic model, the key dimension of an input wire to a multiplication gate is calculated similarly to the above for

Scheme	Garbled Table Size	
Our Bit-Decomp	$4 \cdot \mathsf{lsk} \cdot (\ell + \kappa) \cdot 2 \cdot \log N$	1.9 GB
[BMR16] Decomp	$128 \cdot 2(k-1) \sum_{i=1}^{k}(p_i - 1) + 128 \cdot 4\binom{k}{2}$	5.8 GB

Our scheme assumes the “strong DCR” assumption.

Table 3.4: Comparison of garbled circuit size supporting bit decomposition.

a bit decomposition gate. The difference is that the use of secret sharing in the key extension gadget (Construction 6) doubles the key dimension to $4 \cdot \mathsf{lsk}$. The two input wires to a Mult gate each have a pair of keys of dimension $8 \cdot \mathsf{lsk}$ and $4 \cdot \mathsf{lsk}$, as shown in Figure 3.1. The garbled table tb encrypting two such key pairs consists of $2 \cdot (8 \cdot 2 + 4 \cdot 2) \cdot \mathsf{lsk} = 48 \cdot \mathsf{lsk}$ ring elements in $\mathbb{Z}_P$, where the additional factor of 2 comes again from the use of secret sharing in the key extension gadget. In the DCR instantiation, we set $P = N^4 \geq p^3 2^\kappa N$. (Recall in the concrete setting $\ell = \log(p) = 3808$ bits.)

Scheme	Garbled Table Size	
Ours (per Mult Gate)	$48 \cdot \mathsf{lsk} \cdot 4 \cdot \log N$	12 MB
[RR21] (per Mult Gate)	$2 \cdot 1.5 \cdot 128 \cdot \ell^{1.58}$	20.8 MB
Ours (per +/− Gate)	$32 \cdot \mathsf{lsk} \cdot 4 \cdot \log N$	8 MB
[RR21] (per +/− Gate)	$1.5 \cdot 128 \cdot (\ell^{1.58} + \ell \log \ell)$	11.4 MB

Our scheme assumes the “strong DCR” assumption.

Table 3.5: Comparison of arithmetic garbling for modular arithmetic computation.

Computation Efficiency. We briefly analyze the main computation costs of our scheme, and compare with the scheme of BMR and RR, focusing on bounded integer garbling under our concrete setting.

Scheme	Garbling Computation Cost	
Ours (per Mult Gate)	$12 \cdot \mathsf{lsk} \approx 1.5 \times 10^3$	Mult mod P
[RR21] (per Mult Gate)	$1.5 \cdot \ell^{1.58} \approx 6.8 \times 10^5$	AES calls
[BMR16] (per Mult Gate)	$2 \cdot \sum_{i=1}^{k}(p_i - 1) \approx 10^6$	AES calls
Ours (per $+/-$ Gate)	$6 \cdot \mathsf{lsk} \approx 7.7 \times 10^2$	Mult mod P
[RR21] (per $+/-$ Gate)	$1.5 \cdot \ell \log \ell \approx 6.8 \times 10^4$	AES calls
[BMR16] (per $+/-$ Gate)	Free	Free

Table 3.6: Comparison of computation costs for bounded integer garbling.

In both BMR and RR, the main costs are computing garbled table entries, which are 128-bit AES ciphertexts. Concretely: BMR computes $2 \cdot \sum_{i=1}^{k}(p_i - 1) \approx 10^6$ AES ciphertexts for each Mult gate, and has free addition. The Boolean baseline using RR computes $1.5 \cdot \ell^{1.58} \approx 6.8 \times 10^5$ and $1.5 \cdot \ell \log \ell \approx 6.8 \times 10^4$ AES ciphertexts for each Mult and Add gate respectively. In our scheme, a garbled table for Mult consists of 12 ring elements in $\mathbb{Z}_P$, each a Paillier ciphertext of the form $h^x(1+N)^m$, for some hard group element h, secret exponent x, and message m. Thanks to algebraic properties of Pailler, $(1+N)^m$ can be computed cheaply without exponentiation. The main cost comes from raising h to the exponent x. Let lsk be the bit length of x. The DCR assumption assumes that $\mathsf{lsk} = \log N = 4096$. However, under the "small exponent" assumption as introduced in [ADOS22], we can set $\mathsf{lsk} = 128$, which significantly improves computational efficiency. Concretely, our scheme computes $12 \cdot \mathsf{lsk} \approx 1.5 \times 10^3$ and $6 \cdot \mathsf{lsk} \approx 7.7 \times 10^2$ multiplications mod P for each Mult and Add gate respectively. A comparison is in Table 3.6.

Arithmetic Operation Gadgets

Gadget for Addition $x_i = x_{j_1} + x_{j_2}$**:** At garbling time, given a pair of keys $(\mathbf{k}_0^i, \mathbf{k}_1^i)$ for the output wire i, it produces a pair of keys $(\mathbf{k}_0^{j_1}, \mathbf{k}_1^{j_1})$ and $(\mathbf{k}_0^{j_2}, \mathbf{k}_1^{j_2})$ for each input wire (and no garbled table) as follows:

$$\text{Set } \mathbf{k}_0^{j_1} = \mathbf{k}_0^{j_2} = \mathbf{k}_0^i \qquad \text{Sample additive shares } \mathbf{k}_1^{j_1} + \mathbf{k}_1^{j_2} = \mathbf{k}_1^i\,.$$

At evaluation time, the output label can be obtained as follows

$$\mathbf{L}^{j_1} + \mathbf{L}^{j_2} = (\mathbf{k}_0^{j_1} x_{j_1} + \mathbf{k}_1^{j_1}) + (\mathbf{k}_0^{j_2} x_{j_2} + \mathbf{k}_1^{j_2}) = \mathbf{k}_0^i (x_{j_1} + x_{j_2}) + \mathbf{k}_1^i = \mathbf{L}^i\ .$$

Gadget for Multiplication $x_i = x_{j_1} \times x_{j_2}$**:** At garbling time, given output keys $(\mathbf{k}_0^i, \mathbf{k}_1^i)$, it produces input key pairs $(\mathbf{k}_0^{j_1}, \mathbf{k}_1^{j_1})$ and $(\mathbf{k}_0^{j_2}, \mathbf{k}_1^{j_2})$ (and no garbled table) as follows:

$$\mathbf{k}_0^{j_1} := (\mathbf{k}_0^i, s\mathbf{k}_0^i),\ \mathbf{k}_1^{j_1} := (\mathbf{r}, \mathbf{u})\,, \qquad \mathbf{k}_0^{j_2} := (1, \mathbf{r}),\ \mathbf{k}_1^{j_2} := (s, s\mathbf{r} - \mathbf{k}_1^i - \mathbf{u})\,.$$

where s is a random scalar and $\mathbf{r}, \mathbf{u}$ are random vectors.

At evaluation time, given input labels $\mathbf{L}^{j_1} = (\mathbf{k}_0^i x_{j_1} + \mathbf{r}, s\mathbf{k}_0^i x_{j_1} + \mathbf{u})$ and $\mathbf{L}^{j_2} = (x_{j_2} + s, \mathbf{r}x_{j_2} + s\mathbf{r} - \mathbf{k}_1^i - \mathbf{u})$, the output label can be obtained as follows:

$$\mathbf{L}^i = \mathbf{L}_{\text{left}}^{j_1} \mathbf{L}_{\text{left}}^{j_2} - \mathbf{L}_{\text{right}}^{j_1} - \mathbf{L}_{\text{right}}^{j_2}\ .$$

Figure 3.1: AIK arithmetic operation gadgets.

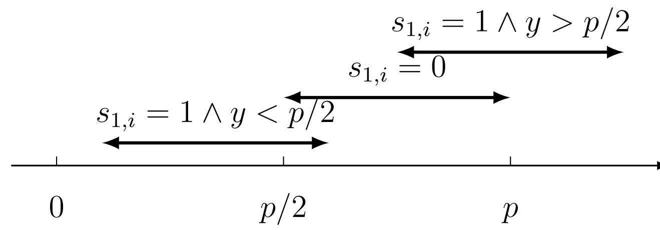


Figure 3.2: The range of $s'_{\mathrm{res},i}$, conditioning on $s_{1,i}$ and y

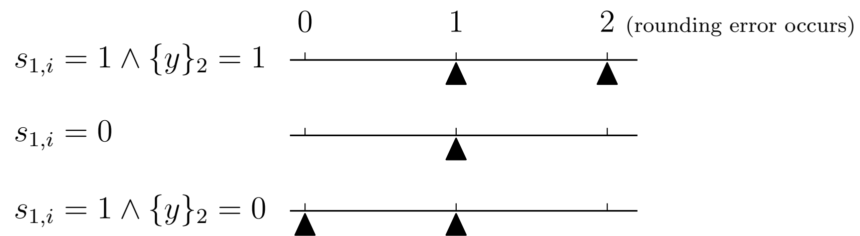


Figure 3.3: The range of $\{s_{1,j}y\}_2 + \{s_{2,j}\}_2$, conditioning on $s_{1,i}$ and $\{y\}_2$

Setup Algorithm of Bounded Integer Garbling

Parameters and Tools: The computation is B-bounded. The construction uses the scheme $\overline{\mathsf{lhe}}$ from Construction 1, which is associated with a bound B_s on the infinity norm of LHE keys sampled by $\overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}$, and a bound B_e on the decryption noise of the scheme lhe underlying $\overline{\mathsf{lhe}}$. All of B, B_s, and B_e are bounded by $2^{\mathrm{poly}(\lambda)}$.

$\mathsf{Setup}(1^\lambda)$ invokes the setup algorithm of the $\overline{\mathsf{lhe}}$ scheme

$$\overline{\mathsf{pp}} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max}),$$

and outputs $\mathsf{pp} = (\overline{\mathsf{pp}}, \ell)$, where the parameters are set as below.

- Parameters of the $\overline{\mathsf{lhe}}$ scheme (with key dimension $\ell_s = \mathrm{poly}(\lambda, \log B_{\max})$):

message modulus	$p = \lambda^{\omega(1)} B \cdot B_s$	(3.4)
smudging noise level	$\alpha = \lambda^{\omega(1)} \max(p, B_e)^2$	(3.5)
maximal noise level	$B_{\max} = p(p + 1 + \alpha + 2B_e)$	
message dimension bound	$\Psi = 2(\ell_s + 1) = 2\ell.$	

- The dimension of keys/labels of the key extension gadget is set to $\ell = \ell_s + 1$.

Figure 3.4: Setup for bounded integer garbling.

Setup Algorithm of Modular Arithmetic Garbling

Parameters and Tools: The computation is modular arithmetic over $\mathbb{Z}_p$. The construction uses two ingredients:

- the scheme $\overline{\mathsf{lhe}}$ from Construction 1, which is associated with a bound B_s on the infinity norm of LHE keys sampled by $\overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}$, and a bound B_e on the decryption noise of the scheme lhe underlying $\overline{\mathsf{lhe}}$.
- a linear seeded $(\ell_{\text{smdg}}, 1/4, \lambda_1, \lambda_2)$-smudger scheme Smudge from Theorem 3.4, which is able to smudge any distribution over $\{0, 2^{\lambda_1}\}$ with $O(2^{-\lambda_2})$ statistical distance, using a $(\ell_{\text{smdg}}, 1/4)$-bit-fixing source.

$\mathsf{Setup}(1^\lambda)$ invokes the setup algorithm of the $\overline{\mathsf{lhe}}$ scheme

$$\overline{\mathsf{pp}} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max}),$$

and outputs $\mathsf{pp} = (\overline{\mathsf{pp}}, \ell)$, where the parameters are set as below.

- Parameters of the $\overline{\mathsf{lhe}}$ scheme (with key dimension $\ell_s = \text{poly}(\lambda, \log B_{\max})$):

$$\begin{aligned} &\text{message modulus} && p = \text{ the modulus of the computation} \\ &\text{smudging noise level} && \alpha = \lambda^{\omega(1)} \max(p, B_e)^4 \\ &\text{maximal noise level} && B_{\max} = p(p + 1 + \alpha + 2B_e) \\ &\text{message dimension bound} && \Psi = 2\ell_s \ell_{\text{smdg}} \end{aligned} \tag{3.11}$$

- The dimension of keys/labels associated with input wires of key extension gate is set to $\ell = \ell_s \ell_{\text{smdg}} + 1$.

Figure 3.5: Setup for modular arithmetic garbling.

Setup Algorithm of Mixed Bounded Integer and Boolean Computation

Parameters and Tools: The computation is B-bounded. The construction uses three ingredients: (All of p, B_s, and B_e are bounded by $2^{\text{poly}(\lambda)}$.)

- the scheme $\overline{\mathsf{lhe}}$ from Construction 1, which is associated with a bound B_s on the infinity norm of LHE keys sampled by $\overline{\mathsf{lhe}}.\overline{\mathsf{KeyGen}}$, and a bound B_e on the decryption noise of the scheme lhe underlying $\overline{\mathsf{lhe}}$;
- a linear seeded $(\ell_{\text{smdg}}, 1/4, \lambda_1, \lambda_2)$-smudger scheme Smudge from Theorem 3.4, which is able to smudge any distribution over $\{0, 2^{\lambda_1}\}$ with $O(2^{-\lambda_2})$ statistical distance, using a $(\ell_{\text{smdg}}, 1/4)$-bit-fixing source.
- a garbling scheme for Boolean circuits Yao, which is associated with a bound $\ell_k = O(\lambda)$ on the bit length of a evaluation key.

$\mathsf{Setup}(1^\lambda)$ invokes the setup algorithm of the $\overline{\mathsf{lhe}}$ scheme

$$\overline{\mathsf{pp}} \leftarrow \overline{\mathsf{lhe}}.\overline{\mathsf{Setup}}(1^\lambda, 1^\Psi, p, B_{\max}),$$

and outputs $\mathsf{pp} = (\overline{\mathsf{pp}}, \ell)$, where the parameters are set as below.

- Parameters of the $\overline{\mathsf{lhe}}$ scheme (with key dimension $\ell_s = \text{poly}(\lambda, \log B_{\max})$):

$$\text{message modulus} \qquad p = \lambda^{\omega(1)} B \cdot B_s \tag{3.14}$$

$$\text{smudging noise level} \qquad \alpha = \lambda^{\omega(1)} \max(p, B_e)^4 \tag{3.15}$$

$$\text{maximal noise level} \qquad B_{\max} = p(p + 1 + \alpha + 2B_e)$$

$$\text{message dimension bound} \qquad \Psi = 2(\ell_s \ell_{\text{smdg}} + \ell_k)$$

- The dimension of keys/labels associated with input wires of key extension gate is set to $\ell = \ell_s + 1$.

Figure 3.6: Setup for mixed bounded integer and Boolean computation garbling.

A circuit is formalize as a DAG.

- Each node represents a gate. For notation simplicity, we define input gates and output gates. Let m be the number of gates. The gates are labeled by $[m]$.

  For each $i \in [m]$, the i-th node belongs to one of the following types

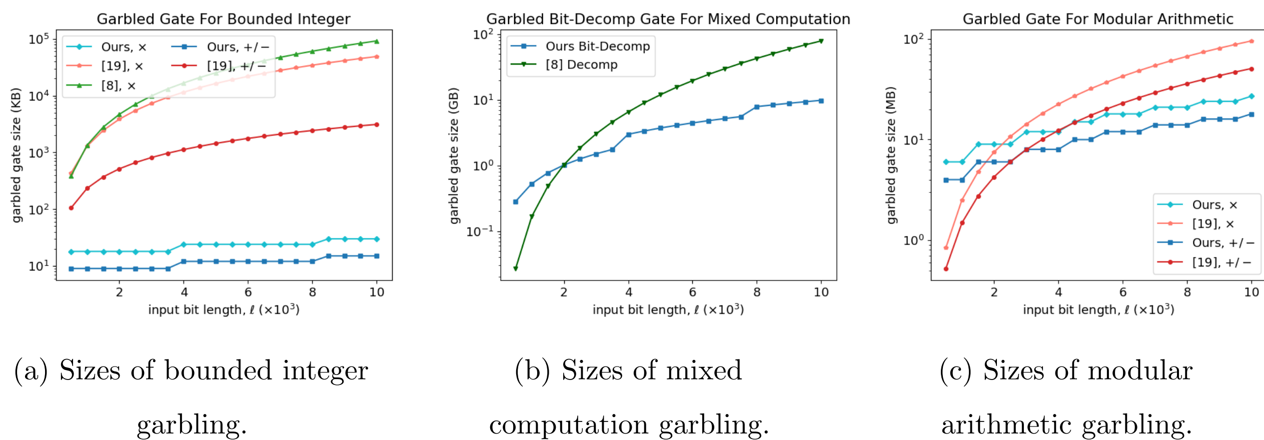
  Input Gate An input gate has no fan-in. The value of the gate equals to the corresponding input of the circuit.

  Computation Gate A computation has an additional attribute specifying its operation. The value of the gate equals the outcome of the operation on the values of its preceding nodes.

  Output Gate An output gate has no fan-out and has fan-in 1. The value of the gate equals the value of its preceding node.

- Wires are presented by directed edges. The value of a wire equals the value of its starting node. W.l.o.g. the nodes are sorted by topological order, so that for every wire, its starting node has a smaller index than its ending node.

Figure 3.7: Formalization of arithmetic circuits.



(a) Sizes of bounded integer garbling.

(b) Sizes of mixed computation garbling.

(c) Sizes of modular arithmetic garbling.

We assume the "small exponent" assumption in the first model, and "strong DCR" in the last two.

Figure 3.8: Garbling sizes in bounded integer, mixed, and modular arithmetic models.

Chapter 4

NEW WAYS TO GARBLE MIXED CIRCUITS

This chapter is adapted from published work [LL24].

4.1 Introduction

Garbled circuit (GC) is introduced in the seminal work of Yao [Yao82], allowing a *garbler* to efficiently transform any Boolean circuit $C : \{0,1\}^{n_{\text{in}}} \to \{0,1\}^{n_{\text{out}}}$ into a *garbled circuit* $\tilde{C}$, along with n_{in} keys $\mathsf{K}_1, \dots, \mathsf{K}_{n_{\text{in}}}$. Each key is a function $\mathsf{K}_i : \{0,1\} \to \{0,1\}^\lambda$, mapping the i-th input bit to a short string. The output of K_i is referred to as the *label* of the i-th input wire. For any (unknown) input x, the garbled circuit $\tilde{C}$ together with input labels $\mathsf{K}_1(x_1), \dots, \mathsf{K}_{n_{\text{in}}}(x_{n_{\text{in}}})$ reveal $C(x)$ but nothing else about x.

GC was originally motivated by the 2-party secure computation problem. Since then, GC has found applications to a large variety of problems, and is recognized as one of the most successful and fundamental tools in cryptography.

For practical applications, people care about the efficiency of GC, especially the communication complexity (i.e., bit length of $\tilde{C}$). A considerable amount of works [BMR90, NPS99, KS08a, PSSW09, KMR14, GLNP18, ZRE15, RR21] have been dedicated to optimize the *concrete* efficiency of Yao's GC construction. In the most recent construction of Rosulek and Roy [RR21], XOR and NOT gates involves no communication, every fan-in-2 AND gate requires $1.5\lambda + 5$ bits of communication. Despite making concrete analytic improvement, they still largely follow Yao's construction, binding tightly with Boolean circuits. The class of arithmetic operations is a featuring example of computations that are expensive to express as Boolean circuits.

The arithmetic setting. The beautiful work of Applebaum, Ishai, and Kushilevitz [AIK11]

initiated the study of garbling arithmetic circuits.

> *Arithmetic GC* over a ring $\mathcal{R}$ is an efficient algorithm that transforms an arithmetic circuit $C : \mathcal{R}^{n_{\text{in}}} \to \mathcal{R}^{n_{\text{out}}}$ into a garbled circuit $\tilde{C}$, along with n_{in} keys $\mathsf{AK}_1, \ldots, \mathsf{AK}_{n_{\text{in}}}$. Each key is an affine function $\mathsf{AK}_i : \mathcal{R} \to \mathcal{R}^{\ell}$. For any (unknown) input x, the garbled circuit $\tilde{C}$ together with input labels $\mathsf{AK}_1(x_1), \ldots, \mathsf{AK}_{n_{\text{in}}}(x_{n_{\text{in}}})$ reveal $C(x)$ but nothing else about x.

The construction of AIK is a natural generalization of Yao's Boolean GC. For each wire, a key $\mathsf{AK} : \mathcal{R} \to \mathcal{R}^{\ell}$ is sampled. The output of AK is called the label of that wire, whose length is roughly the security parameter. For any arithmetic gate g, say $\mathsf{AK}_1, \mathsf{AK}_2$ are the keys of the two input wires and AK is the key of the output wire, the garbler generates a table Tab of this gate, such that for any (unknown) $x, y \in \mathcal{R}$, the evaluator can compute $\mathsf{AK}(g(x, y))$ from $\mathsf{AK}_1(x), \mathsf{AK}_2(y), \mathsf{Tab}$, while learning no other information.

As observed by [AIK11], to keep the table size for each gate constant, it suffices to construct the so-called *key-extension*[1] gadget. Such a gadget consists of a garbling algorithm and an evaluation algorithm. The garbling algorithm $\mathsf{KE.Garb}$ takes a key AK and a long key AK^{L} as input, samples a key-extension table Tab such that, $\mathsf{AK}(x), \mathsf{Tab}$ reveal $\mathsf{AK}^{\mathsf{L}}(x)$ but nothing else about $x, \mathsf{AK}^{\mathsf{L}}$.

[AIK11] presents two constructions of key-extension gadgets. One relies on Chinese remainder theorem, enables garbling of mod-$p_1 p_2 \ldots p_k$ (the product of distinct small primes) computation. The other is based on LWE, supports bounded integer computation (computation over the integer ring $\mathbb{Z}$ when all intermediate values are guaranteed to be bounded).

Follow-up research has made improvements within this framework. Similar to FreeXOR, [BMR16] allows free garbling of addition gates. In a different frontier, [BLLL23] presents a highly efficient arithmetic GC for bounded integer computation based on Paillier encryption. [BLLL23] also presents arithmetic GC for $\mathbb{Z}_p$ based on LWE or Paillier. However, free addition

[1] This module is called "key shrinking" in [AIK11]. The name "key extension" comes from [BLLL23].

is not supported in [BLLL23]. The communication complexity of existing arithmetic GC constructions will be discussed in more detail in Sec. 4.1.1.

Our research proceeds with this line of study within AIK's framework of arithmetic GC. Our starting point is to understand *how to garble mod-2^b arithmetic circuits,* which is not efficiently supported by previous works. In the search for mod-2^b GC, we realize that it is has a few advantage over GC for mod-p or bounded integer computation.

Match popular architectures. In most modern architectures, if not all, the only natively supported arithmetic operation is over $\mathbb{Z}_{2^b}$. Most existing tools (programming languages, compilers, processors, etc) are optimized using/targeting the mod-2^b arithmetic operations. This is our initial motivation to construct the mod-2^b GC.

Mixing Boolean and arithmetic computation. Mixed circuits combine Boolean and arithmetic computations. The basis include Boolean gates, arithmetic operations, together with special gates to convert between Boolean and arithmetic values: arithmetic-to-Boolean conversion (bit-decomposition) and Boolean-to-arithmetic conversion (bit-composition). Previous work [BMR16, BLLL23] has considered the garbling of mixed circuits. But in their constructions, the cost of garbling bit-decomposition is expensive.

It turns out that our mod-2^b GC naturally supports *efficient* garbling of bit-decomposition and bit-composition. In fact, in our construction, the key-extension gadget is the combination of bit-decomposition and bit-composition. For example, to double the arithmetic key/label length, first bit-decompose it into Boolean labels, then use bit-composition twice to obtain a longer label.

Emulate arithmetic computation modulo any modulus N.

For any constant N, mod-N computations can be efficiently emulated by mod-2^{4b} mixed circuits if $b = \lceil \log N \rceil$. To prove such a statement, it suffices to show, given $0 \le x < N^2$, how to compute $x \bmod N$ using a mod-2^{4b} mixed circuit. One step further, it is also sufficient to compute integer division $\lfloor x/N \rfloor$ using a mod-2^{4b} mixed circuit. By the rather standard

multiply-and-shift trick

$$\lfloor x \cdot \lceil 2^{3b}/N \rceil / 2^{3b} \rfloor = \lfloor x/N \rfloor,$$

the quotient can be computed by first multiplying by constant $\lceil 2^{3b}/N \rceil$ then integer division by 2^{3b}. Both operations are efficient in a mod-2^{4b} mixed circuit.

4.1.1 Our Results

Mixed GC in the Random Oracle Model. Using only random oracle, the state-of-the-art garbling scheme for arithmetic circuit is that of [BMR16]. They rely on Chinese remainder theorem (CRT) to garble an arithmetic circuit modulo $N = p_1, \dots p_s \approx 2^b$, by equivalently garbling s copies of the circuit, each modulo a small prime p_i. They allow free addition and each multiplication gate costs $O(\lambda b^2/\log b)$ bits of communication. However, bit-decomposition operation of this scheme is expensive and not explicitly considered in [BMR16].

Our work improves the state-of-the-art in several directions.

- Our first scheme (Theorem 4.1) garbles arithmetic gates modulo $N = p^k \approx 2^b$, for some prime p, with the same asymptotic efficiency as [BMR16]: addition is free, each multiplication costs $O(\lambda b^2/\log b)$ bits of communication. Additionally, our scheme supports efficient bit-decomposition gates at a cost of $O(\lambda b^2/\log b)$ communication, enabling the garbling of mixed circuits.
- Our second scheme (Theorem 4.2) applies CRT in a similar way to [BMR16]. When garbling computations modulo $N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$, our mixed GC supports free addition and relatively efficient bit-decomposition, and garbles every multiplication gate using $O(\lambda b^{1.5})$ communication.
- Our third scheme (Theorem 4.3) further improves the multiplication gate cost to $O(\lambda b \log b/\log\log b)$. However, as a trade-off, addition gates are no longer free and have the same cost as multiplication gates.

		ADD gate table size	MULT gate table size	bit decom-position	ring modulus	assumption besides RO
Boolean	naive	λb	λb^2	free	2^b	
	Karatsuba	λb	$\lambda b^{1.58}$ *	free	2^b	
	FFT-based	λb	$\lambda b \log b$ *	free	2^b	
[BMR16]		free	$\lambda b^2/\log b$	expensive †	$N = p_1 p_2 \dots p_s \approx 2^b$	
Ours (Thm. 4.1)		free	$\lambda b^2/\log b$	$\lambda b^2/\log b$ ‡	$N = p^k \approx 2^b$	
Ours (Lem. 4.6)		$\lambda b^2/\log b$	$\lambda b^2/\log b$	$\lambda b^2/\log b$ ‡	any $N \approx 2^b$	
Ours (Thm. 4.2)		free	$\lambda b^{1.5}$	$\lambda b^2/\log b$ ‡	$N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$	
Ours (Thm. 4.3)		$\frac{\lambda b \log b}{\log \log b}$	$\frac{\lambda b \log b}{\log \log b}$	$\lambda b^2/\log b$ ‡	$N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$	
[BLLL23]		$\lambda_{\mathsf{LWE}} b$	$\lambda_{\mathsf{LWE}} b$	unknown	any $N \approx 2^b$	LWE
[BLLL23]		$\lambda(\lambda_{\mathsf{DCR}} + b)$	$\lambda(\lambda_{\mathsf{DCR}} + b)$	unknown	any $N \approx 2^b$	strong DCR §
[BLLL23]		$\lambda_{\mathsf{LWE}} b$	$\lambda_{\mathsf{LWE}} b$	$\lambda_{\mathsf{LWE}} b^2$	bounded integer	LWE
[BLLL23]		$\lambda_{\mathsf{DCR}} + b$	$\lambda_{\mathsf{DCR}} + b$	$\lambda(\lambda_{\mathsf{DCR}} + b)^2$	bounded integer	strong DCR §
Ours (Thm. 4.4)		free	$\lambda_{\mathsf{DCR}} b$	$\lambda_{\mathsf{DCR}} b$	2^b	DCR
Ours (Cor. 4.1)		$\lambda_{\mathsf{DCR}} b$	$\lambda_{\mathsf{DCR}} b$	$\lambda_{\mathsf{DCR}} b$	any $N \approx 2^b$	DCR

Constant and $\log(\lambda)$ multiplicative factors are ignored. λ_{LWE} and λ_{DCR} denote the LWE dimension and DCR key length respectively. *Due to large hidden constants, the Karatsuba's method outperforms the naive method only when b is at least a few hundreds, the FFT-base method outperforms Karatsuba's only when b is at least tens of thousands. †The cost is not explicitly stated in [BMR16], but is no less the the cost of comparison gate, which is stated to be $O(\lambda b^3/\log b)$. ‡The cost is measured when decomposing to base-p bit representation for some prime p (See Equation 4.2). The cost increases to $O(\lambda b^2)$ when decomposing to base-2 bit representation. §Under the standard DCR assumption, "λ" should be replaced by "λ_{DCR}" in its cost expression.

Table 4.1: Comparison between our GC and previous works.

Mixed GC Based on Computational Assumptions. If allowed to use public key assumptions, the state-of-the-art garbling schemes for arithmetic circuits and mixed circuits are those of [BLLL23]. Under the decisional composite residuosity (DCR) assumption, they construct a garbling scheme for bounded integers where each multiplication gate only costs $O(\lambda_{\mathsf{DCR}} + b)$. In their scheme, the addition gates cost the same as multiplication, the bit-decomposition gates have a more expensive cost of $O(\lambda_{\mathsf{DCR}}^2 \cdot b)$.

Our work improves the state-of-the-art by supporting free addition gates and more efficient bit-decomposition gates. However, as a trade-off, multiplication gates are more expensive, of size $O(\lambda_{\mathsf{DCR}} \cdot b)$.

- Our fourth scheme (Theorem 4.4) garbles mixed circuits modulo 2^b and allows free addition. Each multiplication gate and bit-decomposition gate costs $O(\lambda_{\mathsf{DCR}} \cdot b)$ communication.

4.2 Technical Overview

This section briefly discusses AIK's framework of arithmetic GC (Sec. 4.2.1) and a technically less interesting extension (Sec. 4.2.2) discussing the sufficiency of bit-decomposition and bit-composition. The takeaway is: Mixed circuits can be efficiently garbled, as long as there are efficient garbling gadgets for bit-decomposition and bit-composition.

In Sec. 4.2.3, we presents a naive construction of the two garbling gadgets. The resulting GC does not have superior efficiency, but it is simple enough and will be optimized in later sections.

4.2.1 Background: Key-Extension Implies Arithmetic GC

We recap the framework of AIK [AIK11] for arithmetic GC over some ring $\mathcal{R}$, with the modification that there is a global key $\mathbf{\Delta}$ for all arithmetic wires. As observed by FreeXOR [KS08a] and "FreeADD" [BMR16], the garbling of addition gates will cost no communication if a global key is sampled.

In more detail, an arithmetic key is sampled for each wire as follows (where λ denotes the security parameter):

- A global key $\mathbf{\Delta} \in \mathcal{R}^\ell$ is sampled for all arithmetic wires, where ℓ is the label length. If $\mathcal{R} = \mathbb{Z}_{2^b}$, we will set $\ell = \lambda$. If $\mathcal{R} = \mathbb{Z}_{p^k}$, we will set $\ell = \lceil \lambda / \log p \rceil$.
  For each arithmetic wire, the key is an affine function $\mathsf{AK} : \mathcal{R} \to \mathcal{R}^{\ell+1}$. The output $\mathsf{AK}(x)$ consists of ℓ-dimension label and a *color number*. That is, AK can be represented by $\mathsf{AK} = (\mathbf{A} \in \mathcal{R}^\ell, \alpha \in \mathcal{R})$ such that

$$\mathsf{AK}(x) = (\mathbf{\Delta} x + \mathbf{A}, x + \alpha) \quad (\text{in } \mathcal{R}) .$$

  α is called the mask number of this wire. Set $\alpha = 0$ for every output wire.

The circuit is garbled gate-by-gate. The garbling gadget for arithmetic gate g consists of a garbling algorithm $g.\mathsf{Garb}$, an evaluation algorithm $g.\mathsf{Eval}$ and a simulation algorithm $g.\mathsf{Sim}$. The garbling algorithm $g.\mathsf{Garb}$ takes the keys of input wires $\mathsf{AK}_1, \mathsf{AK}_2$ and a key of output wire AK, outputs a table Tab such that:

- *Correctness.* For any $x, y \in \mathcal{R}$, $g.\mathsf{Eval}(\mathsf{AK}_1(x), \mathsf{AK}_2(y), \mathsf{Tab}) = \mathsf{AK}(g(x, y))$.
- *Handwavy Security.* For any $x, y \in \mathcal{R}$, the distribution of Tab is indistinguishable from $g.\mathsf{Sim}(\mathsf{AK}_1(x), \mathsf{AK}_2(y), \mathsf{AK}(g(x, y)))$ when $\mathsf{AK}_1(x), \mathsf{AK}_2(y)$ are also given to the distinguisher but the global arithmetic key $\mathbf{\Delta}$ is hidden.

If g is addition, note that

$$\begin{aligned} &\mathsf{AK}_1(x) + \mathsf{AK}_2(y) - \mathsf{AK}(x + y) \\ &= (\mathbf{\Delta} x + \mathbf{A}_1, x + \alpha_1) + (\mathbf{\Delta} y + \mathbf{A}_2, y + \alpha_2) - (\mathbf{\Delta}(x + y) + \mathbf{A}, x + y + \alpha) \\ &= (\mathbf{A}_1 + \mathbf{A}_2 - \mathbf{A}, \alpha_1 + \alpha_2 - \alpha) \quad (\text{in } \mathbb{Z}_{2^d}) \end{aligned}$$

is a constant that can always be determined by the input/output labels. Setting it as the table will not violate security and is sufficient for correctness. A smarter solution, as suggested by [BMR16], is to set the table Tab to be *empty*, and to change how the output wire key AK

is generated. Instead of sampling AK at random, set $\mathbf{A} = \mathbf{A}_1 + \mathbf{A}_2$ and $\alpha = \alpha_1 + \alpha_2$, thus $\mathsf{AK}_1(x) + \mathsf{AK}_2(y) \bmod 2^d = \mathsf{AK}(x + y)$.

If g is multiplication, first use *randomized encoding* [IK00, AIK04] to sample two affine functions (long keys) $\mathsf{AK}_1^{\mathsf{L}}, \mathsf{AK}_2^{\mathsf{L}}$ such that $\mathsf{AK}_1^{\mathsf{L}}(x), \mathsf{AK}_2^{\mathsf{L}}(y)$ reveals $\mathsf{AK}(xy)$ but nothing else about x, y, AK. This is formalized as a so-called *affinization gadget* in [AIK11] (called "arithmetic operation gadgets" in [BLLL23]).

The affinization gadget for multiplication can be formalized by a garbling algorithm $\mathsf{Aff}_\times.\mathsf{Garb}$, an evaluation algorithm $\mathsf{Aff}_\times.\mathsf{Eval}$ and a simulation algorithm $\mathsf{Aff}_\times.\mathsf{Sim}$.

- Given an affine function, the garbling algorithm $\mathsf{Aff}_\times.\mathsf{Garb}(\mathsf{AK})$ samples two affine functions $\mathsf{AK}_1^{\mathsf{L}}, \mathsf{AK}_2^{\mathsf{L}}$ such that the output dimension of $\mathsf{AK}_i^{\mathsf{L}}$ is at most twice the output dimension of AK. (The multiplicative factors of $\mathsf{AK}_1^{\mathsf{L}}, \mathsf{AK}_2^{\mathsf{L}}$ are not neccessarily the global $\mathbf{\Delta}$. We represent a "long key" as $\mathsf{AK}^{\mathsf{L}} = (\mathbf{A}, \mathbf{B})$ such that $\mathsf{AK}^{\mathsf{L}}(x) = \mathbf{A}x + \mathbf{B}$.)
- *Correctness.* For any x, y in the ring, given "long labels", the evaluation algorithm $\mathsf{Aff}_\times.\mathsf{Eval}(\mathsf{AK}_1^{\mathsf{L}}(x), \mathsf{AK}_2^{\mathsf{L}}(y))$ always outputs $\mathsf{AK}(xy)$.
- *Security.* For any AK, x, y, the distribution of $(\mathsf{AK}_1^{\mathsf{L}}(x), \mathsf{AK}_2^{\mathsf{L}}(y))$ is perfectly indistinguishable from $\mathsf{Aff}_\times.\mathsf{Sim}(\mathsf{AK}(xy))$. The randomness of the former comes from the randomness tape of $\mathsf{Aff}_\times.\mathsf{Garb}$.

The construction of GC is complete by the *key-extension* gadget, which allows the evaluator to compute $\mathsf{AK}_1^{\mathsf{L}}(x), \mathsf{AK}_2^{\mathsf{L}}(y)$ from $\mathsf{AK}_1(x), \mathsf{AK}_2(y)$.

The key-extension gadget can be formalized by three efficient algorithms $\mathsf{KE.Garb}$, $\mathsf{KE.Eval}$, $\mathsf{KE.Sim}$.

- Given a key AK and an affine function AK^{L}, the garbling algorithm $\mathsf{KE.Garb}(\mathsf{AK}, \mathsf{AK}^{\mathsf{L}})$ samples a table Tab.
- *Correctness.* For any x in the ring, $\mathsf{KE.Eval}(\mathsf{AK}(x), \mathsf{Tab}) = \mathsf{AK}^{\mathsf{L}}(x)$.

- *Handwavy Security.* For any x, the distribution of Tab is indistinguishable from $\mathsf{KE.Sim}(\mathsf{AK}(x), \mathsf{AK}^{\mathsf{L}}(x))$ when $\mathsf{AK}(x)$ are also given to the distinguisher but $\boldsymbol{\Delta}$ is hidden.

The garbling gadget for multiplication gate can be naturally constructed from the affinization gadget and the key-extension gadget.

- Garbling algorithm $\mathsf{Mult.Garb}(\mathsf{AK}_1, \mathsf{AK}_2, \mathsf{AK})$:
  $\mathsf{Aff}_\times.\mathsf{Garb}(\mathsf{AK}) \to (\mathsf{AK}_1^{\mathsf{L}}, \mathsf{AK}_2^{\mathsf{L}})$.
  $\mathsf{KE.Garb}(\mathsf{AK}_i, \mathsf{AK}_i^{\mathsf{L}}) \to \mathsf{Tab}_i$ for $i \in \{1, 2\}$.
  Output $\mathsf{Tab} = (\mathsf{Tab}_1, \mathsf{Tab}_2)$.
- Evaluation algorithm $\mathsf{Mult.Eval}(\mathbf{L}_1, \mathbf{L}_2, \mathsf{Tab})$:
  $\mathsf{KE.Eval}(\mathbf{L}_i, \mathsf{Tab}_i) \to \mathbf{L}_i^{\mathsf{L}}$ for $i \in \{1, 2\}$.
  $\mathsf{Aff}_\times.\mathsf{Eval}(\mathbf{L}_1^{\mathsf{L}}, \mathbf{L}_2^{\mathsf{L}}) \to \mathbf{L}$.
  Output $\mathbf{L}$.
- Simulation algorithm $\mathsf{Mult.Sim}(\mathbf{L}_1, \mathbf{L}_2, \mathbf{L})$:
  $\mathsf{Aff}_\times.\mathsf{Sim}(\mathbf{L}) \to \mathbf{L}_1^{\mathsf{L}}, \mathbf{L}_2^{\mathsf{L}}$.
  $\mathsf{KE.Sim}(\mathbf{L}_i, \mathbf{L}_i^{\mathsf{L}}) \to \mathsf{Tab}_i$ for $i \in \{1, 2\}$.
  Output $\mathsf{Tab} = (\mathsf{Tab}_1, \mathsf{Tab}_2)$.

This arithmetic GC framework [AIK11, BMR16] reduces the problem to constructing a key-extension gadget. As long as there is a secure key-extension gadget that doubles the key length (i.e., the output of AK^{L} can be twice as long as AK), the framework will yield an arithmetic GC of the same complexity.

Lemma 4.1 (informal)**.** *If there is a secure key-extension gadget that doubles the key length whose table size is c_{KE}, there is an arithmetic GC for the same ring such that each addition gate costs no communication, and each multiplication gate costs $2 \cdot c_{\mathrm{KE}}$ communication.*

4.2.2 Bit-Decomposition & Bit-Composition Imply Mixed GC

We extend the AIK framework to support mixed circuit, which consists of arithmetic operation gates as described before, boolean gates such as AND, XOR, and NOT, and two conversion gates, bit-decomposition and bit-compositions.

A wire in the circuit is either an arithmetic wires as described before, or a boolean wire. The keys for arithmetic wires stay unchanged. The keys for boolean wires are sampled as follows:

- A global key $\Delta \in \{0,1\}^\lambda$ is sampled for all boolean wires.

  For each boolean wire, the key is an affine function $\mathsf{K} : \{0,1\} \to \{0,1\}^{\lambda+1}$. The output $\mathsf{K}(x)$ consists of a λ-bit label and a color bit. That is, K can be represented by $\mathsf{K} = (\mathbf{b} \in \{0,1\}^\lambda, \alpha \in \{0,1\})$ such that

  $$\mathsf{K}(x) = (\Delta x \oplus \mathbf{b}, x \oplus \alpha)\,.$$

  α is called the mask bit of this wire. Set $\alpha = 0$ for every output wire.

The arithmetic operation gates are garbled as before, and we skip the rather standard boolean gate garbling gadgets. We describe gadgets for garbling bit-decomposition and bit-composition gates in more detail below.

The bit-decomposition gadget consists of $\mathsf{BD.Garb}, \mathsf{BD.Eval}, \mathsf{BD.Sim}$. The garbling algorithm $\mathsf{BD.Garb}$ takes an arithmetic key AK and b boolean keys $\mathsf{K}_0, \ldots, \mathsf{K}_{b-1}$ as inputs, outputs a table Tab, such that

- *Correctness.* For any $x \in \mathcal{R}$, $\mathsf{BD.Eval}(\mathsf{AK}(x), \mathsf{Tab}) = (\mathsf{K}_0(x_0), \ldots, \mathsf{K}_{b-1}(x_{b-1}))$.
- *Handwavy Security.* For any $x \in \mathcal{R}$, the distribution of $\mathsf{AK}(x), \mathsf{Tab}$ is indistinguishable from $\mathsf{AK}(x), \mathsf{BD.Sim}(\mathsf{AK}(x), \mathsf{K}_0(x_0), \ldots, \mathsf{K}_{b-1}(x_{b-1}))$ when the global arithmetic key $\boldsymbol{\Delta}$ is hidden.

The bit-composition gadget consists of $\mathsf{BC.Garb}, \mathsf{BC.Eval}, \mathsf{BC.Sim}$. The garbling algorithm $\mathsf{BC.Garb}$ takes b boolean keys $\mathsf{K}_0, \ldots, \mathsf{K}_{b-1}$ and an arithmetic affine function AK^L as inputs, outputs a table Tab, such that

- *Correctness.* For any $x \in \mathcal{R}$, $\mathsf{BC.Eval}(\mathsf{K}_0(x_0), \ldots, \mathsf{K}_{b-1}(x_{b-1}), \mathsf{Tab}) = \mathsf{AK}^{\mathsf{L}}(x)$.
- *Handwavy Security.* For any $x \in \mathcal{R}$, the distribution of Tab is indistinguishable from $\mathsf{BC.Sim}(\mathsf{K}_0(x_0), \ldots, \mathsf{K}_{b-1}(x_{b-1}), \mathsf{AK}^{\mathsf{L}}(x))$ when $\mathsf{K}_0(x_0), \ldots, \mathsf{K}_{b-1}(x_{b-1})$ is also given to the adversary but the global key Δ is hidden.

We stress that AK^{L} can be an arbitrary affine function: its multiplicative factor does not have to be the global key; and its output dimension can be larger. Although for simplicity, we assume the output dimension of AK^{L} equals the dimension of a label. In case we need longer AK^{L}, we can always divide it into a few pieces and use the bit-composition gadget multiple times.

It is obvious that bit-decomposition gadget and bit-composition gadget imply key-extension gadget, and thus imply mixed GC. Previous work did not construct the key-extension gadget through this approach because bit-decomposition is expensive in their constructions.

Lemma 4.2 (informal)**.** *If there are a secure bit-decomposition gadget whose table size is c_{BD} and a secure bit-composition gadget whose table size is c_{BC}, then there there is a mixed GC for the same ring such that each addition gate costs no communication, and each multiplication/bit-decomposition/bit-composition gate costs $O(c_{\mathrm{BD}} + c_{\mathrm{BC}})$ communication.*

4.2.3 The Naive Construction

This section presents garbling gadgets for bit-decomposition and bit-composition when the ring is $\mathbb{Z}_{2^b}$. For each $x \in \mathbb{Z}_{2^b}$, let x_i denote the i-th lowest bit of x, so that $x = \sum_i 2^i x_i$. Let $x_{a:b}$ denote $\sum_{a \le i < b} 2^{i-a} x_i$, so that the bit representation of $x_{a:b}$ is a substring of the bit representation of x.

BC. The bit-composition gadget is straight-forward. Given boolean input labels $\mathsf{K}_0(x_0), \ldots,$ $\mathsf{K}_{b-1}(x_{b-1})$, the evaluator need to compute the output label $\mathsf{AK}^{\mathsf{L}}(x) = \mathbf{A}x + \mathbf{B}$ (recall that in bit-composition gadget, the output key can be any affine function). The garbling algorithm $\mathsf{BC.Garb}$ samples additive sharing $\mathbf{B}_0, \ldots, \mathbf{B}_{b-1}$ such that $\sum_i \mathbf{B}_i = \mathbf{B}$, then generates table that allows the evaluator to compute $\mathbf{A}2^i x_i + \mathbf{B}_i$ from $\mathsf{K}_i(x_i)$. The most direct solution is to

Garbling algorithm $\mathsf{BD.Garb}$ takes an arithmetic key $\mathsf{AK} = (\mathbf{A}, \alpha)$ and b boolean keys $\mathsf{K}_0, \ldots, \mathsf{K}_{b-1}$ as inputs.

- Let $\mathbf{A}^{(0)} = \mathbf{A}$. For each $1 \le i < b$, samples $\mathbf{A}^{(i)} \leftarrow (\mathbb{Z}_{2^{b-i}})^\lambda$.
- Let $\alpha^{(0)} = \alpha$. For each $1 \le i < b$, samples $\alpha^{(i)} \leftarrow \mathbb{Z}_{2^{b-i}}$.
- For each $0 \le i < b$, for each $\beta \in \{0, 1\}$, compute

$$\mathsf{C}_{i,\beta+\alpha^{(i)} \bmod 2} \leftarrow \mathsf{H}(\mathbf{\Delta}\beta + \mathbf{A}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus \begin{cases} (\mathsf{K}_i(\beta), \mathbf{\Delta}\beta + \mathbf{A}^{(i)} - 2\mathbf{A}^{(i+1)} \bmod 2^{b-i}, \\ \qquad \beta + \alpha^{(i)} - 2\alpha^{(i+1)} \bmod 2^{b-i}) & \text{if } i < b-1 \\ \mathsf{K}_i(\beta), & \text{if } i = b-1 \end{cases}$$

- Output table $\mathsf{Tab} = (\mathsf{C}_{i,\beta})_{i \in [b], \beta \in \{0,1\}}$

Figure 4.1: The naive bit-decomposition gadget.

let the table contain ciphertexts

$$\mathsf{Enc}(\mathsf{K}_i(\beta), \mathbf{A}2^i\beta + \mathbf{B}_i) \text{ for all } \beta \in \{0, 1\}.$$

The order of the two ciphertexts are permuted according to the mask bit in K_i, so that the evaluator can pick the right ciphertext using the color bit.

BD. The bit decomposition gadget is inspired by the following two observations.

- Let $\mathbf{L} = \mathsf{AK}(x) = \mathbf{\Delta}x + \mathbf{A}$ denote the given arithmetic label. Then

$$\mathbf{L} \bmod 2 = \mathbf{\Delta}x + \mathbf{A} \bmod 2 = \mathbf{\Delta}x_0 + \mathbf{A} \bmod 2\,.$$

  If the table contains $\mathsf{Enc}(\mathbf{\Delta}\beta + \mathbf{A} \bmod 2, \mathsf{K}_0(\beta))$ for $\beta \in \{0, 1\}$, the evaluator can properly decrypt the boolean label $\mathsf{K}_0(x_0)$ of x_0 with $\mathbf{L} \bmod 2$.

Evaluation algorithm $\mathsf{BD.Eval}$ takes input label $(\mathbf{L}, \bar{x})$ and a table Tab as inputs.

- Let $\mathbf{L}^{(0)} := \mathbf{L}$, $\bar{x}^{(0)} = \bar{x}$.
- For $i = 0, 1, 2, \ldots, b-1$:

  Compute $(\mathbf{l}_i, \mathbf{D}^{(i)}, d^{(i)}) \leftarrow \mathsf{H}(\mathbf{L}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus \mathsf{C}_{i, \bar{x}^{(i)} \bmod 2}$. If $i < b-1$, compute

$$\mathbf{L}^{(i+1)} = (\mathbf{L}^{(i)} - \mathbf{D}^{(i)} \bmod 2^{b-i})/2\,, \qquad \bar{x}^{(i+1)} = (\bar{x}^{(i)} - d^{(i)} \bmod 2^{b-i})/2\,.$$

- Output boolean labels $\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l}_{b-1}$.

Simulation algorithm $\mathsf{BD.Sim}$ takes arithmetic label $(\mathbf{L}, \bar{x})$ and boolean labels $\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l}_{b-1}$ as inputs.

- Let $(\mathbf{L}^{(0)}, \bar{x}^{(0)}) = (\mathbf{L}, \bar{x})$.
- Sample random $\mathbf{L}^{(i)} \leftarrow (\mathbb{Z}_{2^{b-i}})^\lambda, \bar{x}^{(i)} \leftarrow \mathbb{Z}_{2^{b-i}}$ for each $1 \le i < b$.
- The active ciphertexts in the table Tab are set as

$$\mathsf{C}_{i, \bar{x}^{(i)} \bmod 2} = \mathsf{H}(\mathbf{L}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus \begin{cases} (\mathbf{l}_i, \mathbf{L}^{(i)} - 2\mathbf{L}^{(i+1)} \bmod 2^{b-i},\ \bar{x}^{(i)} - 2\bar{x}^{(i+1)} \bmod 2^{b-i}) & \text{if } i < b-1 \\ \mathbf{l}_i & \text{if } i = b-1 \end{cases}$$

  The rest are called inactive ciphertexts, and are simulated as random strings.

Figure 4.2: The naive bit-decompositio gadget, continued.

- To continue, the evaluator should be able to compute a mod-2^{b-1} arithmetic label for all but the least significant bit of x

$$\mathbf{L}^{(1)} = \mathbf{\Delta} x_{1:b} + \mathbf{A}^{(1)} \bmod 2^{b-1} .$$

Then the evaluator can iteratively compute all the boolean labels. Note that,

$$\mathbf{L} - 2\mathbf{L}^{(1)} \bmod 2^{b} = \mathbf{\Delta} x_0 + \mathbf{A} - 2\mathbf{A}^{(1)} \bmod 2^{b} . \tag{4.1}$$

If the table also contains ciphertexts

$$\mathsf{Enc}(\mathbf{\Delta}\beta + \mathbf{A} \bmod 2,\ \mathbf{\Delta}\beta + \mathbf{A} - 2\mathbf{A}^{(1)} \bmod 2^{b}) \text{ for } \beta \in \{0,1\} ,$$

the evaluator can decrypt the ciphertext to get (4.1) and compute $\mathbf{L}^{(1)}$.

These observations lead us to the bit-decomposition gadget in Fig. 4.1, 4.2. For simplicity, the encryption is implemented by a secure function H which is modeled as a random oracle

$$\mathsf{Enc}(key, m) = \mathsf{H}(key, \mathsf{aux}) \oplus m, \quad \mathsf{Dec}(key, c) = \mathsf{H}(key, \mathsf{aux}) \oplus c,$$

where aux contains auxiliary information such as the id of current gate. The H queries under some auxiliary information is bounded: For each aux, the construction only queries $\mathsf{H}(key, \mathsf{aux})$ for up to two distinct key.

Lemma 4.3. *There are statistically secure bit-decomposition gadget (Fig. 4.1, 4.2) and bit-composition gadget (a specialization of Fig. 4.3) for ring $\mathbb{Z}_{2^b}$, whose table size is $O(b^2\lambda)$. They yield statistically secure mixed GC for $\mathbb{Z}_{2^b}$ in the random oracle model, where each addition gate costs no communication, and each multiplication/bit-decomposition/bit-composition gate costs $O(b^2\lambda)$ communication.*

Proof. The correctness can be easily verified inductively. The induction hypothesis is $\mathbf{L}^{(i)} = \mathbf{\Delta} x_{i:b} + \mathbf{A}^{(i)} \bmod 2^{b-i}, \bar{x}^{(i)} = x_{i:b} + \alpha^{(i)} \bmod 2^{b-i}$. The base case $i = 0$ holds automatically. Assume the inductive hypothesis holds for $i < b - 1$,

$$\begin{aligned}(\mathbf{l}_i, \mathbf{D}^{(i)}, d^{(i)}) &= \mathsf{H}(\mathbf{L}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus \mathsf{C}_{i, \bar{x}^{(i)} \bmod 2} \\ &= \mathsf{H}(\mathbf{\Delta} x_i + \mathbf{A}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus \mathsf{C}_{i, x_i + \alpha^{(i)} \bmod 2} \\ &= (\mathsf{K}_i(x_i), \mathbf{\Delta} x_i + \mathbf{A}^{(i)} - 2\mathbf{A}^{(i+1)} \bmod 2^{b-i},\ x_i + \alpha^{(i)} - 2\alpha^{(i+1)} \bmod 2^{b-i})\end{aligned}$$

Then the hypothesis also holds for $i+1$ as

$$\begin{aligned}\mathbf{L}^{(i+1)} &= (\mathbf{L}^{(i)} - \mathbf{D}^{(i)} \bmod 2^{b-i})/2 \\ &= ((\boldsymbol{\Delta} x_{i:b} + \mathbf{A}^{(i)}) - (\boldsymbol{\Delta} x_i + \mathbf{A}^{(i)} - 2\mathbf{A}^{(i+1)}) \bmod 2^{b-i})/2 \\ &= (2\boldsymbol{\Delta} x_{i+1:b} + 2\mathbf{A}^{(i+1)} \bmod 2^{b-i})/2 \\ &= \boldsymbol{\Delta} x_{i+1:b} + \mathbf{A}^{(i+1)} \bmod 2^{b-i-1},\end{aligned}$$

similarly $\bar{x}^{(i+1)} = x_{i+1:b} + \alpha^{(i+1)} \mod 2^{b-i-1}$.

For the (handwavy) security, the simulation output is indistinguishable from the real-world distribution as

- In the real world, $\mathbf{L}^{(0)}, \ldots, \mathbf{L}^{(b-1)}, \bar{x}^{(0)}, \ldots, \bar{x}^{b-1}$ are padded by $B^{(0)}, \ldots, B^{(b-1)}, \alpha^{(0)}, \ldots, \alpha^{(b-1)}$, their joint distribution is uniformly random. Thus it is correct to simulate them uniformly at random.
- In the real world, for each $i \in [b]$, the active ciphertext $\mathsf{C}_{i,\bar{x}^{(i)} \bmod 2}$ is uniquely determined by $\mathbf{l}_i, \mathbf{L}^{(i)}, \mathbf{L}^{(i+1)}, \bar{x}^{(i)}, \bar{x}^{(i+1)}$ as stated in the simulator's description. (Otherwise, correctness will be violated.)
- In the real world, for each $i \in [b]$, the inactive ciphertext $\mathsf{C}_{i,\bar{x}^{(i)}+1 \bmod 2}$ is one-time padded by $\mathsf{H}(\mathbf{L}^{(i)} + \boldsymbol{\Delta} \bmod 2, (\mathsf{id}, i))$. As long as Δ is hidden from the distinguisher, the ciphertext can be simulated by a random string. □

Precise Security. As mentioned, the precise security proof is inherently global. We provide a sketch of the proof. Consider the standard global simulator Sim:

- Sim takes circuit C and the circuit output as inputs.
- Sample a random label (including the color bit/number) for each wire. For every output wire, the color bit/number is determined by the given circuit output.
- For each gate, use the corresponding garbling gadget to simulate the table of the gate.

The standard global simulator Sim defines the ideal world. We want to show the indistinguishability between the real world and the ideal world.

Define a hybrid world as follows, the only difference between the hybrid and the ideal world is how the inactive ciphertexts are sampled. (In the proof sketch, we only states the modification of $\mathsf{BD.Sim}$. Similar modifications of simulation algorithm are need in the gadgets for bit-composition and boolean gates.) In the hybrid world, an inactive ciphertext $\mathsf{C}_{i,\bar{x}^{(i)}+1 \bmod 2}$ is not simulated by a random string, instead, its value is set as

$$\mathsf{C}_{i,\bar{x}^{(i)}+1 \bmod 2} = \mathsf{H}(\mathbf{L}^{(i)} + \mathbf{\Delta} \bmod 2,\ (\mathsf{id}, i)) \oplus \begin{cases} (\mathsf{K}_i(x_i \oplus 1), \mathbf{\Delta}(x_i \oplus 1) + \mathbf{A}^{(i)} - 2\mathbf{A}^{(i+1)} \bmod 2^{b-i}, \\ \qquad (x_i \oplus 1) + \alpha^{(i)} - 2\alpha^{(i+1)} \bmod 2^{b-i}) & \text{if } i < b-1 \\ \mathsf{K}_i(x_i \oplus 1) & \text{if } i = b-1 \end{cases}$$

This assignment requires knowing keys, etc., which do not exist in the ideal world. In the hybrid world, however, the actual value x is known. The hybrid world also samples global keys $\mathbf{\Delta}, \delta$, and determines the keys K, AK in reverse from the labels, the actual values and global keys.

It is easy to check that the hybrid world is perfectly indistinguishable from the real world. The indistinguishability between the hybrid world and the ideal world can be shown using the *randomness mapping* technique from [HKT11]. First, we can safely assume the distinguisher to be (non-uniform) deterministic. With overwhelming probability, the distinguisher in the ideal world never queries $\mathsf{H}(\mathbf{L}^{(i)} + \mathbf{\Delta} \bmod 2,\ (\mathsf{id}, i))$. (Technically speaking, this statement is wrong because $\mathbf{\Delta}$ does not exist in the ideal world. To fix it, let the ideal world internally sample global keys.) The randomness mapping π is an injective partial function from the randomness space of the ideal world to the randomness space of the hybrid world, satisfying:

i) π is defined on an overwhelming-probability subset of the randomness space. In our case, π is defined over all samples r in the randomness space, such that the distinguisher never queries $\mathsf{H}(\mathbf{L}^{(i)} + \mathbf{\Delta} \bmod 2,\ (\mathsf{id}, i))$ when r is the randomness of the ideal world.

ii) For all samples r in the support of π, the probability of r in the ideal world is the same as the probability of $\pi(r)$ in the hybrid world.

iii) For all samples r in the support of π, the view of the distinguisher in the ideal world when the randomness is r is identical to the view of the distinguisher in the hybrid world when the randomness is $\pi(r)$.

The way we define the hybrid world hints how to construct the randomness mapping π, though we will not explicitly states the randomness mapping in the paper. The randomness mapping shows that the hybrid world and the ideal world are statistically indistinguishable if the distinguisher makes a bounded number of queries to the random oracle.

4.3 Preliminaries

In this chapter, we will focus on constructing a mixed garbling (Definition 2.5) scheme between Boolean nad arithmetics over $\mathbb{Z}_N$, for some modulus N.

4.3.1 Notations for Labels in Garbling Schemes

Our construction of mixed garbling schemes will have different label spaces $\mathcal{L}, \mathcal{L}'$ for Boolean and arithmetic wire values. To emphasize the differences, we write K, AK to denote key functions for Boolean and arithmetic wires respectively, and write $\mathbf{l}, \mathbf{L}$ to denote Boolean and arithmetic labels (which are results of applying key functions on wire values) respectively.

4.3.2 Definition of Garbling Gadgets.

As in the previous chapter, our garbling scheme garbles a circuit in a gate-by-gate fashion, where different types of gates are handled by different garbling gadgets. Furthermore, as explained in the technical overview a pair of bit-decomposition (BD) and bit-composition (BC) gadget suffices to derive all other required gadgets. Therefore, we will focus on constructing BD and BC gadgets in the rest of this chapter.

We recall details on the BD gadget below. (The BC gadget is analogous and omitted.) The garbling gadget for BD consists of three efficient algorithms $\mathsf{BD.Garb}, \mathsf{BD.Eval}, \mathsf{BD.Sim}$. The

garbling algorithm $\mathsf{BD.Garb}$ takes an arithmetic wire AK and b Boolean wire keys $\mathsf{K}_0, \dots, \mathsf{K}_{b-1}$ as input, and generates a table Tab, such that:

- *Correctness.* For any $x \in \mathcal{R}$, $\mathsf{BD.Eval}(\mathsf{AK}(x), \mathsf{Tab}) = (\mathsf{K}_0(x_0), \dots, \mathsf{K}_{b-1}(x_{b-1}))$.
- *Handwavy Security.* For any $x \in \mathcal{R}$, the distribution of $\mathsf{AK}(x), \mathsf{Tab}$ is indistinguishable from $\mathsf{AK}(x), \mathsf{BD.Sim}(\mathsf{AK}(x), \mathsf{K}_0(x_0), \dots, \mathsf{K}_{b-1}(x_{b-1}))$.

As the name suggested, this security definition is imprecise. The issue is mainly caused by the global key. It can be formalized by a global simulator. The global simulator first samples a label for each wire, then samples the garbling table of each gate using the simulation algorithm of the corresponding gadget. In short, the simulation is modular, but the actual security definition is global. For simplicity, we will work in the random oracle model.[2]

Note that the modular approach from the previous chapter, that allows precise security definitions of each gate garbling gadget, is incompatible with the existence of the global key. The modular approach requires the simulation algorithm of the gate gadget to sample labels on the input wires. This causes another issue that a label can not be reused by multiple gates. Thus extra work is required when a gate has fan-out greater than 1.

4.3.3 Base-p Digit Representation and Bit Representation.

For any $x \in [2^b]$, the *bit representation* of x is the unique boolean vector $(x_0, \dots, x_{b-1}) \in \{0,1\}^n$ such that $x = \sum_i 2^i x_i$. For any $x \in [p^k]$, the *base-p digit representation* of x, is the unique vector $(x_0, \dots, x_{k-1}) \in [p]^k$ such that $x = \sum_i p^i x_i$.

For any $x \in [p^k]$, let $(x_0, \dots, x_{k-1})$ be its base-p digit decomposition, the *base-p bit representation* of x is the unique vector

$$\mathsf{BD}(x) = (x_{i,j})_{i \in [k], j \in \lceil \log p \rceil} \quad \text{s.t. } x = \sum_i p^i \sum_j 2^j x_{i,j}. \tag{4.2}$$

That is, base-p bit representation is the bit representation of the base-p digit decomposition.

[2]In the boolean GC setting, [App16] shows how random oracle can be replaced with symmetric encryption resisting a combined related-key and key-dependent message attack. Their technique are likely to work in the arithmetic GC setting as well.

4.4 Mixed GC for $\mathbb{Z}_{p^k}$ in Random Oracle Model (ROM)

This section presents a mix GC for $\mathbb{Z}_{p^k}$. Recall how the arithmetic key, label, color number are defined for each arithmetic wire (where λ is the security parameter):

- A global key $\mathbf{\Delta} \in \mathbb{Z}_{p^k}^{\ell}$ is sampled for all arithmetic wires, where $\ell = \lceil \lambda / \log p \rceil$ is the label length.

  For each arithmetic wire, the key is an affine function $\mathsf{AK} : \mathbb{Z}_{p^k} \to \mathbb{Z}_{p^k}^{\ell+1}$. The output $\mathsf{AK}(x)$ consists of ℓ-dimension label and a *color number*. That is, AK can be represented by $\mathsf{AK} = (\mathbf{A} \in \mathbb{Z}_{p^k}^{\ell}, \alpha \in \mathbb{Z}_{p^k})$ such that

  $$\mathsf{AK}(x) = (\mathbf{\Delta}x + \mathbf{A}, x + \alpha) \mod p^k .$$

  α is called the mask number of this wire. Set $\alpha = 0$ for every output wire.

As discussed in Sec. 4.2.2, it suffices to construct efficient garbling gadgets for bit-decomposition and bit-composition over ring $\mathbb{Z}_{p^k}$. The construction of the two gadgets for $\mathbb{Z}_{p^k}$ generalizes the constructions for $\mathbb{Z}_{2^b}$ in Sec. 4.2.3.

For each $x \in \mathbb{Z}_{p^k}$, let x_i denote the i-th lowest digit of x, so that $x = \sum_i p^i x_i$. Let $x_{a:b}$ denote $\sum_{a \le i < b} p^{i-a} x_i$, so that the base-$p$ digit representation of $x_{a:b}$ is a substring of the base-p digit representation of x. Let $x_{i,j}$ denote the j-th lowest bit of x_i, so that $x_i = \sum_j 2^j x_{i,j}$.

For each $\beta \in \mathbb{Z}_p$, let β_i denote the i-th lowest bit of β, so that $\beta = \sum_i 2^i \beta_i$. Let $\beta_{a:b}$ denote $\sum_{a \le i < b} 2^{i-a} \beta_i$, so that the bit representation of $\beta_{a:b}$ is a substring of the bit representation of β.

BC. The bit-composition gadget is straight-forward. Given boolean input labels $\mathsf{K}_{i,j}(x_{i,j})$ for $i \in [k], j \in [\log p]$, the evaluator needs to compute the output label $\mathsf{AK}^{\mathsf{L}}(x) = \mathbf{A}x + \mathbf{B}$ (recall that in the bit-composition gadget, the output key can be any affine function). The garbling algorithm $\mathsf{BC.Garb}$ samples additive sharing $\mathbf{B}_{i,j}$ such that $\sum_{i,j} \mathbf{B}_{i,j} = \mathbf{B}$, then generates a table that allows the evaluator to compute $\mathbf{A}p^i 2^j x_{i,j} + \mathbf{B}_{i,j}$ from $\mathsf{K}_{i,j}(x_{i,j})$. The most direct solution is to let the table contain ciphertexts

$$\mathsf{Enc}(\mathsf{K}_{i,j}(\beta), \mathbf{A}p^i 2^j \beta + \mathbf{B}_{i,j}) \text{ for all } \beta \in \{0, 1\}.$$

The order of the two ciphertexts are permuted according to the mask bit in $\mathsf{K}_{i,j}$, so that the evaluator can pick the right ciphertext according to the color bit.

The construction is formalized in Fig. 4.3. The table consists of $O(k \log p)$ ciphertexts, each ciphertext is $k\lambda$-bit long, thus the table size is $O(\lambda k^2 \log p)$ bit.

BD. The bit-decomposition gadget starts with the same observations as the one in Sec. 4.2.3. Let $\mathbf{L} = \mathsf{AK}(x) = \mathbf{\Delta} x + \mathbf{A} \bmod p^k$ denote the given arithmetic label. Define

$$\mathbf{L}^{(i)} = \mathbf{\Delta} x_{i:k} + \mathbf{A}^{(i)} \bmod p^{k-i}\,.$$

where $\mathbf{A}^{(0)} := \mathbf{A}$ and $\mathbf{A}^{(i)}$ are randomly sampled. Thus, $\mathbf{L}^{(0)} = \mathbf{L}$. Note that,

$$\mathbf{L}^{(i)} \bmod p = \mathbf{\Delta} x_{i:k} + \mathbf{A}^{(i)} \bmod p = \mathbf{\Delta} x_i + \mathbf{A}^{(i)} \bmod p\,.$$

If the table contains ciphertext

$$\mathsf{Enc}(\mathbf{\Delta} x_i + \mathbf{A}^{(i)} \bmod p,\ (\text{boolean labels of } x_i, \mathbf{\Delta} x_i + \mathbf{A}^{(i)} - p\mathbf{A}^{(i+1)} \bmod p^{k-i}))$$

the evaluator can, given $\mathbf{L}^{(i)}$, computes all the boolean labels of x_i and the next label $\mathbf{L}^{(i+1)}$. This observation can be formalized as a secure bit-decomposition gadget, who has poor efficiency. The table consists of pk ciphertexts, each ciphertext is $(\lambda \log p + \lambda k)$-bit long, the total length is no less than $\lambda p k^2$. Under constraint $p^k \approx 2^b$, the table size is minimized when $p = O(1)$, which is asymptotically equivalent to the naive construction in Sec. 4.2.3.

The bottleneck is the encryption of $\mathbf{\Delta} x_i + \mathbf{A}^{(i)} - p\mathbf{A}^{(i+1)} \bmod p^{k-i}$. To optimize the efficiency, we replace the long ciphertexts by shorter ciphertexts

$$\mathsf{Enc}(\mathbf{\Delta} x_i + \mathbf{A}^{(i)} \bmod p,\ \text{boolean labels of } x_i)$$

that only encrypts the boolean labels. Since the evaluator can computes the boolean labels of x_i, it uses a mini bit-composition gadget (Fig. 4.4) to compute $\mathbf{\Delta} x_i + \mathbf{A}^{(i)} - p\mathbf{A}^{(i+1)} \bmod p^{k-i}$.

The optimized construction is formalized in Fig. 4.5, 4.6. After optimization, the table consists of $O(kp)$ ciphertexts, each of which is $O(\lambda \log p)$ bit long, and k mini-tables for the mini bit-composition, each of which is $O(\lambda k \log p)$ bit long. The total table size is $O(\lambda k(k + p) \log p)$.

Garbling algorithm $\mathsf{BC.Garb}$ takes boolean keys $\mathsf{K}_{i,j}$ for $i \in [k], j \in [\log p]$, and an arithmetic key $\mathsf{AK}^{\mathsf{L}} = (\mathbf{A}, \mathbf{B})$ as inputs. Let $\alpha_{i,j}$ denote the mask bit of $\mathsf{K}_{i,j}$.

- Sample random $\mathbf{B}_{i,j}$ for $i \in [k], j \in [\log p]$, satisfying $\sum_{i,j} \mathbf{B}_{i,j} \bmod p^k = \mathbf{B}$.
- For each $i \in [k], j \in [\log p]$, for each $\beta \in \{0, 1\}$, compute

$$\mathsf{C}_{i,j,\beta+\alpha_{i,j} \bmod 2} \leftarrow \mathsf{H}(\mathsf{K}_{i,j}(\beta),\ (\mathsf{id}, i, j)) \oplus (\mathbf{A}p^i 2^j \beta + \mathbf{B}_{i,j} \bmod p^k)$$

- Output table $\mathsf{Tab} = (\mathsf{C}_{i,j,\beta})_{i \in [k], j \in [\log p], \beta \in \{0,1\}}$

Evaluation algorithm $\mathsf{BC.Eval}$ takes input labels $(\mathbf{l}_{i,j}, \bar{x}_{i,j})$ for $i \in [k], j \in [\log p]$ and a table Tab as inputs.

- For $i \in [k], j \in [\log p]$, compute $\mathbf{L}_{i,j} \leftarrow \mathsf{H}(\mathbf{l}_{i,j},\ (\mathsf{id}, i, j)) \oplus \mathsf{C}_{i,j,\bar{x}_{i,j}}$.
- Output arithmetic label $\mathbf{L} = \sum_{i,j} \mathbf{L}_{i,j} \bmod p^k$.

Simulation algorithm $\mathsf{BC.Sim}$ takes input labels $(\mathbf{l}_{i,j}, \bar{x}_{i,j})$ for $i \in [k], j \in [\log p]$ and arithmetic label $\mathbf{L}$ as inputs.

- Sample random $\mathbf{L}_{i,j}$ for $i \in [k], j \in [\log p]$, satisfying $\sum_{i,j} \mathbf{L}_{i,j} \bmod p^k = \mathbf{L}$.
- The active ciphertexts in the table Tab are set as

$$\mathsf{C}_{i,j,\bar{x}_{i,j}} = \mathsf{H}(\mathbf{l}_{i,j},\ (\mathsf{id}, i, j)) \oplus \mathbf{L}_{i,j}$$

  The rest are inactive ciphertexts, and are simulated by random strings.

Figure 4.3: The naive bit-composition gadget.

Theorem 4.1. *There are statistically secure bit-composition gadget (Fig. 4.3) for ring $\mathbb{Z}_{p^k}$ whose table size is $O(\lambda k^2 \log p)$ and bit-decomposition gadget (Fig. 4.5, 4.5) for ring $\mathbb{Z}_{p^k}$, whose table size is $O(\lambda k(k + p) \log p)$. They yield a statistically secure mixed GC for $\mathbb{Z}_{p^k}$ in*

the random oracle model, such that each addition gate costs no communication, and each multiplication/bit-decomposition/bit-composition gate costs $O(\lambda k(k+p)\log p)$ communication.

The bit-composition gadget (Fig. 4.3) and the mini bit-composition gadget (Fig. 4.4) are special cases of the linear bit-composition gadget (Fig. 4.7), whose correctness and security will be analyzed in Sec. 4.4.1. The proof of the bit-decomposition gadget is similar to that of Lem. 4.3 in Sec. 4.2.3.

Under the constraint that $p^k \approx 2^b$, the asymptotic cost per gate is minimized when $p \approx b/\log^c b$ for any constant $c \geq 1$. The minimal cost is $O(\lambda b^2/\log b)$.

Further Optimization. The bit-decomposition gadget in Fig. 4.5, 4.6 can be further optimized. Currently, for each $i \in [k]$ the table contains ciphertexts

$$\mathsf{C}_{i,\beta+\alpha^{(i)} \bmod p} \leftarrow \mathsf{H}(\mathbf{\Delta}\beta + \mathbf{A}^{(i)} \bmod p,\ (\mathsf{id}, i)) \oplus (\mathsf{K}_{i,j}(\beta_j) \text{ for } j \in [\log p])$$

for each $j \in [\log p], \beta \in [p]$. Notice that, every potential boolean label, such as $\mathsf{K}_{i,j}(0)$, is encrypted in $O(p)$ ciphertexts. This is rather wasteful.

For better efficiency, $\mathsf{C}_{i,\beta+\alpha^{(i)} \bmod p}$ only encrypts a key $K_{0,\beta}$

$$\mathsf{C}_{i,\beta+\alpha^{(i)} \bmod p} \leftarrow \mathsf{H}(\mathbf{\Delta}\beta + \mathbf{A}^{(i)} \bmod p,\ (\mathsf{id}, i)) \oplus K_{0,\beta}\,.$$

The key $K_{0,\beta}$ is sampled by the garbler, and can decrypt the ciphertext

$$\mathsf{Enc}(K_{0,\beta}, (\mathsf{K}_{i,0}(\beta_0), K_{1,\beta_{1:\log p}}))\,,$$

which reveals the next boolean label and the next key $K_{1,\beta_{1:\log p}}$. That is, the garbler samples keys $K_{j,\beta_{j:\log p}}$ for every $j \in [\log p], \beta \in [p]$, and the table additionally includes ciphertexts

$$\mathsf{Enc}(K_{j,\beta_{j:\log p}}, (\mathsf{K}_{i,j}(\beta_j), K_{j+1,\beta_{j+1:\log p}}))$$

for every $j \in [\log p], \beta \in [p]$. The ciphertexts should be properly shuffled, and some color bits/digits should be introduced to help the evaluation.

After optimization, the table consists of $O(kp)$ ciphertexts, each of which is $O(\lambda)$ bit long, and k mini-tables for the mini bit-composition, each of which is $O(\lambda k \log p)$ bit long.

The total table size is $O(\lambda k(k \log p + p))$. It produces a statistically secure mixed GC in the random oracle model that has a marginal efficiency improvement compared to Thm. 4.1. But we will not explicitly state the further optimized gadget construction. The improvement is not significant enough to change the results in Tab. 4.1.

4.4.1 Extension: Linear BC and General BD

Our mixed GC for $\mathbb{Z}_{p^k}$ (Thm. 4.1) allows conversion between an arithmetic label and boolean labels of its base-p bit representation using bit-decomposition and bit-composition gadgets.

The base-p bit representation is quite useful, for example, it allows comparison between arithmetic numbers. But in many cases, we may need or may want to use the base-p' bit representation for a different base p'. The most naive solution is to use an expensive boolean circuit for base conversion. In this section, we presents an alternative solution.

BC. Let x be an arithmetic value. Given boolean labels of the base-p' bit representation of x, how to compute the $\mathbb{Z}_{p^k}$-arithmetic label of x? We ask a more general question:

> Given boolean labels of $(z_0, \ldots, z_{m-1})$, how to compute the $\mathbb{Z}_{p^k}$-arithmetic label of $\sum_i c_i z_m$, where $c_0, \ldots, c_{m-1}$ are fixed constants?

Essentially, we are asking how to garble gate $f : \{0,1\}^m \to \mathbb{Z}_{p^k}$, which is defined as $f(z_0, \ldots, z_{m-1}) = \sum_i c_i z_m \bmod p^k$.

The construction is rather straightforward. Let $\mathsf{K}_0, \ldots, \mathsf{K}_{m-1}$ be the input wire keys, let $\mathsf{AK}^{\mathsf{L}}(x) = \mathbf{A}x + \mathbf{B} \bmod p^k$ be the output wire key. Let $\mathbf{B}_0, \ldots, \mathbf{B}_{m-1}$ be an additive sharing of $\mathbf{B}$ that are sampled by the garbler. Given $\mathsf{K}_i(z_i)$, the evaluator can compute $\mathbf{L}_i = \mathbf{A}c_i z_i + \mathbf{B}_i \bmod p^k$ because the table contains

$$\mathsf{Enc}(\mathsf{K}_i(\beta), \mathbf{A}c_i\beta + \mathbf{B}_i)$$

for all $i \in [m], \beta \in \{0,1\}$. The evaluator outputs

$$\begin{aligned} \mathbf{L} := \sum_i \mathbf{L}_i \bmod p^k &= \sum_i (\mathbf{A}c_i z_i + \mathbf{B}_i) \bmod p^k \\ &= \mathbf{A}f(z_0, \ldots, z_{m-1}) + \mathbf{B} \bmod p^k \,. \end{aligned} \tag{4.3}$$

This is formalized in Fig. 4.7.

Lemma 4.4. *For any* $f(z_0, \ldots, z_{m-1}) = \sum_i c_i z_m \bmod p^k$*, there is a secure garbling gadget for general linear bit-composition function* f *(Fig. 4.7), called* linear bit-composition gadget*, in the random oracle model. The table size is* $O(\lambda mk)$*, assume the output label dimension is* $\lambda/\log p$*.*

Proof. For any input $z_0, \ldots, z_{m-1}$, the evaluator computes $\mathbf{L}_i \leftarrow \mathsf{H}(\mathbf{l}_i, (\mathsf{id}, i)) \oplus \mathsf{C}_{i,z_i \oplus \alpha_i}$, then $\mathbf{L}_i = \mathbf{A} c_i \beta + \mathbf{B}_i \bmod p^k$. The correctness of the output is guaranteed by (4.3).

To prove security, is suffices to notice that $\mathbf{B}_0, \ldots, \mathbf{B}_{m-1}$ is an additive sharing implies $\mathbf{L}_0, \ldots, \mathbf{L}_{m-1}$ is an additive sharing. In other words, we know $\mathbf{L}_0, \ldots, \mathbf{L}_{m-2}$ is i.i.d. uniform in the real world because they are one-time padded by i.i.d. uniform $\mathbf{B}_0, \ldots, \mathbf{B}_{m-2}$. And $\mathbf{L}_{m-1}$ is determined by $\mathbf{L}_0, \ldots, \mathbf{L}_{m-2}$ and $\mathbf{L}$ from $\mathbf{L} := \sum_i \mathbf{L}_i \bmod p^k$. □

BD.

Given the $\mathbb{Z}_{p^k}$-arithmetic label of x, if we want to compute the boolean labels of the base-p' bit representation of x:

- First compute the boolean labels of the base-p bit representation of x, using bit-decomposition gadget.
- Compute the $\mathbb{Z}_{p'^{k'}}$-arithmetic label of x, using linear bit-composition gadget.
- Compute the boolean labels of the base-p' bit representation of x, using bit-decomposition gadget.

In particular, the cost of conversion from base-p bit representation to base-2 representation is $O(\lambda b^2)$ where $2^b \approx p^k$. This is much cheaper than using the boolean circuit for base conversion.

4.4.2 Extension: Emulating Computations for $\mathbb{Z}_N$

Our mixed GC for $\mathbb{Z}_{p^k}$ can emulate arithmetic mod-N operations if $p^k > N^2$ and there is an efficient garbling gadget for the modulo gate $\mathsf{mod}_N : \mathbb{Z}_{p^k} \to \mathbb{Z}_{p^k}$, which is defined as $\mathsf{mod}_N(x) = x \bmod N$. The emulation is rather straightforward:

- Every number in $\mathbb{Z}_N$ is emulated by the same number in $\mathbb{Z}_{p^k}$
- Every mod-N arithmetic operation (ADD or MULT) is emulated the by the same operation over $\mathbb{Z}_{p^k}$, followed by mod_N.

  Remark: The cost of emulating addition gates can be dramatically optimized. Instead of appending mod_N after every addition gate, append mod_N only if the accumulated magnitude is close to $p^k/2$ or when the fan-out includes a multiplication gate.

Garbling the the modulo gate mod_N is mostly equivalent to garbling the integer division gate $\mathsf{div}_N : \mathbb{Z}_{p^k} \to \mathbb{Z}_{p^k}$, which is defined as $\mathsf{div}_N(x) = \lfloor x/N \rfloor$, since $\mathsf{mod}_N(x) = x - N \cdot \mathsf{div}_N(x)$. Unfortunately, the garbling gadget for div_N is hard to construct.[3] We will define a similar gate div^*_N whose garbling gadget is efficient and also suffices for emulating mod-N computations. The definition of $\mathsf{div}^*_N(x)$ is inspired by a well-known optimization that reduce division by constant to multiplication and shifting.

Lemma 4.5 (Generalization of [GM94]). *For any positive integers $N, p, k_\mathsf{I}, k_\mathsf{E}, m$ satisfying $p^{k_\mathsf{I}+k_\mathsf{E}} \leq mN < p^{k_\mathsf{I}+k_\mathsf{E}} + p^{k_\mathsf{E}}$,*

$$\left\lfloor \frac{x}{N} \right\rfloor = \left\lfloor \frac{mx}{p^{k_\mathsf{I}+k_\mathsf{E}}} \right\rfloor \qquad \textit{for all } 0 \leq x < p^{k_\mathsf{I}}.$$

Proof. $p^{k_\mathsf{I}+k_\mathsf{E}} \leq mN < p^{k_\mathsf{I}+k_\mathsf{E}} + p^{k_\mathsf{E}}$ implies, by multiplying $\frac{x}{p^{k_\mathsf{I}+k_\mathsf{E}}N}$,

$$\frac{x}{N} \leq \frac{mx}{p^{k_\mathsf{I}+k_\mathsf{E}}} < \frac{x}{N} + \frac{x}{Np^{k_\mathsf{I}}} < \frac{x+1}{N}.$$

□

Now we are ready to define the gate $\mathsf{div}^*_N : \mathbb{Z}_{p^{2k+1}} \to \mathbb{Z}_{p^{2k+1}}$. Let $k_\mathsf{E} := \lceil \log_p(N) \rceil$ be the minimum integer satisfying $p^{k_\mathsf{E}} \geq N$. Let $m = \lceil \frac{p^{k_\mathsf{I}+k_\mathsf{E}}}{N} \rceil$, thus $p^{k_\mathsf{I}+k_\mathsf{E}} \leq mN < p^{k_\mathsf{I}+k_\mathsf{E}} + N \leq p^{k_\mathsf{I}+k_\mathsf{E}} + p^{k_\mathsf{E}}$. By Lem. 4.5,

$$\left\lfloor \frac{x}{N} \right\rfloor = \left\lfloor \frac{mx}{p^{k+k_\mathsf{E}}} \right\rfloor$$

for any $0 \leq x < p^k$. Therefore we define $\mathsf{div}^*_N : \mathbb{Z}_{p^{2k+1}} \to \mathbb{Z}_{p^{2k+1}}$ as

$$\mathsf{div}^*_N(x) = \left\lfloor \frac{mx \bmod p^{2k+1}}{p^{k+k_\mathsf{E}}} \right\rfloor.$$

[3] An efficient garbling gadget of div_N can be constructed based on the garbling gadget of div^*_N.

It satisfies $\mathsf{div}_N^*(x) = \lfloor x/N \rfloor$ for all $x < p^k$. Since div_N^* is the composition of multiplication in $\mathbb{Z}_{p^{2k+1}}$ and digit shifting, it can be efficiently garbled by our mixed GC for $\mathbb{Z}_{p^{2k+1}}$.

Define gate $\mathsf{mod}_N^* : \mathbb{Z}_{p^{2k+1}} \to \mathbb{Z}_{p^{2k+1}}$ as $\mathsf{mod}_N^*(x) = x - N \cdot \mathsf{div}_N^*(x)$. Then mod_N^* can be efficiently garbled by our mixed GC for $\mathbb{Z}_{p^{2k+1}}$, and $\mathsf{mod}_N^*(x) = x \bmod N$ for all $x < p^k$.

Lemma 4.6. *For any $N \leq 2^b$, there is a statistically secure mixed GC for $\mathbb{Z}_N$ in the random oracle model, such that each addition/multiplication/bit-decomposition/bit-composition gate costs $O(\lambda b^2/\log b)$ communication. The bit-decomposition is over a prime base $p = \Theta(b/\log b)$.*

Proof. Mod-N computations can be emulated in a $\mathbb{Z}_{p^{2k+1}}$-mixed circuits. Combing with Thm. 4.1, the cost per gate is $O(\lambda k(k+p)\log p)$. The cost is minimized by letting $p = \Theta(b/\log b)$. □

Remarks. Although Lem. 4.6 does not claim free addition, we observe from its construction that addition is free up to a certain extent.

In this mixed GC for $\mathbb{Z}_N$, the bit decomposition gate outputs base-p bit representations. In case a (base-2) bit representation is needed, it can be computed from the base-p bit representation by a cost of $O(\lambda b^2)$, using the trick stated in Sec. 4.4.1.

4.5 Mixed GC from Chinese Remainder Theorem in ROM

Chinese remainder theorem (CRT) is used in [BMR16] to solve the following natural task: Given b, find an efficient arithmetic GC over ring $\mathbb{Z}_N$ for some $N \approx 2^b$.

Since there is no more specific constraints on N, [BMR16] sets $N = p_1 p_2 \dots p_s$ being the product of the first s primes. Then $s = \Theta(b/\log b)$ and $p_s = \Theta(b)$. Consider an arithmetic circuit over $\mathbb{Z}_{p_i}$, denoted by "$C \bmod p_i$", that is identical to C except the ring is replaced by $\mathbb{Z}_{p_i}$. Then

$$C(x) \bmod p_i = (C \bmod p_i)(x \bmod p_i)\,.$$

Therefore, by CRT, the task of evaluating $C(x)$ is reduced to evaluating mod-p_i arithmetic circuit $(C \bmod p_i)(x \bmod p_i)$ for all $1 \leq i \leq s$. In [BMR16], the reduction is combined with

mixed GC for every ring $\mathbb{Z}_{p_i}$, resulting in an arithmetic GC for $\mathbb{Z}_N$ where each multiplication gate costs about $O(\lambda b^2/\log b)$ bits.

In this section, we will strengthen the result in two dimensions.

Based on Mod-p^k Mixed GC. [BMR16] sets $N = p_1 p_2 \dots p_s$ because their basic GC only supports computation modulo a prime number. In Sec. 4.4, we have already construct relatively efficient mixed GC for prime power rings. Therefore, we will set

$$N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$$

and reduce the problem of garbling mod-N computation to garbling mod-$p_i^{k_i}$ computations for each $1 \le i \le s$.

Efficient BD. In the CRT framework, if the actual value of a $\mathbb{Z}_N$-wise is x, it is not hard to get the boolean labels of the bit representation of $x \bmod p_i^{k_i}$ (via Theorem 4.1), for each $1 \le i \le s$. To compute the bit representation of x, the naive idea is garble the CRT algorithm.

For more efficient bit-decomposition, we make the following observation. There are constants $c_1, \dots, c_s \in \mathbb{Z}_N$ such that, for any $x \in \mathbb{Z}_N$

$$x = \sum_i c_i x^{(i)} \bmod N\,,$$

where $x^{(i)} := x \bmod p_i^{k_i}$ denotes the mod-$p_i^{k_i}$ component of x. $(x^{(1)}, \dots, x^{(s)})$ is usually called the *CRT representation* of x. The fact that x is a linear function (modulo N) on its CRT representation suggests a more efficient bit-decomposition construction in the "CRT framework".

Our new bit-decomposition construction is essentially a mixed circuit over the ring $\mathbb{Z}_{p^{2k+1}}$, where p, k satisfy $p^k > N^2 > \sum_i c_i x^{(i)}$. The input of the mixed circuits consists of the bit representation of $x^{(i)}$ for all $1 \le i \le s$. All the input wires can be merged into $\sum_i c_i x^{(i)}$ through the generalized linear BC gate (Fig. 4.7). Then next step is mod_N^*, whose output $\sum_i c_i x^{(i)} \bmod N$ always equals x. The last gate is the standard bit-decomposition of $\mathcal{C}_{\mathsf{mix}}(\mathbb{Z}_{p^k})$, producing the base-$p$ bit representation of x.

The linear BC costs λmk bits, where $m = \sum_i k_i \log p_i = O(b)$. The modulo gate mod_N^* and bit-decomposition gate cost $O(\lambda b(k+p))$. The overall cost is $O(\lambda b(k+p))$, which can be minimized as $O(\lambda b^2/\log b)$ by setting $p = \Theta(b/\log b)$.

If (base-2) bit representation of x is required, the overall cost of BD is $O(\lambda b^2)$.

By combining the "CRT framework" with Thm. 4.1 and Lem. 4.6 respectively, we have two more efficient mixed GC for $\mathbb{Z}_N$.

Theorem 4.2. *For any b, there exist $N > 2^b$ and a statistically secure mixed GC for $\mathbb{Z}_N$ in the random oracle model, such that each addition gate costs no communication, and each multiplication gate costs $O(\lambda b^{1.5})$ communication, and each bit-decomposition/bit-composition gate costs $O(\lambda b^2/\log b)$ communication.*

Proof. Set $N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$. The task of garbling mod-N mixed circuits is reduced to garbling mod-$p_i^{k_i}$ mixed circuits for all $1 \le i \le s$. Each mod-$p_i^{k_i}$ mixed circuit will be garbled the mixed GC in Thm. 4.1.

Thus each mod-N addition gate will cost nothing.

Each mod-N multiplication gate costs

$$\sum_i O(\lambda k_i (k_i + p_i) \log p_i) \,.$$

We want to minimize the cost, under the constraint that $p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$.

For any i, if k_i increases by 1, then $\log N$ will increase by $\log p_i$, the total cost will increase by $O(\lambda(k_i + p_i) \log p_i)$. The "marginal cost increase per bit of N by changing k_i" is

$$\frac{\partial \mathrm{cost}(k_1, \dots, k_s)}{\partial k_i} \Bigg/ \frac{\partial \log N(k_1, \dots, k_s)}{\partial k_i} = O(\lambda(k_i + p_i)) \,.$$

To minimize the cost, this ratio should be roughly the same for all i.

Following this intuitive argument, we choose a constant c and let $p_i + k_i = c$ for all i. The value of c is determined by the constraint $N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$.

$$b \le \log \prod_{i \le s} p_i^{k_i} = \sum_{i \le s} k_i \log p_i = \sum_{i \le s} (c - p_i) \log p_i \approx \sum_{p=2}^{c} (c - p) = \Theta(c^2) \,.$$

Thus we set $c = \Theta(\sqrt{b})$.

The cost per multiplication gate is

$$\sum_i \lambda k_i(k_i + p_i) \log p_i = \sum_i \lambda(c - p_i)c \log p_i \approx \sum_{p=2}^{c} \lambda(c-p)c = O(\lambda c^3) = O(\lambda b^{1.5}).$$

The total cost of having one BD gate in the mod-$p_i^{k_i}$ part for all $1 \le i \le s$ is also $O(\lambda b^{1.5})$. But these parallel BD gates only compute (the bit representation of) the CRT representation. To compute the bit representation, an additional cost of $O(\lambda b^2)$ (or $O(\lambda b^2/\log b)$, if the representation can use any base) is needed.

For BC, say the boolean representation of the number has at most $O(b)$ bits. Applying linear BC (Fig. 4.7) for all $1 \le i \le s$ will cost $O(\sum_i \lambda b k_i)$ bits.

$$\sum_i \lambda b k_i = \lambda b \sum_i (c - p_i) \le \lambda bcs = O(\lambda b^2/\log b) \qquad \square$$

Theorem 4.3. *For any b, there exist $N > 2^b$ and a statistically secure mixed GC for $\mathbb{Z}_N$ in the random oracle model, such that each addition/multiplication gate costs $O(\lambda b \log b/\log\log b)$ communication, each bit-decomposition costs $O(\lambda b^2/\log b)$ communication, each bit-composition gate costs $O(\lambda b^2/\log\log b)$ communication.*

Proof. Set $N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$. The task of garbling mod-N mixed circuits is reduced to garbling mod-$p_i^{k_i}$ mixed circuits for all $1 \le i \le s$. Each mod-$p_i^{k_i}$ mixed circuit will be garbled with the mixed GC in Lem. 4.6.

Each mod-N addition/multiplication gate costs

$$\sum_i O(\lambda d_i^2/\log d_i), \text{ where } 2^{d_i} > p_i^{k_i}.$$

We want to minimize the cost, under the constraint that $p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$.

We choose a constant d such that $d = d_1 = d_2 = \dots = d_s$, and let $k_i = \lfloor d_i/\log p_i \rfloor$. So all primes are smaller than 2^d and $s = \Theta(2^d/d)$. The value of d is determined by the constraint $N = p_1^{k_1} p_2^{k_2} \dots p_s^{k_s} \approx 2^b$.

$$b \le \log \prod_{i \le s} p_i^{k_i} = \sum_{i \le s} k_i \log p_i \le \frac{1}{2} \sum_{i \le s} d = \Theta(sd) = \Theta(2^d).$$

Thus we set $d = \log b + O(1)$. Then $s = O(b/\log b)$.

The cost of each mod-N addition/multiplication gate is

$$\sum_i O\Big(\frac{\lambda d_i^2}{\log d_i}\Big) = O\Big(\frac{s\lambda d^2}{\log d}\Big) = O\Big(\frac{b\lambda \log b}{\log\log b}\Big).$$

The cost of BD, by the same analysis as in the proof of Thm. 4.2, is $O(\lambda b^2)$ if the outcome is base-2 bit representation, $O(\lambda b^2/\log b)$ if the representation can use any base.

The cost of BC is trickier to trace. For each i, the mod-$p_i^{k_i}$ computations are emulated, according to the construction of Lem. 4.6, by a mod-p^k mixed circuit. Such that $k = O(d/\log d) = O(\log b/\log\log b)$. For each i, using linear BC to compute the arithmetic value costs λbk. The total cost is $s\lambda bk = \lambda b^2/\log\log b$. But linear BC computes a linear function modulo p^k, rather than the desired modulus $p_i^{k_i}$. This issue is resolve by slightly enlarge p^k to some $\text{poly}(p_i^{k_i}, b) = b^{\Theta(1)}$ so that linear BC computes the linear function over $\mathbb{Z}$. This modification over enlarge k by a constant factor, thus will not asymptotically increase the cost of any operations. □

4.6 Mixed GC from DCR-Based LHE in ROM

In this section, we show how to improve the efficiency of our mixed GC construction by relying on computational assumption. The new construction is most similar to the naive mixed construction (Lem. 4.3 in Sec. 4.2.3) over ring $\mathbb{Z}_{2^b}$.

The construction will be based on the computational assumption of decisional composite residuosity (DCR, Definition 2.9). In particular, we consider the following two private-key variants of the Damgård-Jurik Encryption [Pai99, DJ01], whose security is known to be equivalent to the DCR assumption. We note the notation mapping in Fig. 4.8

Construction 12 (First Variant of Damgård-Jurik Encryption)**.**

- $\mathsf{DamJur.Setup}(1^\lambda, 1^\zeta)$ Sample two safe primes $p = 2p' + 1$, $q = 2q' + 1$ of $\lambda_{\mathsf{DCR}}(\lambda)$-bit long each. Compute $M = pq$, sample a random generator $g \in \mathsf{HC}_{M^{\zeta+1}}$. Output public parameter $\mathsf{pp} = (M, \zeta, g)$ and trapdoor $\mathsf{tp} = p'q'$.

- $\mathsf{DamJur.Gen}(\mathsf{pp})$ Samples a key $\mathsf{sk} \leftarrow [M/4]$.
- $\mathsf{DamJur.Enc}(\mathsf{sk}, m)$ takes a key $\mathsf{sk} \in \mathbb{Z}$ and a message $m \in [M^\zeta]$. Output

$$\mathsf{DamJur.Enc}(\mathsf{sk}, m) := g^{\mathsf{sk}}(M+1)^m \bmod M^{\zeta+1}\,.$$

- $\mathsf{DamJur.Dec}(\mathsf{sk}, c)$ takes a key $\mathsf{sk} \in \mathbb{Z}$ and a ciphertext $c \in \mathbb{Z}_{M^{\zeta+1}}$. Output

$$\mathsf{DamJur.Dec}(\mathsf{sk}, c) := \mathsf{DLog}_{M+1}(c/g^{\mathsf{sk}}) \quad (\text{over } \mathbb{Z}_{M^{\zeta+1}})\,.$$

Construction 13 (Second Variant of Damgård-Jurik Encryption)**.**

- $\overline{\mathsf{DamJur}}.\mathsf{Setup}(1^\lambda, 1^\zeta)$ Sample two safe primes $p = 2p' + 1$, $q = 2q' + 1$ of $\lambda_{\mathsf{DCR}}(\lambda)$-bit long each. Compute $M = pq$. Output public parameter $\mathsf{pp} = (M, \zeta)$ and trapdoor $\mathsf{tp} = p'q'$.
- $\overline{\mathsf{DamJur}}.\mathsf{Gen}(\mathsf{pp})$ Samples a key $g \leftarrow \mathbb{Z}_M^*$.
- $\overline{\mathsf{DamJur}}.\mathsf{Enc}(g, m)$ takes a key $g \in \mathbb{Z}_M^*$ and a message $m \in [M^\zeta]$. Output

$$\mathsf{ct} = \overline{\mathsf{DamJur}}.\mathsf{Enc}(g, m) := \pi_{M^{\zeta+1}}(g) \cdot (M+1)^m \bmod M^{\zeta+1}\,.$$

- $\overline{\mathsf{DamJur}}.\mathsf{Dec}(\mathsf{sk}, c)$ takes a key $g \leftarrow \mathbb{Z}_M^*$ and a ciphertext $c \in \mathbb{Z}_{M^{\zeta+1}}$. Output

$$\overline{\mathsf{DamJur}}.\mathsf{Dec}(\mathsf{sk}, c) := \mathsf{DLog}_{M+1}(c/\pi_{M^{\zeta+1}}(g)) \quad (\text{over } \mathbb{Z}_{M^{\zeta+1}})\,.$$

- $\overline{\mathsf{DamJur}}.\mathsf{Inv}(\mathsf{tp}, c)$ takes a ciphertext $c \in \mathbb{Z}_{M^{\zeta+1}}$, output the unique $g \in \mathbb{Z}_M^*$, $m \in [M^\zeta]$ such that $c = \pi_{M^{\zeta+1}}(g) \cdot (M+1)^m \bmod M^{\zeta+1}$:

$$g = (c \bmod M)^{(M^\zeta)^{-1}} \bmod M, \quad m = \overline{\mathsf{DamJur}}.\mathsf{Dec}(g, c).$$

  Note that computing the inverse of M^ζ modulo $\varphi(M)$ requires knowledge of the trapdoor. See [BDGM20] for more detail on the correctness of the inversion algorithm.

We note the following linear homomorphism in both constructions. For any message m_1, m_2 and keys $\mathsf{sk}_1, \mathsf{sk}_2, g_1, g_2$.

$$\begin{aligned}&\mathsf{DamJur.Enc}(\mathsf{sk}_1, m_1) \cdot \mathsf{DamJur.Enc}(\mathsf{sk}_2, m_2) \bmod M^{\zeta+1}\\ &\quad = \mathsf{DamJur.Enc}(\mathsf{sk}_1 + \mathsf{sk}_2 \text{ over } \mathbb{Z},\ m_1 + m_2 \bmod M^\zeta)\end{aligned}$$

$$\begin{aligned}&\overline{\mathsf{DamJur}}.\mathsf{Enc}(g_1, m_1) \cdot \overline{\mathsf{DamJur}}.\mathsf{Enc}(g_2, m_2) \bmod M^{\zeta+1}\\ &\quad = \overline{\mathsf{DamJur}}.\mathsf{Enc}(g_1 \cdot g_2 \bmod M,\ m_1 + m_2 \bmod M^\zeta)\end{aligned}$$

4.6.1 Bit-Composition Based on DamJur

As observed in Sec. 4.4.1, the more general bit-composition function $(x_0, \ldots, x_{m-1}) \to \sum_i c_i x_i \bmod 2^b$ is not harder to garble. Thus we will directly construct this more general bit-composition.

Let $\mathsf{K}_0, \ldots, \mathsf{K}_{m-1}$ be the boolean keys. Let $\mathsf{AK}^\mathsf{L} = (\mathbf{A} \in \mathbb{Z}_{2^b}^\ell, \mathbf{B} \in \mathbb{Z}_{2^b}^\ell)$ be the arithmetic key. In the analysis of the complexity, we will assume $m = O(b)$ and $\ell = O(\lambda)$. For any $x_0, \ldots, x_{m-1} \in \{0,1\}$, given $\mathsf{K}_0(x_0), \ldots, \mathsf{K}_{m-1}(x_{m-1})$ and the table, the evaluator of the bit-composition gadget should output the arithmetic label $\mathbf{L} = \mathsf{AK}^\mathsf{L}(x) = x\mathbf{A} + \mathbf{B} \bmod 2^b$ where $x = \sum_i c_i x_i \bmod 2^b$.

The construction is based on the following intuition (informally): Allow the evaluator to decrypts $x + r$ and $(x+r)\mathsf{sk}^A + \mathsf{sk}^B$. Let the table contain

$$\mathsf{ct}^A = \mathsf{Enc}(\mathsf{sk}^A, \mathbf{A}), \quad \mathsf{ct}^B = \mathsf{Enc}(-r\mathsf{sk}^B, -r\mathbf{A} + \mathbf{B})$$

using some homomorphic encryption. Then the evaluator can compute

$$(\mathsf{ct}^A)^{x+r}\mathsf{ct}^B = \mathsf{Enc}((x+r)\mathsf{sk}^A + \mathsf{sk}^B, x\mathbf{A} + \mathbf{B})$$

which can be decrypted into $x\mathbf{A} + \mathbf{B}$.

To formalize the intuition: i) We will add large random noise $\mathbf{R}$, and let the evaluator get $x\mathbf{A} + \mathbf{B} + 2^b\mathbf{R}$ instead. ii) We need to construct an encryption scheme that has the required homomorphism.

As the section name suggested, the encryption scheme is (almost) DamJur, except that we want the scheme to encrypt a vector rather than a number. We consider the following natural encoding $\mathsf{encode} : \mathbb{Z}^\ell \to \mathbb{Z}$, parameterized by ℓ and B,

$$\mathsf{encode}(v_0, \ldots, v_{\ell-1}) = \sum_{i \in [\ell]} B^i v_i \,,$$

together with an efficient decoder $\mathsf{decode} : [B^\ell] \to [B]^\ell$, satisfying

- For any $\mathbf{A}, \mathbf{B} \in \mathbb{Z}^\ell$, $\mathsf{encode}(\mathbf{A} + \mathbf{B}) = \mathsf{encode}(\mathbf{A}) + \mathsf{encode}(\mathbf{B})$.

- For any $\mathbf{A} \in [B]^\ell$, $\mathsf{encode}(\mathbf{A}) \in [B^\ell]$ and $\mathsf{decode}(\mathsf{encode}(\mathbf{A})) = \mathbf{A}$.

Set the parameter of the encoder by $B = 2^{2b+2\lambda+1}$. Define the following encryption scheme vDJ,

- $\mathsf{vDJ.Setup}(1^\lambda)$ is $\mathsf{DamJur.Setup}(1^\lambda, 1^\zeta)$, by choosing smallest ζ s.t. $M^\zeta \geq B^\ell$.
- $\mathsf{vDJ.Gen}$ is $\mathsf{DamJur.Gen}$.
- $\mathsf{vDJ.Enc}(\mathsf{sk}, \mathbf{V}) = \mathsf{DamJur.Enc}(\mathsf{sk}, \mathsf{encode}(\mathbf{V}))$.
- $\mathsf{vDJ.Dec}(\mathsf{sk}, c) = \mathsf{decode}(\mathsf{DamJur.Dec}(\mathsf{sk}, c))$.

Using vDJ, our intuition can be formalized as a bit-composition gadget.

Lemma 4.7. *For any linear bit-composition function* $f(z_0, \dots, z_{m-1}) = \sum_i c_i z_m \bmod 2^b$ *satisfying* $\sum_i c_i \leq 2^b$ *(otherwise the construction should be slightly modified), there is a secure garbling gadget for* f *(Fig. 4.9, 4.10), under DCR assumption in the random oracle model. The table size is* $O(m\lambda_{\mathsf{DCR}} + \ell(b+\lambda))$*, which is* $O(\lambda_{\mathsf{DCR}} b + \lambda^2)$ *when* $\ell = O(\lambda)$ *and* $m = O(b)$*.*

Proof. Verify the correctness in the real world: For each i, $\hat{x}_i = x_i + r_i$. Thus $\hat{x} = \sum_i c_i x_i = x + r$. For each i, $\mathsf{sk}_i = c_i(x_i + r_i)\mathsf{sk}^A + \mathsf{sk}_i^B \bmod p'q'$, thus

$$\mathsf{sk} = \sum_i \Big(c_i(x_i + r_i)\mathsf{sk}^A + \mathsf{sk}_i^B \bmod p'q' \Big) = (x+r)\mathsf{sk}^A + \mathsf{sk}^B + p'q't$$

for some $t \in \mathbb{Z}$. By the homomorphism of encryption,

$$\begin{aligned}
(\mathsf{ct}^A)^{\hat{x}}\mathsf{ct}^B &= (\mathsf{vDJ.Enc}(\mathsf{sk}^A, \mathbf{A}))^{\hat{x}}(\mathsf{vDJ.Enc}(\mathsf{sk}^B, (2^{2b+\lambda} - r\mathbf{A}) + \mathbf{B} + 2^b\mathbf{R})) \\
&= \mathsf{vDJ.Enc}(\hat{x}\mathsf{sk}^A + \mathsf{sk}^B, \hat{x}\mathbf{A} + 2^{2b+\lambda} - r\mathbf{A} + \mathbf{B} + 2^b\mathbf{R}) \\
&= \mathsf{vDJ.Enc}(\mathsf{sk}, x\mathbf{A} + \mathbf{B} + 2^{2b+\lambda} + 2^b\mathbf{R})
\end{aligned}$$

which can be decrypted by sk.

$$\hat{\mathbf{L}} = \mathsf{vDJ.Dec}(\mathsf{sk}, (\mathsf{ct}^A)^{\hat{x}}\mathsf{ct}^B) = x\mathbf{A} + \mathbf{B} + 2^{2b+\lambda} + 2^b\mathbf{R} \tag{4.4}$$

Finally, $\mathbf{L} = \hat{\mathbf{L}} \bmod 2^b = x\mathbf{A} + \mathbf{B} \bmod 2^b$.

Security follows from the following arguments:

- sk^A is uniformly sampled in the real world.
- Since $\hat{x}_i = x_i + r_i$ is smudged by random $r_i \in [2^\lambda]$ in the real world. Simulating it as $\hat{x} \leftarrow [2^\lambda]$ only introduces $2^{-\lambda}$ statistical error.
- sk_i is uniformly distributed among $[p'q']$, because it is one-time padded by sk_i^B in the real world. Simulating sk_i by $\mathsf{sk}_i \leftarrow [M/4]$ introduces a statistical error of $O(2^{-\lambda_{\mathsf{DCR}}})$.
- In the real world $\hat{\mathbf{L}} = (x\mathbf{A} + \mathbf{B}) + 2^{2b+\lambda} + 2^b\mathbf{R}$. Note that

$$(x\mathbf{A} + \mathbf{B}) + 2^{2b+\lambda} = (x\mathbf{A} + \mathbf{B} \bmod 2^b) + 2^b\mathbf{T} = \mathbf{L} + 2^b\mathbf{T}$$

  for some $\mathbf{T} \in [2^{b+\lambda+1}]^\ell$. The randomly sampled $\mathbf{R}$ who has magnitude $2^{b+2\lambda}$ smudges $\mathbf{T}$. Simulating $\hat{\mathbf{L}}$ by $\mathbf{L} + 2^b\mathbf{R}$ introduces a statistical error of magnitude $2^{-\lambda}$.
- Combine the arguments so far, in the real world, the joint distribution of

$$\mathsf{sk}^A, \quad (\hat{x}_i)_{i\in[m]}, \quad (\mathsf{sk}_i)_{i\in[m]}, \quad \mathbf{R} = \frac{\hat{\mathbf{L}} - \mathbf{L}}{2^b}$$

  is $O(\frac{\text{poly}}{2^\lambda})$-close to a uniform distribution over $[p'q'] \times [2^\lambda]^m \times [M/4]^m \times [2^{b+2\lambda}]$. That is, in the real world, the distribution of sk^A is close to uniform even conditioning all the values simulated so far. Thus ct^A can be simulated by a random ciphertext in $\mathsf{QR}_{M^{\zeta+1}}$, by the DCR assumption.
- ct^B is uniquely determined by the correctness guarantee, (determined by Eq. (4.4), in particular).

The table consists of $O(m)$ one-time pad ciphertexts, each of which is $O(\lambda_{\mathsf{DCR}})$ bit long, and two vDJ ciphertexts, each of which is of length

$$\zeta \log M \leq \lambda_{\mathsf{LWE}} + \ell(2b + 2\lambda)\,.$$

So the total table size is $O(m\lambda_{\mathsf{DCR}} + \ell(b + \lambda))$. □

4.6.2 Bit-Decomposition Based on $\overline{\mathsf{DamJur}}$

In the bit-decomposition gadget, the evaluator is given an arithmetic label $\mathbf{L} = \mathsf{AK}(x) = x\mathbf{\Delta} + \mathbf{A} \bmod 2^b$, and its color number $\bar{x} = x + \alpha \bmod 2^b$ together with a table generated by the garbler from $\mathsf{AK}, \mathsf{K}_0, \ldots, \mathsf{K}_{b-1}$, and should output $\mathsf{K}_0(x_0), \ldots, \mathsf{K}_{b-1}(x_b)$.

Recall our intuition behind the naive BD (Fig. 4.1, 4.2): In each inductive step, the evaluator gets $\mathbf{L}^{(i)} = x_{i:b}\mathbf{\Delta} + \mathbf{A}^{(i)}$ and computes

$$\mathbf{L}^{(i)} \bmod 2 = x_i\mathbf{\Delta} + \mathbf{A}^{(i)} \bmod 2.$$

Using $\mathbf{L}^{(i)} \bmod 2$ as the key, the evaluator decrypts a ciphertext

$$\mathsf{H}(x_i\mathbf{\Delta} + \mathbf{A}^{(i)} \bmod 2) \oplus (\mathsf{K}(x_i), x_i\mathbf{\Delta} + \mathbf{S})$$

in the table, gets $\mathsf{K}(x_i)$ and $x_i\mathbf{\Delta} + \mathbf{S}$. The latter allows the evaluator to compute $\mathbf{L}^{(i+1)}$ and proceed to the next step.

The bottleneck is the ciphertext size. Let us replace the ciphertext by

$$\mathsf{H}(x_i\mathbf{\Delta} + \mathbf{A}^{(i)} \bmod 2) \oplus (\mathsf{K}(x_i), x_i + r, (x_i + r)\mathsf{sk}^\Delta + \mathsf{sk}^S)\,.$$

And let the table additionally contains two ciphertexts

$$\mathsf{ct}^\Delta = \mathsf{Enc}(\mathsf{sk}^\Delta, \mathbf{\Delta})\,, \quad \mathsf{ct}^S = \mathsf{Enc}(\mathsf{sk}^S, -r\mathbf{\Delta} + \mathbf{S} + 2^b\mathbf{R})\,,$$

using a homomorphic encryption scheme. Then the evaluator can instead compute $x_i\mathbf{\Delta} + \mathbf{S} + 2^b\mathbf{R}$ from

$$\mathsf{Dec}((x_i + r)\mathsf{sk}^\Delta + \mathsf{sk}^S, (\mathsf{ct}^\Delta)^{x_i+r}/\mathsf{ct}^S)\,.$$

Such modification does not improves the complexity yet, because $\mathsf{ct}^\Delta, \mathsf{ct}^S$ become the new dominating part. Notice that, all tables may share a global ct^Δ as it only depends on the global key.

For the last bottleneck ct^S, we require its distribution to be *"dense"*, in the sense that, the distribution of ct^S is statistically close to the uniform distribution over a samplable domain. This requires i) a "dense" encryption scheme, and ii) the distribution of the message $-r\mathbf{\Delta} + \mathbf{S} + 2^b\mathbf{R}$ is statistically close to uniform over the message space.

If our requirement is satisfied, the garbler can instead sample a random seed, and let $\mathsf{ct}^S = \mathsf{H}(\mathsf{seed})$. The ciphertext ct^S in the table can be replaced by seed. For correctness, the garbler need to reversely compute the key and message behind the ciphertext ct^S.

As discussed in [BDGM20], all of our requirements are satisfied by $\overline{\mathsf{DamJur}}$.

- *Density:* For random $g \leftarrow \mathbb{Z}_M^*$ and random $m \leftarrow [M^\zeta]$, the distribution of ciphertext $\overline{\mathsf{DamJur}}.\mathsf{Enc}(g, m)$ is uniform in $\mathbb{Z}^*_{M^{\zeta+1}}$.
- *Invertibility:* There is an efficient algorithm Inv, which takes a ciphertext $\mathsf{ct} \in \mathbb{Z}^*_{M^{\zeta+1}}$ and the trapdoor tp, computes g, m such that $\overline{\mathsf{DamJur}}.\mathsf{Enc}(g, m) = \mathsf{ct}$.

$\overline{\mathsf{DamJur}}$ encrypts a number rather a vector. Similar to Sec. 4.6.1, we need a encoder-decoder pair between vectors and numbers. The encoder has to be dense in the sense that almost all encodings in the codomain are valid. Again, consider the natural encoding $\mathsf{encode} : \mathbb{Z}^{\lambda+1} \to \mathbb{Z}$, parameterized by B,

$$\mathsf{encode}(v_0, \dots, v_\lambda) = \sum_{i \in [\lambda+1]} B^i v_i \,,$$

together with an efficient decoder $\mathsf{decode} : [B^\lambda] \to [B]^\lambda$.

- For security, set $B \geq 2^{b+2\lambda}$.
- For density, ensure $M^\zeta \geq B^{\lambda+1} \geq M^\zeta(1 - 2^{-\lambda})$.

Define the following encryption scheme $\overline{\mathsf{vDJ}}$,

- $\overline{\mathsf{vDJ}}.\mathsf{Setup}(1^\lambda)$ is $\overline{\mathsf{DamJur}}.\mathsf{Setup}(1^\lambda, 1^\zeta)$, by choosing smallest ζ s.t. $M^\zeta \geq (2^{2b+\lambda+1})^{\lambda+1}$. Also let B be the largest multiple of 2^b satisfying $M^\zeta \geq B^{\lambda+1}$. Then all the three requirements on B can be satisfied.
- $\overline{\mathsf{vDJ}}.\mathsf{Gen}$ is $\overline{\mathsf{DamJur}}.\mathsf{Gen}$.
- $\overline{\mathsf{vDJ}}.\mathsf{Enc}(\mathsf{sk}, \mathbf{V}) = \overline{\mathsf{DamJur}}.\mathsf{Enc}(\mathsf{sk}, \mathsf{encode}(\mathbf{V}))$.
- $\overline{\mathsf{vDJ}}.\mathsf{Dec}(\mathsf{sk}, c) = \mathsf{decode}(\overline{\mathsf{vDJ}}.\mathsf{Dec}(\mathsf{sk}, c))$.
- $\overline{\mathsf{vDJ}}.\mathsf{Inv}(\mathsf{tp}, c) = (g, \mathsf{decode}(v))$ for $(g, v) = \overline{\mathsf{DamJur}}.\mathsf{Inv}(\mathsf{tp}, c)$.

Now we are ready to present the bit-decomposition gadget in Fig. 4.11, 4.12.

Lemma 4.8. *There is a secure bit-decomposition gadget (Fig. 4.11, 4.12) over ring $\mathbb{Z}_{2^b}$, under DCR assumption in the programable random oracle model. The table size is $O(b\lambda_{\mathsf{DCR}})$.*

Proof. First verify correctness in the real world. Inductively, the evaluator gets $(\mathbf{L}^{(i)}, \bar{x}^{(i)}) = (x_{i:b}\mathbf{\Delta} + \mathbf{A}^{(i)}, x_{i:b} + \alpha^{(i)})$. The least significant bits of $\mathbf{L}^{(i)}$ allows the evaluator to decrypt $\mathsf{C}_{i,\bar{x}^{(i)}}$, and gets

$$\mathbf{l}_i = \mathsf{K}(x_i), \quad \hat{x}_i = x_i + r_i, \quad h^{(i)} = (g^{\Delta})^{x_i+r_i} g^{(i)}, .$$

$$\begin{aligned}
&(\mathsf{ct}^{\Delta})^{\hat{x}_i}\mathsf{ct}^{(i)} \\
&= \Big(\overline{\mathsf{vDJ}}.\mathsf{Enc}(g^{\Delta}, (\mathbf{\Delta}, 1))\Big)^{x_i+r_i} \cdot \overline{\mathsf{vDJ}}.\mathsf{Enc}(g^{(i)}, 2^{b+\lambda} - r_i(\mathbf{\Delta}, 1) + (\mathbf{S}^{(i)}, s^{(i)})) \\
&= \overline{\mathsf{vDJ}}.\mathsf{Enc}((g^{\Delta})^{x_i+r_i} g^{(i)}, x_i(\mathbf{\Delta}, 1) + (\mathbf{S}^{(i)}, s^{(i)}) + 2^{b+\lambda})
\end{aligned}$$

From the decryption of $\overline{\mathsf{vDJ}}$ ciphertext, the evaluator gets

$$(\mathbf{D}^{(i)}, d^{(i)}) = (x_i\mathbf{\Delta} + \mathbf{S}^{(i)}, x_i + s^{(i)}) + 2^{b+\lambda}$$

The label of the next inductive step is correctly computed as

$$\begin{aligned}
(\mathbf{L}^{(i+1)}, \bar{x}^{(i+1)}) &= \lfloor \frac{(\mathbf{L}^{(i)}, \bar{x}^{(i)}) - (\mathbf{D}^{(i)}, d^{(i)}) \bmod 2^{b-i}}{2} \rfloor \\
&= \lfloor \frac{(2x_{i+1:b}\mathbf{\Delta} + \mathbf{A}^{(i)} - \mathbf{S}^{(i)}, 2x_{i+1:b} + \alpha^{(i)} - s^{(i)}) \bmod 2^{b-i}}{2} \rfloor \\
&= \Big(x_{i+1:b}\mathbf{\Delta} + \lfloor \frac{\mathbf{A}^{(i)} - \mathbf{S}^{(i)}}{2} \rfloor, x_{i+1:b} + \lfloor \frac{\alpha^{(i)} - s^{(i)}}{2} \rfloor\Big) \bmod 2^{b-i-1} \\
&= (x_{i+1:b}\mathbf{\Delta} + \mathbf{A}^{(i+1)}, x_{i+1:b} + \alpha^{(i+1)}) \bmod 2^{b-i-1} .
\end{aligned}$$

The security follows from the following arguments:

- $\hat{x}_i = x_i + r_i$ is smudged by $r_i \leftarrow [2^{\lambda}]$ in the real world. Simulating it as $\hat{x} \leftarrow [2^{\lambda}]$ only introduces $2^{-\lambda}$ statistical error.
- In the real world, $(g^{(i)}, \mathbf{S}^{(i)}, s^{(i)})$ is statistically close to uniform, the randomness comes from the outcome of $\mathsf{H}(\mathsf{seed}^{(i)}, \mathsf{id}, i)$. Thus $h^{(i)}, \mathbf{D}^{(i)}, d^{(i)}$ can be simulated at random, because $h^{(i)}$ is one-time padded $g^{(i)}$ and $(\mathbf{D}^{(i)}, d^{(i)})$ is smudged by $(\mathbf{S}^{(i)}, s^{(i)})$.
- $\mathsf{seed}^{(i)}$ can be simulated at random, because it is a fresh uniform sample in the real world.
- The programing of H is on the random point $\mathsf{seed}^{(i)}$, which has not been queried by the distinguisher with overwhelming probability. The programmed value is determined from the correctness guarantee.

The table consists of $O(b)$ ciphertexts, each of which is $O(\lambda_{\mathsf{DCR}})$ bit long, and $O(b)$ seeds, each of which is $O(\lambda)$ bit long. The total table size is $O(\lambda_{\mathsf{DCR}} b)$. □

Combining the bit-composition gadget in Lem. 4.7 and the bit-decomposition gadget in Lem. 4.8 produces a mix GC scheme, as stated by the following theorem.

Theorem 4.4. *There is a secure mixed GC for $\mathbb{Z}_{2^b}$ under DCR assumption in the programmable random oracle model, such that each addition gate costs no communication, each multiplication/bit-decomposition gate costs $O(\lambda_{\mathsf{DCR}} b)$ communication, and each bit-composition gate costs $O(\lambda_{\mathsf{DCR}} b + \lambda^2)$ communication.*

Our mixed GC for $\mathbb{Z}_{2^b}$ implies a mixed GC for any $\mathbb{Z}_N$ for any $N \approx 2^b$, using the emulation technique discussed in Sec. 4.4.2.

Corollary 4.1. *For any $N \leq 2^b$, there is a secure mixed GC for $\mathbb{Z}_N$ under DCR assumption in the programmable random oracle model, such that each addition/multiplication/bit-decomposition gate costs $O(\lambda_{\mathsf{DCR}} b)$ communication, and each bit-composition gate costs $O(\lambda_{\mathsf{DCR}} b + \lambda^2)$ communication.*

Garbling algorithm $\mathsf{miniBC}_k.\mathsf{Garb}$ takes boolean keys K_j for $j \in [\log p]$, and an arithmetic key $\mathsf{AK}^{\mathsf{L}} = (\mathbf{A}, \mathbf{B})$ as inputs. Let α_j denote the mask bit of K_j.

- Sample random $\mathbf{B}_j$ for $j \in [\log p]$, satisfying $\sum_j \mathbf{B}_j \bmod p^k = \mathbf{B}$.
- For each $j \in [\log p]$, for each $\beta \in \{0,1\}$, compute

$$\mathsf{C}_{j,\beta+\alpha_j \bmod 2} \leftarrow \mathsf{H}(\mathsf{K}_j(\beta),\ (\mathsf{id}, j)) \oplus (\mathbf{A}2^j\beta + \mathbf{B}_j \bmod p^k)$$

- Output table $\mathsf{Tab} = (\mathsf{C}_{j,\beta})_{j\in[\log p],\beta\in\{0,1\}}$

Evaluation algorithm $\mathsf{miniBC}_k.\mathsf{Eval}$ takes input labels $(\mathbf{l}_j, \bar{x}_j)$ for $j \in [\log p]$ and a table Tab as inputs.

- For $j \in [\log p]$, compute $\mathbf{L}_j \leftarrow \mathsf{H}(\mathbf{l}_j,\ (\mathsf{id}, j)) \oplus \mathsf{C}_{j,\bar{x}_j}$.
- Output arithmetic label $\mathbf{L} = \sum_j \mathbf{L}_j \bmod p^k$.

Simulation algorithm $\mathsf{miniBC}_k.\mathsf{Sim}$ takes input labels $(\mathbf{l}_j, \bar{x}_j)$ for $j \in [\log p]$ and arithmetic label $\mathbf{L}$ as inputs.

- Sample random $\mathbf{L}_j$ for $j \in [\log p]$, satisfying $\sum_j \mathbf{L}_j \bmod p^k = \mathbf{L}$.
- The active ciphertexts in the table Tab are set as

$$\mathsf{C}_{j,\bar{x}_j} = \mathsf{H}(\mathbf{l}_j,\ (\mathsf{id}, j)) \oplus \mathbf{L}_j$$

  The rest are simulated by random strings.

Figure 4.4: The mini bit-composition gadget.

Garbling algorithm $\mathsf{BD.Garb}$ takes an arithmetic key $\mathsf{AK} = (\mathbf{A}, \alpha)$ and $k \cdot \lceil \log p \rceil$ boolean keys $\mathsf{K}_{i,j}$ for $i \in [p], j \in [\log p]$ as inputs.

- Let $\mathbf{A}^{(0)} = \mathbf{A}$. For each $1 \le i < k$, samples $\mathbf{A}^{(i)} \leftarrow (\mathbb{Z}_{p^{k-i}})^{\lambda}$.
- Let $\alpha^{(0)} = \alpha$. For each $1 \le i < k$, samples $\alpha^{(i)} \leftarrow \mathbb{Z}_{p^{k-i}}$.
- For each $i \in [k], j \in [\log p]$, for each $\beta \in [p]$, compute

$$\mathsf{C}_{i,\beta+\alpha^{(i)} \bmod p} \leftarrow \mathsf{H}(\mathbf{\Delta}\beta + \mathbf{A}^{(i)} \bmod p,\ (\mathsf{id}, i)) \oplus (\mathsf{K}_{i,j}(\beta_j) \text{ for } j \in [\log p])$$

- For each $0 \le i < k-1$, define affine function $\mathsf{DK}^{(i)}$

$$\mathsf{DK}^{(i)}(\beta) = (\mathbf{\Delta}\beta + \mathbf{A}^{(i)} - p\mathbf{A}^{(i+1)},\ \beta + \alpha^{(i)} - p\alpha^{(i+1)}) \bmod p^{k-i},$$

  compute table $\mathsf{tb}_i \leftarrow \mathsf{miniBC}_{k-i}.\mathsf{Garb}(\mathsf{K}_{i,j} \text{ for } j \in [\log p], \mathsf{DK}^{(i)})$.
- Output table Tab consisting of $(\mathsf{C}_{i,\beta})_{i \in [k], \beta \in [p]}$ and $(\mathsf{tb}_i)_{i \in [k-1]}$

Figure 4.5: The bit-decomposition gadget in ring $\mathbb{Z}_{p^k}$.

Evaluation algorithm $\mathsf{BD.Eval}$ takes input label $(\mathbf{L}, \bar{x})$ and a table Tab as inputs.

- Let $\mathbf{L}^{(0)} := \mathbf{L}$, $\bar{x}^{(0)} = \bar{x}$.
- For $i = 0, 1, 2, \ldots, k-1$:

  Compute $(\mathbf{l}_{i,j} \text{ for } j \in [\log p]) \leftarrow \mathsf{H}(\mathbf{L}^{(i)} \bmod p,\ (\mathsf{id}, i)) \oplus \mathsf{C}_{i,\bar{x}^{(i)} \bmod p}$.

  If $i < k-1$, compute $(\mathbf{D}^{(i)}, d^{(i)}) \leftarrow \mathsf{miniBC}_{k-i}.\mathsf{Eval}(\mathbf{l}_{i,j} \text{ for } j \in [\log p], \mathsf{tb}_i)$

  $$\mathbf{L}^{(i+1)} = (\mathbf{L}^{(i)} - \mathbf{D}^{(i)} \bmod p^{k-i})/p\,, \qquad \bar{x}^{(i+1)} = (\bar{x}^{(i)} - d^{(i)} \bmod p^{k-i})/p\,.$$

- Output boolean labels $\mathbf{l}_{i,j}$ for $i \in [p], j \in [\log p]$.

Simulation algorithm $\mathsf{BD.Sim}$ takes arithmetic label $(\mathbf{L}, \bar{x})$ and boolean labels $\mathbf{l}_{i,j}$ for $i \in [p], j \in [\log p]$ as inputs.

- Let $(\mathbf{L}^{(0)}, \bar{x}^{(0)}) = (\mathbf{L}, \bar{x})$.
- Sample random $\mathbf{L}^{(i)} \leftarrow (\mathbb{Z}_{p^{k-i}})^{\lambda}, \bar{x}^{(i)} \leftarrow \mathbb{Z}_{p^{k-i}}$ for each $1 \le i < k$.
- The active ciphertexts in the table Tab are set as

  $$\mathsf{C}_{i,\bar{x}^{(i)} \bmod p} = \mathsf{H}(\mathbf{L}^{(i)} \bmod p,\ (\mathsf{id}, i)) \oplus (\mathbf{l}_{i,j} \text{ for } j \in [\log p])$$

  The rest are inactive ciphertexts, and are simulated by random strings.
- For each $0 \le i < k-1$, compute

  $$(\mathbf{D}^{(i)}, d^{(i)}) \leftarrow (\mathbf{L}^{(i)} - p\mathbf{L}^{(i+1)},\ \bar{x}^{(i)} - p\bar{x}^{(i+1)}) \bmod p^{k-i}$$

  and simulate tb_i by $\mathsf{tb}_i \leftarrow \mathsf{miniBC}_{k-i}.\mathsf{Sim}(\mathbf{l}_{i,j} \text{ for } j \in [\log p], (\mathbf{D}^{(i)}, d^{(i)}))$.

Figure 4.6: The bit-decomposition gadget in ring $\mathbb{Z}_{p^k}$, continued.

The gadget is parameterized by coefficients $c_0, \ldots, c_{m-1} \in \mathbb{Z}_{p^k}$.

Garbling algorithm $\mathsf{linBC.Garb}$ takes boolean keys $\mathsf{K}_0, \ldots, \mathsf{K}_{m-1}$, and an arithmetic key $\mathsf{AK}^{\mathsf{L}} = (\mathbf{A}, \mathbf{B})$ as inputs. Let α_i denote the mask bit of K_i.

- Sample random $\mathbf{B}_i$ for $i \in [m]$, satisfying $\sum_i \mathbf{B}_i \bmod p^k = \mathbf{B}$.
- For each $i \in [m]$, for each $\beta \in \{0, 1\}$, compute

$$\mathsf{C}_{i,\beta+\alpha_i \bmod 2} \leftarrow \mathsf{H}(\mathsf{K}_i(\beta),\ (\mathsf{id}, i)) \oplus (\mathbf{A} c_i \beta + \mathbf{B}_i \bmod p^k)$$

- Output table $\mathsf{Tab} = (\mathsf{C}_{i,\beta})_{i \in [m], \beta \in \{0,1\}}$.

Evaluation algorithm $\mathsf{linBC.Eval}$ takes input labels $(\mathbf{l}_i, \bar{x}_i)$ for $i \in [m]$ and a table Tab as inputs.

- For $i \in [m]$, compute $\mathbf{L}_i \leftarrow \mathsf{H}(\mathbf{l}_i,\ (\mathsf{id}, i)) \oplus \mathsf{C}_{i,\bar{x}_i}$.
- Output arithmetic label $\mathbf{L} = \sum_i \mathbf{L}_i \bmod p^k$.

Simulation algorithm $\mathsf{linBC.Sim}$ takes input labels $(\mathbf{l}_i, \bar{x}_i)$ for $i \in [m]$ and arithmetic label $\mathbf{L}$ as inputs.

- Sample random $\mathbf{L}_i$ for $i \in [m]$, satisfying $\sum_i \mathbf{L}_i \bmod p^k = \mathbf{L}$.
- The active ciphertexts in the table Tab are set as

$$\mathsf{C}_{i,\bar{x}_i} = \mathsf{H}(\mathbf{l}_i,\ (\mathsf{id}, i)) \oplus \mathbf{L}_i$$

  The rest are inactive ciphertexts, and are simulated by random strings.

Figure 4.7: The linear bit-composition gadget over ring $\mathbb{Z}_{p^k}$.

Standard Public-Key Notation	Private-Key Notation
public key	public parameter pp
secret key	trapdoor tp
encryption randomness	secret key sk
decryption using encryption randomness	decryption Dec
decryption	inversion using trapdoor Inv

Figure 4.8: Notation mapping for variants of Damgård-Jurik encryption.

The gadget is parameterized by coefficients $c_0, \ldots, c_{m-1} \in \mathbb{Z}_{2^b}$.

Garbling algorithm $\mathsf{BC.Garb}$ takes boolean keys $\mathsf{K}_0, \ldots, \mathsf{K}_{m-1}$, and an arithmetic key $\mathsf{AK}^{\mathsf{L}} = (\mathbf{A}, \mathbf{B})$ as inputs. Let α_i denote the mask bit of K_i.

- (global step) Generate $M, \zeta, g, p'q'$ using $\mathsf{vDJ.Setup}$, while setting ζ such that $M^\zeta \geq 2^{\ell(2b+\lambda+1)}$. Add (M, ζ, g) to the beginning of the garbled circuit.
- Sample keys $\mathsf{sk}^A, \mathsf{sk}^B_0, \ldots, \mathsf{sk}^B_{m-1} \leftarrow [p'q']$. Let $\mathsf{sk}^B := \sum_i \mathsf{sk}^B_i$.

  Sample masks $r_0, \ldots, r_{m-1} \leftarrow [2^\lambda]$, $\mathbf{R} \leftarrow [2^{b+2\lambda}]^\ell$. Let $r = \sum_i c_i r_i$. Compute

  $$\mathsf{ct}^A = \mathsf{vDJ.Enc}(\mathsf{sk}^A, \mathbf{A}), \qquad \mathsf{ct}^B = \mathsf{vDJ.Enc}(\mathsf{sk}^B, (2^{2b+\lambda} - r\mathbf{A}) + \mathbf{B} + 2^b\mathbf{R}).$$

- For each $i \in [m]$, for each $\beta \in \{0,1\}$, compute

  $$\mathsf{C}_{i,\beta+\alpha_i \bmod 2} \leftarrow \mathsf{H}(\mathsf{K}_i(\beta),\ (\mathsf{id}, i)) \oplus (x_i + r_i, c_i(x_i + r_i)\mathsf{sk}^A + \mathsf{sk}^B_i \bmod p'q')$$

- Output table $\mathsf{Tab} = ((\mathsf{C}_{i,\beta})_{i\in[m],\beta\in\{0,1\}}, \mathsf{ct}^A, \mathsf{ct}^B)$.

The modifications with respect to Fig. 4.7 are highlighted.

Figure 4.9: The bit-composition gadget based on vDJ.

Evaluation algorithm $\mathsf{BC.Eval}$ takes input labels $(\mathbf{l}_i, \bar{x}_i)$ for $i \in [m]$ and a table Tab as inputs.

- For $i \in [m]$, compute $(\hat{x}_i, \mathsf{sk}_i) \leftarrow \mathsf{H}(\mathbf{l}_i,\ (\mathsf{id}, i)) \oplus \mathsf{C}_{i,\bar{x}_i}$.
- Compute $\mathsf{sk} = \sum_i \mathsf{sk}_i$, $\hat{x} = \sum_i c_i \hat{x}_i$.
- Output label $\mathbf{L} = \hat{\mathbf{L}} \bmod 2^b$, where $\hat{\mathbf{L}} = \mathsf{vDJ.Dec}(\mathsf{sk}, (\mathsf{ct}^A)^{\hat{x}} \mathsf{ct}^B)$.

Simulation algorithm $\mathsf{BC.Sim}$ takes input labels $(\mathbf{l}_i, \bar{x}_i)$ for $i \in [m]$ and arithmetic label $\mathbf{L}$ as inputs.

- (global step) Sample (M, ζ, g) using the $\mathsf{vDJ.Setup}$.
- Sample random $\hat{x}_0, \ldots, \hat{x}_{m-1} \leftarrow [2^\lambda]$. Let $\hat{x} = \sum_i c_i \hat{x}_i$.

  Sample random $\mathsf{sk}_0, \ldots, \mathsf{sk}_{m-1} \leftarrow [M/4]$. Let $\mathsf{sk} = \sum_i \mathsf{sk}_i$.
- Sample masks $\mathbf{R} \leftarrow [2^{b+\lambda}]^\ell$, let $\hat{\mathbf{L}} = \mathbf{L} + 2^b \mathbf{R}$.

  Simulate ct^A by randomly sample $\mathsf{ct}^A \leftarrow \mathsf{QR}_{M^{\zeta+1}}$. Simulate ct^B as

$$\mathsf{ct}^B = \mathsf{vDJ.Enc}(\mathsf{sk}, \hat{\mathbf{L}})/(\mathsf{ct}^A)^{\hat{x}} \quad (\text{over } \mathbb{Z}^*_{M^{\zeta+1}}).$$

- The active ciphertexts in the table Tab are set as

$$\mathsf{C}_{i,\bar{x}_i} = \mathsf{H}(\mathbf{l}_i,\ (\mathsf{id}, i)) \oplus (\hat{x}_i, \mathsf{sk}_i)$$

  The rest are inactive ciphertexts, and are simulated by random strings.

Figure 4.10: The bit-composition gadget based on vDJ, continued.

Garbling algorithm $\mathsf{BD.Garb}$ takes an arithmetic key $\mathsf{AK} = (\mathbf{A}, \alpha)$ and b boolean keys $\mathsf{K}_0, \ldots, \mathsf{K}_{b-1}$ as inputs.

- (global step) Generate $M, \zeta, p'q'$ using $\overline{\mathsf{vDJ}}.\mathsf{Setup}$, while setting ζ, B properly. Sample key $g^{\Delta} \leftarrow \mathbb{Z}_M^*$ and compute $\mathsf{ct}^{\Delta} = \overline{\mathsf{vDJ}}.\mathsf{Enc}(g^{\Delta}, (\mathbf{\Delta}, 1))$.

  Add $M, \zeta, \mathsf{ct}^{\Delta}$ to the beginning of the garbled circuit.
- Let $\mathbf{A}^{(0)} = \mathbf{A}$, $\alpha^{(0)} = \alpha$.
- For each $0 \le i < b$, sample $r_i \leftarrow [2^{\lambda}]$, $\mathsf{seed}^{(i)} \leftarrow \{0,1\}^{\lambda}$.

  Compute $\mathsf{ct}^{(i)} = \mathsf{H}(\mathsf{seed}^{(i)}, (\mathsf{id}, i)) \in \mathbb{Z}_{M^{\zeta+1}}$. Find $g^{(i)}, \mathbf{S}^{(i)}, s^{(i)}$ satisfying

$$(g^{(i)}, (2^{b+\lambda} - r_i(\mathbf{\Delta}, 1)) + (\mathbf{S}^{(i)}, s^{(i)}) = \overline{\mathsf{vDJ}}.\mathsf{Inv}(\mathsf{tp}, \mathsf{ct}^{(i)}) \,.$$

  Resample $\mathsf{seed}^{(i)}$ if $(\mathbf{S}^{(i)}, s^{(i)}) \notin [B - 2^{b+\lambda}]^{\lambda+1}$ to prevent overflow. Set

$$\mathbf{A}^{(i+1)} = \lfloor \frac{\mathbf{A}^{(i)} - \mathbf{S}^{(i)}}{2} \rfloor \bmod 2^{b-i-1} \qquad \alpha^{(i+1)} = \lfloor \frac{\alpha^{(i)} - s^{(i)}}{2} \rfloor \bmod 2^{b-i-1} \,.$$

- For each $0 \le i < b$, for each $\beta \in \{0,1\}$, compute

$$\mathsf{C}_{i,\beta+\alpha^{(i)} \bmod 2} \leftarrow \mathsf{H}(\mathbf{\Delta}\beta + \mathbf{A}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus (\mathsf{K}_i(\beta), \beta + r_i, (g^{\Delta})^{\beta+r_i} g^{(i)})$$

- Output table $\mathsf{Tab} = ((\mathsf{C}_{i,\beta})_{i\in[b],\beta\in\{0,1\}}, (\mathsf{seed}^{(i)})_{i\in[b-1]})$

The modifications with respect to Fig. 4.1, 4.2 are highlighted.

Figure 4.11: The bit-decomposition gadget based on $\overline{\mathsf{vDJ}}$

Evaluation algorithm $\mathsf{BD.Eval}$ takes input label $(\mathbf{L}, \bar{x})$ and a table Tab as inputs.

- Let $\mathbf{L}^{(0)} := \mathbf{L}$, $\bar{x}^{(0)} = \bar{x}$.
- For $i = 0, 1, 2, \ldots, b-1$:

  Compute $(\mathbf{l}_i, \hat{x}_i, h^{(i)}) \leftarrow \mathsf{H}(\mathbf{L}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus \mathsf{C}_{i, \bar{x}^{(i)} \bmod 2}$. If $i < b-1$, compute $\mathsf{ct}^{(i)} = \mathsf{H}(\mathsf{seed}^{(i)}, \mathsf{id}, i)$, $(\mathbf{D}^{(i)}, d^{(i)}) \leftarrow \overline{\mathsf{vDJ}}.\mathsf{Dec}(h^{(i)}, (\mathsf{ct}^{\Delta})^{\hat{x}_i}\mathsf{ct}^{(i)})$

  $$(\mathbf{L}^{(i+1)}, \bar{x}^{(i+1)}) = \lfloor((\mathbf{L}^{(i)}, \bar{x}^{(i)}) - (\mathbf{D}^{(i)}, d^{(i)}) \bmod 2^{b-i})/2\rfloor,$$

- Output boolean labels $\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l}_{b-1}$.

Simulation algorithm $\mathsf{BD.Sim}$ takes arithmetic label $(\mathbf{L}, \bar{x})$ and boolean labels $\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l}_{b-1}$ as inputs.

- (global step) Generate $M, \zeta, p'q'$ using $\overline{\mathsf{vDJ}}.\mathsf{Setup}$, while setting ζ, B properly. Simulate ct^{Δ} as a random ciphertext.
- Let $(\mathbf{L}^{(0)}, \bar{x}^{(0)}) = (\mathbf{L}, \bar{x})$.
- Sample random $\hat{x}_i \leftarrow [2^{\lambda}]$, $\mathsf{seed}^{(i)} \leftarrow \{0,1\}^{\lambda}$, $\mathbf{D}^{(i)} \leftarrow [B]^{\lambda}$, $d^{(i)} \leftarrow [B]$ for each $i \in [b-1]$. Program H so that

  $$\overline{\mathsf{vDJ}}.\mathsf{Enc}(h^{(i)}, (\mathbf{D}^{(i)}, d^{(i)})) = (\mathsf{ct}^{\Delta})^{\hat{x}_i}\, \mathsf{H}(\mathsf{seed}^{(i)}, \mathsf{id}, i) \quad (\text{in } \mathbb{Z}_{M^{\zeta+1}})$$

- The active ciphertexts in the table Tab are set as

  $$\mathsf{C}_{i, \bar{x}^{(i)} \bmod 2} = \mathsf{H}(\mathbf{L}^{(i)} \bmod 2,\ (\mathsf{id}, i)) \oplus (\mathbf{l}_i, \hat{x}_i, h^{(i)})$$

  The rest are inactive ciphertexts, and are simulated by random strings.

Figure 4.12: The bit-decomposition gadget based on $\overline{\mathsf{vDJ}}$, continued.

Chapter 5

HOW TO COMPRESS GARBLE CIRCUIT LABELS, EFFICIENTLY

This chapter is adapted from published work [DLL25].

5.1 Introduction

Introduced by Yao in the 1980s [Yao82, BHR12], a garbling scheme allows efficiently transforming a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ into a *garbled circuit* $\widehat{C}$ and a pair of short, λ-bit, keys $(\mathbf{K}[i,0], \mathbf{K}[i,1])$ for every input bit i, where λ is the security parameter. This transformation ensures that $\widehat{C}$, along with the input labels $\mathbf{K}[\mathbf{x}] = (\mathbf{K}[1,x_1], \cdots, \mathbf{K}[n,x_n])$, reveals only the output $\mathbf{y} = C(\mathbf{x})$ and no other information about the input $\mathbf{x}$. Over the years, Garbled circuits have found a diverse set of applications, in particular for building efficient constant-round multi-party computation protocols [Yao86, BMR90].

Although originally viewed as impractical, tremendous efficiency improvements in the past two decades has brought it to the forefront of practical and deployable cryptography. Much of this research focused on reducing the size of garbled circuits, and hence the communication costs of transferring them, providing a deep understanding on both the theoretical and practical limits [BMR90, NPS99, KS08a, PSSW09, KMR14, GLNP15, ZRE15, RR21, GKP+13b, HLL23, AJS17, KLW15, BCG+18, JLL23]. However, efficiency related to the other "half" of garbled circuits, namely the input labels, has been relatively neglected so far, and is the focus of this work.

Consider the following simple use case: Many research institutes each want to perform different analyses, described by circuits C_i, over a common sensitive database $\mathbf{x}$, e.g. patient records. Due to the sensitive nature of the data, only the results $C_i(\mathbf{x})$ of approved analysis

may be revealed. Garbled circuits offer a non-interactive solution: The data provider can first provide different garbled circuits $\widehat{C}_i$ to the research institutes according to their proposed analysis, and later broadcast a single set of input labels $\mathbf{K}[\mathbf{x}]$, e.g. through a blockchain. Since the dataset $\mathbf{x}$ may be very large, it's desirable to minimize the size of labels $\mathbf{K}[\mathbf{x}]$, which, naively, would require $\lambda \cdot |\mathbf{x}|$ bits. Applebaum, Ishai, Kushilevitz, and Waters [AIKW13] (AIKW) were the first to ask the following natural question: Is rate-λ, i.e., sending λ bits per input bit, optimal?

Optimal Rate $1 + o(1)$. It turns out that rate-λ is far from optimal. AIKW achieves compression of input labels in an *online-offline setting* based on a variety of public key cryptography assumptions including RSA, LWE, or DDH. They showed that after collecting the data $\mathbf{x}$, i.e., in the *online phase*, it suffices to send just $|\mathbf{x}| + \text{poly}(\lambda)$ bits. This requires publishing a potentially large amount of information before seeing $\mathbf{x}$, i.e., in the *offline phase*. Subsequent works [GS18a, GOS18] show how to achieve this under weaker assumptions (Factoring or CDH), which however results in expensive non-black-box use of cryptographic tools. While all of these results achieve an *optimal* online rate of $1 + o(1)$, their techniques come at the cost of an offline communication $\Omega(|\mathbf{x}| \cdot \lambda)$.

Therefore, a question left open by AIKW and [GS18a, GOS18] is whether the *overall* rate (combining offline and online) can actually be reduced to less than λ. At first sight, it may seem impossible to communicate the labels $\mathbf{K}[\mathbf{x}]$ (which have size $|\mathbf{x}| \cdot \lambda$) using less than $|\mathbf{x}| \cdot \lambda$ bits overall. This intuition is however incorrect, because the $2|\mathbf{x}|$ input keys might be generated in a pseudorandom way that is still sufficient for garbled circuits security.

The above intuition can indeed be realized using a tool called *projective PRG* (pPRG) proposed recently by Applebaum et al. (ABI+) [ABI+23]. A pPRG with public parameter pp is a pseudorandom generator with the new power that the seed sd can be "projected" to a subset T of all output bits, such that the projected seed sd_T expands to these output bits indexed by T, while keeping the remaining output bits pseudorandom. pPRG with both succinct, i.e., poly(λ)-size, public parameters and projected seeds promises to eliminate the offline phase altogether, and simultaneously maintaining $1 + o(1)$ online rate. Built upon the

techniques in AIKW, ABI+ [ABI+23] constructed pPRGs with different levels of succinctness based on several public key assumptions.

Summarizing [AIKW13, ABI+23], we now have compression techniques with two levels of communication efficiency. *1)* Optimal overall-rate $1 + o(1)$ is achieved, based either on RSA, or on indistinguishability obfuscation (iO) combined with somewhere statistically binding hash functions (SSBH). This means each label can be transferred by sending essentially a single bit! *2)* Optimal online rate $1 + o(1)$ is attained, albeit with sub-optimal $\Omega(|\mathbf{x}| \cdot \lambda)$ offline cost, based on LWE and DDH. The DDH-based technique can be further improved using bilinear groups, enabling *reusing* the large offline communication, achieving amortized overall-rate $1 + o(1)$ over multiple instances of garbled circuits. See Table 5.1 for detailed comparison.

Theory vs Practice. Given the importance of practically efficient garbled circuits, it is exciting to try to apply these compression techniques in the wild. Unfortunately, the computational costs of current techniques are extremely high, establishing only feasibility, but not practicality.

Specifically, the methods described in [AIKW13, ABI+23] require performing a large number of expensive public key operations, namely group exponentiations (in either RSA or DDH groups), inner products of long vectors over $\mathbb{Z}_p$ (LWE), or pairing operations (bilinear DDH), while the methods in [GS18a, GOS18] are prohibitively expensive due to non-black-box cryptography. To underline these issues, take the RSA-based scheme as an example, which is one of the currently most efficient methods. It requires the receiver to run $O(|\mathbf{x}| \cdot \log |\mathbf{x}|)$ RSA-exponentiations[1] (with large exponents) in order to recover the input labels. However, the concrete cost of this is very high: Suppose the input has $100K$ bits i.e., $|\mathbf{x}| \geq 10^5$, and one RSA-exponentiation consumes $1ms$. Then the receiver can only process 60 input labels per second, and this rate decreases even further for larger inputs. While the DDH-based scheme could have similar performance (assuming $0.1ms$ per scalar multiplication on elliptic

[1] This assumes some algorithmic optimization not described in [AIKW13, ABI+23].

curves [BCLN16]), it does come with significant offline communication. All other schemes have even more computational overhead.

Motivated by the state-of-affairs, the overarching goal of this work is:

Can we transfer input labels in online- or overall-rate less than λ, efficiently*?*

We remark that the state-of-affairs is reminiscent to the current landscape of research on the minimal size of garbled circuits. In theory, optimal poly(λ, n, m)-size garbled circuits is feasible (with n, m the input/output length), based on LWE or indistinguishability obfuscation and puncturable PRF [GKP+13b, HLL23, AJS17, KLW15, BCG+18, JLL23]. However, these schemes are computationally prohibitive. In practice, optimized versions of Yao's garbled circuits are far more efficient, despite their large communication costs. Similarly, current theoretically optimal input-label compression techniques have not yet brought benefits to practical efficiency. Narrowing the gap is the aim of this work.

Our Results in a Nutshell. We make progress towards the above overarching goal by presenting a new efficient compression technique with optimal overall-rate, $1 + o(1)$, based on RingLWE (depending on the choice of ring parameters), in the Random Oracle Model (ROM).

- *Lightweight communication:* The optimal overall-rate $1+o(1)$ is achieved. While, unlike pPRG-based solutions, our scheme does make use of offline communication, it has the novel feature that information sent in the offline phase is either reusable or lightweight, with amortized offline-rate $o(1)$.
- *Concretely Efficient:* By leveraging RingLWE packing, our online computation is dominated by $O(\frac{\log |\mathbf{x}| \cdot \lambda}{n \log q})$ ring operations per input bit (where n is the RingLWE degree), a potentially *fractional* number. Indeed, in concrete settings an average of 0.14 ring operations is performed per input bit in our implementation.
- *Implementation:* Our implementation demonstrates the practical efficiency of our method. For transferring the labels of $700K$ input bits, it takes about $0.90\mu s$ per

bit for the garbler to compress and $1.77\mu s$ per bit for the evaluator to reconstruct, on a single-core PC.

As a further application, we construct *preprocessing garbled circuits*: relying on an offline phase with $\tilde{O}(\lambda|C|)$ communication, the online phase (that starts once the plaintext circuit C becomes known to both parties) has a reduced communication cost of only poly(λ) bits, independent of the circuit size. These ideas naturally yield maliciously secure two-party computation protocols in the preprocessing model, with optimally *succinct online communication*. Namely, only poly(λ) bits are exchanged after the circuit C becomes available, and $|\mathbf{x}_A| + |\mathbf{x}_B| + \text{poly}(\lambda)$ bits are exchanged after the two parties receive their respective inputs $\mathbf{x}_A$ and $\mathbf{x}_B$. While previous techniques for sublinear-communication 2PC did not rely on preprocessing, all of them require computationally expensive tools, such as, FHE, iO, HSS. Our protocols are much more concretely efficient.

On the Importance of Compressing Input Labels. Before moving on to describing our results in more detail, we want to call for further study on the efficiency of transferring input labels. This topic has so far had a limited history of study, perhaps due to the school of thought that the cost of transferring the input labels is dominated by that of transferring the garbled circuit. However, there are several strong reasons motivating its study. First, performance optimization entails saving costs in every possible avenue in practice. In big data applications the input might even be almost as large as the circuit. Second, in applications that allow preprocessing, the real-time performance becomes mostly dominated by the time required for transferring input labels. Third, the lack of theoretical understanding of this basic and natural question about garbled circuits is unsatisfactory; any progress here may lead to other applications. Finally, we are optimistic that in the future, practical garbled circuits with less than λ bit per gate, or even fixed polynomial size, might become possible. Then, the costs for transferring input labels may even outweigh that of the garbled circuit.

Our Results in More Detail.

The key tool in this work, following AIKW techniques, are what we call *Batch-Select* schemes[2]. It enables the sender, Alice, to encode the keys $\mathbf{K}$ in the offline stage:

$$\mathsf{Sel.Enc}_1(\{\mathbf{K}[i,1]-\mathbf{K}[i,0]\}) \to (\mathsf{ct}_1,\mathsf{st}_1)\ , \quad \mathsf{Sel.Enc}_2(\{\mathbf{K}[i,0]\}) \to (\mathsf{ct}_2,\mathsf{st}_2)\ .$$

(Note that the 0-keys $\mathbf{K}[i,0]$ and the key offsets $\mathbf{K}[i,1]-\mathbf{K}[i,0]$ are encrypted separately, which will be convenient later.) Alice publishes the ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ and keeps the secret state $\mathsf{st} = (\mathsf{st}_1, \mathsf{st}_2)$. After $\mathbf{x}$ arrives, Alice generates a very succinct key $\mathsf{Sel.KeyGen}(\mathsf{st}, \mathbf{x}) \to \mathsf{sk}_\mathbf{x}$ of just $\mathrm{poly}(\lambda)$ size. The pair $(\mathsf{sk}_\mathbf{x}, \mathbf{x})$ enables (partial) decryption of ct, revealing exactly the input labels $\mathsf{Sel.Dec}(\mathsf{ct}, \mathsf{sk}_\mathbf{x}, \mathbf{x}) \to \mathbf{K}[\mathbf{x}] = \{(\mathbf{K}[i,1]-\mathbf{K}[i,0]) \cdot x_i + \mathbf{K}[i,0]\}$, and nothing else. Security is formalized via simulation of ciphertext ct and secret key $\mathsf{sk}_\mathbf{x}$, given only the revealed labels $\mathbf{K}[\mathbf{x}]$.

Batch-Select enables achieving optimal online rate when transferring input labels, because online Alice just sends $\mathbf{x}$ and $\mathsf{sk}_\mathbf{x}$ of length $|\mathbf{x}| + |\mathsf{sk}_\mathbf{x}| = |\mathbf{x}| + \mathrm{poly}(\lambda)$[3]. In this work, we present a concretely efficient Batch-Select scheme with the novel features that part of the offline communication ct_1 can be *reused*, and the rest ct_2 can be *compressed* using the random oracle, leading to small offline-rate $o(1)$. More precisely:

Theorem 5.1 (Informal, New Batch-Select Scheme)**.** *Let λ be the security parameter. Assuming 2^λ-secure RingLWE with dimension n, modulus $\log q = O(\lambda)$, there exists a 2^λ-secure Batch-Select scheme. The scheme has the following asymptotic efficiency when the input length satisfies $|\mathbf{x}| = \Omega(n)$:*

- $|\mathsf{ct}_1| = O(|\mathbf{x}| \log |\mathbf{x}| \cdot \lambda)$. *Ciphertext ct_1 can be reused for $T = \sqrt{q}$ times.* (If the modulus q is super-polynomial, this lets us reuse ct_1 for any polynomial number of times.)

[2] In [AIKW13], this gadget is "special randomized encoding for the *selection function*" and has a slightly different syntax. Nevertheless, it can be used in an analogous way to compress input labels.

[3] In order to achieve input hiding, Alice can apply a one-time pad $\mathbf{r}$ to the input. She would send $\mathbf{x} = \mathbf{x}' + \mathbf{r}$, where $\mathbf{x}'$ is the *actual* input. The one time pad can be removed by modifying the computation to $C'_\mathbf{r}(\mathbf{x}) = C(\mathbf{x} - \mathbf{r})$.

- $|\mathsf{ct}_2| = O(|\mathbf{x}| \cdot \lambda)$ *in the plain model. When* ct_2 *is used to encrypt* random *keys* $\mathbf{K}[i,0]$*, its size can be reduced to* $O(\frac{\lambda}{q^{1/2-\epsilon} \cdot \log q} \cdot |\mathbf{x}|)$ *bits in the ROM for any constant* $\epsilon \in (0, \frac{1}{2})$*. In particular:*
  - *For sufficiently large polynomial modulus* $q = \mathrm{poly}(n, \lambda)$*, the rate is* $o(1)$*.*
  - *For exponentially large modulus* $q > 2^{2(1+\epsilon')\lambda}$ *for any positive constant* $\epsilon' > 0$*,* ct_2 *can be transferred for free (with no communication).*
- $|\mathsf{sk}_{\mathbf{x}}| = n \log q = \mathrm{poly}(\lambda)$*, i.e., the secret key consists of a single ring element.*
- *The computational efficiency of all algorithms is* quasilinear *in the input length.*

Note that 2^λ-security is only assumed for the sake of fair comparison to the naive garbled circuit approach, and can be replaced by any super-polynomial value.

Lightweight Offline Communication. The offline communication of our scheme transfers ct_1 and ct_2. Their absolute sizes are large, $O(|\mathbf{x}| \log |\mathbf{x}| \lambda)$ and $O(|\mathbf{x}|\lambda)$, exceeding λ-rate. We crucially rely on the reusability of ct_1 to establish that the amortized communication rate for transferring ct_1 is $o(1)$ or below. While ct_2 is not reusable, it can be compressed in the ROM to rate $o(1)$ or below. Below we give more details on these two optimizations, and summarize the amortized size of ct_1 and compressed size of ct_2 in Table 5.2.

First, T-time reusability of ct_1 immediately reduces the per-instance cost of sending ct_1 by a factor of T, assuming Alice sends T garbled circuits and corresponding input labels. This amortization only works whenever ct_1 is encrypting the same values across all instances, but this is not an issue when using batch-select as a way to transfer input labels. The reason is that garbled circuits stay secure even when the key offset $\mathbf{K}[i,1] - \mathbf{K}[i,0]$ is repeated for multiple indices i; or even when it is repeated across several garbled circuits. This assumes correlation-robust hash functions [CKKZ12], as in typical garbling with Free-XOR labels [KS08a].

Alternatively (when it is not desired to garble several circuits), we can still amortize within the same instance, by treating the input $\mathbf{x}$ as $k \leq T$ shorter inputs $\mathbf{x}_1, \cdots \mathbf{x}_k$ of length $|\mathbf{x}|/k$. Then, the size of ct_1 becomes $O(|\mathbf{x}| \log |\mathbf{x}| \cdot \lambda/k)$. Using an optimized Batch-Select

construction (which only yields benefits if not amortization across several instances; and it needs to assume identical secret offsets $\Delta = \mathbf{K}[i,1] - \mathbf{K}[i,0]$ for all i) we can furthermore avoid the $\log|\mathbf{x}|$-factor. Thus, by choosing $k = \omega(\lambda)$, the size of ct_1 has rate $o(1)$, even for just a single garbling instance. While Alice now needs to send k secret keys $\mathsf{sk}_{\mathbf{x}_j}$ in the online phase (for a total size of $k \cdot n \cdot \log q$), this cost is still bounded by $\text{poly}(\lambda)$.

A second optimization applies (in the ROM) whenever the keys $\{\mathbf{K}[i,0]\}$ that are encrypted by ct_2 are *uniformly random* (which is the case for most constructions of garbled circuits). This allows compressing ct_2 by a factor of $\tilde{O}(q^{1/2-\epsilon})$ for any constant $\epsilon \in (0, \frac{1}{2})$. Hence, even with sufficiently large *polynomial* RingLWE modulus q, we reduce the size of $|\mathsf{ct}_2|$ to e.g. $|\mathbf{x}|/\lambda$, which implies an $o(1)$-rate. Furthermore, when the modulus is exponentially large $q = 2^{2(1+\epsilon')\lambda}$ for some constant $\epsilon' > 0$, we can "transfer" ct_2 entirely for free, with no communication.

Putting both reusability and ct_2 compression together, the batch-select offline rate approaches 0, while the online rate remains at $1 + o(1)$.

Concretely Efficient Computation. The online computation cost of our scheme is dominated by $O(\frac{\log|\mathbf{x}|\cdot\lambda}{n\log q})$ ring operations per input bit. This fractional number is achieved by packing multiple input keys into a single RingLWE element.

We demonstrate concrete efficiency based on our batch-select implementation. To prioritize computational cost and simplicity of implementation, we choose a 109-bit modulus, and implement a simplified version of our scheme that requires $6|\mathbf{x}|$ bits of offline communication (and $\text{poly}(\lambda)$ bits of online communication). Clearly, this is not the theoretically optimal overall $o(1)$-rate (see Table 5.2), but nevertheless a large improvement over the baseline.

Concretely, our implementation has an online computation time of $3\mu s$ per input bit, which we estimate to be 10^4 times faster than the AIKW RSA-based scheme (assuming that each RSA exponentiation takes $1ms$). To demonstrate the practicality of our method, we show (in Table 5.3) that the computation time (using a single CPU thread with 2.10GHz) from compressing and reconstructing input labels roughly equals the latency of naively transmitting un-compressed input labels over a network that has bandwidth 50 Mbps. But even when the network is faster than this, our method is still valuable e.g. when the input

labels need to be broadcast to multiple evaluators, or when size plays more crucial role such as in blockchain-settings. Details on our evaluation can be found in Section 5.9.

Application to Two Party Computation (2PC). In the literature there is a wealth of constructions of 2PC protocols. They can be roughly categorized into *i)* ones that are concretely efficient but have large communication linear in the complexity of the computation (e.g., [DPSZ12, LP11, WRK17, DILO22]), *ii)* ones with laconic communication depending only on the input/output lengths, but rely on computationally expensive tools like FHE (e.g., [Gen09, BV11, BGV12, GSW13]), or iO (e.g., [HW15]), etc, and *iii)* ones that enjoy both concrete efficiency and laconic communication, but are restricted to low depth computation like NC_1, using homomorphic secret sharing (e.g., [BGI16]). The latter protocols can be extended to evaluating circuits, but only achieving mildly sublinear communication proportional to $|C|/\log|C|$ or $|C|/\log\log|C|$. Another drawback of protocols in ii) and iii) is that they rely on expensive generic techniques, such as, succinct communication ZK to achieve malicious security.

Using our compression technique, we explore building concretely efficient malicious 2PC with laconic *online* communication. We start with considering a new notion called preprocessing garbled circuits: By sending information of length proportional to the circuit size $\tilde{O}(\lambda|C|)$ in an *instance-independent stage* that can be run before knowing C (depending only on certain meta-information such as circuit size), in the online stage after knowing C, the garbled circuit can be compressed to $\mathrm{poly}(\lambda)$ bits. This is achieved by simply sending a garbled universal circuit $\widehat{U}$ in the instance-independent offline stage, and using our compression technique to transfer input labels for $\widehat{U}$ that allow evaluating the circuit C. Since C is not hidden, the online stage only sends sk_C of $\mathrm{poly}(\lambda)$ bits.

By combining preprocessing garbled circuits and authenticated garbled circuits [WRK17, DILO22], we obtain maliciously secure 2PC with succinct online communication in the ROM based on RingLWE: The function-dependent online stage has communication $\mathrm{poly}(\lambda)$, while the input-dependent online stage has communication $\mathrm{poly}(\lambda) + |\mathbf{x}| + |\mathbf{y}|$. The instance-independent preprocessing phase has quasilinear complexity in the circuit size. Importantly,

the protocol is concretely efficient, as it only invokes authenticated garbling and our compression technique in a black-box way. (Of course it would also be possible to merge the function- and input-dependent steps into a single phase with communication $\mathrm{poly}(\lambda)+|\mathbf{x}|+|\mathbf{y}|$. However, in practice the function C might become known much earlier than the inputs $\mathbf{x}, \mathbf{y}$, and for concrete efficiency reasons it would make sense to start processing it as soon as possible.)

Prior works [IKM+13, Cou19] have explored using *function-dependent* (instead of instance-independent) preprocessing to reduce the communication after the inputs are known. However, they are in the information-theoretic regime and their main messages have been relatively negative, either the correlated randomness produced by preprocessing is exponentially long, or the online communication is only slightly sublinear, $|C|/O(\log\log|C|)$.

5.2 Technical Overview

Our focus is the efficient construction of a *batch-select* scheme Sel over a message space $\mathcal{M}$ that forms a ring. This primitive is a special case of functional encryption: The encryptor, given *two* message vectors $\mathbf{l}_1, \mathbf{l}_2 \in \mathcal{M}^w$, first computes one (large) ciphertext ct. Later, given any selection vector $\mathbf{y} \in \{0,1\}^w$, they compute a (small) decryption key $\mathsf{sk}_\mathbf{y}$. This allows the decryptor (who has the ciphertext ct, key $\mathsf{sk}_\mathbf{y}$, and selection vector $\mathbf{y}$), to compute the evaluation $\mathbf{l}_{\mathsf{res}} := \mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$. Here, $\odot$ denotes component-wise vector multiplication. The (simplified) batch-select syntax is as follows:

Encryptor	Decryptor
① $\mathsf{Sel.Enc}(\mathbf{l}_1, \mathbf{l}_2) \to \mathsf{ct}, \mathsf{st}$	③ $\mathsf{Sel.Dec}(\mathsf{sk}_\mathbf{y}, \mathsf{ct}, \mathbf{y}) \to \mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$.
② $\mathsf{Sel.KeyGen}(\mathsf{st}, \mathbf{y}) \to \mathsf{sk}_\mathbf{y}$	

Security states that no information about $\mathbf{l}_1$ and $\mathbf{l}_2$ beyond $\mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$ is learned. Importantly, we require the size of key $\mathsf{sk}_\mathbf{y}$ to be independent of the message dimension w.

We note that this primitive has already been studied in [AIKW13] (see Table 5.1), where it was called "randomized encoding for subset functions" (we use the slightly different syntax

described above, which is going to be conceptually closer to our construction). We are going to present a new and concretely efficient construction, along with new applications.

5.2.1 Instantiating Batch-Select

We start by describing a generic way of instantiating the batch-select primitive, given linearly homomorphic encryption (LHE) and linear laconic encryption (LEnc), both working on the batch-select message space $\mathcal{M}$.

Linearly Homomorphic Encryption. The desired batch-select output $\mathbf{l}_{\mathsf{res}}$ for messages $\mathbf{l}_1, \mathbf{l}_2$ and a selection vector $\mathbf{y}$, is visualized on the left-hand side:

$$\underbrace{\begin{pmatrix} \vdots \\ \mathbf{l}_1[i] \\ \vdots \end{pmatrix} \odot \begin{pmatrix} \vdots \\ y_i \\ \vdots \end{pmatrix} + \begin{pmatrix} \vdots \\ \mathbf{l}_2[i] \\ \vdots \end{pmatrix}}_{\text{Desired}} \quad \text{vs.} \quad \underbrace{\begin{pmatrix} \vdots \\ \mathbf{l}_1[i] \\ \vdots \end{pmatrix} \cdot y + \begin{pmatrix} \vdots \\ \mathbf{l}_2[i] \\ \vdots \end{pmatrix}}_{\text{Achievable with LHE}} \tag{5.1}$$

The difficulty of achieving this comes from *component-wise* multiplication of message $\mathbf{l}_1$ with some long vector $\mathbf{y}$. In particular, if only *vector-scalar* multiplication of $\mathbf{l}_1$ with a single element y were necessary (see right-hand side above), then we could easily achieve this using a key-and-message linearly homomorphic encryption scheme (LHE): The ciphertext returned by $\mathsf{Sel.Enc}$ would contain encryptions $\mathsf{ct}_1^i \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_1, \mathbf{l}_1[i])$ and $\mathsf{ct}_2^i \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_2, \mathbf{l}_2[i])$ for all $i \in [w]$. Given the *succinct* combined key $\mathsf{sk} := \mathsf{sk}_1 \cdot y + \mathsf{sk}_2$, one could decrypt all linearly-combined ciphertext $\mathsf{ct}_{\mathsf{res}}^i := \mathsf{ct}_1^i \cdot y + \mathsf{ct}_2^i$ to obtain $\mathbf{l}_{\mathsf{res}}[i] = \mathbf{l}_1[i] \cdot y + \mathbf{l}_2[i]$.

In [AIKW13], their DDH-based and LWE-based schemes achieve component-wise multiplication by encrypting each column of the square matrix $\mathrm{diag}(\mathbf{l}_1)$ using an LHE, but this results in quadratic ciphertext size.

Linear Laconic Encryption. Our idea for efficiently closing the gap between the two terms in Equation 5.1 is inspired by the recently proposed laconic encryption scheme from [DKL+23] (it is based on RingLWE, but for now we describe it for any message space $\mathcal{M}$). We are not

going to use their scheme as-is. Instead, we use some of the underlying ideas towards our batch-select construction.

Specifically, our observation is that [DKL+23] implicitly uses a primitive that we denote by LEnc. It allows a sender to encrypt a vector $\mathbf{s} \in \mathcal{M}^w$ of ring elements, and anyone else to locally evaluate the resulting ciphertext ct to obtain the *component-wise multiplication* $\mathbf{s} \odot \mathbf{a}$ between the encrypted vector $\mathbf{s}$ and any chosen vector $\mathbf{a}$. However, this outcome is *masked* by the term $\mathbf{r} \cdot d_{\mathbf{a}}$, for some $\mathbf{r} \in \mathcal{M}^w$ generated during encryption, and a publically computable digest $d_{\mathbf{a}} \leftarrow \mathsf{LEnc.Digest}(\mathbf{a})$ that is also an element in $\mathcal{M}$:

Sender	Receiver
① $\mathsf{LEnc.Enc}(\mathbf{s}) \to (\mathbf{r}, \mathsf{ct})$	② $\mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}) \to \mathbf{r} \cdot d_{\mathbf{a}} - \mathbf{s} \odot \mathbf{a}$

This gadget is helpful, because the evaluation outcome can be seen as "replacing" the difficult-to-achieve component-wise multiplication $\mathbf{s} \odot \mathbf{a}$ by a simple vector-scalar multiplication $\mathbf{r} \cdot d_{\mathbf{a}}$.

Putting It Together. Our batch-select scheme applies the LEnc gadget to message $\mathbf{s} := \mathbf{l}_1$, which will allow the decryptor to obtain $\mathbf{r} \cdot d_{\mathbf{a}} - \mathbf{l}_1 \odot \mathbf{y}$, i.e., the desired component-wise multiplication $\mathbf{l}_1 \odot \mathbf{y}$ masked by the "decryption term" $\mathbf{r} \cdot d_{\mathbf{y}}$. In order to remove this decryption term and simultaneously take care of the second message vector $\mathbf{l}_2$, our scheme additionally produces LHE encryptions of vectors $\mathbf{r}$ and $\mathbf{l}_2$. The encryptor, once selection vector $\mathbf{y}$ is known, computes a *short* combined secret key $\mathsf{sk}_{\mathbf{y}}$ that may be used to evaluate $\mathbf{r} \cdot d_{\mathbf{y}} + \mathbf{l}_2$. Subtracting this from the LEnc outcome yields $\mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$ as desired. We summarize

the scheme as follows.

Encryptor

① $\mathsf{Sel.Enc}(\mathbf{l}_1, \mathbf{l}_2) \quad \to \quad \mathsf{ct} := (\mathsf{LEnc.ct}, \mathsf{LHE.ct}_1, \mathsf{LHE.ct}_2)$ and $\mathsf{st} := (\mathsf{sk}_1, \mathsf{sk}_2)$,

$$\text{where} \quad (\mathbf{r}, \mathsf{LEnc.ct}) \leftarrow \mathsf{LEnc.Enc}(\mathbf{l}_1) \ ,$$

$$\mathsf{LHE.ct}_1 \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_1, \mathbf{r}) \ ,$$

$$\mathsf{LHE.ct}_2 \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_2, \mathbf{l}_2) \ .$$

② $\mathsf{Sel.KeyGen}(\mathsf{st}, \mathbf{y}) \quad \to \quad \mathsf{sk}_{\mathbf{y}} \leftarrow \mathsf{sk}_1 \cdot d_{\mathbf{y}} + \mathsf{sk}_2$,

$$\text{where} \quad d_{\mathbf{y}} \leftarrow \mathsf{LEnc.Digest}(\mathbf{y}) \ .$$

Decryptor

③ $\mathsf{Sel.Dec}(\mathsf{sk}_{\mathbf{y}}, \mathsf{ct}, \mathbf{y}) \quad \to \quad \underbrace{\mathsf{LHE.Dec}(\mathsf{sk}_{\mathbf{y}}, \mathsf{ct}_{\mathsf{res}})}_{\mathbf{r} \cdot d_{\mathbf{y}} + \mathbf{l}_2} - \underbrace{\mathsf{LEnc.Eval}(\mathsf{LEnc.ct}, \mathbf{y})}_{\mathbf{r} \cdot d_{\mathbf{y}} - \mathbf{l}_1 \odot \mathbf{y}}$,

$$\text{where} \quad d_{\mathbf{y}} \leftarrow \mathsf{LEnc.Digest}(\mathbf{y}) \ ,$$

$$\mathsf{ct}_{\mathsf{res}} := \mathsf{LHE.ct}_1 \cdot d_{\mathbf{y}} + \mathsf{LHE.ct}_2 \ .$$

As desired, the resulting batch-select decryption key $\mathsf{sk}_{\mathbf{y}}$ is only a single LHE key, independently of w.

Leaving the idealized setting. So far we omitted several details that arise once we replace the generic message space $\mathcal{M}$ by an actual RingLWE ring $\mathcal{R}_p$.

First, we ignored RingLWE-induced noise. As usual, we can deal with this by multiplying the message vectors $\mathbf{l}_1, \mathbf{l}_2$ with a sufficiently large scaling factor Δ and then work over the ring $\mathcal{R}_q$ for $q = p\Delta$ instead. The final step of the decryptor is to divide and round the output.

We note that this complicates the security proof: since the decryptor will know both the *exact* message $\mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$ as well as its *noisy* variant, some function of the RingLWE noise will be leaked. Therefore, we use *noise flooding* to statistically hide the leakage. To keep its cost low, we use a result from [DKL+23], which shows that (for the type of leakage that our scheme produces) it is sufficient to sample the flooding noise from a distribution that is only polynomially larger than the actual RingLWE noise. This saves us from an exponentially

large RingLWE modulus when desired.

Second, naive LHE incurs large noise growth. In particular, the noise resulting from naive multiplication $\mathsf{ct}_1 \cdot d_{\mathbf{y}}$ during LHE evaluation will exceed the modulus q. Therefore, we apply a standard decomposition trick to split ct_1 into $m = \lceil \log_g q \rceil$ ciphertexts encrypting $\mathbf{r}, \mathbf{r} \cdot g, \mathbf{r} \cdot g^2, \ldots$, which can then be multiplied with a decomposition of $d_{\mathbf{y}}$. Performing multiplication in this form will increase existing noise by only a factor of $q^{1/m} \cdot \mathrm{poly}(\lambda)$, which allows us to set $m = O(1)$ without breaking correctness. Applying the same trick inside LEnc, the batch-select ciphertext for message space $\mathcal{M} = \mathcal{R}_p$ has the following size:

$$|\mathsf{LEnc.ct}| = O(w \log w \cdot |\mathcal{R}_q|), \quad |\mathsf{LHE.ct}_1| = O(w \cdot |\mathcal{R}_q|), \quad |\mathsf{LHE.ct}_2| = O(w \cdot |\mathcal{R}_q|).$$

Fortunately, the noise growth of our scheme still allows a large plaintext modulus such as $p \geq q^{1/c}$ for constant c (even with polynomial modulus-to-noise ratio). Therefore, $\log |\mathcal{R}_q| = O(\log |\mathcal{R}_p|)$, implying that a batch-select ciphertext asymptotically has the same size as its plaintext.

Third, for our garbling application, we require batch-select with message space $\mathcal{M} = \mathbb{Z}_p$ instead of ring $\mathcal{R}_p$. This can be achieved easily and efficiently with *packing*: for compatible primes p, the Chinese Remainder Theorem shows that each $\mathcal{R}_p$ element is isomorphic to $\mathbb{Z}_p^n$ [SV10, GHS12] (where n is the degree of ring elements in $\mathcal{R}$). Thus, whenever $w = \Omega(n)$, we also get a batch-select scheme for message space $\mathcal{M} = \mathbb{Z}_p$ with

$$|\mathsf{LEnc.ct}| = O(w \log w \log p), \quad |\mathsf{LHE.ct}_1| = O(w \log p), \quad |\mathsf{LHE.ct}_2| = O(w \log p).$$

5.2.2 Making batch-select practical

Overall, batch-select on messages in $\mathcal{M}^w = \mathbb{Z}_p^w$ with $p = \Theta(2^\lambda)$ (as in our application for garbled circuits in Section 5.2.3) would cost $|\mathsf{ct}| = O(w \log w \lambda)$ bits. For certain settings, we present two optimizations to bring down this cost to essentially 0.

Reusability. Our first observation is that garbled circuit labels under free-XOR type assumptions [KS08a] have a special label format $\mathbf{l}_1 = \Delta \cdot \mathbf{1}_w$ (for some fixed $\Delta \in \mathcal{M}$). Even

when garbling several circuits, the same label offset may be used for all of them, and hence the message vector $\mathbf{l}_1$ will always be the same. Only $\mathbf{l}_2$ and the selection vector $\mathbf{y}$ change across multiple runs.

The only component of the batch-select ciphertext that depends on $\mathbf{l}_2$ is $\mathsf{LHE.ct}_2$; the other two components $\mathsf{LEnc.ct}$, $\mathsf{LHE.ct}_1$ can be reused. More precisely, we split $\mathsf{Sel.Enc}$ into two parts:

$$\begin{aligned}
\text{Reusable: } \text{(1a) } \mathsf{Sel.Enc}_1(\mathbf{l}_1) \quad &\rightarrow \quad \mathsf{ct}_1 := (\mathsf{LEnc.ct}, \mathsf{LHE.ct}_1) \text{ and } \mathsf{st}_1 := \mathsf{sk}_1 \ , \\
&\quad\quad \text{where } \ (\mathbf{r}, \mathsf{LEnc.ct}) \leftarrow \mathsf{LEnc.Enc}(\mathbf{l}_1) \ , \\
&\quad\quad\quad\quad \mathsf{LHE.ct}_1 \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_1, \mathbf{r}) \ . \\
\text{Non-reusable: } \text{(1b) } \mathsf{Sel.Enc}_2(\mathbf{l}_2) \quad &\rightarrow \quad \mathsf{ct}_2 := \mathsf{LHE.ct}_2 \text{ and } \mathsf{st}_2 := \mathsf{sk}_2 \ , \\
&\quad\quad \text{where } \ \mathsf{LHE.ct}_2 \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_2, \mathbf{l}_2) \ .
\end{aligned}$$

Later, $\mathsf{Sel.KeyGen}$ will be called with both states st_1 and st_2, and $\mathsf{Sel.Dec}$ will be called with both ciphertexts ct_1 and ct_2. The amortized cost for applying batch-select T times becomes $O((\frac{\log w}{T} + 1) w \log p)$. For sufficiently large $T = \Omega(\log w)$, the cost of $|\mathsf{ct}_1|$ vanishes, and only that of $|\mathsf{ct}_2| = O(w \log p)$ remains.

We also note an orthogonal optimization: Whenever $\mathbf{l}_1 = \Delta \cdot \mathbf{1}_w$, we can apply an optimized LEnc construction (Section 5.5.1) that directly yields cost $|\mathsf{LEnc.ct}| = O(w \cdot |\mathcal{R}_q|) = O(w \log p)$ bits, even without amortization. However, using this construction does not allow us to get below the $|\mathsf{ct}_1| = O(w \log p)$ barrier. Only when applying amortization, the following optimization will yield asymptotical advantages.

Compressing Random LHE Ciphertexts. Our second observation is that when transferring common garbled circuit labels, e.g. Yao's garbling, batch-select is used with *completely random* $\mathbf{l}_2$ message vectors. In this case, we may simply sample them within batch-select in the following optimized way: we first sample a *random LHE ciphertext* $\mathsf{LHE.ct}_2$ (this can be done using the random oracle), and reversely derive a message vector $\mathbf{l}_2 \leftarrow \mathsf{LHE.Dec}(\mathsf{sk}_2, \mathsf{LHE.ct}_2)$. It then suffices to publish a seed of λ bits in place of the entire ciphertext $\mathsf{LHE.ct}_2$.

A complication arises as a randomly sampled $\mathsf{LHE.ct}_2$ may not decrypt correctly. Roughly, decryption correctness requires the noise level in $\mathsf{LHE.ct}_2$ to be much smaller than the scaling factor Δ, while a randomly sampled one has uniform noise level in $[0, \Delta)$. This only happens with negligible probability when the modulus $q = 2^{\Theta(\lambda)}$ is sufficiently exponentially large. For general modulus, we utilize rejection sampling to indicate whether a ciphertext is good. It requires sending $|\mathsf{ct}_2| = O(\frac{w \cdot \lambda}{q^{1/2-\varepsilon} \cdot \log q})$ bits given plaintext modulus $p = q^\varepsilon$.

See Section 5.4.4 for more detail and our security analysis.

Putting It Together. In summary, when applying both discussed optimizations, the cost of batch-select (amortized across T instances with identical $\mathbf{l}_1$ and *uniformly random* $\mathbf{l}_2$ for each instance) becomes

$$|\mathsf{LEnc.ct}| = O(w \log w \log p/T), \quad |\mathsf{LHE.ct}_1| = O(w \log p/T), \quad |\mathsf{LHE.ct}_2^*| = O(w)$$

for general modulus q. With $T = \Omega(\log w \cdot \log p)$, the amortized offline cost will be $O(w)$, which may be much smaller than the plaintext that has size $w \cdot \log p$. With exponential modulus $q = 2^{2(1+\varepsilon')\lambda}$, we even get $|\mathsf{LHE.ct}_2^*| = 0$, and therefore the amortized cost converges towards 0.

5.2.3 Application: Preprocessing Garbling

Now we sketch a natural application of batch-select: a garbling scheme in which the costly garbling splits into a *function-independent* part, and a subsequent *succinct, function-dependent* part. In other words, the main work can be performed before the circuit is even known.

Our approach is conceptually simple: In a function-independent step ($\mathsf{GarbleU}$), without knowing the function description, the garbler prepares a standard garbling $\widehat{C_U}$ (e.g. using Yao's GC) of a *universal circuit* C_U. This circuit $C_U(f, \mathbf{x})$, when given the description f and input $\mathbf{x}$, returns $f(\mathbf{x})$. We denote the s input keys corresponding to the function description f by $\mathbf{K}^{\mathsf{Fn}} \in \mathcal{K}^{s \times 2}$, and the input keys corresponding to the input $\mathbf{x}$ by $\mathbf{K} \in \mathcal{K}^{|\mathbf{x}| \times 2}$.

In a function-dependent step ($\mathsf{GarbleFunc}$), we only need to make sure that the evaluator

obtains the function's input labels

$$\mathbf{K}^{\mathsf{Fn}}[\mathbf{f}] = (\mathbf{K}^{\mathsf{Fn}}[\mathbf{1}] - \mathbf{K}^{\mathsf{Fn}}[\mathbf{0}]) \odot \mathbf{f} + \mathbf{K}^{\mathsf{Fn}}[\mathbf{0}] \ ,$$

where $\mathbf{f} \in \{0,1\}^s$ is the bit-representation of function f. We can easily do so, *succinctly*, using batch-select message vectors $\mathbf{l}_1 = \mathbf{K}^{\mathsf{Fn}}[\mathbf{1}] - \mathbf{K}^{\mathsf{Fn}}[\mathbf{0}]$ and $\mathbf{l}_2 = \mathbf{K}^{\mathsf{Fn}}[\mathbf{0}]$. Besides $\widehat{C_U}$, we place the (large) ciphertext ct into the function-independent garbling (denoted by $\widehat{U} := (\mathsf{ct}, \widehat{C_U})$), and only need to send the short key $\mathsf{sk}_\mathbf{f}$ in the function-dependent step.

We summarize this process below. SG denotes a standard model garbling (such as Yao's garbling scheme). The evaluation function $\mathsf{Eval}((\widehat{C_U}, \mathsf{sk}_\mathbf{f}), f, \mathbf{k}_\mathbf{x})$ takes the function f, the entire garbling $(\widehat{U}, \mathsf{sk}_\mathbf{f})$, and the input labels $\mathbf{k}_\mathbf{x}$ for input $\mathbf{x}$. See Section 5.7 for details.

$$\mathsf{GarbleU}(\mathbf{K}) \quad \begin{cases} \widehat{C_U} \leftarrow \mathsf{SG.Garble}(C_U, (\mathbf{K}^{\mathsf{Fn}}, \mathbf{K})) \text{ for random keys } \mathbf{K}^{\mathsf{Fn}} \\ \mathsf{ct}, \mathsf{st} \leftarrow \mathsf{Sel.Enc}(\mathbf{K}^{\mathsf{Fn}}[\mathbf{1}] - \mathbf{K}^{\mathsf{Fn}}[\mathbf{0}], \mathbf{K}^{\mathsf{Fn}}[\mathbf{0}]) \\ \text{output } \widehat{U} := (\mathsf{ct}, \widehat{C_U}) \text{ and } \mathsf{st} \end{cases}$$

$$\mathsf{GarbleFunc}(\mathsf{st}, \mathbf{f}) \quad \big\{ \text{output } \mathsf{sk}_\mathbf{f} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}, \mathbf{f})$$

$$\mathsf{Eval}(f, (\widehat{U}, \mathsf{sk}_\mathbf{f}), \mathbf{k}_\mathbf{x}) \quad \begin{cases} \mathbf{k}_\mathbf{f}^{\mathsf{Fn}} \leftarrow \mathsf{Sel.Dec}(\mathsf{sk}_\mathbf{f}, \mathsf{ct}, \mathbf{f}) \\ \text{output } \mathbf{y} \leftarrow \mathsf{SG.Eval}(C_U, \widehat{C_U}, (\mathbf{k}_\mathbf{f}^{\mathsf{Fn}}, \mathbf{k}_\mathbf{x})) \end{cases}$$

The function-dependent garbling size will be $|\mathsf{sk}_\mathbf{f}| = \text{poly}(\lambda)$. Using the optimizations from Section 5.2.2, the size of function-independent garbling is $|\widehat{U}| = O(|\widehat{C_U}| + |\mathbf{f}| \cdot \lambda)$ for a single instance, and converges towards $|\widehat{C_U}|$ amortized across a large number of instances (in the ROM).

We can naturally transform the garbling scheme above into a 2-party computation protocol with succinct function-dependent cost. One may additionally use batch-select to transfer input labels $\mathbf{k}_\mathbf{x}$ with cost $\text{poly}(\lambda) + |\mathbf{x}|$ in the online phase, instead of the naive cost $|\mathbf{x}| \cdot \lambda$.

5.2.4 Adaptive Security

In this work we mainly focus on the *selective* simulation security of batch-select, where the attacker must choose all inputs $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(T)})$ before seeing any offline component.

Similarly, for preprocessing garbling, all functions $C^{(t)}$ and inputs $\mathbf{x}^{(t)}$ must be chosen statically.

A stronger security requirement would be adaptive security, where the attacker can choose each $\mathbf{x}^{(t)}$ and/or $C^{(t)}$ adaptively, dependent on previously generated components. This notion is important for attaining *maliciously* secure two-party computation protocols, where malicious parties' inputs are not defined at the beginning of the execution. In Appendix 5.8, we show that in the ROM, it is simple to modify our batch-select scheme to become adaptively secure, using similar techniques as in [BHR12]. This requires an adaptive version of error-leakage RingLWE (see Definition 5.3), which follows directly from plain RingLWE with exponential modulus-to-noise ratio using noise flooding.

We apply the adaptively secure batch-select to construct a malicious 2-party computation protocol with only $\text{poly}(\lambda)$ bits of communication in the function dependent phase, and $|\mathbf{x}| + |\mathbf{y}| + \text{poly}(\lambda)$ in the online phase.

5.3 Preliminaries

In this chapter, we mainly focus on obtaining an optimized solution for compressing input labels of Boolean garbling schemes (Definition 2.1), though our techniques should also generalize to compressing labels of arithmetic garbling and mixed garbling schemes.

5.3.1 Notations for Labels in Garbling Schemes

In common Boolean garbling schemes (e.g. Yao's garbling), the i-th bit of an input $\mathbf{x} \in \{0,1\}^{\ell_x}$ is associated with a pair of *keys* $\mathbf{K}[i,0], \mathbf{K}[i,1] \in \mathcal{K}$. Here, $\mathcal{K}$ is the key space associated with the garbling scheme (e.g., $\mathcal{K} = \mathbb{Z}_2^{\lambda}$). We call the selected key $\mathbf{K}[i, x_i]$ the *label* of the i-th bit. We write the set of all keys as a matrix $\mathbf{K} \in \mathcal{K}^{\ell_x \times 2}$. We use short-hands $\mathbf{K}[b] = (\mathbf{K}[1,b], \ldots, \mathbf{K}[\ell_x, b])$ and $\mathbf{K}[\mathbf{x}] = (\mathbf{K}[1,x_1], \ldots, \mathbf{K}[\ell_x, x_w])$ (for $b \in \{0,1\}$ and $\mathbf{x} \in \{0,1\}^{\ell_x}$), to denote the set of all b-keys, and the set of all labels for input $\mathbf{x}$, respectively. Note that both $\mathbf{K}[b]$ and $\mathbf{K}[\mathbf{x}]$ are in $\mathcal{K}^{\ell_x}$.

5.3.2 Notations for Indistinguishability

We denote by λ the security parameter. For two families of distributions $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$, we say that they are 2^λ-indistinguishable, if for any probabilistic adversary $\mathcal{A}$ with running time bounded by $t(\lambda)$, we have

$$|\Pr[\mathcal{A}(X_\lambda) = 1] - \Pr[\mathcal{A}(Y_\lambda) = 1]| \leq \frac{\text{poly}(\lambda)}{2^\lambda} \cdot t(\lambda)$$

for some polynomial $\text{poly}(\lambda)$. We also write this as $X \approx_{\text{c}}^{2^\lambda} Y$. Furthermore, if the *statistical* distance between X and Y is bounded by $\frac{\text{poly}(\lambda)}{2^\lambda}$, then we write $X \approx_{\text{s}}^{2^\lambda}$.

5.3.3 RingLWE and Error-leakage RingLWE

In this work, we consider the polynomial rings $\mathcal{R}(\lambda) = \mathbb{Z}[X]/(X^n + 1)$, where $n(\lambda)$ is a power-of-2. Let $\Phi : \mathcal{R} \to \mathbb{R}^n$ be an embedding of $\mathcal{R}$ in $\mathbb{R}^n$. For example, the coefficient embedding maps $\Phi(\sum_{i=0}^{n-1} a_i X^i) = (a_0, \ldots, a_{n-1})^T$. We define the *infinity norm* of a ring element $a \in \mathcal{R}$ with respect to its coefficient embedding, i.e., $\|a\|_\infty = \|\Phi(a)\|_\infty = \max_{i=0,\ldots,d_\mathcal{R}-1} |a_i|$ for $a = \sum_{i=0}^{d_\mathcal{R}-1} a_i X^i$. The *ring expansion factor* of $\mathcal{R}$ is given by $\gamma_\mathcal{R} := \max_{a,b\in\mathcal{R}\setminus\{0\}} \frac{\|a\cdot b\|_\infty}{\|a\|_\infty \cdot \|b\|_\infty}$.

Lemma 5.1 ([AL21], Proposition 2)**.** *If the cyclotomic $\mathcal{R}$ has degree $n = 2^k$ (i.e., the degree is a power of 2), then the expansion factor is bounded by $\gamma_\mathcal{R} \leq n$.*

Given a modulus $q \in \mathbb{N}$, we use $\mathcal{R}_q$ to denote $\mathcal{R}/q\mathcal{R}$. For an element $a \in \mathcal{R}_q$, we use $\|a\|_\infty$ in the ring modulo q, to denote the infinity norm of the element $a \in \mathcal{R}$, where we identify each coefficient $a_i \in \mathbb{Z}_q$ of a with the corresponding representative element in $[\frac{q}{2}, \frac{q}{2}) \subseteq \mathbb{Z}$. We use $\lfloor \frac{a}{\Delta} \rceil$ to denote the ring element $b \in \mathcal{R}_q$, where the i-th coefficient b_i is equal to $\frac{a_i}{\Delta}$, rounded to the nearest integer.

For any modulus $q(\lambda)$, number of samples $m(\lambda)$, and distributions $\chi(\lambda)$ over $\mathcal{R}$, we write $\mathsf{LWE}_{\mathcal{R},m,q,\chi}$ to mean the RingLWE assumption (Definition 2.11) with a fixed $m(\lambda)$ number of samples, an error distribution χ, and a uniform secret distribution over $\mathcal{R}_q$.

Distributions over $\mathcal{R}$. We will use discrete Gaussians for error distributions in RingLWE (and the uniform distribution over $\mathcal{R}_q$ for the secret distribution). Over $\mathbb{R}$, the *continuous*

Gaussian distribution with parameter s is defined by the density function $\rho_s(x) = e^{-\pi x^2/s^2}$. Over $\mathbb{Z}$, this yields the *discrete* Gaussian distribution with parameter s by defining the density function $\mathcal{D}_{\mathbb{Z},s}(x) = \frac{\rho_s(x)}{\sum_{x' \in \mathbb{Z}} \rho_s(x')}$. For a ring $\mathcal{R}$ with embedding $\Phi : \mathcal{R} \to \mathbb{R}^n$, the Gaussian distribution $\mathcal{D}_{\mathcal{R},s}$ samples a vector in $\mathbb{R}^n$ using $\mathcal{D}^n_{\mathbb{Z},s}$, and then maps the result back to an element in $\mathcal{R}$. For example, if Φ is the coefficient embedding, $\mathcal{D}_{\mathcal{R},s}$ samples individual coefficients $a_i \leftarrow \mathcal{D}_{\mathbb{Z},s}$, which then yields the ring element $a = \sum_{i=0}^{n-1} a_i X^i$. Gaussians may also be generalized to a *covariance matrix* $\sigma \in \mathbb{R}^{n \times n}$, which allows individual coordinates to be correlated.

To ensure perfect correctness of our constructions, we will actually use *truncated* discrete Gaussians, whose distribution $\overline{\mathcal{D}}_{\mathcal{R},s}$ is based on a security parameter λ. This distribution $\overline{\mathcal{D}}_{\mathcal{R},s}$ is identical to $\mathcal{D}_{\mathcal{R},s}$, except that it rejects the sample if $\|a\|_\infty > \sqrt{\lambda} \cdot s$, and instead continues sampling until it finds an $a \in \mathcal{R}$ with $\|a\|_\infty \leq \sqrt{\lambda} \cdot s$.

Lemma 5.2. *For a ring $\mathcal{R}$ with degree $n = \mathrm{poly}(\lambda)$, and for any parameter s, the following two distributions are 2^λ-statistically close: $\mathcal{D}_{\mathcal{R},s} \approx_{\mathrm{s}}^{2^\lambda} \overline{\mathcal{D}}_{\mathcal{R},s}$.*

Proof. By [DKL+23, Lemma 2], we have

$$\Pr[\|e\| > \sqrt{\lambda} \cdot s \mid e \leftarrow \mathcal{D}_{\mathcal{R},s}] < 2 \cdot n \cdot e^{-\pi \cdot \lambda} < \frac{\mathrm{poly}(\lambda)}{2^\lambda} .$$

Conditioned on the event given in the probability above not happening, $\overline{\mathcal{D}}_{\mathcal{R},s}$ is identical to $\mathcal{D}_{\mathcal{R},s}$, and hence the statistical distance between these two distributions is bounded by $\frac{\mathrm{poly}(\lambda)}{2^\lambda}$. □

Given any distribution χ, we will use "$\max \chi$" to denote the maximum norm $\|a\|_\infty$ that any value $a \leftarrow \chi$ sampled from this distribution can have (e.g., $\max \overline{\mathcal{D}}_{\mathcal{R},s} = \sqrt{\lambda} \cdot s$).

Error-leakage RingLWE. In [DKL+23], the following variant of RingLWE (called *error-leakage* RingLWE) was introduced. It provides the adversary with some additional information: $\mathcal{A}$ does not only receive the RingLWE sample $(\mathbf{a}, \mathbf{y})$, but also some *leakage* $\mathbf{l} = \mathbf{Z} \cdot \mathbf{e} + \bar{\mathbf{e}}$, where $\mathbf{e}$ is the error used to compute $\mathbf{y} = \mathbf{a} \cdot s + \mathbf{e}$, and $\bar{\mathbf{e}}$ is a fresh error from a different

distribution $\overline{\chi}$. The *leakage matrix* $\mathbf{Z}$ may be adversarially chosen, but it is restricted to a subset $\mathcal{L} \subset \mathcal{R}^{k \times m}$.

Definition 5.1 ($\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}}$ Assumption, [DKL+23])**.** *We say that the* $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}}$ *assumption (with some set* $\mathcal{L} \subset \mathcal{R}^{k \times m}$*) holds, if for any probabilistic adversary* $\mathcal{A}$ *with running time bounded by* $t(\lambda)$*, we have*

$$\left|\Pr[\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}} = 1] - \Pr[\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}} = 1]\right| \leq \frac{\mathrm{poly}(\lambda)}{2^{\lambda}} \cdot t(\lambda) ,$$

where the $\mathsf{elLWE}^{\mathcal{A},b}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}}$ *experiment is defined as follows.*

1. *The game samples a public parameters* $\mathbf{a} \leftarrow \mathcal{R}_q^m$*, and sends* $\mathbf{a}$ *to* $\mathcal{A}$*.*

2. $\mathcal{A}$ *outputs a matrix* $\mathbf{Z} \in \mathcal{L}$*.*

3. *The game samples* $\mathbf{e} \leftarrow \chi^m$*,* $\overline{\mathbf{e}} \leftarrow \chi^k$ *and computes the leakage* $\mathbf{l} = \mathbf{Z} \cdot \mathbf{e} + \overline{\mathbf{e}}$*. It sends to* $\mathcal{A}$ *the pair* $(\mathbf{y}, \mathbf{l})$*, where* $\mathbf{y}$ *is computed as follows.*

   - *If* $b = 0$*, sample* $s \leftarrow \mathcal{R}_q$ *and define* $\mathbf{y}^T = s \cdot \mathbf{a}^T + \mathbf{e}^T \bmod q$*.*
   - *If* $b = 1$*, sample* $\mathbf{y} \leftarrow \mathcal{R}_q^m$ *uniformly at random.*

4. $\mathcal{A}$ *outputs a bit* b'*, which is also the output of the game.*

In [DKL+23], the authors give a hardness result of elLWE under the standard assumption of RingLWE. The required parameters for standard RingLWE depend on spectral properties guaranteed for all leakage matrices $\mathbf{Z} \in \mathcal{R}^{k \times m}$ in $\mathcal{L}$ (when viewed as a matrix in $\mathbb{R}^{kn \times mn}$). While these propertiese are only explicitly considered in the special case of $k = 1$ in [DKL+23], their proof of elLWE hardness actually works for any matrix $\mathbf{Z} \in \mathcal{R}^{k \times m}$. We will use this more general version, but this also requires us to define the following formalization of a maximal singular value $\sigma_{\max}(\mathcal{L})$. It is based on Lemma 9 of [DKL+23].

Definition 5.2. *Let* $\mathcal{R}$ *be a ring of degree* n*, and let* $\Phi : \mathcal{R} \to \mathbb{R}^n$ *be an embedding of* $\mathcal{R}$ *in* $\mathbb{R}^n$*. For a vector* $\mathbf{x} \in \mathcal{R}^{\ell}$*, we write* $\Phi(\mathbf{x}) \in \mathbb{R}^{\ell n}$ *to mean the vector containing all* $\Phi(x_i)$ *for* $\mathbf{x} = (x_1, \ldots, x_{\ell})^T$*.*

For a matrix $\mathbf{Z} \in \mathcal{R}^{k\times m}$*, we may define a linear transformation* $\Theta_{\mathbf{Z}} : \mathbb{R}^{mn} \to \mathbb{R}^{kn}$ *with corresponding matrix* $\mathbf{A}_{\mathbf{Z}} \in \mathbb{R}^{kn\times mn}$ *as*

$$\Theta_{\mathbf{Z}}(\mathbf{x}) = \Phi(\mathbf{Z} \cdot \Phi^{-1}(\mathbf{x})) = \mathbf{A}_{\mathbf{Z}} \cdot \mathbf{x} \ .$$

For a set $\mathcal{L} \subset \mathcal{R}^{m\times k}$*, we define the* maximal singular value $\sigma_{max}(\mathcal{L})$ *as*

$$\sigma_{max}(\mathcal{L}) = \max_{\mathbf{Z}\in\mathcal{L}} \sigma_{max}(\mathbf{A}_{\mathbf{Z}}) \ .$$

Theorem 5.2 ([DKL+23], Theorem 3)**.** *Let* $\epsilon > 0$ *be negligible. Let* $\mathcal{R}$ *be a suitable ring with an embedding as a lattice in* $\mathbb{R}^n$*, and let* $\mathbf{\Sigma}_0$ *be a covariance matrix with* $\sqrt{\mathbf{\Sigma}_0} \geq \eta_\epsilon(\mathcal{R})$*. Let* $\mathcal{L} \subset \mathcal{R}^{k\times m}$ *be an efficiently decidable set. Let* $s, t \geq 2\sqrt{2}$ *be such that*

$$t^2 \sigma_{min}(\mathbf{\Sigma}_0) \geq \frac{(s^2+1)(s^2+2)}{s^2} \sigma_{max}(\mathbf{\Sigma}_0) \cdot (\sigma_{max}(\mathcal{L}))^2$$

Now let $\chi = \mathcal{D}_{\mathcal{R},\sqrt{(s^2+1)\mathbf{\Sigma}_0}}$*,* $\overline{\chi} = \mathcal{D}_{\mathcal{R},\sqrt{(t^2+1)\mathbf{\Sigma}_0}}$ *and* $\chi^* = \mathcal{D}_{\mathcal{R},s/2\cdot\sqrt{\mathbf{\Sigma}_0}}$*. Then, assuming that* $\mathsf{LWE}_{\mathcal{R},m,q,\chi^*}$ *is hard,* $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}}$ *is also hard. More precisely, if there exists an adversary* $\mathcal{A}$ *with advantage* δ *against* $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}}$*, then there exists an adversary* $\mathcal{A}'$ *with roughly the same running time and adavantage* $\delta - 44\epsilon$ *against* $\mathsf{LWE}_{\mathcal{R},m,q,\chi^*}$*.*

Leakage Considered in This Work. In this work, we will consider the set $\mathcal{L}_{k,m}(\beta) \subset \mathcal{R}^{k\times m}$ of leakage matrices such that all contained ring elements have infinity norm bounded by β (under coefficient embedding). For such matrices, we can show the following bound on the maximal singular values as required by Theorem 5.2:

Lemma 5.3 ([DKL+23], Lemma 9 generalized)**.** *Let* $\mathcal{R}$ *be a cyclotomic ring of degree* n *that is a power-of-2, and let* $\mathbf{Z} \in \mathcal{R}^{k\times m}$ *be a matrix, s.t. each coefficient of* $\mathbf{Z}$ *has* $\|\cdot\|_\infty$*-norm bounded by* β *(in the coefficient embedding). Then, the maximal singular value of the matrix* $\mathbf{A}_{\mathbf{Z}} \in \mathbb{R}^{kn\times mn}$ *is bounded by* $\sigma_{max}(\mathbf{A}_{\mathbf{Z}}) \leq \beta n\sqrt{km}$*. This immediately implies*

$$\sigma_{max}(\mathcal{L}_{k,m}(\beta)) \leq \beta n\sqrt{km} \ .$$

Proof. For any ring element $a \in \mathcal{R}$ with $a = \sum_{i=1}^{n} a_i X^i$ that fulfills $\|\Phi(a)\|_\infty \leq \beta$, and for any $b \in \mathcal{R}$, we have

$$\begin{aligned}\|\Phi(a \cdot b)\|_2 &\leq \sum_{i=1}^{n} |a_i| \cdot \|\Phi(X^i \cdot b)\|_2 = \sum_{i=1}^{n} |a_i| \cdot \|\Phi(b)\|_2 \leq \beta \cdot \sum_{i=1}^{n} \|\Phi(b)\|_2 \\ &= \beta n \|\Phi(b)\|_2 \ .\end{aligned}$$

Here we used the fact that $X^i \cdot b$ is just a rotation of b which does not change its norm.

We use this result in the following inequality. We split $\mathbf{A_Z}$ into $k \cdot m$ blocks $\mathbf{A_Z}[i,j] \in \mathbb{R}^{n \times n}$ for $i \in [k]$, $j \in [m]$ (note that $\mathbf{A_Z}[i,j]$ is exactly the negacyclic matrix corresponding to the ring element in the i-th row and j-th column of $\mathbf{Z}$). Furthermore, let $\mathbf{x} \in \mathbb{R}^{dm}$ be any vector, which we split into $\mathbf{x}^T = (\mathbf{x}_1^T, \ldots, \mathbf{x}_m^T)^T$, i.e., m smaller vectors $\mathbf{x}_j \in \mathbb{R}^d$, each corresponding to one ring element. Then,

$$\begin{aligned}\|\mathbf{A_Z} \cdot \mathbf{x}\|_2 &\leq \sqrt{k} \cdot \max_{i \in [k]} \| \sum_{j \in [m]} \mathbf{A_Z}[i,j] \cdot \mathbf{x}_j \|_2 \leq \sqrt{k} \cdot \max_{i \in [k]} \sum_{j \in [m]} \|\mathbf{A_Z}[i,j] \cdot \mathbf{x}_j\|_2 \\ &= \sqrt{k} \cdot \max_{i \in [k]} \sum_{j \in [m]} \|\Phi(\mathbf{Z}[i,j] \cdot \Phi^{-1}(\mathbf{x}_j))\|_2 \\ &= \beta n \sqrt{k} \cdot \sum_{j \in [m]} \|\Phi(\Phi^{-1}(\mathbf{x}_j))\|_2 = \beta n \sqrt{k} \cdot \sum_{j \in [m]} \|\mathbf{x}_j\|_2 \\ &\leq \beta n \sqrt{km} \|\mathbf{x}\|_2 \ .\end{aligned}$$

□

We will also consider the following, *sparse*, version of leakage matrices: Let $\mathcal{L}_{k,m}^{\times w}(\beta) \subset \mathcal{R}^{wk \times wm}$ be the set of matrices consisting of w separate blocks (whose infinity norm are each bounded by β) of size $k \times m$ on the diagonal, i.e.,

$$\mathcal{L}_{k,m}^{\times w}(\beta) = \{\mathrm{diag}(\mathbf{Z}_1, \ldots, \mathbf{Z}_w) \mid \mathbf{Z}_i \in \mathcal{L}_{k,m}(\beta)\} \ .$$

Intuitively, a leakage matrix in $\mathcal{L}_{k,m}^{\times w}(\beta)$ says that there are w leakages, each one computed separately from m fresh RingLWE samples (but with the same secret key).

Viewing $\mathcal{L}_{k,m}^{\times w}(\beta)$ as a subset of $\mathcal{L}_{wk,wm}(\beta)$ and then directly applying Lemma 5.3 would incur a factor of w in the upper bound of $\sigma_{\max}(\mathcal{L}_{wk,wm}(\beta))$. Instead, we use the following lemma that shows that this upper bound is independent of w.

Lemma 5.4. *Let $\mathcal{R}$ be a cyclotomic ring of degree n that is a power-of-2. Then, the maximal singular value of any matrix in $\mathcal{L}_{k,m}^{\times w}(\beta)$ is bounded by*

$$\sigma_{max}(\mathcal{L}_{k,m}^{\times w}(\beta)) \leq \beta n \sqrt{km} \; .$$

Proof. Applying Lemma 5.3, we get for any $\mathbf{Z} = \mathrm{diag}(\mathbf{Z}_1, \ldots, \mathbf{Z}_w)$ with $\mathbf{Z}_i \in \mathcal{L}_{k,m}(\beta)$ and $\mathbf{x} = (\mathbf{x}_1^T, \ldots, \mathbf{x}_w^T)^T \in \mathbb{R}^{wmd}$ the inequality

$$\|\mathbf{A}_{\mathbf{Z}} \cdot \mathbf{x}\|_2 = \sqrt{\sum_{i \in [w]} \|\mathbf{A}_{\mathbf{Z}_i} \cdot \mathbf{x}_i\|_2^2} \leq \beta n \sqrt{km} \sqrt{\sum_{i \in [w]} \|\mathbf{x}_i\|_2^2} = \beta n \sqrt{km} \cdot \|\mathbf{x}\|_2 \; .$$

□

Adaptive error-leakage RingLWE. In this work, we are additionally interested in an *adaptive* version of error-leakage RingLWE. Here, the adversary may choose the leakage matrix $\mathbf{Z} \in \mathcal{L}$ after already seeing the RingLWE sample $\mathbf{y}$, and then obtains the leakage. $\mathbf{Z} \cdot \mathbf{e} + \overline{\mathbf{e}}$. Furthermore, this step may be repeated adaptively up to T times, with a separate leakage matrix $\mathbf{Z}^{(t)}$ and noise $\overline{\mathbf{e}}^{(t)}$.

Definition 5.3 (Adaptive $\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}}$ Assumption)**.** *We say that the adaptive $\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}}$ assumption (with some set $\mathcal{L} \subset \mathcal{R}^{k \times m}$) holds, if for any probabilistic adversary $\mathcal{A}$ with running time bounded by $t(\lambda)$, we have*

$$\left| \Pr[\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}}^{\mathcal{A},0} = 1] - \Pr[\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}}^{\mathcal{A},1} = 1] \right| \leq \frac{\mathrm{poly}(\lambda)}{2^\lambda} \cdot t(\lambda) \; ,$$

where the $\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}}^{\mathcal{A},b}$ experiment is defined as follows.

1. *The game samples public parameters $\mathbf{a} \leftarrow \mathcal{R}_q^m$ and noises $\mathbf{e} \leftarrow \chi^m$. It sends to $\mathcal{A}$ the pair $(\mathbf{a}, \mathbf{y})$, where the RingLWE sample $\mathbf{y}$ is computed as follows.*

- *If $b = 0$, sample $s \leftarrow \mathcal{R}_q$ and define $\mathbf{y}^T = s \cdot \mathbf{a}^T + \mathbf{e}^T \bmod q$.*
- *If $b = 1$, sample $\mathbf{y} \leftarrow \mathcal{R}_q^m$ uniformly at random.*

2. *For each $t \in [T]$:*

   - *$\mathcal{A}$ outputs leakage matrix $\mathbf{Z}^{(t)} \in \mathcal{L}$.*
   - *The game samples $\overline{\mathbf{e}}^{(t)} \leftarrow \chi^k$ and sends to $\mathcal{A}$ the leakage $\mathbf{l}^{(t)} = \mathbf{Z}^{(t)} \cdot \mathbf{e} + \overline{\mathbf{e}}^{(t)}$.*

3. *$\mathcal{A}$ outputs a bit b', which is also the output of the game.*

Note that the adaptive version is easily implied by standard RingLWE whenever the noise $\overline{\mathbf{e}}$ is sampled from a distribution $\overline{\chi}$ that statistically hides $\mathbf{Z} \cdot e$. This variant, called *noise flooding*, has the disadvantage of increasing the bitlength of modulus q by λ.

g-ary gadget vector. In order to constraint noise growth, it will sometimes be necessary to "decompose" an element $a \in \mathcal{R}_q$ (of arbitrary norm) into several ring elements $a_1, \ldots, a_m$ of norm $\|a\|_\infty < g$. We denote the number of elements in the decomposition by $m := \lceil \log_g q \rceil$. By $\mathbf{g}^{-1}$ we denote such a decomposition (written as a column vector), and its inverse is given by the vector

$$\mathbf{g}^T = \begin{pmatrix} 1 & g & g^2 & \ldots & g^{m-1} \end{pmatrix} \in \mathcal{R}_q^m .$$

5.4 Construction of Main Tool: Batch-Select

In this section, we discuss our new ideas towards an efficient *batch-select* construction. We first give its definition, then define two crucial building blocks LEnc and LHE in Sections 5.4.1 and 5.4.2, and present our full construction in Section 5.4.3. Finally, we discuss our RO-based optimization in Section 5.4.4.

Definition 5.4 (Batch-Select)**.** *A* batch-select functional encryption scheme *with abelian message space $\mathcal{M}(\lambda)$ consists of five efficient algorithms:*

- $\mathsf{Sel.Setup}(1^\lambda, 1^w)$ *takes the security parameter λ and dimension w. It outputs public parameters* pp*, implicitly given as input to all remaining algorithms.*

- $\mathsf{Sel.Enc}_1(\mathbf{l}_1)$ *takes a message vector* $\mathbf{l}_1 \in \mathcal{M}^w$*, and outputs a ciphertext* ct_1 *and a state* st_1.
- $\mathsf{Sel.Enc}_2(\mathbf{l}_2)$ *takes a message vector* $\mathbf{l}_2 \in \mathcal{M}^w$*, and outputs a ciphertext* ct_2 *and a state* st_2.
- $\mathsf{Sel.KeyGen}(\mathsf{st}_1, \mathsf{st}_2, \mathbf{y})$ *takes the two states* $\mathsf{st}_1, \mathsf{st}_2$ *returned by* $\mathsf{Enc}_1, \mathsf{Enc}_2$*, and a selection vector* $\mathbf{y} \in \{0,1\}^w$*. It outputs a decryption key* $\mathsf{sk}_{\mathbf{y}}$.
- $\mathsf{Sel.Dec}(\mathsf{sk}_{\mathbf{y}}, \mathsf{ct}_1, \mathsf{ct}_2, \mathbf{y})$ *takes the decryption key* $\mathsf{sk}_{\mathbf{y}}$*, two ciphertexts* $\mathsf{ct}_1, \mathsf{ct}_2$*, and the selection vector* $\mathbf{y}$*. It outputs a message vector* $\mathbf{l} \in \mathcal{M}^w$ *(which should be* $\mathbf{l} = \mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$*, where* $\odot$ *denotes component-wise multiplication).*

Correctness. *The scheme* Sel *is correct if for all* $\lambda \in \mathbb{N}$*,* $w \le 2^\lambda$*, message vectors* $\mathbf{l}_1, \mathbf{l}_2 \in \mathcal{M}^w$*, and selection vectors* $\mathbf{y} \in \{0,1\}^w$*, the following holds:*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_{\mathbf{y}}, \mathsf{ct}_1, \mathsf{ct}_2, \mathbf{y}) = \mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2 \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^w) \\ (\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1) \\ (\mathsf{ct}_2, \mathsf{st}_2) \leftarrow \mathsf{Sel.Enc}_2(\mathbf{l}_2) \\ \mathsf{sk}_{\mathbf{y}} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_1, \mathsf{st}_2, \mathbf{y}) \end{array}\right] = 1.$$

Definition 5.5 (*T*-times Simulation Security)**.** *A* Sel *scheme is* $T(\lambda)$-times 2^λ-simulation secure *if there exists an efficient simulator* $\mathsf{Sel.Sim}$*, such that for all* $\{w_\lambda\}_{\lambda\in\mathbb{N}}$*,* $\{\mathbf{l}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$*,* $\{\mathbf{l}_{2,\lambda}^{(1)}, \dots, \mathbf{l}_{2,\lambda}^{(T)}\}_{\lambda\in\mathbb{N}}$*, and* $\{\mathbf{y}_\lambda^{(1)}, \dots, \mathbf{y}_\lambda^{(T)}\}_{\lambda\in\mathbb{N}}$*, where* $w_\lambda \le \mathrm{poly}(\lambda)$*,* $\mathbf{l}_{1,\lambda}, \mathbf{l}_{2,\lambda}^{(t)} \in \mathcal{M}(\lambda)^{w_\lambda}$*, and* $\mathbf{y}_\lambda^{(t)} \in \{0,1\}^{w_\lambda}$*, the following holds (where we suppress the subscript* λ*):*

$$\left\{\mathsf{pp}, \mathsf{Sel.Sim}(\mathsf{pp}, \{\mathbf{l}^{(t)}, \mathbf{y}^{(t)}\}_{[T]}) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^w) \\ \mathbf{l}^{(t)} = \mathbf{l}_1 \odot \mathbf{y}^{(t)} + \mathbf{l}_2^{(t)} \quad \forall t \in [T] \end{array}\right\}_\lambda$$

$$\approx_{\mathrm{c}}^{2^\lambda} \left\{\begin{array}{l} \mathsf{pp}, (\mathsf{ct}_1, \\ \{\mathsf{ct}_2^{(t)}, \mathsf{sk}_{\mathbf{y}}^{(t)}\}_{[T]}) \end{array} \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^w) \\ (\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1) \\ (\mathsf{ct}_2^{(t)}, \mathsf{st}_2^{(t)}) \leftarrow \mathsf{Sel.Enc}_2(\mathbf{l}_2^{(t)}) \quad \forall t \in [T] \\ \mathsf{sk}_{\mathbf{y}}^{(t)} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_1, \mathsf{st}_2^{(t)}, \mathbf{y}^{(t)}) \quad \forall t \in [T] \end{array}\right\}_\lambda$$

We define two variations of batch-select: a *weak* version in which all entries of the reused message vector $\mathbf{l}_1$ need to be identical (this will allow for a more efficient instantiation), and a *random* version in which the message vector $\mathbf{l}_2$ is required to be uniformly random.

Definition 5.6 (*Weak* Batch-Select)**.** *A* weak *batch-select scheme is defined identically to Definition 5.4, except that it is guaranteed that all messages in $\mathbf{l}_1$ are identical. More specifically,* correctness *is only required to hold for messages $\mathbf{l}_1$ of the form $\mathbf{l}_1 = \Delta \cdot \mathbf{1}_w$ for some $\Delta \in \mathcal{M}$.*

Furthermore, for T-times 2^λ-simulation security *to hold, it suffices if the indistinguishability in Definition 5.5 holds for all $\{\mathbf{l}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$ where $\mathbf{l}_{1,\lambda} = \Delta \cdot \mathbf{1}_w$ for some $\Delta \in \mathcal{M}$.*

Definition 5.7 (*Random* Batch-Select)**.** *A* random *batch-select scheme is defined identically to Definition 5.4, except that the syntax of encryption algorithm $\mathsf{Sel.Enc}_2$ changes to:*

- $\mathsf{Sel.Enc}_2()$ *takes no input, and outputs a ciphertext ct_2, a message vector $\mathbf{l}_2 \in \mathcal{M}^w$, and a state st_2.*

The correctness property does not quantify over message vector $\mathbf{l}_2$. Instead, $\mathbf{l}_2$ is generated by $\mathsf{Sel.Enc}_2()$. Correctness is only required to hold with overwhelming probability $1 - \frac{\mathrm{poly}(\lambda)}{2^\lambda}$.

We say that a random batch-select scheme satisfies T-times 2^λ-simulation security*, if there exists an efficient simulator $\mathsf{Sel.Sim}$, such that for all $\{w_\lambda\}_{\lambda \in \mathbb{N}}$, $\{\mathbf{l}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$, and $\{\mathbf{y}_\lambda^{(1)}, \ldots, \mathbf{y}_\lambda^{(T)}\}_{\lambda \in \mathbb{N}}$, where $w_\lambda \leq \mathrm{poly}(\lambda)$, $\mathbf{l}_{1,\lambda} \in \mathcal{M}(\lambda)^{w_\lambda}$, and $\mathbf{y}_\lambda^{(t)} \in \{0,1\}^{w_\lambda}$, the following holds (where we suppress the subscript λ):*

$$\left\{ \mathsf{pp}, \{\mathbf{l}^{(t)}\}_{[T]}, \mathsf{Sel.Sim}(\mathsf{pp}, \{\mathbf{l}^{(t)}, \mathbf{y}^{(t)}\}_{[T]}) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^w) \\ \mathbf{l}^{(t)} \leftarrow \mathcal{M}^w \quad \forall t \in [T] \end{array} \right\}_\lambda$$

$$\approx_c^{2^\lambda} \left\{ \begin{array}{l} \mathsf{pp}, \\ \{\mathbf{l}_1 \odot \mathbf{y}^{(t)} + \mathbf{l}_2^{(t)}\}_{[T]}, \\ (\mathsf{ct}_1, \{\mathsf{ct}_2^{(t)}, \mathsf{sk}_{\mathbf{y}}^{(t)}\}_{[T]}) \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^w) \\ (\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1) \\ (\mathsf{ct}_2^{(t)}, \mathbf{l}_2^{(t)}, \mathsf{st}_2^{(t)}) \leftarrow \mathsf{Sel.Enc}_2() \quad \forall t \in [T] \\ \mathsf{sk}_{\mathbf{y}}^{(t)} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_1, \mathsf{st}_2^{(t)}, \mathbf{y}^{(t)}) \quad \forall t \in [T] \end{array} \right\}_\lambda$$

5.4.1 *First building block:* LEnc

Our first tool, *linear laconic encryption*, is formally defined in Definition 5.8. Intuitively, it can be viewed as a functional encryption for (noisily) evaluating $\mathbf{s} \odot \mathbf{a}$ over an encrypted vector $\mathbf{s}$ and a public vector $\mathbf{a}$, but with a specific secret key format and (noisy) decryption procedure as illustrated below.

$$\begin{aligned} &\text{Encrypt } \mathbf{s}: && (\mathsf{st} := \mathbf{r}, \mathsf{ct}) \leftarrow \mathsf{LEnc.Enc}(\mathbf{s}), \\ &\text{Generate } \mathsf{sk} \text{ w.r.t. } \mathbf{a}: && \mathsf{sk} = \mathbf{r} \cdot d_{\mathbf{a}}, \quad d_{\mathbf{a}} \leftarrow \mathsf{LEnc.Digest}(\mathbf{a}), \\ &\text{Decrypt } \mathsf{res} = \mathbf{s} \odot \mathbf{a}: && \mathsf{res} + \text{noise} = \boldsymbol{\delta} - \mathsf{sk}, \quad \boldsymbol{\delta} \leftarrow \mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}) \end{aligned}$$

In particular, the sk is required to be a vector-scalar product between a secret state $\mathbf{r}$ and a public digest of the vector $\mathbf{a}$. During decryption, the ciphertext can first be evaluated to $\boldsymbol{\delta}$ using only the public vector $\mathbf{a}$. The decryption result is then simply $\boldsymbol{\delta} - \mathsf{sk}$.

Definition 5.8 (Linear Laconic Encryption)**.** *A* LEnc *scheme with associated message space $\mathcal{R}_q(\lambda)$ and error bound $B_{\mathsf{LEnc}}(\lambda)$ consists of the following efficient algorithms:*

- $\mathsf{LEnc.Setup}(1^\lambda, w)$ *takes the security parameter λ and a message dimension w. It outputs public parameters* pp*, which is implicitly given as input to all remaining algorithms.*
- $\mathsf{LEnc.Enc}(\mathbf{s})$ *takes output keys $\mathbf{s} \in \mathcal{R}_q^w$. It returns input keys $\mathbf{r} \in \mathcal{R}_q^w$ and ciphertext* ct*.*
- $\mathsf{LEnc.Digest}(\mathbf{a})$ *takes the database $\mathbf{a} \in \mathcal{R}_q^w$. It outputs,* deterministically*, a digest $d_{\mathbf{a}} \in \mathcal{R}_q$.*
- $\mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a})$ *takes a ciphertext* ct *and database $\mathbf{a} \in \mathcal{R}_q^w$. It outputs $\boldsymbol{\delta} \in \mathcal{R}_q^w$.*

Correctness. *The* LEnc *scheme is correct if for all $\lambda \in \mathbb{N}$, $w \leq 2^\lambda$ output keys $\mathbf{s} \in \mathcal{R}_q^w$, and database $\mathbf{a} \in \mathcal{R}_q^w$ with $d_{\mathbf{a}} = \mathsf{LEnc.Digest}(\mathbf{a})$:*

$$\Pr\left[|\boldsymbol{\delta} - (\mathbf{r} \cdot d_{\mathbf{a}} - \mathbf{s} \odot \mathbf{a})| \leq B_{\mathsf{LEnc}} \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{LEnc.Setup}(1^\lambda, w) \\ \mathbf{r}, \mathsf{ct} \leftarrow \mathsf{LEnc.Enc}(\mathbf{s}) \\ \boldsymbol{\delta} \leftarrow \mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}) \end{array} \right] = 1 .$$

Security, as captured by the following definition, states that the secret vector $\mathbf{s}$ remains hidden even if the noise $\mathbf{e}$ accumulated by evaluation (i.e., the difference between the *actual* outcome $\boldsymbol{\delta}$ and the *expected* outcome $\mathbf{r} \cdot d_{\mathbf{a}} - \mathbf{s} \odot \mathbf{a}$) may be leaked. However, this only holds as long as this leakage is hidden by another noise $\overline{\mathbf{e}}$ sampled from some distribution $\overline{\chi}_{\mathsf{LEnc}}$.

Definition 5.9 (Simulation Security with T-noise Leakage)**.** *A* LEnc *scheme is* 2^λ-simulation secure under $T(\lambda)$-noise leakage w.r.t. to noise-hiding distribution $\overline{\chi}_{\mathsf{LEnc}}$ *if there exists an efficient simulator* $\mathsf{LEnc.Sim}$*, s.t. for all* $\{w_\lambda\}_{\lambda\in\mathbb{N}}$*,* $\{\mathbf{s}_\lambda\}_{\lambda\in\mathbb{N}}$*,* $\{\mathbf{a}_\lambda^{(t)}\}_{\lambda\in\mathbb{N},t\in[T]}$*, where* $w_\lambda \le \mathrm{poly}(\lambda)$*,* $\mathbf{s}_\lambda \in \mathcal{R}_q^{w_\lambda}$*, and* $\mathbf{a}_\lambda^{(t)} \in \mathcal{R}_q^{w_\lambda}$*, the following holds (where we suppress the subscript* λ*):*

$$\left\{ (\mathsf{pp}, \mathsf{ct}, \{\mathbf{e}^{(t)} + \overline{\mathbf{e}}^{(t)}\}_{[T]}) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{LEnc.Setup}(1^\lambda, w) \\ \mathbf{r}, \mathsf{ct} \leftarrow \mathsf{LEnc.Enc}(\mathbf{s}) \\ d_{\mathbf{a}}^{(t)} \leftarrow \mathsf{LEnc.Digest}(\mathbf{a}^{(t)}) \quad \forall t \in [T] \\ \boldsymbol{\delta}^{(t)} \leftarrow \mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}^{(t)}) \quad \forall t \in [T] \\ \mathbf{e}^{(t)} := \boldsymbol{\delta}^{(t)} - (\mathbf{r} \cdot d_{\mathbf{a}}^{(t)} - \mathbf{s} \odot \mathbf{a}^{(t)}) \quad \forall t \in [T] \\ \overline{\mathbf{e}}^{(t)} \leftarrow \overline{\chi}_{\mathsf{LEnc}}^{w} \quad \forall t \in [T] \end{array} \right\}_\lambda$$

$$\approx_c^{2^\lambda} \left\{ (\mathsf{pp}, \widetilde{\mathsf{ct}}, \{\widetilde{\mathbf{e}}^{(t)} + \overline{\mathbf{e}}^{(t)}\}_{[T]}) \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{LEnc.Setup}(1^\lambda, w) \\ \widetilde{\mathsf{ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]} \leftarrow \mathsf{LEnc.Sim}(\mathsf{pp}, \{\mathbf{a}^{(t)}\}_{[T]}) \\ \overline{\mathbf{e}}^{(t)} \leftarrow \overline{\chi}_{\mathsf{LEnc}}^{w} \quad \forall t \in [T] \end{array} \right\}_\lambda$$

The following variant of LEnc may be used to construct *weak* batch-select (Definition 5.6) instead of the full batch-select (Definition 5.4).

Definition 5.10 (*Weak* Linear Laconic Encryption)**.** *A* Weak *Linear Laconic Encryption scheme is defined identically to Definition 5.8, except that it is guaranteed that* all elements of the output key $\mathbf{s}$ are identical*. More specifically,* correctness *is only required to hold for output keys* $\mathbf{s}$ *of the form* $\mathbf{s} = s \cdot \mathbf{1}_w$ *for some* $s \in \mathcal{R}_q$*.*

Furthermore, for 2^λ-simulation security with $T(\lambda)$-noise leakage *to hold, it suffices if the indistinguishability in Definition 5.9 holds for all* $\{\mathbf{s}_\lambda\}_{\lambda\in\mathbb{N}}$ *where* $\mathbf{s}_\lambda = s_\lambda \cdot \mathbf{1}_w$ *for some* $s_\lambda \in \mathcal{R}_q$*.*

In Section 5.5, we present a construction of LEnc (adapted from [DKL+23]) based on the RingLWE assumption, as well as a modified version of a *weak* LEnc that is more efficient, but also requires a more agressive parameter choice. We summarize the results in the following lemma.

Lemma 5.5 (LEnc from RingLWE). *Assume $\mathsf{LWE}_{\mathcal{R},O(1),q,\mathcal{D}_{\mathcal{R},q^{0.1}}}$ holds (for a cyclotomic ring $\mathcal{R}$ of degree $n \geq \lambda$ that is a power-of-2, modulus $q \geq n^{15}$, and modulus-to-noise ratio $q^{0.9}$). Then, there exists an LEnc scheme over message space $\mathcal{R}_q$ with error bound $B_{\mathsf{LEnc}} \leq \sqrt{q}$ that is 2^λ-simulation secure under $\sqrt{q}$-noise leakage w.r.t. to a noise distribution $\overline{\chi}_{\mathsf{LEnc}}$ whose values are bounded by $\max \overline{\chi}_{\mathsf{LEnc}} = 1000\sqrt{q}$. Furthermore, when encrypting w-dimension messages, the components of the scheme have size (in terms of ring elements)*

$$|\mathsf{pp}| = O(1), \quad |\mathsf{ct}| = O(w \log w) \ ,$$

and its algorithms run in time (in terms of ring multiplication operations)

$$\mathsf{Enc} : O(w \log w), \quad \mathsf{Digest} : O(w), \quad \mathsf{Eval} : O(w \log w).$$

Under the same assumptions, there exists a weak LEnc *scheme secure under $\frac{\sqrt{q}}{w}$-noise leakage. Its ciphertext contains only $|\mathsf{ct}| = O(w)$ ring elements, and the* Enc *algorithm only requires $O(w)$ ring multiplications.*

5.4.2 Second building block: LHE *for Rings*

The second tool, *linearly homomorphic encryption*, is a functional encryption for (noisily) evaluating $\mathbf{m}_1 \cdot y + \mathbf{m}_2$ over encrypted vectors $\mathbf{m}_1, \mathbf{m}_2$ and a public element y.

Definition 5.11 (Noisy Linearly Homomorphic Encryption over Rings). *An* LHE *scheme with associated message space $\mathcal{R}_q(\lambda)$ and error bound $B_{\mathsf{LHE}}(\lambda)$ consists of the following efficient algorithms:*

- $\mathsf{LHE.Setup}(1^\lambda, 1^w)$ *takes the security parameter λ and a message dimension w. It outputs public parameters* pp*, which is implicitly given as input to all remaining algorithms.*

- $\mathsf{LHE.Enc}_1(\mathbf{m}_1)$ *takes a message vector* $\mathbf{m}_1 \in \mathcal{R}_q^w$*, and outputs a ciphertext* ct_1 *and a state* st_1*.*
- $\mathsf{LHE.Enc}_2(\mathbf{m}_2)$ *takes a message vector* $\mathbf{m}_2 \in \mathcal{R}_q^w$*, and outputs a ciphertext* ct_2 *and a state* st_2*.*
- $\mathsf{LHE.KeyGen}(\mathsf{st}_1, \mathsf{st}_2, y)$ *takes the two states* $\mathsf{st}_1, \mathsf{st}_2$ *output by* $\mathsf{Enc}_1, \mathsf{Enc}_2$*, and a ring element* $y \in \mathcal{R}_q$*. It outputs a decryption key* sk_y*.*
- $\mathsf{LHE.Dec}(\mathsf{sk}_y, \mathsf{ct}_1, \mathsf{ct}_2, y)$ *takes decryption key* sk_y *and the two ciphertexts* $\mathsf{ct}_1, \mathsf{ct}_2$*. It outputs a decrypted message* $\mathbf{m}_{\mathsf{res}} \in \mathcal{R}_q^w$*.*

Correctness. *The* LHE *scheme is correct if for all* $\lambda, w \in \mathbb{N}$*, messages* $\mathbf{m}_1, \mathbf{m}_2 \in \mathcal{R}_q^w$*, and coefficients* $y \in \mathcal{R}_q$*:*

$$\Pr\left[\|\mathbf{m}_{\mathsf{res}} - (\mathbf{m}_1 \cdot y + \mathbf{m}_2)\|_\infty \leq B_{\mathsf{LHE}} \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^w) \\ \mathsf{ct}_1, \mathsf{st}_1 \leftarrow \mathsf{LHE.Enc}_1(\mathbf{m}_1) \\ \mathsf{ct}_2, \mathsf{st}_2 \leftarrow \mathsf{LHE.Enc}_2(\mathbf{m}_2) \\ \mathsf{sk}_y \leftarrow \mathsf{LHE.KeyGen}(\mathsf{st}_1, \mathsf{st}_2, y) \\ \mathbf{m}_{\mathsf{res}} \leftarrow \mathsf{LHE.Dec}(\mathsf{sk}_y, \mathsf{ct}_1, \mathsf{ct}_2, y) \end{array}\right] = 1 \ .$$

Definition 5.12 (T-times Simulation Security)**.** *An* LHE *scheme is* $T(\lambda)$-time 2^λ-simulation secure *if there exists an efficient simulator* $\mathsf{LHE.Sim}$*, s.t. for all* $\{w_\lambda\}_{\lambda\in\mathbb{N}}$*,* $\{\mathbf{m}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$*,* $\{\mathbf{m}_{2,\lambda}^{(1)}, \ldots, \mathbf{m}_{2,\lambda}^{(T)}\}_{\lambda\in\mathbb{N}}$*,* $\{y_\lambda^{(1)}, \ldots, y_\lambda^{(T)}\}_{\lambda\in\mathbb{N}}$*, where* $w_\lambda \leq \mathrm{poly}(\lambda)$*,* $\mathbf{m}_{1,\lambda}, \mathbf{m}_{2,\lambda}^{(t)} \in \mathcal{R}_q^w$*, and* $y_\lambda^{(t)} \in \mathcal{R}_q$*, the following holds (where we suppress the subscript* λ*):*

$$\left\{\mathsf{pp}, \mathsf{LHE.Sim}(\mathsf{pp}, \{\mathbf{m}_{\mathsf{res}}^{(t)}, y^{(t)}\}_{[T]}) \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^w) \\ \mathbf{m}_{\mathsf{res}}^{(t)} := \mathbf{m}_1 \cdot y^{(t)} + \mathbf{m}_2^{(t)} \quad \forall t \in [T] \end{array}\right\}_\lambda$$

$$\approx_{\mathrm{c}}^{2^\lambda} \left\{ \mathsf{pp}, (\mathsf{ct}_1, \{\mathsf{ct}_2^{(t)}, \mathsf{sk}_y^{(t)}\}_{[T]}) \;\middle|\; \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^w) \\ \mathsf{ct}_1, \mathsf{st}_1 \leftarrow \mathsf{LHE.Enc}_1(\mathbf{m}_1) \\ \mathsf{ct}_2^{(t)}, \mathsf{st}_2^{(t)} \leftarrow \mathsf{LHE.Enc}_2(\mathbf{m}_2^{(t)}) \quad \forall t \in [T] \\ \mathsf{sk}_y^{(t)} \leftarrow \mathsf{LHE.KeyGen}(\mathsf{st}_1, \mathsf{st}_2^{(t)}, y^{(t)}) \quad \forall t \in [T] \end{array}\right\}_\lambda$$

In Section 5.6, we present a construction of LHE based on RingLWE, and summarize the result in the following lemma.

Lemma 5.6 (LHE from RingLWE)**.** *Assume* $\mathsf{LWE}_{\mathcal{R},w,q,\mathcal{D}_{\mathcal{R},q^{0.1}}}$ *holds (for a cyclotomic ring* $\mathcal{R}$ *of degree* $n \geq \lambda$ *that is a power-of-2, modulus* $q \geq n^{15}$*, and modulus-to-noise ratio* $q^{0.9}$*). Then, there exists an* LHE *scheme over message space* $\mathcal{R}_q$ *with error bound* $B_{\mathsf{LHE}} \leq 10\sqrt{q}$ *that is* $\sqrt{q}$*-times* 2^{λ}*-simulation secure. Furthermore, when encrypting* w*-dimension messages, the components of the scheme have size*

$$|\mathsf{pp}| = O(w), \quad |\mathsf{ct}_1| = O(w), \quad |\mathsf{ct}_2| = w, \quad |\mathsf{sk}| = 1 ,$$

and its algorithms run in time (in terms of ring multiplications operations)

$$\mathsf{Enc}_1 : O(w), \quad \mathsf{Enc}_2 : O(w), \quad \mathsf{KeyGen} : O(1), \quad \mathsf{Dec} : O(w).$$

5.4.3 *Instantiating Batch-Select from* LEnc *and* LHE

We now present a construction of *batch-select* (Definition 5.4) based on LEnc and LHE. We will support the message space $\mathcal{M} = \mathbb{Z}_p$, by working over a cyclotomic ring $\mathcal{R}$ of a power-of-2 degree n and composite modulus $q = p \cdot \Delta$. To ensure correctness, Δ must fulfill $\Delta/2 > B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}}$ (where B_{LEnc} and B_{LHE} denote the maximum error incurred by LEnc and LHE, and $\overline{\chi}_{\mathsf{LEnc}}$ is the distribution of noise required to hide any noise leakage from LEnc; see Definition 5.9).

Plaintext encoding. We use the Chinese Remainder Theorem (CRT) to pack n elements of $\mathbb{Z}_p$ elements into a single plaintext in $\mathcal{R}_p$. Specifically, in our choice of the ring $\mathcal{R} = \mathbb{Z}[X]/\Phi(m)$, the m-th cyclotomic polynomial (of degree $n = \phi(m)$) splits mod p if $p \equiv 1 \bmod m$, i.e., $\Phi(m) = F_1 \cdot \ldots \cdot F_n$ over $\mathbb{F}_p$, where F_i are degree 1 polynomials. By CRT, the plaintext space is isomorphic to n times $\mathbb{Z}_p$:

$$\mathcal{R}_p = \mathbb{Z}_p[X]/\phi(m) \cong \mathbb{Z}_p[X]/F_1(X) \times \ldots \times \mathbb{Z}_p[X]/F_n(X) \cong \mathbb{Z}_p^n.$$

We write $\mathsf{crt} : \mathbb{Z}_p^n \to \mathcal{R}_p$ to denote a ring isomorphism that "encodes" n plaintexts from the space $\mathbb{Z}_p$ as a single ring element in $\mathcal{R}_p$. Applying CRT to the plaintext space of RingLWE

is a common idea in the literature [SV10, GHS12]. We refer readers to [GHS12] for more details of the underlying algebra.

Using packing, we obtain an efficient encoding Encode that maps w-dimensional messages $\mathbf{l} \in \mathcal{M}^w$ to $w' = \lceil \frac{w}{n} \rceil$-dimensional ring elements in $\mathcal{R}_p^{w'}$. Importantly, this mapping fulfills the identity $\mathsf{Encode}(\mathbf{a} \odot \mathbf{b}) = \mathsf{Encode}(\mathbf{a}) \odot \mathsf{Encode}(\mathbf{b})$ for any vectors $\mathbf{a}, \mathbf{b} \in \mathcal{M}^w$, and similarly for addition. Note that while Encode maps to $\mathcal{R}_p^{w'}$, in our construction we will treat the resulting vectors as elements in the larger space $\mathcal{R}_q^{w'}$.

Construction 14 (Batch-Select Sel)**.** $\underline{\mathsf{Setup}(1^\lambda, 1^w) \to \mathsf{pp}}$**:** Run $\mathsf{pp}_{\mathsf{LHE}} \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^{w'})$, $\mathsf{pp}_{\mathsf{LEnc}} \leftarrow \mathsf{LEnc.Setup}(1^\lambda, w')$ with $w' = \lceil \frac{w}{n} \rceil$ as defined above, and output $\mathsf{pp} := (\mathsf{pp}_{\mathsf{LHE}}, \mathsf{pp}_{\mathsf{LEnc}})$.

$\underline{\mathsf{Enc}_1(\mathbf{l}_1) \to \mathsf{ct}_1, \mathsf{st}_1}$**:** Encode messages $\mathbf{l}_1 \in \mathcal{M}^w = \mathbb{Z}_p^w$ as a vector $\widehat{\mathbf{l}}_1 = \mathsf{Encode}(\mathbf{l}_1) \in \mathcal{R}_p^{w'}$, and then compute the laconic encryption ciphertext $\mathsf{LEnc.ct}$ for the resulting vector of ring elements:

$$\mathbf{r}, \mathsf{LEnc.ct} \leftarrow \mathsf{LEnc.Enc}(\widehat{\mathbf{l}}_1 \cdot \Delta)$$

Encrypt the corrosponding LEnc keys $\mathbf{r}$ using the *reusable* LHE component:

$$\mathsf{LHE.ct}_1, \mathsf{st}_1 \leftarrow \mathsf{LHE.Enc}_1(\mathbf{r})$$

Return ciphertext $\mathsf{ct}_1 := (\mathsf{LEnc.ct}, \mathsf{LHE.ct}_1)$ and state st_1.

$\underline{\mathsf{Enc}_2(\mathbf{l}_2) \to \mathsf{ct}_2, \mathsf{st}_2}$**:** Encode messages $\mathbf{l}_2 \in \mathcal{M}^w = \mathbb{Z}_p^w$ as a vector $\widehat{\mathbf{l}}_2 = \mathsf{Encode}(\mathbf{l}_2) \in \mathcal{R}_p^{w'}$. In addition, sample noise $\overline{\mathbf{e}}_{\mathsf{LEnc}} \leftarrow \overline{\chi}_{\mathsf{LEnc}}^{w'}$ (recall that LEnc-security only holds as long as its leakage is hidden by noise sampled from $\overline{\chi}_{\mathsf{LEnc}}$).

Then, encrypt $\widehat{\mathbf{l}}_2 \cdot \Delta + \overline{\mathbf{e}}_{\mathsf{LEnc}}$ using the *non-reusable* LHE component.

$$\mathsf{LHE.ct}_2, \mathsf{st}_2 \leftarrow \mathsf{LHE.Enc}_2(\widehat{\mathbf{l}}_2 \cdot \Delta + \overline{\mathbf{e}}_{\mathsf{LEnc}})$$

Return ciphertext $\mathsf{ct}_2 := \mathsf{LHE.ct}_2$ and state st_2.

$\underline{\mathsf{KeyGen}(\mathsf{st}_1, \mathsf{st}_2, \mathbf{y}) \to \mathsf{sk}_\mathbf{y}}$**:** Encode selection vector $\mathbf{y} \in \{0,1\}^w \subseteq \mathbb{Z}_p^w$ as a vector $\widehat{\mathbf{y}} = \mathsf{Encode}(\mathbf{y}) \in \mathcal{R}_p^{w'}$. Then, compute the digest $d_{\widehat{\mathbf{y}}} \leftarrow \mathsf{LEnc.Digest}(\widehat{\mathbf{y}})$, and return the LHE secret key $\mathsf{sk}_\mathbf{y} \leftarrow \mathsf{LHE.KeyGen}(\mathsf{st}_1, \mathsf{st}_2, d_{\widehat{\mathbf{y}}})$.

$\underline{\mathsf{Dec}(\mathsf{sk}_\mathbf{y}, \mathsf{ct}_1, \mathsf{ct}_2, \mathbf{y}) \to \mathbf{l}_\mathbf{y}}$**:** As in KeyGen, encode selection vector $\mathbf{y} \in \{0,1\}^w$ as $\widehat{\mathbf{y}} = \mathsf{Encode}(\mathbf{y}) \in \mathcal{R}_p^{w'}$ and compute its digest $d_{\widehat{\mathbf{y}}} \leftarrow \mathsf{LEnc.Digest}(\widehat{\mathbf{y}})$. Parse $\mathsf{ct}_1 = (\mathsf{LEnc.ct}, \mathsf{LHE.ct}_1)$, $\mathsf{ct}_2 = \mathsf{LHE.ct}_2$.

To decrypt, first evaluate the LHE to obtain res':

$$\mathsf{res}' \leftarrow \mathsf{LHE.Dec}(\mathsf{sk}_\mathbf{y}, \mathsf{LHE.ct}_1, \mathsf{LHE.ct}_2, d_{\widehat{\mathbf{y}}})$$
$$// \text{ s.t. } \mathsf{res}' = \mathbf{r} \cdot d_{\widehat{\mathbf{y}}} + \widehat{\mathbf{l}}_2 \cdot \Delta + \text{noise}$$

Now we will evaluate the laconic encryption to obtain $\mathbf{r} \cdot d_{\widehat{\mathbf{y}}} - (\widehat{\mathbf{l}}_1 \odot \widehat{\mathbf{y}}) \cdot \Delta$ (plus noise). Subtracting this from res' yields the encoding res of the desired message $\mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$:

$$\mathsf{res} := \mathsf{res}' - \mathsf{LEnc.Eval}(\mathsf{LEnc.ct}, \widehat{\mathbf{y}})$$
$$// \text{ s.t. } \mathsf{res} = (\widehat{\mathbf{l}}_1 \odot \widehat{\mathbf{y}}) \cdot \Delta + \widehat{\mathbf{l}}_2 \cdot \Delta + \text{noise}$$

Finally, return the decoded result $\mathsf{Encode}^{-1}(\lfloor \frac{\mathsf{res}}{\Delta} \rceil)$.

Correctness. First, denote by $\mathbf{l} = \mathbf{l}_1 \odot \mathbf{y} + \mathbf{l}_2$ (over $\mathbb{Z}_p$) be the required output of $\mathsf{Sel.Dec}$, with encoding $\widehat{\mathbf{l}} := \mathsf{Encode}(\mathbf{l})$, which (by the properties of Encode) is equal to $\widehat{\mathbf{l}}_1 \odot \widehat{\mathbf{y}} + \widehat{\mathbf{l}}_2$ over $\mathcal{R}_p$. Considering the larger ring $\mathcal{R}_q$ that our construction works on, we have the identity $\widehat{\mathbf{l}} \cdot \Delta = (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}} + \widehat{\mathbf{l}}_2 \cdot \Delta$, because of

$$\begin{aligned} \widehat{\mathbf{l}} \cdot \Delta &= [\widehat{\mathbf{l}}_1 \odot \widehat{\mathbf{y}} + \widehat{\mathbf{l}}_2]_p \cdot \Delta = (\widehat{\mathbf{l}}_1 \odot \widehat{\mathbf{y}} + \widehat{\mathbf{l}}_2 - p \cdot \mathbf{z}) \cdot \Delta \\ &= (\widehat{\mathbf{l}}_1 \odot \widehat{\mathbf{y}}) \cdot \Delta + \widehat{\mathbf{l}}_2 \cdot \Delta - \underbrace{(p \cdot \Delta)}_{=q} \cdot \mathbf{z} \\ &= (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}} + \widehat{\mathbf{l}}_2 \cdot \Delta \pmod{q} \,, \end{aligned}$$

where $[.]_p$ denotes reduction modulo p, and $\mathbf{z}$ is some vector in $\mathcal{R}_q^{w'}$. Note that this equality crucially depends on the fact that the plaintext modulus p divides the larger ring modulus q.

Now, by correctness of the underlying LHE, we have

$$\|\mathsf{res}' - (\mathbf{r} \cdot d_{\widehat{\mathbf{y}}} + \widehat{\mathbf{l}}_2 \cdot \Delta + \overline{\mathbf{e}}_{\mathsf{LEnc}})\|_\infty \leq B_{\mathsf{LHE}} \ ,$$

and by correctness of the underlying LEnc, we have

$$\| \underbrace{\mathsf{LEnc.Eval}(\mathsf{LEnc.ct}, \widehat{\mathbf{y}})}_{=:\boldsymbol{\delta}} - (\mathbf{r} \cdot d_{\widehat{\mathbf{y}}} - (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}})\|_\infty \leq B_{\mathsf{LEnc}} \ .$$

Combining these two bounds, the overall error is at most

$$\begin{aligned}
\|\mathsf{res} - \widehat{\mathbf{l}} \cdot \Delta\|_\infty &= \|\mathsf{res}' - \boldsymbol{\delta} - ((\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}} + \widehat{\mathbf{l}}_2 \cdot \Delta)\|_\infty \\
&= \|(\mathsf{res}' - \mathbf{r} \cdot d_{\widehat{\mathbf{y}}} - \widehat{\mathbf{l}}_2 \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}) - (\boldsymbol{\delta} - \mathbf{r} \cdot d_{\widehat{\mathbf{y}}} + (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}) + \overline{\mathbf{e}}_{\mathsf{LEnc}}\|_\infty \\
&\leq B_{\mathsf{LHE}} + B_{\mathsf{LEnc}} + \|\overline{\mathbf{e}}_{\mathsf{LEnc}}\|_\infty \\
&\leq B_{\mathsf{LHE}} + B_{\mathsf{LEnc}} + \max \overline{\chi}_{\mathsf{LEnc}} < \frac{\Delta}{2} \ ,
\end{aligned}$$

and therefore the output $\mathsf{Decode}(\lfloor \frac{\mathsf{res}}{\Delta} \rceil)$ will be equal to $\mathsf{Decode}(\widehat{\mathbf{l}}) = \mathbf{l}$.

Lemma 5.7 (Security of Batch-Select, Construction 14)**.** *Assuming that the underlying* LHE *scheme is T-times 2^λ-simulation secure, and the underlying* LEnc *scheme is 2^λ-simulation secure with T-noise leakage, Construction 14 is a correct batch-select scheme that fulfills T-times 2^λ-simulation security.*

Proof. We define the simulator $\mathsf{Sel.Sim}$, which receives input $\mathsf{pp} = (\mathsf{pp}_{\mathsf{LHE}}, \mathsf{pp}_{\mathsf{LEnc}})$ as well as T message and selection vectors $\{\mathbf{l}^{(t)}, \mathbf{y}^{(t)}\}_{[T]}$, as follows (where we define encoded values as in the construction: $\widehat{\mathbf{y}}^{(t)} := \mathsf{Encode}(\mathbf{y}^{(t)})$ and $\widehat{\mathbf{l}}^{(t)} := \mathsf{Encode}(\mathbf{l}^{(t)})$):

- First, run the laconic encryption simulator

$$\widetilde{\mathsf{LEnc.ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]} \leftarrow \mathsf{LEnc.Sim}(\mathsf{pp}_{\mathsf{LEnc}}, \{\widehat{\mathbf{y}}^{(t)}\}_{[T]}) \ .$$

- Second, compute the digest $d_{\widehat{\mathbf{y}}}^{(t)}$ and evaluate the laconic encryption $\boldsymbol{\delta}^{(t)}$:

$$\begin{aligned}
d_{\widehat{\mathbf{y}}}^{(t)} &\leftarrow \mathsf{LEnc.Digest}(\widehat{\mathbf{y}}^{(t)}) \quad \forall i \in [T] \\
\boldsymbol{\delta}^{(t)} &\leftarrow \mathsf{LEnc.Eval}(\widetilde{\mathsf{LEnc.ct}}, \widehat{\mathbf{y}}^{(t)}) \quad \forall i \in [T]
\end{aligned}$$

- Third, simulate the output $\widetilde{\mathsf{res}}'$ of the LHE, and invoke the LHE simulator to simulate all LHE ciphertexts and keys (note that $\boldsymbol{\delta}^{(t)} - \widetilde{\mathbf{e}}^{(t)}$ is equal to the "noise-free" $\mathbf{r} \cdot d_{\widehat{\mathbf{y}}}^{(t)} - ((\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)})$, and therefore $\widetilde{\mathsf{res}}'^{(t)}$ is equal to $\mathbf{r} \cdot d_{\widehat{\mathbf{y}}}^{(t)} + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)}$):

$$\overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} \leftarrow \overline{\chi}_{\mathsf{LEnc}}^{w'} \quad \forall i \in [T]$$
$$\widetilde{\mathsf{res}}'^{(t)} := \boldsymbol{\delta}^{(t)} - \widetilde{\mathbf{e}}^{(t)} + \widehat{\mathbf{l}}^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} \quad \forall i \in [T]$$
$$\widetilde{\mathsf{LHE.ct}}_1, \{\widetilde{\mathsf{LHE.ct}}_2^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{y}^{(t)}}^{(t)}\}_{[T]} \leftarrow \mathsf{LHE.Sim}(\mathsf{pp}, \{\widetilde{\mathsf{res}}'^{(t)}, d_{\widehat{\mathbf{y}}}^{(t)}\}_{[T]})$$

- Finally, return $\widetilde{\mathsf{ct}}_1 := (\widetilde{\mathsf{LEnc.ct}}, \widetilde{\mathsf{LHE.ct}}_1)$ and $\{\widetilde{\mathsf{LHE.ct}}_2^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{y}^{(t)}}^{(t)}\}_{[T]}$.

We show a series of hybrid experiments that transitions from Hyb_0 (the "real" distribution as defined in Definition 5.5) to Hyb_3 (the "simulated" distribution). We abuse notation to also write Hyb_i as the output of the distribution of the corresponding experiment.

Hyb_0 To recall, Hyb_0 (omitting generation of public parameters pp) computes $\mathsf{ct}_1 = (\mathsf{LEnc.ct}, \mathsf{LHE.ct}_1)$ and $\{\mathsf{LHE.ct}_2^{(t)}, \mathsf{sk}_{\mathbf{y}^{(t)}}^{(t)}\}_{[T]}$ in the following way (where, as in the construction, we define the digests $d_{\widehat{\mathbf{y}}}^{(t)} := \mathsf{LEnc.Digest}(\widehat{\mathbf{y}}^{(t)})$, and encoded values $\widehat{\mathbf{l}}_1 := \mathsf{Encode}(\mathbf{l}_1)$ and $\widehat{\mathbf{l}}_2 := \mathsf{Encode}(\mathbf{l}_2)$):

$$\mathbf{r}, \boxed{\mathsf{LEnc.ct}} \leftarrow \mathsf{LEnc.Enc}(\widehat{\mathbf{l}}_1 \cdot \Delta)$$
$$\boxed{\mathsf{LHE.ct}_1}, \mathsf{st}_1 \leftarrow \mathsf{LHE.Enc}_1(\mathbf{r})$$

For $t \in [T]$:

$$\boxed{\mathsf{LHE.ct}_2^{(t)}}, \mathsf{st}_2^{(t)} \leftarrow \mathsf{LHE.Enc}_2(\widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)}) \quad \Big| \quad \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} \leftarrow \overline{\chi}_{\mathsf{LEnc}}^{w'}$$
$$\boxed{\mathsf{sk}_{\mathbf{y}^{(t)}}^{(t)}} \leftarrow \mathsf{LHE.KeyGen}(\mathsf{st}_1, \mathsf{st}_2^{(t)}, d_{\widehat{\mathbf{y}}}^{(t)})$$

Hyb_1 We now use the simulation security of LHE to replace generation of $\widetilde{\mathsf{LHE.ct}}_1, \{\widetilde{\mathsf{LHE.ct}}_2^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{y}^{(t)}}^{(t)}\}_{[T]}$ (i.e., the final three lines in Hyb_0) by the following:

$$\widetilde{\mathsf{LHE.ct}}_1, \{\widetilde{\mathsf{LHE.ct}}_2^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{y}^{(t)}}^{(t)}\}_{[T]} \leftarrow \mathsf{LHE.Sim}(\mathsf{pp}, \{\mathsf{res}'^{(t)}, d_{\widehat{\mathbf{y}}}^{(t)}\}_{[T]})$$
$$\Big| \quad \mathsf{res}'^{(t)} \leftarrow \mathbf{r} \cdot d_{\widehat{\mathbf{y}}}^{(t)} + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} \quad \forall t \in [T]$$

The indistinguishability $\mathsf{Hyb}_0 \approx_c^{2^\lambda} \mathsf{Hyb}_1$ follows directly from the T-times simulation security of LHE.

Hyb_2 In Hyb_2, we now rewrite $\mathsf{res}'^{(t)}$ (for every $t \in [T]$) as

$$\begin{aligned}\boldsymbol{\delta}^{(t)} &\leftarrow \mathsf{LEnc.Eval}(\mathsf{LEnc.ct}, \widehat{\mathbf{y}}^{(t)})\\ \mathbf{e}^{(t)} &:= \boldsymbol{\delta}^{(t)} - (\mathbf{r} \cdot d_{\widehat{\mathbf{y}}}^{(t)} - (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)})\\ \mathsf{res}'^{(t)} &\leftarrow \boldsymbol{\delta}^{(t)} - \mathbf{e}^{(t)} + \widehat{\mathbf{l}}^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)}\end{aligned}$$

(Note that $\mathbf{e}^{(t)}$ denotes the LEnc "evaluation error", i.e., the difference between the expected result $\mathbf{r} \cdot y^{(t)} - (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)}$ and the noisy outcome $\boldsymbol{\delta}^{(t)}$.)

This way of defining $\mathsf{res}'^{(t)}$ is identical to the previous one:

$$\begin{aligned}\mathbf{r} \cdot d_{\widehat{\mathbf{y}}}^{(t)} + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} &= (\boldsymbol{\delta}^{(t)} + (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)} - \mathbf{e}^{(t)}) + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)}\\ &= \boldsymbol{\delta}^{(t)} - \mathbf{e}^{(t)} + (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)} + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)}\\ &= \boldsymbol{\delta}^{(t)} - \mathbf{e}^{(t)} + \widehat{\mathbf{l}}^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)}\ ,\end{aligned}$$

where the last equality follows from $\widehat{\mathbf{l}}^{(t)} \cdot \Delta = (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)} + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta$, which was shown in the correctness section.

Therefore, we get $\mathsf{Hyb}_1 \equiv \mathsf{Hyb}_2$.

Hyb_3 Note that Hyb_2 does not make use of $\widehat{\mathbf{l}}_2^{(t)}$ anymore, but only $\widehat{\mathbf{l}}_1$ (in $\mathsf{LEnc.ct}$ and when computing the LEnc error $\mathbf{e}^{(t)}$). We now use the simulation security of LEnc in order to eliminate the final usages of $\widehat{\mathbf{l}}_1$, by simulating $\mathsf{LEnc.ct}$ and $\mathbf{e}^{(t)}$. Specifically, in Hyb_3, we replace the lines that compute $\mathsf{LEnc.ct}$ and $\mathbf{e}^{(t)}$ by

$$\widetilde{\mathsf{LEnc.ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]} \leftarrow \mathsf{LEnc.Sim}(\mathsf{pp}_{\mathsf{LEnc}}, \{\widehat{\mathbf{y}}^{(t)}\}_{[T]})\ .$$

Because $\mathbf{e}^{(t)}$ is only used as part of the quantity $\mathbf{e}^{(t)} + \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)}$ (in the definition of $\mathsf{res}'^{(t)}$) with $\overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)}$ being a fresh noise generated from $\overline{\chi}_{\mathsf{LEnc}}^{w'}$, we get the indistinguishability $\mathsf{Hyb}_2 \approx_c^{2^\lambda} \mathsf{Hyb}_3$.

Observe that Hyb_3 proceeds identically as the simulator $\mathsf{Sel.Sim}$. By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_c^{2\lambda} \mathsf{Hyb}_3$, which proves the security. $\square$

Our construction above works over message space $\mathbb{Z}_p$, where p depends on RingLWE parameters. However, in our garbling application, we require a message space of size at least 2^λ (so that messages may hold input labels). To handle this even with small RingLWE modulus, we may easily turn our construction into a batch-select scheme with message space $\mathbb{Z}_p^\ell$ (s.t. $p^\ell \geq 2^\lambda$) by splitting each message across ℓ slots. Then, by combining this with underlying LEnc and LHE (Lemmas 5.5 and 5.6), we can e.g. construct a batch-select scheme with the following parameters:

Theorem 5.3 (Batch-Select)**.** *Assume* $\mathsf{LWE}_{\mathcal{R},\infty,q,\mathcal{D}_{\mathcal{R},q^{0.1}}}$ *holds (for a cyclotomic ring* $\mathcal{R}$ *of degree* $n \geq \lambda$ *that is a power-of-2, modulus* $q \geq n^{15}$ *with* $q \geq 2^{60}$ *of length* $\log q = O(\lambda)$*, and modulus-to-noise ratio* $q^{0.9}$*). Furthermore, suppose modulus* q *is a composite number* $q = p \cdot \Delta$ *for some plaintext modulus* $p \leq q^{1/4}$ *(with* $p = q^{\Theta(1)}$*) that is an "NTT friendly" prime, i.e.,* n *divides* $(p-1)$*. Then, there exists a batch-select scheme with message space* $\mathcal{M} := \mathbb{Z}_p^\ell$ *of size* $|\mathcal{M}| \geq 2^\lambda$ *that is* $\sqrt{q}$*-times* 2^λ*-simulation secure. When encrypting* w*-dimensional messages (with* $w \geq \Omega(n)$*), the scheme's components have size*

$$|\mathsf{pp}| = O(w \cdot \lambda), \quad |\mathsf{ct}_1| = O(w \log w \cdot \lambda), \quad |\mathsf{ct}_2| = O(w \cdot \lambda), \quad |\mathsf{sk}_{\mathbf{y}}| = \mathrm{poly}(\lambda) ,$$

and its algorithms run in time (in terms of ring multiplication operations):

$$\mathsf{Enc}_1 : O\left(\frac{w \log w}{n}\right), \quad \mathsf{Enc}_2 : O\left(\frac{w}{n}\right), \quad \mathsf{KeyGen} : O\left(\frac{w}{n}\right), \quad \mathsf{Dec} : O\left(\frac{w \log w}{n}\right) .$$

A weak *batch-select scheme with* $\frac{\sqrt{q}}{w}$*-times* 2^λ*-simulation security can be constructed under the same parameters. Then, ciphertext* ct_1 *has only size* $|\mathsf{ct}_1| = O(w \cdot \lambda)$*, and the* Enc_1 *algorithm only requires* $O(\frac{w}{n})$ *ring multiplications.*

The latter also implies a weak batch-select scheme with 1*-time* 2^λ*-simulation security whenever* $q \geq \omega((w\lambda)^2)$*, where the size of* ct_1 *reduces to* $|\mathsf{ct}_1| = o(w)$*.*

Proof of Theorem 5.3. Under the premises of this theorem, both primitives LEnc and LHE underlying our batch-select construction fulfill the error bounds as described in Lemmas 5.5 and 5.6. Thus, due to $p \leq q^{1/4}$, i.e., $\Delta \geq q^{3/4}$, we get

$$B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}} \leq \sqrt{q} + 10\sqrt{q} + 1000\sqrt{q} < \frac{1}{2} q^{3/4} \leq \Delta/2 \ ,$$

and the batch-select construction fulfills correctness (here we used $q \geq 2^{60}$).

The claimed efficienies follow from those of LEnc and LHE: recall that these two building blocks are applied to dimension $w' = \frac{w\ell}{n}$, where $\ell = \lceil \frac{\lambda}{\log p} \rceil$. Due to $\log p = \Omega(\log q)$ and $\log q = O(\lambda)$, this means $\ell = \Theta(\frac{\lambda}{\log q})$. Each ring element has size $n \cdot \log q$, and therefore we get ciphertext sizes

$$|\mathsf{ct}_1| = |\mathsf{LEnc}.\mathsf{ct}| + |\mathsf{LHE}.\mathsf{ct}_1| = O((w' \log w' + w') \cdot n \cdot \log q) = O(w \log w \cdot \lambda) \quad \text{and}$$
$$|\mathsf{ct}_2| = |\mathsf{LHE}.\mathsf{ct}_2| = O(w' \cdot n \cdot \lambda) = O(w \cdot \lambda) \ ,$$

and the running time follows in a similar manner.

Weak batch-select with $\frac{\sqrt{q}}{w}$-times simulation security and $|\mathsf{ct}_1| = O(w \cdot \lambda)$ follows with our weak LEnc construction.

A weak batch-select scheme with 1-time simulation security for modulus $q \geq \omega((w\lambda)^2)$ with $|\mathsf{ct}_1| = o(w)$ follows by simply applying the previous weak batch-select scheme in parallel for $k = \omega(\lambda)$ times on smaller instances of size $\frac{w}{k}$, which yields $|\mathsf{ct}_1| = O(w \cdot \lambda / k) = o(w)$ due to $w \leq 2^{\lambda}$. The new secret key sk will contain k individual keys of size $\mathrm{poly}(\lambda)$ each, and hences its size is still in $\mathrm{poly}(\lambda)$. Note that this trick does not change the size of $|\mathsf{ct}_2|$, because this ciphertext now consists of k individual ciphertexts of size $O(\frac{w}{k} \cdot \lambda)$. □

5.4.4 Sending encryptions of random strings with a random oracle

We now present an optimization that only allows constructing a *random* batch-select scheme in the ROM. For this construction, we cannot use LHE as a black-box anymore. Instead, we need to look inside $\mathsf{LHE}.\mathsf{Enc}_2$, to revisit the way this algorithm is encrypting the messages $\widehat{\mathbf{l}}_2 \cdot \Delta + \overline{\mathbf{e}}_{\mathsf{LEnc}}$. Denote by $\mathbf{a}$ the public parameters used inside LHE, and by s_2 and $\overline{\mathbf{e}} \leftarrow \overline{\mathcal{D}}^{w}_{\mathcal{R},\overline{s}}$

the secrets and noises sampled inside $\mathsf{LHE.Enc}_2$. Then, the ciphertext $\mathsf{LHE.ct}_2$ will have the form

$$\mathsf{LHE.ct}_2 = \underbrace{\mathbf{a} \cdot s_2 + \overline{\mathbf{e}}_{\mathsf{LEnc}} + \overline{\mathbf{e}}}_{\mathbf{c}} \; + \; \widehat{\mathbf{l}}_2 \cdot \Delta$$

However, regardless of the exact way this ciphertext is generated, we note that for the batch-select output to be correct it already suffices if $\mathsf{LHE.ct}_2 = \mathbf{a} \cdot s_2 + \widehat{\mathbf{l}}_2 \cdot \Delta + \mathbf{e}$ for *some* error $\mathbf{e}$ with

$$\|\mathbf{e}\|_\infty + B_{\mathsf{LEnc}} + (B_{\mathsf{LHE}} - \max \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}) < \Delta/2 \; . \tag{5.2}$$

This is because the result res computed by the batch-select decryptor Dec will be $\widehat{\mathbf{l}}_2 \cdot \Delta + \mathbf{e} +$ LEnc-noise + LHE-noise (note that we do not need to accomodate for the $\leq \max \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}$ noise resulting from $\overline{\mathbf{e}}$ as it is already included in $\mathsf{LHE.ct}_2$; hence LHE evaluation contributes at most another $B_{\mathsf{LHE}} - \max \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}$ noise).

Modifying batch-select. Our idea to reduce the size of $\mathsf{LHE.ct}_2$ is to reverse the order of sampling. Instead of calculating $\mathsf{LHE.ct}_2$ based on a random message $\widehat{\mathbf{l}}_2$, we sample some $\mathsf{LHE.ct}_2^*$ using the random oracle H (with image $\mathbb{Z}_q$), and reversely determine $\widehat{\mathbf{l}}_2$. We replace $\mathsf{Sel.Enc}_2$ by the following process (note that it does not take any input except pp, since we are constructing a *random* batch-select scheme):

(1) As before, sample noise $\overline{\mathbf{e}}_{\mathsf{LEnc}} \leftarrow \overline{\chi}_{\mathsf{LEnc}}^{w'}$ (used to hide the noise leakage resulting from LEnc evaluation). Further sample LHE secret $s_2 \leftarrow \mathcal{R}_q$ and noise $\overline{\mathbf{e}} \leftarrow \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}^{w'}$ (this was previously done inside $\mathsf{LHE.Enc}_2$). Use these values to compute $\mathbf{c} := \mathbf{a} \cdot s_2 + \overline{\mathbf{e}} + \overline{\mathbf{e}}_{\mathsf{LEnc}}$ (where $\mathbf{a}$ are the LHE public parameters). In addition, sample an RO-seed $\mathsf{seed} \leftarrow \{0,1\}^\lambda$.

(2) We now sample the ciphertext $\mathsf{LHE.ct}_2^*$ in the following way (where we abuse notation to denote by $\mathbf{z}[i,j] \in \mathbb{Z}_q$ the j-th coefficient of the i-th entry in a vector $\mathbf{z} \in \mathcal{R}_q^{w'}$ of ring elements): for all indices $i \in [w']$ and $j \in [n]$, find the smallest integer $d_{i,j} \in \mathbb{N}$ s.t.

$$\|H(\mathsf{seed}, i, j, d_{i,j}) - \mathbf{c}[i,j] \bmod \Delta\|_\infty + B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}} < \Delta/2 \; . \tag{5.3}$$

(We can view this step as a form of rejection sampling: we continue increasing $d_{i,j} = 0$ until it reaches a value for which the sampled value has sufficiently small error.)

Choose $\mathsf{LHE.ct}_2^*[i,j] := H(\mathsf{seed}, i, j, d_{i,j})$.

(3) Finally, we reversely compute messages $\widehat{\mathbf{l}}_2$ from $\mathsf{LHE.ct}_2^*$ by computing

$$\widehat{\mathbf{l}}_2 = \left\lfloor \frac{\mathsf{LHE.ct}_2^* - \mathbf{c}}{\Delta} \right\rceil \in \mathcal{R}_p^{w'}. \tag{5.4}$$

(4) Return ciphertext $\mathsf{ct}_2 := (\mathsf{seed}, (d_{i,j})_{i\in[w'],j\in[n]})$, labels $\widehat{\mathbf{l}}_2$, and state $\mathsf{st}_2 := s_2$.

We modify $\mathsf{Sel.Dec}$ in the following way:

(1) Before anything else, recover the ciphertext $\mathsf{LHE.ct}_2^*$ given only $\mathsf{ct}_2 = (\mathsf{seed}, (d_{i,j})_{i\in[w'],j\in[n]})$, by computing $\mathsf{LHE.ct}_2^*[i,j] := H(\mathsf{seed}, i, j, d_{i,j})$.

(2) Then, continue as before.

Correctness. We verify that correctness still holds: fix any ciphertext $\mathsf{LHE.ct}_2^*$ generated by the process above, and denote by $\mathbf{e} \in \mathcal{R}_q^{w'}$ the unique vector for which $\mathsf{LHE.ct}_2^* = \mathbf{a}\cdot s_2 + \widehat{\mathbf{l}}_2\cdot\Delta + \mathbf{e}$. Note that $\mathsf{LHE.ct}_2^* - \mathbf{c} - \widehat{\mathbf{l}}_2 \cdot \Delta = (\mathsf{LHE.ct}_2^* - \mathbf{c}) \bmod \Delta$, see Equation 5.4. Furthermore, by Equation 5.3 we have $\|(\mathsf{LHE.ct}_2^* - \mathbf{c}) \bmod \Delta\|_\infty < \Delta/2 - B_{\mathsf{LEnc}} - B_{\mathsf{LHE}} - \max \overline{\chi}_{\mathsf{LEnc}}$. By definition of $\mathbf{c} = \mathbf{a}\cdot s_2 + \overline{\mathbf{e}} + \overline{\mathbf{e}}_{\mathsf{LEnc}}$, we thus get

$$\begin{aligned}\|\mathbf{e}\|_\infty = \|\mathsf{LHE.ct}_2^* - \mathbf{c} - \widehat{\mathbf{l}}_2 \cdot \Delta + \overline{\mathbf{e}} + \overline{\mathbf{e}}_{\mathsf{LEnc}}\|_\infty &\le \|(\mathsf{LHE.ct}_2^* - \mathbf{c}) \bmod \Delta\|_\infty + \underbrace{\|\overline{\mathbf{e}}\|_\infty}_{\le \max \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}} + \underbrace{\|\overline{\mathbf{e}}_{\mathsf{LEnc}}\|_\infty}_{\le \max \overline{\chi}_{\mathsf{LEnc}}} \\ &\le \Delta/2 - B_{\mathsf{LEnc}} - (B_{\mathsf{LEnc}} - \max \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}) ,\end{aligned}$$

and therefore Equation 5.2 is fulfilled.

Efficiency. Revisiting Equation 5.3, we see that (for fixed i, j, $d_{i,j}$) rejection independently happens with probability $P := \frac{B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}}}{\Delta/2}$. We can use this together with Chernoff in order to bound the total number of "rejections" $R := \sum_{i\in[w'],j\in[n]} d_{i,j}$, i.e., the number of

times where Equation 5.3 does not hold during encryption, by $R \leq O(P \cdot w'n + \lambda^2)$ with probability $1 - \text{negl}(\lambda)$. (Whenever the expected number of rejections is $P \cdot w'n > \lambda$, then with overwhelming probability, the total number of rejections will be $\leq O(P \cdot w'n)$. Whenever the expected number of rejections is $P \cdot w'n \leq \lambda$, then with overwhelming probability, the total number of rejections will be $\leq O(\lambda^2)$.)

Assuming $P \leq \frac{1}{2}$, the size of the list $(d_{i,j})_{i \in [w'], j \in [n]}$ (and therefore also the size of ciphertext $|\mathsf{ct}_2|$) is roughly $2w'n + \text{poly}(\lambda)$. Alternatively, for small P it is more beneficial to express $(d_{i,j})_{i \in [w'], j \in [n]}$ within $R \cdot \log(w'n)$ bits (by listing all pairs (i, j), potentially including duplicates, for which a rejection took place). Hence, assuming $w \geq \Omega(n)$ and therefore $w'n \leq O(w)$, the new size of $\mathsf{Sel.ct}_2$ is now (with overwhelming probability)

$$|\mathsf{seed}| + |(d_{i,j})_{i \in [w'], j \in [n]}| \leq \lambda + R \cdot \log(w'n) \leq O(P \cdot w \log w) + \text{poly}(\lambda)$$

instead of $O(w \log q)$.

Furthermore, whenever $B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}} < \Delta/2^\lambda$, rejection happens only with probability $P \leq O(\frac{1}{2^\lambda})$. Thus, we can choose ct_2 to be *completely empty* (as all $d_{i,j}$ will be 0 with overwhelming probability, and we may insert the λ-size seed as a new component into $\mathsf{sk}_{\mathbf{y}}$) while still maintaining correctness with probability $1 - \frac{\text{poly}(\lambda)}{2^\lambda}$.

Security. Intuitively, security follows from the fact that rejection happens with *public* probability, and therefore each $d_{i,j}$ can be simulated easily. However, we additionally need to ensure that all $H(\mathsf{seed}, i, j, d')$ for $d' < d_{i,j}$ are programmed in such a way that Equation 5.3 actually fails. We prove the following security lemma.

Lemma 5.8 (Security of Random Batch-Select, Section 5.4.4)**.** *Assuming that the underlying* LHE *scheme is T-times 2^λ-simulation secure, and the underlying* LEnc *scheme is 2^λ-simulation secure with T-noise leakage, the Random Batch-Select scheme as described in Section 5.4.4 fulfills T-times 2^λ-simulation security.*

Proof. To argue security, we describe a simulator Sim^* that first runs the batch-select simulator Sim for our original construction (see proof of Lemma 5.7), and then additionally simulates

the rejection sampling results $d_{i,j}$ and programs the random oracle. It takes as input public parameters pp, messages $\widehat{\mathbf{l}}^{(t)}$, and selection vectors $\mathbf{y}^{(t)}$.

- First, run Sim to obtain

$$(\widetilde{\mathsf{ct}}_1, \{\widetilde{\mathsf{LHE.ct}_2^*}^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{y}}^{(t)}\}) \leftarrow \mathsf{Sim}(\mathsf{pp}, \{\widehat{\mathbf{l}}^{(t)}, \mathbf{y}^{(t)}\}_{[T]}) .$$

- Next, sample a random $\mathsf{seed}^{(t)} \leftarrow \{0,1\}^\lambda$, and simulate the rejection sampling results $d_{i,j}^{(t)}$ as follows. For $t \in [T]$, $i \in [w]$, $j \in [n]$:
  - Initially set the number of rejections $d_{i,j}^{(t)} = 0$.
  - Sample a noise $e \leftarrow [\Delta]$ and check if

$$|e| < \Delta/2 - (B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}}). \tag{5.5}$$

  - If no, sample a random value $l \leftarrow \mathbb{Z}_p$ and program

$$H(\mathsf{seed}^{(t)}, i, j, d_{i,j}^{(t)}) := \widetilde{\mathsf{LHE.ct}_2^*}^{(t)}[i,j] + l \cdot \Delta + e \bmod q.$$

    Increase $d_{i,j}^{(t)}$ by 1 and repeat.
  - If yes, program

$$H(\mathsf{seed}^{(t)}, i, j, d_{i,j}^{(t)}) := \widetilde{\mathsf{LHE.ct}_2^*}^{(t)}[i,j] + e \bmod q .$$

- Finally, output the simulation results $(\widetilde{\mathsf{ct}}_1, \{\widetilde{\mathsf{ct}}_2^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{y}}^{(t)}\}_{[T]})$, where we choose $\widetilde{\mathsf{ct}}_2^{(t)} := (\mathsf{seed}^{(t)}, (d_{i,j}^{(t)})_{i\in[w], j\in[n]})$.

We show a series of hybrids that transitions from Hyb_0 (the "real" distribution as defined in Definition 5.7) to Hyb_4 (the "simulated" distribution), focusing on how the rejection results are computed, and how the random oracle is programmed. We abuse the notation to also write Hyb_i as the output distribution of the corresponding experiment.

Hyb_0**:** We recall how the rejection results are generated in Hyb_0 (where we surpress the superscript (t) in the following for brevity). First, a vector $\mathbf{c} \in \mathcal{R}_q^w$ is computed as

$$\mathbf{c} = \mathbf{a} \cdot s_2 + \overline{\mathbf{e}} + \overline{\mathbf{e}}_{\mathsf{LEnc}}, \quad s_2 \leftarrow \mathcal{R}_q, \quad \overline{\mathbf{e}} \leftarrow \overline{\chi}^w, \quad \overline{\mathbf{e}}_{\mathsf{LEnc}} \leftarrow \overline{\chi}_{\mathsf{LEnc}}^w,$$

where $\mathbf{a}$ is part of the public parameters $\mathsf{Sel.pp}$, and a seed is sampled $\mathsf{seed} \leftarrow \{0,1\}^\lambda$. Finally, an intermediate vector $\mathsf{LHE.ct}_2^*$ is computed as follows. For $i \in [w]$, $j \in [n]$:

- Initially set the number of rejections $d_{i,j} = 0$.
- Check whether Equation 5.3 is fulfilled (this involves computing $H(\mathsf{seed}, i, j, d_{i,j})$).
- If no, increase $d_{i,j}$ and repeat.
- If yes, set $\mathsf{LHE.ct}_2^*[i, j] = H(\mathsf{seed}, i, j, d_{i,j})$.

Finally, the messages $\widehat{\mathbf{l}}_2$ are computed as $\widehat{\mathbf{l}}_2 = \lfloor (\mathsf{LHE.ct}_2^* - \mathbf{c})/\Delta \rceil \in \mathcal{R}_p^w$.

Hyb_1**:** Instead of checking Equation 5.3 for the RO-value $H(\mathsf{seed}, i, j, d_{i,j})$, we do so for a random value $r \leftarrow \mathbb{Z}_q$ and then program $H(\mathsf{seed}, i, j, d_{i,j}) := r$. Furthermore, we sample the value r in the following special (but still uniformly random) way:

$$r := \mathbf{c}[i, j] + l \cdot \Delta + e, \quad \text{where } l \leftarrow \mathbb{Z}_p \text{ and } e \leftarrow [\Delta].$$

We have that $\mathsf{Hyb}_1 \equiv \mathsf{Hyb}_0$.

Hyb_2**:** Note that whether a value r passes the check in Equation 5.3 only depends on the error e but not l (both defined in the previous step). Hence, instead of checking Equation 5.3, we will directly check Equation 5.5 on error e. Further, depending on whether the check is successful, we set r differently:

$$\begin{aligned} &\text{if no:} && r = \mathbf{c}[i, j] + \widehat{\mathbf{l}}_2[i, j] \cdot \Delta + l \cdot \Delta + e, \quad l \leftarrow \mathbb{Z}_p \\ &\text{if yes:} && r = \mathbf{c}[i, j] + \widehat{\mathbf{l}}_2[i, j] \cdot \Delta + e, \end{aligned}$$

where $\mathbf{l}_2 \leftarrow \mathcal{R}_q^w$ is a vector sampled only once (for any given t) and reused when a check fails and $d_{i,j}$ is increased.

We again have that $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_1$ (because in the no-case r looks uniformly random conditioned on not passing Equation 5.3, and in the yes-case r looks uniformly random conditioned on passing Equation 5.3).

Hyb_3**:** In the previous hybrid, we eliminate the generation of $\mathbf{c} \in \mathcal{R}_q^w$ as well as its underlying secret $s_2 \leftarrow \mathcal{R}_q$ and errors $\overline{\mathbf{e}}, \overline{\mathbf{e}}_{\mathsf{LEnc}}$. Furthermore, instead of computing r from the term $\mathbf{c}[i, j] + \widehat{\mathbf{l}}_2[i, j] \cdot \Delta$ as in the previous hybrid, we use the term $\mathsf{Sel.ct}_2[i, j]$. These values are generated globally using $(\mathsf{Sel.st}_2, \mathsf{Sel.ct}_2) \leftarrow \mathsf{Sel.Enc}_2(\widehat{\mathbf{l}}_2)$.

Examining the construction of Sel and the underlying construction of LHE, we have that $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_2$.

Hyb_4**:** To summarize, in Hyb_3, the rejection results and programmed entries of the random oracle are entirely derived from $\mathsf{Sel.ct}_2$. Instead of computing $\mathsf{Sel.ct}_2$, we can therefore apply simulation security of the original batch-select scheme. In particular, $\mathsf{Sel.ct}_1$, $\mathsf{Sel.ct}_2$, and $\mathsf{sk}_{\mathbf{y}}$ can all be alternatively generated from $\mathsf{Sel.Sim}(\mathsf{Sel.pp}, \{\widehat{\mathbf{l}}^{(t)}, \mathbf{y}^{(t)}\}_{[T]})$. By Lemma 5.7, we get $\mathsf{Hyb}_4 \approx_c^{2^\lambda} \mathsf{Hyb}_3$.

Observe that Hyb_4 proceeds identically as the simulated world with simulator Sim^*. By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_c^{2^\lambda} \mathsf{Hyb}_4$, which proves the security. □

For our construction above, we can e.g. get the following parameter setting when considering a message space $\mathcal{M} = \mathbb{Z}_p^\ell$ with $\log |\mathcal{M}| = \ell \cdot \log p \geq \lambda$.

Theorem 5.4 (*Random* Batch-Select)**.** *Consider the same assumptions and parameters as in Theorem 5.3, except that $q \geq n^{25}$ and $p = \Theta(q^\varepsilon)$ for some $\varepsilon > 0$. Then, there exists a $\sqrt{q}$-times 2^λ-simulation secure* random *batch-select scheme with message space $\mathbb{Z}_p^\ell$ of size $|\mathcal{M}| \geq 2^\lambda$ in the ROM with the same efficiencies as described in Theorem 5.3, except that the ciphertext ct_2 has only size $O(\frac{w \cdot \lambda}{q^{1/2-\varepsilon} \cdot \log q}) + \mathrm{poly}(\lambda)$.*

Furthermore, when $q > 2^{2(1+\varepsilon')\lambda}$ while $p = \Theta(q^{\varepsilon' \lambda})$ for some $\varepsilon' > 0$, then there is a batch-select scheme with the same properties, except that the size of ct_2 is 0.

Proof of Theorem 5.4. Under the given parameters (using $q \geq n^{25}$ instead of $q \geq n^{15}$) and $\log w \leq \lambda \leq q^{1/25}$, we get the slightly stronger error bound $B_{\mathsf{LEnc}} + B_{\mathsf{LHE}} + \max \overline{\chi}_{\mathsf{LEnc}} \leq O(\frac{q^{1/2}}{\log w})$ in a similar manner as in Lemmas 5.5 and 5.6. Therefore, due to $\Delta = q/p \geq q^{1-\epsilon}$, the individual rejection probability is $P \leq O(\frac{q^{1/2}}{\Delta \cdot \log w}) \leq O(\frac{1}{q^{1/2-\epsilon}})$, and the size of $|\mathsf{ct}_2|$ will be, with overwhelming probability, bounded by $O(\frac{w\lambda}{\log q} \cdot \frac{1}{q^{1/2-\epsilon}}) + \mathrm{poly}(\lambda)$.

When $q > 2^{2(1+\varepsilon')\lambda}$ with $\Delta \geq 2^{(2+\varepsilon')\lambda}$ and $p \leq 2^{\varepsilon' \lambda}$, then rejection only happens with probability $\leq O(1/2^\lambda)$, and we may omit ct_2 completely while correctness is satisfied with overwhelming probability. □

5.5 Construction of Building Block: LEnc

We now present a construction of LEnc over a ring $\mathcal{R}_q$. It is parameterized by

(1) a base g, used for decomposing a ring element into $m = \lceil \log_g q \rceil$ shorter ring elements of norm bounded by g,

(2) the parameter s for the internally used noise distribution $\chi = \mathcal{D}_{\mathcal{R},s}$, and

(3) the parameter $\overline{s}$ for the noise distribution $\overline{\chi}_{\mathsf{LEnc}} = \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}$ used to hide the error resulting from LEnc evaluation.

The correctness error will be bounded by

$$B_{\mathsf{LEnc}} \leq g \cdot m \cdot \gamma_{\mathcal{R}} \cdot \log_2 w \cdot s \cdot \sqrt{\lambda} \, ,$$

where $m = \lceil \log_g q \rceil$, and the construction fulfills simulation security with T-noise leakage w.r.t. noise distribution $\overline{\chi}_{\mathsf{LEnc}}$ under the assumption of $\mathsf{eILWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T,2m}(g)}$.

The construction will essentially be a simplified version of laconic encryption as presented in [DKL+23]. We first briefly recap the important details of the underlying ideas, and then give the full construction.

We will, w.l.o.g., assume that w is a power of 2. If this is not the case, the database may be padded by sufficiently many zeroes. We identify each entry in the database $\mathbf{a}$ with a leaf of the complete binary tree of height $\ell := \log_2 w$. Let us further index the leaves by ℓ-bit bitstrings $\mathsf{ind} \in \{0,1\}^\ell$, and all other nodes on the tree by a prefix $\mathsf{pre} \in \{0,1\}^{<\ell}$. The empty string ϵ denotes the root of the tree.

To compute the digest, we first assign $y_{\mathsf{ind}} := \mathbf{a}[i]$ to the leaves (where ind is the bitstring corresponding to index $i \in [w]$), and then assign a value y_{pre} to each node $\mathsf{pre} \in \{0,1\}^{<\ell}$, computed as a hash $f(\cdot,\cdot)$ of its two children as

$$y_{\mathsf{pre}} := f(y_{\mathsf{pre}\|0}, y_{\mathsf{pre}\|1}) := \mathbf{b}_0^T \cdot (-\mathbf{g}^{-1}(y_{\mathsf{pre}\|0})) + \mathbf{b}_1^T \cdot (-\mathbf{g}^{-1}(y_{\mathsf{pre}\|1})) \, .$$

The root value y_ϵ will represent the LEnc scheme's digest.

Given only a ring element $s \in \mathcal{R}_q$ and an index $\mathsf{ind} \in \{0,1\}^\ell$, $\mathsf{LEnc.Enc}$ needs to output $r_0 \in \mathcal{R}_q$ and a ciphertext ct, such that an evaluator can obtain the result $r_0 \cdot y_\epsilon - s \cdot y_{\mathsf{ind}}$ when given $\mathbf{y}$ and ct. We do so by choosing random $r_0, \ldots, r_{\ell-1} \leftarrow \mathcal{R}_q$ and selecting $\mathsf{ct} = (\mathbf{c}_0, \ldots, \mathbf{c}_{\ell-1})$, s.t. for each $i = 0, \ldots, \ell-1$,

$$\mathbf{c}_i \approx \begin{cases} r_i \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + r_{i+1} \cdot \begin{pmatrix} \mathbf{g}^T & \mathbf{0}^T \end{pmatrix} & \text{if } \mathsf{ind}_i = 0 \\ r_i \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + r_{i+1} \cdot \begin{pmatrix} \mathbf{0}^T & \mathbf{g}^T \end{pmatrix} & \text{if } \mathsf{ind}_i = 1 \end{cases},$$

where ind_i is the i-th bit of ind. Now, by the way in which the hash function f was constructed, we can evaluate

$$\mathbf{c}_i \cdot \begin{pmatrix} -\mathbf{g}^{-1}(y_{\mathsf{ind}_{:i}\|0}) \\ -\mathbf{g}^{-1}(y_{\mathsf{ind}_{:i}\|1}) \end{pmatrix} \approx r_i \cdot y_{\mathsf{ind}_{:i}} - r_{i+1} \cdot y_{\mathsf{ind}_{:i+1}} ,$$

where $\mathsf{ind}_{:i}$ is the length-i prefix of ind. Therefore, summing up the terms above for all $i = 0, \ldots, \ell-1$ yields $r_0 \cdot y_\epsilon - r_\ell \cdot y_{\mathsf{ind}}$, which is the desired result if the encryptor chooses $r_\ell := s$.

The following construction implements this idea in a *vectorized* fashion, meaning that the ciphertexts above are created for *all* database entries $\mathsf{ind} \in \{0,1\}^\ell$ simultaneously, and evaluation will produce the vector $\mathbf{r}_0 \cdot y_\epsilon - \mathbf{s} \odot \mathbf{y}$.

Construction 15 (Linear Laconic Encryption LEnc)**.** **$\underline{\mathsf{Setup}(1^\lambda, w) \to \mathsf{pp}}$:** Sample and output RingLWE public matrices $\mathsf{pp} := \mathbf{b}_0, \mathbf{b}_1 \leftarrow \mathcal{R}_q^m$.

$\underline{\mathsf{Enc}(\mathbf{s}) \to \mathbf{r}, \mathsf{ct}}$: Sample RingLWE secrets $\mathbf{r}_i \leftarrow \mathcal{R}_q^w$ for each layer $i = 0, \ldots, \ell-1$. For notational convenience, set $\mathbf{r}_\ell := \mathbf{s}$. Also sample *truncated* RingLWE noises $\mathbf{E}_i \leftarrow \overline{\mathcal{D}}_{\mathcal{R},s}^{w \times 2m}$.

Then, for each of the w rows, indexed by $\mathsf{ind} \in \{0,1\}^\ell$, compute for each $i = 0, \ldots, \ell-1$ the ciphertext

$$\mathbf{C}_i[\mathsf{ind}] := \mathbf{r}_i[\mathsf{ind}] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + \mathbf{r}_{i+1}[\mathsf{ind}] \cdot \begin{pmatrix} \overline{\mathsf{ind}_i} \cdot \mathbf{g}^T & \mathsf{ind}_i \cdot \mathbf{g}^T \end{pmatrix} + \mathbf{E}_i[\mathsf{ind}] .$$

We write the results as ℓ matrices $\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1} \in \mathcal{R}_q^{w \times 2m}$.

Return input keys $\mathbf{r} := \mathbf{r}_0$ and ciphertexts $\mathsf{ct} := (\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1})$.

$\underline{\mathsf{Digest}(\mathbf{a}) \to d_{\mathbf{a}}}$**:** Build an $\ell = \log w$-level hash tree as follows, and output the root y_ϵ.

Choose the hash values on the leaves of the tree as $y_{\mathsf{ind}} := \mathbf{a}[\mathsf{ind}]$. (where $\mathbf{a}[\mathsf{ind}]$ denotes the ind-th row of $\mathbf{a}$, where ind is interpreted as the binary integer corresponding to bitstring $\mathsf{ind} \in \{0,1\}^\ell$).

Compute the hash values on the remaining binary tree as

$$y_{\mathsf{pre}} := \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} \cdot \begin{pmatrix} -\mathbf{g}^{-1}(y_{\mathsf{pre}\|0}) \\ -\mathbf{g}^{-1}(y_{\mathsf{pre}\|1}) \end{pmatrix} \quad \forall \mathsf{pre} \in \{0,1\}^{<\ell} ,$$

and return the digest that is chosen to be the root $d_{\mathbf{a}} := y_\epsilon$.

$\underline{\mathsf{Eval}(\mathsf{ct}, \mathbf{a}) \to \boldsymbol{\delta}}$**:** Parse $\mathsf{ct} := (\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1})$.

As in Digest, compute y_{pre} for all $\mathsf{pre} \in \{0,1\}^{\leq \ell}$ from database $\mathbf{a}$.

Then, for each $\mathsf{ind} \in \{0,1\}^\ell$, compute the result

$$\boldsymbol{\delta}[\mathsf{ind}] := \sum_{i=0}^{\ell-1} \mathbf{C}_i[\mathsf{ind}] \cdot \begin{pmatrix} -\mathbf{g}^{-1}(y_{\mathsf{ind}_{:i}\|0}) \\ -\mathbf{g}^{-1}(y_{\mathsf{ind}_{:i}\|1}) \end{pmatrix} ,$$

and return the vector $\boldsymbol{\delta}$.

We now prove correctness and security of Construction 15, LEnc.

Correctness. In the following analysis of LEnc, whenever the public parameters $\mathbf{b}_0, \mathbf{b}_1$ and the database $\mathbf{a} \in \mathcal{R}_q^w$ are fixed, we define (for any prefix $\mathsf{pre} \in \{0,1\}^{<\ell}$) the hash tree recursively as

$$y_{\mathsf{pre}} = \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} \cdot \mathbf{z}_{\mathsf{pre}} \quad \text{where} \quad \mathbf{z}_{\mathsf{pre}} := \begin{pmatrix} -\mathbf{g}^{-1}(y_{\mathsf{pre}\|0}) \\ -\mathbf{g}^{-1}(y_{\mathsf{pre}\|1}) \end{pmatrix} ,$$

and the leaves y_{ind} for $\mathsf{ind} \in \{0,1\}^\ell$ are given by $\mathbf{a}$.

We first show correctness with error bound $B_{\mathsf{LEnc}} = g \cdot m \cdot \gamma_{\mathcal{R}} \cdot \log_2 w \cdot s \cdot \sqrt{\lambda}$. Note that for any $\mathsf{ind} \in \{0,1\}^\ell$, evaluation yields

$$\begin{aligned}
\boldsymbol{\delta}[\mathsf{ind}] &= \sum_{i=0}^{\ell-1} \mathbf{C}_i[\mathsf{ind}] \cdot \ \mathbf{z}_{\mathsf{ind}_{:i}} \\
&= \sum_{i=0}^{\ell-1} \Big(\mathbf{r}_i[\mathsf{ind}] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + \mathbf{r}_{i+1}[\mathsf{ind}] \cdot \begin{pmatrix} \overline{\mathsf{ind}_i} \mathbf{g}^T & \mathsf{ind}_i \mathbf{g}^T \end{pmatrix} + \mathbf{E}_i[\mathsf{ind}] \Big) \cdot \mathbf{z}_{\mathsf{ind}_{:i}} \\
&= \sum_{i=0}^{\ell-1} \mathbf{r}[\mathsf{ind}] \cdot y_{\mathsf{ind}_{:i}} - \mathbf{r}_{i+1}[\mathsf{ind}] \cdot y_{\mathsf{ind}_{:i+1}} + \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}_{\mathsf{ind}_{:i}} \\
&= \mathbf{r}_0[\mathsf{ind}] \cdot y_\epsilon - \mathbf{s}[\mathsf{ind}] \cdot \mathbf{a}[\mathsf{ind}] + \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}_{\mathsf{ind}_{:i}} \ .
\end{aligned}$$

Therefore, we just need to prove that $\| \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}_{\mathsf{ind}_{:i}} \|_\infty < B_{\mathsf{LEnc}}$. Note that $\|\mathbf{E}_i\|_\infty \leq \sqrt{\lambda} \cdot s$ and $\|\mathbf{z}_{\mathsf{ind}_{:i}}\|_\infty < g$. We get

$$\begin{aligned}
\| \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}_{\mathsf{ind}_{:i}} \|_\infty &\leq \sum_{i=0}^{\ell-1} \sum_{j \in [m]} \| \mathbf{E}_i[\mathsf{ind}, j] \cdot \mathbf{z}_{\mathsf{ind}_{:i}}[j] \|_\infty \\
&\leq \sum_{i=0}^{\ell-1} \sum_{j \in [m]} \gamma_{\mathcal{R}} \cdot \|\mathbf{E}_i\|_\infty \cdot \|\mathbf{z}_{\mathsf{ind}_{:i}}\|_\infty \\
&< \ell \cdot m \cdot \gamma_{\mathcal{R}} \cdot \sqrt{\lambda} \cdot s \cdot g = B_{\mathsf{LEnc}} \ .
\end{aligned}$$

Lemma 5.9 (Security of Construction 15)**.** *Assuming* $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T,2m}(g)}$*, Construction 15 (*$\mathsf{LEnc}$*) fulfills* 2^λ*-simulation security with* T*-noise leakage (Definition 5.9).*

Proof. The simulator Sim takes the public parameters and the databases $(\mathsf{pp}, \{\mathbf{a}^{(t)}\})$, and simulates ciphertext and noise $(\widetilde{\mathsf{ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]})$ as follows:

- First, sample all ciphertexts purely random

$$\widetilde{\mathbf{C}}_i \leftarrow \mathcal{R}_q^{w \times 2m} \quad \text{for } i = 0, \ldots, \ell - 1 \ ,$$

  and choose $\widetilde{\mathsf{ct}} := (\widetilde{\mathbf{C}}_0, \ldots, \widetilde{\mathbf{C}}_{\ell-1})$.

- Then, simulate the noise leakages as follows: sample

$$\mathbf{E}_i \leftarrow \chi^{w \times 2m} \quad \text{for } i = 0, \ldots, \ell - 1 \ ,$$

and for each $\mathsf{ind} \in \{0,1\}^\ell$ and $t \in [T]$, compute

$$\widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}:i} \ .$$

Return $(\widetilde{\mathsf{ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]})$.

In order to prove that the *real world* is indistinguishable from the *simulated world*, we define Hyb'_0 to output $((\mathbf{b}_0, \mathbf{b}_1), (\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1}), \{\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}\}_{[T]})$, where these values are computed as follows:

$$\boxed{\mathbf{b}_0, \mathbf{b}_1} \leftarrow \mathcal{R}_q^m \tag{5.6}$$

For $i = 0, \ldots, \ell - 1$ and $\mathsf{ind} \in \{0,1\}^\ell$:

$$\boxed{\mathbf{C}_i[\mathsf{ind}]} := \mathbf{r}_i[\mathsf{ind}] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + \mathbf{r}_{i+1}[\mathsf{ind}] \cdot \begin{pmatrix} \overline{\mathsf{ind}_i} \cdot \mathbf{g}^T & \mathsf{ind}_i \cdot \mathbf{g}^T \end{pmatrix} + \mathbf{E}_i[\mathsf{ind}] \tag{5.7}$$

$$\left| \begin{array}{l} \mathbf{r}_i[\mathsf{ind}] \leftarrow \mathcal{R}_q \text{ and } \mathbf{r}_\ell := \mathbf{s} \\ \mathbf{E}_i[\mathsf{ind}] \leftarrow \overline{\mathcal{D}}^{2m}_{\mathcal{R},s} \end{array} \right.$$

For $t \in [T]$ and $\mathsf{ind} \in \{0,1\}^\ell$:

$$\boxed{\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}[\mathsf{ind}]} := \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] + \overline{\mathbf{e}}^{(t)}[\mathsf{ind}] \ \left| \begin{array}{l} \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \displaystyle\sum_{0 \le i < \ell} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}:i} \\ \overline{\mathbf{e}}^{(t)}[\mathsf{ind}] \leftarrow \overline{\chi}_{\mathsf{LEnc}} \end{array} \right. \tag{5.8}$$

Note that Hyb'_0 is identical to the real world. To see this, consider the real noise leakage $\mathbf{e}^{(t)} = \boldsymbol{\delta}^{(t)} - (\mathbf{r} \cdot y^{(t)} - \mathbf{s} \odot \mathbf{a}^{(t)})$ with $\boldsymbol{\delta}^{(t)} \leftarrow \mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}^{(t)})$. Then, as in the correctness analysis above which calculates $\boldsymbol{\delta}^{(t)}[\mathsf{ind}]$, we get

$$\mathbf{e}^{(t)}[\mathsf{ind}] = \boldsymbol{\delta}^{(t)}[\mathsf{ind}] - (\mathbf{r}[\mathsf{ind}] \cdot y^{(t)} - \mathbf{s} \odot \mathbf{a}^{(t)}[\mathsf{ind}]) = \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}:i} = \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}]$$

for any $\mathsf{ind} \in \{0,1\}^\ell$ and $t \in [T]$.

As a first step, we define a hybrid Hyb_0 that is identical to the previous Hyb'_0, except that it samples the error matrices $\mathbf{E}_i[\mathsf{ind}]$ from the Gaussian $\chi^{2m} = \mathcal{D}^{2m}_{\mathcal{R},s}$ instead of the *truncated* Gaussian $\overline{\mathcal{D}}^{2m}_{\mathcal{R},s}$, and similarly $\overline{\mathbf{e}}^{(t)}[\mathsf{ind}]$ from the Gaussian $\mathcal{D}_{\mathcal{R},\overline{s}}$ instead of $\overline{\chi}_{\mathsf{LEnc}} = \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}$. By Lemma 5.2, we have $\mathsf{Hyb}_0 \approx^{2^\lambda}_{\mathrm{s}} \mathsf{Hyb}'_0$.

Next, for any $i^* \in [\ell]$, we define Hyb_{i^*} to be identical to Hyb_0, except that the ciphertexts of the first i^* layers are sampled as

$$\widetilde{\mathbf{C}}_i \leftarrow \mathcal{R}_q^{w \times 2m} \quad \text{for } i < i^* .$$

Note that Hyb_ℓ is identical to the simulated world.

It remains to show that $\mathsf{Hyb}_{i^*} \approx^{2^\lambda}_{\mathrm{c}} \mathsf{Hyb}_{i^*+1}$ for any $i^* = 0, \ldots, \ell - 1$. To do so, we further divide the hybrid Hyb_{i^*} into several sub-hybrids $\mathsf{Hyb}_{i^*,j}$ for $0 \leq j \leq w$, which are identical to Hyb_{i^*}, except that $\mathbf{C}_{i^*}[\mathsf{ind}]$ is sampled as

$$\widetilde{\mathbf{C}}_{i^*}[\mathsf{ind}] \leftarrow \mathcal{R}_q^{2m} \quad \text{for } j' < j ,$$

where ind is the bitstring representing j'. We can see that $\mathsf{Hyb}_{i^*} \equiv \mathsf{Hyb}_{i^*,0}$ and that $\mathsf{Hyb}_{i^*,w} \equiv \mathsf{Hyb}_{i^*+1}$, and therefore it just remains to show that $\mathsf{Hyb}_{i^*,j} \approx^{2^\lambda}_{\mathrm{c}} \mathsf{Hyb}_{i^*,j+1}$ for any $0 \leq j < w$.

We do so by a reduction to the $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T,2m}(g)}$ assumption. Specifically, assuming an adversary $\mathcal{D}$ that distinguishes between $\mathsf{Hyb}_{i^*,j}$ and $\mathsf{Hyb}_{i^*,j+1}$, we construct an adversary $\mathcal{A}$ for elLWE as follows (where the running time of $\mathcal{A}$ additively increases by $\mathrm{poly}(\lambda)$ when compared to that of $\mathcal{D}$). We use $\mathsf{ind} \in \{0,1\}^\ell$ as the bitstring representing the integer j.

- After receiving public parameters $\mathbf{b}_0, \mathbf{b}_1 \in \mathcal{R}_q^m$, choose the *leakage matrix* $\mathbf{Z} \in \mathcal{R}_q^{T \times 2m}$ in such a way that the t-th row is equal to the transpose of $\mathbf{z}^{(t)}_{\mathsf{ind}_{:i^*}}$.
- After receiving the LWE sample $\mathbf{y} \in \mathcal{R}_q^{2m}$ and leakage $\mathbf{l} \in \mathcal{R}_q^T$, run and output the result of $\mathcal{D}$ on a simulation of $\mathsf{Hyb}_{i^*,j}$, with two modifications:
  - The ind-th row of $\mathbf{C}_{i^*}$ is replaced by

$$\mathbf{C}_{i^*}[\mathsf{ind}] := \mathbf{y}^T + \mathbf{r}_{i^*+1}[\mathsf{ind}] \cdot \left(\overline{\mathsf{ind}_{i^*}} \cdot \mathbf{g}^T \quad \mathsf{ind}_{i^*} \cdot \mathbf{g}^T\right)$$

    instead of being computed from the key $\mathbf{r}_{i^*}[\mathsf{ind}]$.

– For every $t \in [T]$, the ind-th error is replaced by

$$\widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{\substack{0 \leq i < \ell \\ i \neq i^*}} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} \quad \text{and} \quad \widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}[\mathsf{ind}] := \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] + \mathbf{l}[t] \; .$$

The error-leakage RingLWE experiment with adversary $\mathcal{A}$ has the following distributions:

- Consider the experiment $\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T,2m}(g)}$. Here, $\mathbf{y}$ corresponds to a real LWE sample, i.e., $\mathbf{y}^T = \mathbf{r}_{i^*}[\mathsf{ind}] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + \mathbf{E}_{i^*}[\mathsf{ind}]$ with leakage $\mathbf{l}[t] = \mathbf{E}_{i^*}[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i^*}} + \overline{e}^{(t)}[\mathsf{ind}]$ (for some fresh key $\mathbf{r}_{i^*}[\mathsf{ind}] \leftarrow \mathcal{R}_q$ and errors $\mathbf{E}_{i^*}[\mathsf{ind}] \leftarrow \chi^{2m}$ and $\overline{e}^{(t)}[\mathsf{ind}] \leftarrow \mathcal{D}_{\mathcal{R},\overline{s}}$).

  Therefore, $\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T,2m}(g)}$ is identically distributed as the output of $\mathcal{D}$ when given input generated from distribution $\mathsf{Hyb}_{i^*,j}$.
- Consider the experiment $\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T,2m}(g)}$. Here, $\mathbf{y} \leftarrow \mathcal{R}_q^{2m}$ is uniformly random and the leakage is equal to $\mathbf{l}[t] = \mathbf{E}_{i^*}[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} + \overline{e}^{(t)}[\mathsf{ind}]$ (for fresh errors $\mathbf{E}_{i^*}[\mathsf{ind}] \leftarrow \chi^{2m}$ and $\overline{e}^{(t)}[\mathsf{ind}] \leftarrow \mathcal{D}_{\mathcal{R},\overline{s}}$).

  Note that $\mathbf{C}_{i^*}[\mathsf{ind}]$ will be uniformly random (due to the randomness of $\mathbf{y}$). Therefore, $\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T,2m}(g)}$ is identically distributed as the output of $\mathcal{D}$ when given input generated from distribution $\mathsf{Hyb}_{i^*,j+1}$.

Therefore, the distinguisher $\mathcal{D}$ has exactly the same advantage as the adversary $\mathcal{A}$. By the $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{T,2m}(g)}$ assumption, this implies that $\mathsf{Hyb}_{i^*,j} \approx^{2^\lambda}_{\mathrm{c}} \mathsf{Hyb}_{i^*,j+1}$. Our hybrid argument thus shows $\mathsf{Hyb}'_0 \approx^{2^\lambda}_{\mathrm{c}} \mathsf{Hyb}_\ell$, which concludes the security proof. □

5.5.1 Construction of Weak Linear Laconic Encryption

We now present a slight modification of Construction 15, to give a *weak* LEnc over a ring $\mathcal{R}_q$ (see Definition 5.10). It utilizes the same parameters as Construction 15 and has the same correctness error. However, simulation security with T-noise leakage w.r.t. noise distribution $\overline{\chi}_{\mathsf{LEnc}}$ requires the assumption of $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{Tw,2m}(g)}$.

Construction The main motivation is the observation that in Construction 15, for *every* single $\mathsf{ind} \in \{0,1\}^\ell$, we created *independent* ciphertexts allowing us to obtain $\mathbf{r}[\mathsf{ind}] \cdot y_\epsilon - \mathbf{s}[\mathsf{ind}] \cdot \mathbf{a}[\mathsf{ind}]$.

Hence, a natural question to ask is whether there is potential for optimization by exploiting the repeated usage of the same functionality.

Indeed, we are able to construct an optimized *weak* LEnc, i.e., when the output keys $\mathbf{s}$ provided to the encryption algorithm $\mathsf{LEnc.Enc}(\mathbf{s})$ are guaranteed to consist of w *identical* ring elements ($\mathbf{s} = s \cdot \mathbf{1}_w$ for some $s \in \mathcal{R}_q$). The size of the ciphertext will be reduced by a factor of $\frac{\ell}{2} = \frac{\log w}{2}$. However, it also requires a slightly stronger assumption of elLWE in which the leakage matrix has dimensions $Tw \times 2m$ instead of $T \times 2m$.

To get some intuition of where the improvement is coming from, consider the ciphertext $\mathsf{ct} = (\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1})$ in Construction 15. Taking a closer look at $\mathbf{C}_{\ell-1}$, we can see that it will be equal to

$$\mathbf{C}_{\ell-1} = \begin{pmatrix} \mathbf{r}_{\ell-1}[1] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + \mathbf{s}[1] \cdot \begin{pmatrix} \mathbf{g}^T & \mathbf{0}^T \end{pmatrix} \\ \mathbf{r}_{\ell-1}[2] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + \mathbf{s}[2] \cdot \begin{pmatrix} \mathbf{0}^T & \mathbf{g}^T \end{pmatrix} \\ \vdots \\ \mathbf{r}_{\ell-1}[w-1] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + \mathbf{s}[w-1] \cdot \begin{pmatrix} \mathbf{g}^T & \mathbf{0}^T \end{pmatrix} \\ \mathbf{r}_{\ell-1}[w] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + \mathbf{s}[w] \cdot \begin{pmatrix} \mathbf{0}^T & \mathbf{g}^T \end{pmatrix} \end{pmatrix} + \mathbf{E}_{\ell-1} \ .$$

Now, because $\mathbf{s}$ only consists of identical elements s, the "ciphertexts" corresponding to rows $1, 3, 5, \ldots$ are all used to hide the same vector $s \cdot \begin{pmatrix} \mathbf{g}^T & \mathbf{0}^T \end{pmatrix}$. Additionally, the ciphertexts corresponding to rows $2, 4, 6, \ldots$ are all used to hide the same vector $s \cdot \begin{pmatrix} \mathbf{0}^T & \mathbf{1}^T \end{pmatrix}$.

To exploit this pattern, we will sample $\mathbf{r}_{\ell-1}$ in such a way that $\mathbf{r}_{\ell-1}[1] = \mathbf{r}_{\ell-1}[3] = \ldots$ and $\mathbf{r}_{\ell-1}[2] = \mathbf{r}_{\ell-1}[4] = \ldots$, which allows us to use a "compressed" $\mathbf{C}_{\ell-1}$ of the form

$$\widehat{\mathbf{C}}_{\ell-1} = \begin{pmatrix} \widehat{\mathbf{r}}_{\ell-1}[1] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + s \cdot \begin{pmatrix} \mathbf{g}^T & \mathbf{0}^T \end{pmatrix} \\ \widehat{\mathbf{r}}_{\ell-1}[2] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + s \cdot \begin{pmatrix} \mathbf{0}^T & \mathbf{g}^T \end{pmatrix} \end{pmatrix} + \widehat{\mathbf{E}}_{\ell-1} \ ,$$

which consists only of two rows. We get the "decompressed" version by computing

$$\mathbf{C}_{\ell-1} = \mathbf{1}_{2^{\ell-1}} \otimes \widehat{\mathbf{C}}_{\ell-1} \ ,$$

and they correspond to keys $\mathbf{r}_{\ell-1} = \mathbf{1}_{2^{\ell-1}} \otimes \widehat{\mathbf{r}}_{\ell-1}$.

Note that we reduced the number of rows in $\widehat{\mathbf{C}}_{\ell-1}$ to just 2. This, in turn, can be used to see that in $\mathbf{C}_{\ell-2}$, only 4 different patterns need to be encrypted, and so on. In general, through this compression trick and reusing the same keys $\widehat{\mathbf{r}}_i \in \mathcal{R}_q^{\frac{w}{2^i}\times 2m}$ for 2^i times, $\widehat{\mathbf{C}}_i$ will consist of $\frac{w}{2^i}$ rows. The only potential issue with this approach is that the same noise, e.g. $\widehat{\mathbf{E}}_{\ell-1}$, is leaked w times, and therefore we will slightly modify the security proof and depend on a variant of elLWE that allows more leakage than before.

The resulting construction differs from that in Section 5.5 only in the encryption and evaluation algorithms:

Construction 16 (Weak Linear Laconic Encryption)**.**

$\mathsf{Enc}(\mathsf{s} = s \cdot \mathbf{1}_w) \to \mathbf{r}, \mathsf{ct}$**:** Sample RingLWE secrets $\widehat{\mathbf{r}}_i \leftarrow \mathcal{R}_q^{\frac{w}{2^i}}$ for each layer $i = 0, \ldots, \ell - 1$. For notational convenience, set $\widehat{\mathbf{r}}_\ell := s$. Also sample *truncated* RingLWE noises $\widehat{\mathbf{E}}_i \leftarrow \chi^{\frac{w}{2^i}\times 2m}$.

Then, for each of the $\frac{w}{2^i}$ rows, indexed by $\mathsf{suf} \in \{0,1\}^{\ell-i}$, compute for each $i = 0, \ldots, \ell-1$ the compressed ciphertext

$$\widehat{\mathbf{C}}_i[\mathsf{suf}] := \widehat{\mathbf{r}}_i[\mathsf{suf}] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + \widehat{\mathbf{r}}_{i+1}[\mathsf{suf}_{1:}] \cdot \begin{pmatrix} \overline{\mathsf{suf}_0} \cdot \mathbf{g}^T & \mathsf{suf}_0 \cdot \mathbf{g}^T \end{pmatrix} + \widehat{\mathbf{E}}_i[\mathsf{suf}] \,.$$

We write the results as ℓ matrices $\widehat{\mathbf{C}}_i \in \mathcal{R}_q^{\frac{w}{2^i}\times 2m}$.

Return input keys $\mathbf{r} := \mathbf{r}_0$ and ciphertexts $\mathsf{ct} := (\widehat{\mathbf{C}}_0, \ldots, \widehat{\mathbf{C}}_{\ell-1})$.

$\mathsf{Eval}(\mathsf{ct}, \mathbf{a}) \to \boldsymbol{\delta}$**:** Parse $\mathsf{ct} := (\widehat{\mathbf{C}}_0, \ldots, \widehat{\mathbf{C}}_{\ell-1})$. Decompress these ciphertexts in the following way:

$$\mathbf{C}_i := \mathbf{1}_{2^i} \otimes \widehat{\mathbf{C}}_i \in \mathcal{R}_q^{w\times 2m} \,.$$

As in Digest, compute y_{pre} for all $\mathsf{pre} \in \{0,1\}^{\leq \ell}$ from database $\mathbf{a}$.

Then, for each $\mathsf{ind} \in \{0,1\}^\ell$, compute the result

$$\boldsymbol{\delta}[\mathsf{ind}] := \sum_{i=0}^{\ell-1} \mathbf{C}_i[\mathsf{ind}] \cdot \begin{pmatrix} -\mathbf{g}^{-1}(y_{\mathsf{ind}_{:i}\|0}) \\ -\mathbf{g}^{-1}(y_{\mathsf{ind}_{:i}\|1}) \end{pmatrix} \,,$$

and return the vector $\boldsymbol{\delta}$.

Correctness. Correctness of Construction 16 follows from the fact that the decompressed ciphertexts are equal to "normal" ciphertexts as created in Construction 15:

$$\begin{aligned}\mathbf{C}_i[\mathsf{ind}] &= \widehat{\mathbf{C}}_i[\mathsf{ind}_{i:}] \\ &= \widehat{\mathbf{r}}_i[\mathsf{ind}_{i:}] \cdot \begin{pmatrix}\mathbf{b}_0^T & \mathbf{b}_1^T\end{pmatrix} + \widehat{\mathbf{r}}_{i+1}[\mathsf{ind}_{i+1:}] \cdot \begin{pmatrix}\overline{\mathsf{ind}_i} \cdot \mathbf{g}^T & \mathsf{ind}_i \cdot \mathbf{g}^T\end{pmatrix} + \widehat{\mathbf{E}}_i[\mathsf{ind}_{i:}] \\ &= \mathbf{r}_i[\mathsf{ind}] \cdot \begin{pmatrix}\mathbf{b}_0^T & \mathbf{b}_1^T\end{pmatrix} + \mathbf{r}_{i+1}[\mathsf{ind}] \cdot \begin{pmatrix}\overline{\mathsf{ind}_i} \cdot \mathbf{g}^T & \mathsf{ind}_i \cdot \mathbf{g}^T\end{pmatrix} + \mathbf{E}_i[\mathsf{ind}] ,\end{aligned}$$

where

$$\mathbf{r}_i := \mathbf{1}_{2^i} \otimes \widehat{\mathbf{r}}_i \quad \text{and} \quad \mathbf{E}_i := \mathbf{1}_{2^i} \otimes \widehat{\mathbf{E}}_i$$

denote the "decompressed" keys and noise. Now we can apply correctness of the original Construction 15 (which does make any assumptions on entries of $\mathbf{r}_i$ or $\mathbf{E}_i$ being independent of each other).

Lemma 5.10 (Security of Construction 16)**.** *Assuming* $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{Tw,2m}(g)}$*, Construction 15 fulfills* 2^λ*-simulation security with* T*-noise leakage (Definition 5.9).*

Proof. The simulator Sim takes the public parameters and the databases $(\mathsf{pp}, \{\mathbf{a}^{(t)}\})$, and simulates ciphertext and noise $(\widetilde{\mathsf{ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]})$ as follows:

- First, sample all compressed ciphertexts purely random

$$\widetilde{\widehat{\mathbf{C}}}_i \leftarrow \mathcal{R}_q^{\frac{w}{2^i} \times 2m} \quad \text{for } i = 0, \ldots, \ell - 1 ,$$

  and choose $\widetilde{\mathsf{ct}} := (\widetilde{\widehat{\mathbf{C}}}_0, \ldots, \widetilde{\widehat{\mathbf{C}}}_{\ell-1})$.
- Then, simulate the noise leakages as follows: sample

$$\widehat{\mathbf{E}}_i \leftarrow \chi^{\frac{w}{2^i} \times 2m} \quad \text{for } i = 0, \ldots, \ell - 1 ,$$

  "decompress" these noise matrices by choosing

$$\mathbf{E}_i := \mathbf{1}_{2^i} \otimes \widehat{\mathbf{E}}_i \quad \text{for } i = 0, \ldots, \ell - 1 ,$$

and for each $\mathsf{ind} \in \{0,1\}^\ell$ and $t \in [T]$, compute

$$\widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} .$$

Return $(\widetilde{\mathsf{ct}}, \{\widetilde{\mathbf{e}}^{(t)}\}_{[T]})$.

In order to prove that the *real world* is indistinguishable from the *simulated world*, we define Hyb'_0 to output $((\mathbf{b}_0, \mathbf{b}_1), (\widehat{\mathbf{C}}_0, \ldots, \widehat{\mathbf{C}}_{\ell-1}), \{\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}\}_{[T]})$, where these values are computed as follows:

$$\boxed{\mathbf{b}_0, \mathbf{b}_1} \leftarrow \mathcal{R}_q^m$$

For $i = 0, \ldots, \ell-1$ and $\mathsf{suf} \in \{0,1\}^{\ell-i}$:

$$\boxed{\widehat{\mathbf{C}}_i[\mathsf{suf}]} := \widehat{\mathbf{r}}_i[\mathsf{suf}] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + \widehat{\mathbf{r}}_{i+1}[\mathsf{suf}_{1:}] \cdot \begin{pmatrix} \overline{\mathsf{suf}_0} \cdot \mathbf{g}^T & \mathsf{suf}_0 \cdot \mathbf{g}^T \end{pmatrix} + \widehat{\mathbf{E}}_i[\mathsf{suf}]$$

$$\mathbf{r}_i := \mathbf{1}_{2^i} \otimes \widehat{\mathbf{r}}_i \quad \text{for } i = i^*, \ldots, \ell \text{ with } \widehat{\mathbf{r}}_i \leftarrow \mathcal{R}_q^{\frac{w}{2^i}} \text{ and } \widehat{\mathbf{r}}_\ell := s$$

$$\mathbf{E}_i := \mathbf{1}_{2^i} \otimes \widehat{\mathbf{E}}_i \quad \text{for } i = 0, \ldots, \ell-1 \text{ with } \widehat{\mathbf{E}}_i \leftarrow \overline{\mathcal{D}}_{\mathcal{R},s}^{\frac{w}{2^i} \times 2m}$$

For $t \in [T]$ and $\mathsf{ind} \in \{0,1\}^\ell$:

$$\boxed{\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}[\mathsf{ind}]} := \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] + \overline{\mathbf{e}}^{(t)}[\mathsf{ind}] \quad \Bigg| \quad \begin{aligned} \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] &:= \sum_{0 \le i < \ell} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} \\ \overline{\mathbf{e}}^{(t)}[\mathsf{ind}] &\leftarrow \overline{\chi}_{\mathsf{LEnc}} \end{aligned}$$

Note that Hyb'_0 is identical to the real world. To see this, consider the real noise leakage $\mathbf{e}^{(t)} = \boldsymbol{\delta}^{(t)} - (\mathbf{r} \cdot y^{(t)} - \mathbf{s} \odot \mathbf{a}^{(t)})$ with $\boldsymbol{\delta}^{(t)} \leftarrow \mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}^{(t)})$. Then, as in the correctness analysis above which calculates $\boldsymbol{\delta}^{(t)}[\mathsf{ind}]$, we get

$$\mathbf{e}^{(t)}[\mathsf{ind}] = \boldsymbol{\delta}^{(t)}[\mathsf{ind}] - (\mathbf{r}[\mathsf{ind}] \cdot y^{(t)} - \mathbf{s} \odot \mathbf{a}^{(t)}[\mathsf{ind}]) = \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} = \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}]$$

for any $\mathsf{ind} \in \{0,1\}^\ell$ and $t \in [T]$.

As a first step, we define a hybrid Hyb_0 that is identical to the previous Hyb'_0, except that it samples the error matrices $\widehat{\mathbf{E}}_i[\mathsf{suf}]$ from the Gaussian $\chi^{2m} = \mathcal{D}^{2m}_{\mathcal{R},s}$ instead of the *truncated*

Gaussian $\overline{\mathcal{D}}_{\mathcal{R},s}^{2m}$, and similarly $\overline{\mathbf{e}}^{(t)}[\mathsf{ind}]$ from the Gaussian $\mathcal{D}_{\mathcal{R},\bar{s}}$ instead of $\overline{\chi}_{\mathsf{LEnc}} = \overline{\mathcal{D}}_{\mathcal{R},\bar{s}}$. By Lemma 5.2, we have $\mathsf{Hyb}_0 \approx_{\mathrm{s}}^{2^\lambda} \mathsf{Hyb}_0'$.

Next, for any $i^* \in [\ell]$, we define Hyb_{i^*} to be identical to Hyb_0, except that the ciphertexts of the first i^* layers are sampled as

$$\widetilde{\widehat{\mathbf{C}}}_i \leftarrow \mathcal{R}_q^{\frac{w}{2^i} \times 2m} \quad \text{for } i < i^* .$$

Note that Hyb_ℓ is identical to the simulated world.

It remains to show that $\mathsf{Hyb}_{i^*} \approx_{\mathrm{c}} \mathsf{Hyb}_{i^*+1}$ for any $i^* = 0, \ldots, \ell-1$. To do so, we further divide the hybrid Hyb_{i^*} into several sub-hybrids $\mathsf{Hyb}_{i^*,j}$ for $0 \le j \le \frac{w}{2^{i^*}}$, which are identical to Hyb_{i^*}, except that $\widehat{\mathbf{C}}_{i^*}[\mathsf{ind}]$ is sampled as

$$\widetilde{\widehat{\mathbf{C}}}_{i^*}[\mathsf{suf}] \leftarrow \mathcal{R}_q^{2m} \quad \text{for } j' < j ,$$

where suf is the bitstring representing j'. We can see that $\mathsf{Hyb}_{i^*} \equiv \mathsf{Hyb}_{i^*,0}$ and that $\mathsf{Hyb}_{i^*,w} \equiv \mathsf{Hyb}_{i^*+1}$, and therefore it just remains to show that $\mathsf{Hyb}_{i^*,j} \approx_{\mathrm{c}} \mathsf{Hyb}_{i^*,j+1}$ for any $0 \le j < w$.

We do so by a reduction to the $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T2^{i^*},2m}(g)}$ assumption. Specifically, assuming an adversary $\mathcal{D}$ that distinguishes between $\mathsf{Hyb}_{i^*,j}$ and $\mathsf{Hyb}_{i^*,j+1}$, we construct an adversary $\mathcal{A}$ for elLWE as follows (where the running time of $\mathcal{A}$ additively increases by $\mathrm{poly}(\lambda)$ when compared to that of $\mathcal{D}$). We use $\mathsf{suf} \in \{0,1\}^{\ell-i}$ as the bitstring representing the integer j.

- After receiving public parameters $\mathbf{b}_0, \mathbf{b}_1 \in \mathcal{R}_q^m$, choose the *leakage matrix* $\mathbf{Z} \in \mathcal{R}_q^{T2^{i^*} \times 2m}$, with rows indexed by (t, pre), for $t \in [T]$ and $\mathsf{pre} \in \{0,1\}^{i^*}$, in such a way that the (t, pre)-th row is equal to the transpose of $\mathbf{z}_{\mathsf{pre}}^{(t)}$.
- After receiving the LWE sample $\mathbf{y} \in \mathcal{R}_q^{2m}$ and leakage $\mathbf{l} \in \mathcal{R}_q^{T2^{i^*}}$, run and output the result of $\mathcal{D}$ on a simulation of $\mathsf{Hyb}_{i^*,j}$, with two modifications:
  - The suf-th row of $\widehat{\mathbf{C}}_{i^*}$ is replaced by

$$\widehat{\mathbf{C}}_{i^*}[\mathsf{suf}] := \mathbf{y}^T + \widehat{\mathbf{r}}_{i^*+1}[\mathsf{suf}_{1:}] \cdot \left(\overline{\mathsf{suf}_0} \cdot \mathbf{g}^T \quad \mathsf{suf}_0 \cdot \mathbf{g}^T \right)$$

    instead of being computed from the key $\widehat{\mathbf{r}}_{i^*}[\mathsf{suf}]$.

– For every $t \in [T]$, and every $\mathsf{pre} \in \{0,1\}^{i^*}$, the $\mathsf{ind} = \mathsf{pre}\|\mathsf{suf}$-th error is replaced by

$$\widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{\substack{0 \le i < \ell \\ i \neq i^*}} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} \quad \text{and} \quad \widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}[\mathsf{ind}] := \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] + \mathbf{l}[t, \mathsf{pre}] \; .$$

The error-leakage RingLWE experiment with adversary $\mathcal{A}$ has the following distributions:

- Consider the experiment $\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T2^{i^*},2m}(g)}$. Here, $\mathbf{y}$ corresponds to a real LWE sample, $\mathbf{y}^T = \widehat{\mathbf{r}}_{i^*}[\mathsf{suf}] \cdot \begin{pmatrix} \mathbf{b}_0^T & \mathbf{b}_1^T \end{pmatrix} + \widehat{\mathbf{E}}_{i^*}[\mathsf{suf}]$ with leakage $\mathbf{l}[t, \mathsf{pre}] = \widehat{\mathbf{E}}_{i^*}[\mathsf{suf}] \cdot \mathbf{z}^{(t)}_{\mathsf{pre}} + \bar{e}^{(t)}[\mathsf{pre}\|\mathsf{suf}]$ (for some fresh key $\widehat{\mathbf{r}}_{i^*}[\mathsf{suf}] \leftarrow \mathcal{R}_q$ and errors $\widehat{\mathbf{E}}_{i^*}[\mathsf{suf}] \leftarrow \chi^{2m}$ and $\bar{e}^{(t)}[\mathsf{pre}\|\mathsf{suf}] \leftarrow \mathcal{D}_{\mathcal{R},\bar{s}}$).

  Therefore, $\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T2^{i^*},2m}(g)}$ is identically distributed as the output of $\mathcal{D}$ when given input generated from distribution $\mathsf{Hyb}_{i^*,j}$.
- Consider the experiment $\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T2^{i^*},2m}(g)}$. Here, $\mathbf{y} \leftarrow \mathcal{R}_q^{2m}$ is uniformly random and the leakage is equal to $\mathbf{l}[t, \mathsf{pre}] = \widehat{\mathbf{E}}_{i^*}[\mathsf{suf}] \cdot \mathbf{z}^{(t)}_{\mathsf{pre}} + \bar{e}^{(t)}[\mathsf{pre}\|\mathsf{suf}]$ (for fresh errors $\mathbf{E}_{i^*}[\mathsf{suf}] \leftarrow \chi^{2m}$ and $\bar{e}^{(t)}[\mathsf{pre}\|\mathsf{suf}] \leftarrow \mathcal{D}_{\mathcal{R},\bar{s}}$).

  Note that $\mathbf{C}_{i^*}[\mathsf{ind}]$ will be uniformly random (due to the randomness of $\mathbf{y}$). Therefore, $\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T2^{i^*},2m}(g)}$ is identically distributed as the output of $\mathcal{D}$ when given input generated from distribution $\mathsf{Hyb}_{i^*,j+1}$.

Therefore, the distinguisher $\mathcal{D}$ has exactly the same advantage as the adversary $\mathcal{A}$. By the $\mathsf{elLWE}_{\mathcal{R},q,\chi,\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{Tw,2m}(g)}$ assumption, this implies that $\mathsf{Hyb}_{i^*,j} \approx^{2^\lambda}_{\mathrm{c}} \mathsf{Hyb}_{i^*,j+1}$. Our hybrid argument thus shows $\mathsf{Hyb}'_0 \approx^{2^\lambda}_{\mathrm{c}} \mathsf{Hyb}_\ell$, which concludes the security proof. □

5.5.2 LEnc Parameter Setting

We now prove the claimed efficiencies for the parameter setting described in Lemma 5.5.

of Lemma 5.5. Under assumption $\mathsf{LWE}_{\mathcal{R},O(1),q,\mathcal{D}_{\mathcal{R},q^{0.1}}}$, we may conclude (using Theorem 5.2 and Lemma 5.3) that also the error-leakage version $\mathsf{elLWE}_{\mathcal{R},q,\mathcal{D}_{\mathcal{R},s},\mathcal{D}_{\mathcal{R},\bar{s}},\mathcal{L}_{T,2m}(g)}$ holds, with parameters $s = 3q^{0.1}$ and $\bar{s} = 2s \cdot g \cdot n\sqrt{2Tm}$.

To obtain the claimed parameters, we choose the gadget-base $g = \lceil q^{0.05} \rceil$ (as used inside of our LEnc construction) and hence $m \leq 20$. Thus, we get

$$\begin{aligned} \max \overline{\chi}_{\mathsf{LEnc}} = \sqrt{\lambda} \cdot \overline{s} &= q^{0.1} \cdot \underbrace{g}_{\leq 2q^{0.05}} \cdot \underbrace{n}_{\leq q^{1/15}} \cdot \underbrace{\sqrt{T}}_{\leq q^{1/4}} \cdot \underbrace{\sqrt{m}}_{\leq 20} \cdot \underbrace{\sqrt{\lambda}}_{\leq q^{1/30}} \cdot 6\sqrt{2} \\ &\leq 1000\sqrt{q} \ , \end{aligned}$$

and similarly

$$\begin{aligned} B_{\mathsf{LEnc}} &= \underbrace{g}_{2q^{0.05}} \cdot \underbrace{m}_{\leq 20} \cdot \underbrace{\gamma_{\mathcal{R}}}_{\leq q^{1/15}} \cdot \underbrace{\log_2 w}_{\leq \lambda \leq q^{1/15}} \cdot q^{0.1} \cdot \underbrace{\sqrt{\lambda}}_{\leq q^{1/30}} \cdot 2 \\ &\leq 80q^{0.35} \leq \sqrt{q} \ . \end{aligned}$$

Similarly, we also get a *weak* LEnc scheme with $T = \frac{\sqrt{q}}{w}$-noise leakage under the same parameters, because the $\mathsf{elLWE}_{\mathcal{R},q,\mathcal{D}_{\mathcal{R},s},\mathcal{D}_{\mathcal{R},\overline{s}},\mathcal{L}_{Tw,2m}(g)}$ assumption holds under $\mathsf{LWE}_{\mathcal{R},O(1),q,\mathcal{D}_{\mathcal{R},q^{0.1}}}$, with the same parameters as above: $s = 3q^{0.1}$ and $\overline{s} = 2s \cdot g \cdot n\sqrt{2m} \cdot q^{1/4}$. $\square$

5.6 Construction of Building Block: LHE

In this section, we give a construction of LHE over a ring $\mathcal{R}_q$ (see Definition 5.11). It is parameterized by

(1) a base g, used for decomposing a ring element into $m = \lceil \log_g q \rceil$ shorter ring elements of norm bounded by g,

(2) the parameter s for the noise distribution $\chi = \mathcal{D}_{\mathcal{R},s}$, and

(3) the parameter $\overline{s}$ for the noise distribution $\overline{\chi} = \mathcal{D}_{\mathcal{R},\overline{s}}$.

The correctness error will be bounded by

$$B_{\mathsf{LHE}} \leq (g \cdot m \cdot \gamma_{\mathcal{R}} \cdot s + \overline{s}) \cdot \sqrt{\lambda} \ ,$$

where $m = \lceil \log_g q \rceil$, and the construction fulfills T-times simulation security under the assumption of $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}^{\times w}_{T,1}(g)}$.

Construction 17 (LHE over rings LHE)**.** $\underline{\mathsf{Setup}(1^\lambda, 1^w) \to \mathsf{pp}}$**:** Output a public random vector $\mathsf{pp} = \mathbf{a} \leftarrow \mathcal{R}_q^w$ of ring elements.

$\underline{\mathsf{Enc}_1(\mathbf{m}_1) \to \mathsf{ct}_1, \mathsf{st}_1}$**:** Sample RingLWE secrets $\mathbf{s}_1 \leftarrow \mathcal{R}_q^m$ and *truncated* noises $\mathbf{E} \leftarrow \overline{\mathcal{D}}_{\mathcal{R},s}^{w \times m}$. Output a ciphertext

$$\mathsf{ct}_1 := \mathbf{a} \cdot \mathbf{s}_1^T + \mathbf{m}_1 \cdot \mathbf{g}^T + \mathbf{E} \quad \in \mathcal{R}_q^{w \times m} ,$$

together with state $\mathsf{st}_1 = \mathbf{s}_1$.

$\underline{\mathsf{Enc}_2(\mathbf{m}_2) \to \mathsf{ct}_2, \mathsf{st}_2}$**:** Sample a RingLWE secret $s_2 \leftarrow \mathcal{R}_q$ and *truncated* noises $\overline{\mathbf{e}} \leftarrow \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}^{w}$. Output a ciphertext

$$\mathsf{ct}_2 := \mathbf{a} \cdot s_2 + \mathbf{m}_2 + \overline{\mathbf{e}} \quad \in \mathcal{R}_q^{w} ,$$

together with state $\mathsf{st}_2 = \mathbf{s}_2$.

$\underline{\mathsf{KeyGen}(\mathsf{st}_1, \mathsf{st}_2, y) \to \mathsf{sk}_y}$**:** Parse the states $\mathsf{st}_1 = \mathbf{s}_1 \in \mathcal{R}_q^m$ and $\mathsf{st}_2 = s_2 \in \mathcal{R}_q$. Output a decryption key

$$\mathsf{sk}_y := \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y) + s_2 \quad \in \mathcal{R}_q .$$

$\underline{\mathsf{Dec}(\mathsf{sk}_y, \mathsf{ct}_1, \mathsf{ct}_2, y) \to \mathbf{m}_{\mathsf{res}}}$**:** First combine the ciphertexts into

$$\mathsf{ct}_{\mathsf{res}} := \mathsf{ct}_1 \cdot \mathbf{g}^{-1}(y) + \mathsf{ct}_2 \quad \in \mathcal{R}_q^w ,$$
$$// \text{ s.t. } \mathsf{ct}_{\mathsf{res}} = \mathbf{a} \cdot (\mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y) + s_2) + (\mathbf{m}_1 \cdot \mathbf{g}^T \cdot \mathbf{g}^{-1}(y) + \mathbf{m}_2) + \text{noise}$$

Then, use sk_y to decrypt and return $\mathbf{m}_{\mathsf{res}} = \mathsf{ct}_{\mathsf{res}} - \mathbf{a} \cdot \mathsf{sk}_y$, which is supposed to equal $\mathbf{m}_{\mathsf{res}} = \mathbf{m}_1 \cdot y + \mathbf{m}_2 + \text{noise}$.

We now prove correctness and security of Construction 17, LHE.

Correctness. We verify the decryption process, and show that the magnitude of the noise is bounded by $B_{\mathsf{LHE}}(\lambda) = (s \cdot m \cdot \gamma_{\mathcal{R}} \cdot g + \overline{s}) \cdot \sqrt{\lambda}$:

$$\begin{aligned}
\mathbf{m}_{\mathsf{res}} &= \overbrace{\mathsf{ct}_1 \cdot \mathbf{g}^{-1}(y) + \mathsf{ct}_2}^{\mathsf{ct}_{\mathsf{res}}} \;-\; \mathbf{a} \cdot \overbrace{(\mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y) + s_2)}^{\mathsf{sk}_y} \\
&= (\mathbf{m}_1 \cdot y + \mathbf{m}_2) + (\mathbf{E} \cdot \mathbf{g}^{-1}(y) + \overline{\mathbf{e}}) \ , \\
\|\mathbf{E} \cdot \mathbf{g}^{-1}(y) + \overline{\mathbf{e}}\|_\infty &\leq m \cdot \gamma_{\mathcal{R}} \cdot \|\mathbf{E}\|_\infty \cdot \|\mathbf{g}^{-1}(y)\|_\infty + \|\overline{\mathbf{e}}\|_\infty \\
&< m \cdot \gamma_{\mathcal{R}} \cdot g \cdot \max \overline{\mathcal{D}}_{\mathcal{R},s} + \max \overline{\mathcal{D}}_{\mathcal{R},\overline{s}} \\
&< (m \cdot \gamma_{\mathcal{R}} \cdot g \cdot s + \overline{s}) \cdot \sqrt{\lambda} = B_{\mathsf{LHE}} \ .
\end{aligned}$$

Lemma 5.11 (Security of Construction 17)**.** *Assuming* $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}^{\times w}_{T,1}(g)}$*, Construction 17 (*LHE*) fulfills* T-times 2^λ-simulation security *(Definition 5.12).*

Proof. The simulator Sim takes the public parameters pp, the evaluated messages and ring elements $\{\mathbf{m}^{(t)}_{\mathsf{res}}, y^{(t)}\}$, and simulates ciphertexts and decryption keys $(\mathsf{ct}_1, \{\mathsf{ct}^{(t)}_2, \mathsf{sk}^{(t)}_y\})$ as follows:

- Sample the first ciphertext ct_1 and all decryption keys $\mathsf{sk}^{(t)}_y$ at random:

$$\widetilde{\mathsf{ct}}_1 \leftarrow \mathcal{R}_q^{w \times m} \ , \quad \widetilde{\mathsf{sk}}^{(t)}_y \leftarrow \mathcal{R}_q \quad \forall t \in [T] \ .$$

- Then, simulate the remaining ciphertexts $\mathsf{ct}^{(t)}_2$ by sampling noises $\mathbf{E} \leftarrow \chi^{w \times m}$ and $\overline{\mathbf{e}}^{(t)} \leftarrow \overline{\chi}^w$ as in an honest execution (except that they are not truncated), and computing

$$\begin{aligned}
\widetilde{\mathsf{ct}}^{(t)}_{\mathsf{res}} &:= \mathbf{a} \cdot \mathsf{sk}^{(t)}_y + \mathbf{m}^{(t)}_{\mathsf{res}} + (\mathbf{E} \cdot \mathbf{g}^{-1}(y^{(t)}) + \overline{\mathbf{e}}^{(t)}) \quad \forall t \in [T] \\
\widetilde{\mathsf{ct}}^{(t)}_2 &:= \widetilde{\mathsf{ct}}^{(t)}_{\mathsf{res}} - \widetilde{\mathsf{ct}}_1 \cdot \mathbf{g}^{-1}(y^{(t)}) \quad \forall t \in [T]
\end{aligned}$$

We show a series of hybrid experiments that transitions from Hyb_0 (the "real" distribution as defined in Definition 5.12) to Hyb_4 (the "simulated" distribution). We abuse notation to also write Hyb_i as the output of the distribution of the corresponding experiment.

Hyb_0 To recall, Hyb_0 generates ciphertexts $\mathsf{ct}_1, \{\mathsf{ct}_2^{(t)}\}$ and decryption keys $\{\mathsf{sk}_y^{(t)}\}$ in the following way:

$$\boxed{\mathbf{a}} \leftarrow \mathcal{R}_q^m \tag{5.9}$$

$$\boxed{\mathsf{ct}_1} := \mathbf{a} \cdot \mathbf{s}_1^T + \mathbf{m}_1 \cdot \mathbf{g}^T + \mathbf{E} \quad \left| \quad \begin{array}{l} \mathbf{s}_1 \leftarrow \mathcal{R}_q^m \\ \mathbf{E} \leftarrow \overline{\mathcal{D}}_{\mathcal{R},s}^{w\times m} \end{array} \right. \tag{5.10}$$

For $t \in [T]$:

$$\boxed{\mathsf{ct}_2^{(t)}} := \mathbf{a} \cdot s_2^{(t)} + \mathbf{m}_2^{(t)} + \overline{\mathbf{e}}^{(t)} \quad \left| \quad \begin{array}{l} s_2^{(t)} \leftarrow \mathcal{R}_q \\ \overline{\mathbf{e}}^{(t)} \leftarrow \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}^{w} \end{array} \right. \tag{5.11}$$

$$\boxed{\mathsf{sk}_y^{(t)}} := \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)} \tag{5.12}$$

Hyb_1 As a first step, in Hyb_1 we sample the errors $\mathbf{E}$ and $\overline{\mathbf{e}}^{(t)}$ as Gaussians from $\chi^{w\times m} = \mathcal{D}_{\mathcal{R},s}^{w\times m}$ and $\overline{\chi}^w = \mathcal{D}_{\mathcal{R},\overline{s}}^w$, instead of truncated Gaussians from $\overline{\mathcal{D}}_{\mathcal{R},s}^{w\times m}$ and $\overline{\mathcal{D}}_{\mathcal{R},\overline{s}}^w$, respectively. By Lemma 5.2, we have $\mathsf{Hyb}_1 \approx_s^{2^\lambda} \mathsf{Hyb}_0$.

Hyb_2 Now, we switch the order of computation: instead of computing $\mathsf{ct}_2^{(t)}$ directly from $\mathbf{m}_2^{(t)}$, we first define the *combined* ciphertext $\mathsf{ct}_{\mathsf{res}}^{(t)}$ (which is an encryption of $\mathbf{m}_{\mathsf{res}}$ under $\mathsf{sk}_y^{(t)}$), and then simulate $\mathsf{ct}_2^{(t)}$ as a combination of $\mathsf{ct}_{\mathsf{res}}^{(t)}$ and ct_1.

More precisely, we leave ct_1 and $\mathsf{sk}_y^{(t)}$ unchanged from the previous hybrid, but instead compute $\widetilde{\mathsf{ct}}_2^{(t)}$ as follows:

$$\boxed{\mathsf{sk}_y^{(t)}} := \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)} \quad \left| \quad s_2^{(t)} \leftarrow \mathcal{R}_q \right.$$

$$\boxed{\widetilde{\mathsf{ct}}_2^{(t)}} := \widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} - \widetilde{\mathsf{ct}}_1 \cdot \mathbf{g}^{-1}(y^{(t)}) \quad \left| \quad \begin{array}{l} \overline{\mathbf{e}}^{(t)} \leftarrow \overline{\chi}^w \\ \widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} := \mathbf{a} \cdot \mathsf{sk}_y^{(t)} + \mathbf{m}_{\mathsf{res}}^{(t)} + (\mathbf{E} \cdot \mathbf{g}^{-1}(y^{(t)}) + \overline{\mathbf{e}}^{(t)}) \end{array} \right.$$

By definition of $\mathsf{sk}_y^{(t)} = \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)}$ and $\mathbf{m}_{\mathsf{res}}^{(t)} = \mathbf{m}_1 \cdot y^{(t)} + \mathbf{m}_2^{(t)}$, this way of defining ct_2 is identical to the previous one. Therefore, we have $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_1$.

Hyb_3 Note that in the previous hybrid, the uniformly random key $s_2^{(t)}$ is used nowhere but in the definition of $\mathsf{sk}_y^{(t)} = \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)}$. Therefore, we may equivalently generate

$$\boxed{\widetilde{\mathsf{sk}}_y^{(t)}} \leftarrow \mathcal{R}_q \quad \forall t \in [T]$$

uniformly random instead of computing it from $s_2^{(t)}$. We get $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_2$.

Hyb_4 In this hybrid, we replace the first ciphertext $\mathsf{ct}_1 = \mathbf{a} \cdot \mathbf{s}_1^T + \mathbf{m}_1 \cdot \mathbf{g}^T + \mathbf{E}$ by a uniformly generated vector

$$\widetilde{\mathsf{ct}}_1 \leftarrow \mathcal{R}_q^{w \times m} .$$

While we would like to use the RingLWE assumption for this step, note that the noise matrix $\mathbf{E}$ is still used in the definition of $\widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)}$. Therefore, we need to use the more powerful elLWE, which allows us to utilize the noise leakage $\mathbf{E} \cdot \mathbf{g}^{-1}(y^{(t)}) + \overline{\mathbf{e}}^{(t)}$.

In more detail, we define sub-hybrids $\mathsf{Hyb}_{3,i}$ for $i = 0, \ldots, m$, where $\mathsf{Hyb}_{3,i}$ is identical to Hyb_3, except that the first i columns of ct_1 are sampled uniformly random. Note that $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_{3,0}$ and $\mathsf{Hyb}_{3,m} \equiv \mathsf{Hyb}_4$. Therefore, in order to prove $\mathsf{Hyb}_4 \approx_{\mathrm{c}} \mathsf{Hyb}_3$, it only remains to show $\mathsf{Hyb}_{3,i-1} \approx_{\mathrm{c}} \mathsf{Hyb}_{3,i}$ for $i = 1, \ldots, m$. To do so, assume there exists a distinguisher $\mathcal{D}$ for $\mathsf{Hyb}_{3,i-1}$ and $\mathsf{Hyb}_{3,i}$. We construct an adversary $\mathcal{A}$ for $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$ in the following way (where the running time of $\mathcal{A}$ additively increases by $\mathrm{poly}(\lambda)$ when compared to that of $\mathcal{D}$):

- After receiving public parameters $\mathbf{a} \in \mathcal{R}_q^w$, choose the *leakage matrix* $\mathbf{Z} \in \mathcal{R}_q^{Tw \times w}$ in the following way. Let $\mathbf{g}^{-1}(y^{(t)})[i]$ be the i-th entry of the decomposition $\mathbf{g}^{-1}(y^{(t)})$. Then, select

$$\mathbf{z}^T := \begin{pmatrix} \mathbf{g}^{-1}(y^{(1)})[i] & \ldots & \mathbf{g}^{-1}(y^{(T)})[i] \end{pmatrix}$$

and choose

$$\mathbf{Z} := \mathrm{diag}(\underbrace{\mathbf{z}, \ldots, \mathbf{z}}_{w \text{ times}}) .$$

Note that this is saying that the noise $\mathbf{E}[j,i]$ *(i.e., j-th row and i-th column of* $\mathbf{E}$*) is leaked T times; once in the j-th row of each* $\mathsf{ct}_{\mathsf{res}}^{(t)}$*.*

- After receiving the LWE sample $\mathbf{y} \in \mathcal{R}_q^w$ and leakage $\mathbf{l} \in \mathcal{R}_q^{Tw}$, first split the leakage $\mathbf{l}^T = (\mathbf{l}[1], \ldots, \mathbf{l}[Tw])$ into T separate vectors

$$\mathbf{l}_t^T = (\mathbf{l}[t], \mathbf{l}[T+t], \ldots, \mathbf{l}[(w-1)T+t]) \quad \forall t \in [T] \,.$$

Note that the vector $\mathbf{l}_t$ *will correspond to a leakage of the noise vector* $\mathbf{E}[:,i]$ *when multiplied with* $\mathbf{g}^{-1}(y^{(t)})[i]$*, and then hidden by a larger noise* $\overline{\mathbf{e}}^{(t)}$*.*

Then, run and output the result of $\mathcal{D}$ on a simulation of $\mathsf{Hyb}_{3,i}$, with two modifications:

  - the i-th column of ct_1 is computed as $\mathbf{y}^T + \mathbf{m}_1 \cdot g^i$, and
  - for any $t \in [T]$, the ciphertext $\mathsf{ct}_{\mathsf{res}}^{(t)}$ is computed as

$$\mathsf{ct}_{\mathsf{res}}^{(t)} := \mathbf{a} \cdot \mathsf{sk}_y^{(t)} + \mathbf{m}_{\mathsf{res}}^{(t)} + \left(\sum_{i' \in [T] \setminus \{i\}} \mathbf{E}[:,i'] \cdot \mathbf{g}^{-1}(y^{(t)})[i'] + \mathbf{l}_t \right) \,.$$

The error-leakage RingLWE experiment with adversary $\mathcal{A}$ has the following distributions:

- Consider the experiment $\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}^{\times w}_{T,1}(g)}$. Here, $\mathbf{y}$ corresponds to a real LWE sample, i.e., $\mathbf{y} = \mathbf{a} \cdot \mathbf{s}_1[i] + \mathbf{E}[:,i]$ with leakage $\mathbf{l}_t = \mathbf{E}[:,i] \cdot \mathbf{g}^{-1}(y^{(t)})[i] + \overline{\mathbf{e}}^{(t)}$ (for some fresh key $\mathbf{s}_1[i] \leftarrow \mathcal{R}_q$ and errors $\mathbf{E}[:,i] \leftarrow \mathcal{R}_q^w$ and $\overline{\mathbf{e}}^{(t)} \leftarrow \overline{\chi}^w$).

  Therefore, $\mathsf{elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}^{\times w}_{T,1}(g)}$ is identically distributed as the output of $\mathcal{D}$ when given input generated from distribution $\mathsf{Hyb}_{3,i-1}$.
- Consider the experiment $\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}^{\times w}_{T,1}(g)}$. Here, $\mathbf{y} \leftarrow \mathcal{R}_q^{2m}$ is uniformly random and the leakage is equal to $\mathbf{l}_t = \mathbf{E}[:,i] \cdot \mathbf{g}^{-1}(y^{(t)})[i] + \overline{\mathbf{e}}^{(t)}$ (for fresh errors $\mathbf{E}[:,i] \leftarrow \mathcal{R}_q^w$ and $\overline{\mathbf{e}}^{(t)} \leftarrow \overline{\chi}^w$).

  Note that the i-th column of ct_1 will be uniformly random (due to the randomness of $\mathbf{y}$). Therefore, $\mathsf{elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}^{\times w}_{T,1}(g)}$ is identically distributed as the output of $\mathcal{D}$ when given input generated from distribution $\mathsf{Hyb}_{3,i}$.

Therefore, the distinguisher $\mathcal{D}$ has exactly the same advantage as the adversary $\mathcal{A}$. By the $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}^{\times w}_{T,1}(g)}$ assumption, this implies that $\mathsf{Hyb}_{3,i-1} \approx_{\mathrm{c}}^{2^\lambda} \mathsf{Hyb}_{3,i}$.

By the hybrid argument, we get $\mathsf{Hyb}_0 \approx_{c}^{2^\lambda} \mathsf{Hyb}_4$. Because Hyb_4 is identical to the simulated world, T-times simulation security follows. □

5.6.1 LHE Parameter Setting

We now prove the claimed efficiencies for the parameter setting in Lemma 5.6.

of Lemma 5.6. Under assumption $\mathsf{LWE}_{\mathcal{R},w,q,\mathcal{D}_{\mathcal{R},q^{0.1}}}$, we may conclude (using Theorem 5.2 and Lemma 5.4) that also the error-leakage version $\mathsf{elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$ holds, with error distributions χ and $\overline{\chi}$ of parameters $s = 3q^{0.1}$ and $\overline{s} = 2s \cdot g \cdot n\sqrt{T}$.

To obtain the claimed parameters, we choose the gadget-base $g = \lceil q^{0.05} \rceil$ (as used inside of our LEnc construction) and hence $m \leq 20$. Thus, we get

$$\begin{aligned} B_{\mathsf{LHE}} &\leq (\underbrace{g}_{\leq 2q^{0.05}} \cdot \underbrace{m}_{\leq 20} \cdot \underbrace{\gamma_{\mathcal{R}}}_{\leq q^{1/15}} + 2 \cdot \underbrace{g}_{\leq 2q^{0.05}} \cdot \underbrace{n}_{\leq q^{1/15}} \cdot \underbrace{\sqrt{T}}_{\leq q^{1/4}}) \cdot q^{0.1} \underbrace{\sqrt{\lambda}}_{\leq q^{1/30}} \cdot 3 \\ &\leq 10 q^{1/2} \, . \end{aligned}$$

□

5.7 Application: Preprocessing Garbling

In this section, we apply our construction of batch-select towards the notion of *preprocessing garbling* with succinct function-dependent garbling. We first describe the model in Section 5.7.1 and then our desired primitive in Section 5.7.2. Afterwards, we define some existing building blocks that we require (apart from batch-select) in Section 5.7.3, and give the final construction in Section 5.7.4.

5.7.1 Computational Model

A *preprocessing garbling scheme* will handle computations specified by a *universal* function and a *function* description. For example, a universal function may specify parameters of a Boolean circuit such as input and output lengths as well as circuit size, while the function description may specify the gate types and how they are connected.

We formalize this model of computation as a family $\mathcal{U} = \{\mathcal{U}_{\ell_x,\ell_y}\}_{\ell_x,\ell_y \in \mathbb{N}}$ of classes of *universal* functions $U \in \mathcal{U}_{\ell_x,\ell_y}$ with the format

$$U : \mathcal{F}_U \times \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_y} ,$$

where $f \in \mathcal{F}_U$ is the *function* description. The two components U, f together with an input bit string $\mathbf{x} \in \{0,1\}^{\ell_x}$ determine the output $\mathbf{y} = U(f, \mathbf{x})$. Our preprocessing garbling construction will take as input the universal function U in as a Boolean circuit, and $f \in \mathcal{F}_U$, a string.

In the case where U is a universal function capable of handling any function f that is specified by a binary circuit, we may use the following result from prior work, which shows that there exists efficient universal circuits.

Lemma 5.12 ([ZYZL19]). *There is a Boolean circuit C_U of size $17.75n \log n$ (with $4.5n \log n$ AND gates) that takes a function description $f : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_y}$ (represented by a Boolean circuit with at most n gates) and an input $\mathbf{x} \in \{0,1\}^{\ell_x}$, and outputs $f(\mathbf{x})$.*

5.7.2 Definition

In the following definition for *preprocessing garbling*, the algorithms RDGen, InputKeyGen, and GarbleU together form the *function-independent offline* phase that only depends on the universal function U. The *function-dependent offline* phase corresponds to GarbleFunc, which then takes the function description f. The ultimate goal, as achieved by our instantiation, is that the output of GarbleFunc is *succinct*, i.e., its length is independent of that of the function description f.

Definition 5.13 (Preprocessing Garbling). *Let $\mathcal{U} = \{\mathcal{U}_{\ell_x,\ell_y}\}$ be a class of computation, where each universal function $U \in \mathcal{U}_{\ell_x,\ell_y}$ has a signature $U : \mathcal{F}_U \times \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_y}$. A preprocessing garbling scheme (with reusability) for $\mathcal{U}$ consists of five efficient algorithms, and is associated with a key space $\mathcal{K}(\lambda)$, which is an abelian group with size $|\mathcal{K}(\lambda)| \geq 2^{\lambda}$.*

- $\mathsf{RDGen}(U)$ *takes the universal function* U *and outputs the reusable part* $\widehat{U}_{\mathsf{rd}}$ *of the function-independent garbling, together with a reusable state* $\mathsf{st}_{\mathsf{rd}}$.
- $\mathsf{InputKeyGen}(1^\lambda, 1^{\ell_x})$ *takes an input length* ℓ_x*, and returns input keys* $\mathbf{K} \in \mathcal{K}^{\ell_x \times 2}$.
- $\mathsf{GarbleU}(\mathsf{st}_{\mathsf{rd}}, \mathbf{K})$ *takes the reusable state* $\mathsf{st}_{\mathsf{rd}}$*, and the input keys* $\mathbf{K}$*. It outputs the non-reusable part* $\widehat{U}$ *of the function-independent garbling, and state* st.
- $\mathsf{GarbleFunc}(\mathsf{st}, f \in \mathcal{F}_U)$ *takes the state* st*, a function description* f*, and outputs a* hint*, i.e., the function-dependent garbling. We refer to* $(\widehat{U}_{\mathsf{rd}}, \widehat{U}, \mathsf{hint})$ *as the garbling of* (U, f).
- $\mathsf{Eval}(f, \widehat{U}_{\mathsf{rd}}, \widehat{U}, \mathsf{hint}, \mathbf{k_x} \in \mathcal{K}^{\ell_x})$ *takes a function description* f*, the garbling* $(\widehat{U}_{\mathsf{rd}}, \widehat{U}, \mathsf{hint})$*, and input labels* $\mathbf{k_x}$*, and outputs the computation result* $\mathbf{y} \in \{0,1\}^{\ell_y}$.

Correctness. *The scheme is correct if for all* $\lambda, \ell_x, \ell_y \in \mathbb{N}$*, universal functions* $U \in \mathcal{U}_{\ell_x,\ell_y}$ *and function descriptions* $f \in \mathcal{F}_U$*, input keys* $\mathbf{K} \in \mathcal{K}^{\ell_x \times 2}$*, and inputs* $\mathbf{x} \in \{0,1\}^{\ell_x}$*, the following holds:*

$$\Pr\left[\begin{array}{l} \mathsf{Eval}(f, \widehat{U}_{\mathsf{rd}}, \widehat{U}, \mathsf{hint}, \mathbf{K}[\mathbf{x}]) \\ = U(f, \mathbf{x}) \end{array} \middle| \begin{array}{r} (\widehat{U}_{\mathsf{rd}}, \mathsf{st}_{\mathsf{rd}}) \leftarrow \mathsf{RDGen}(U) \\ (\widehat{U}, \mathsf{st}) \leftarrow \mathsf{GarbleU}(\mathsf{st}_{\mathsf{rd}}, \mathbf{K}) \\ \mathsf{hint} \leftarrow \mathsf{GarbleFunc}(\mathsf{st}, f) \end{array}\right] = 1.$$

Definition 5.14 (T-times Input Privacy)**.** *The scheme fulfills* T*-times input privacy if there exists an efficient simulator* $\mathsf{Sim}_{\mathsf{Priv}}$ *s.t. for any efficient adversary* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{Exp}^{\mathcal{A},0}_{\mathsf{Priv}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathcal{A},1}_{\mathsf{Priv}}(\lambda) = 1]\right| \leq \mathrm{negl}(\lambda),$$

where the game $\mathsf{Exp}^{\mathcal{A},b}_{\mathsf{Priv}}(\lambda)$ *is defined below.*

1. $\mathcal{A}(1^\lambda)$ *selects* $1^{\ell_x}, 1^{\ell_y}$*, a universal function* $U \in \mathcal{U}_{\ell_x,\ell_y}$*, and* T *function descriptions* $f^{(t)} \in \mathcal{F}_U$*, and inputs* $\mathbf{x}^{(t)} \in \{0,1\}^{\ell_x}$*, for* $t \in [T]$.

2. *The game sends to* $\mathcal{A}$ *the reusable garbling* $\widehat{U}_{\mathsf{rd}}$ *and the* T *individual garblings with their input keys* $\{\widehat{U}^{(t)}, \mathsf{hint}^{(t)}, \mathbf{k}^{(t)}_{\mathbf{x}}\}_{t \in [T]}$*, computed as follows.*

   - *If* $b = 0$*, first run* $(\widehat{U}_{\mathsf{rd}}, \mathsf{st}_{\mathsf{rd}}) \leftarrow \mathsf{RDGen}(U)$*. Then for* $t \in [T]$ *run*

$$\mathbf{K}^{(t)} \leftarrow \mathsf{InputKeyGen}(1^\lambda, 1^{\ell_x}),$$
$$(\widehat{U}^{(t)}, \mathsf{st}^{(t)}) \leftarrow \mathsf{GarbleU}(\mathsf{st}_{\mathsf{rd}}, \mathbf{K}^{(t)}), \qquad \mathsf{hint}^{(t)} \leftarrow \mathsf{GarbleFunc}(\mathsf{st}^{(t)}, f^{(t)})$$

Finally, set $\mathbf{k}_{\mathbf{x}}^{(t)} = \mathbf{K}^{(t)}[\mathbf{x}]$.

- *If* $b = 1$*, run* $(\widehat{U}_{\mathsf{rd}}, \{\widehat{U}^{(t)}, \mathsf{hint}^{(t)}, \mathbf{k}_{\mathbf{x}}^{(t)}\}) \leftarrow \mathsf{Sim}_{\mathsf{Priv}}(U, \{f^{(t)}, U(f^{(t)}, \mathbf{x}^{(t)})\}_{t\in[T]})$.

3. *$\mathcal{A}$ outputs a bit b', which is also the output of the game.*

5.7.3 Ingredients

Standard Model Garbling. Traditional garbling (i.e., the standard model in which there is no separation between universal function and function description) may be seen as a special case of preprocessing garbling. It will be required for our construction of preprocessing garbling.

Definition 5.15 (Standard Model Garbling)**.** *Let* $\mathcal{C} = \{\mathcal{C}_{\ell_x,\ell_y}\}_{\ell_x,\ell_y\in\mathbb{N}}$ *be the family of functions that do not take any function description, i.e.,* $\mathcal{C}_{\ell_x,\ell_y}$ *consists of all two-input functions* $C : \emptyset \times \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_y}$*, specified by their Boolean circuit representation.*

For this class, a corresponding garbling scheme (which we denote by SG*, "standard garbling") without support for preprocessing or reusability has the following simplified syntax, where* RDGen *and* $\mathsf{GarbleFunc}$ *are not needed anymore, and* $\mathsf{GarbleU}$ *is renamed to* Garble*.*

- $\mathsf{SG.InputKeyGen}(1^\lambda, 1^{\ell_x})$ *takes an input length* ℓ_x*, and returns input keys* $\mathbf{K} \in \mathcal{K}^{\ell_x\times 2}$*.*
- $\mathsf{SG.Garble}(C, \mathbf{K})$ *takes the circuit* C *and the input keys* $\mathbf{K}$*, and outputs the garbling* $\widehat{C}$*.*
- $\mathsf{SG.Eval}(C, \widehat{C}, \mathbf{k}_{\mathbf{x}} \in \mathcal{K}^{\ell_x})$ *takes the circuit* C*, its garbling* $\widehat{C}$ *and input labels* $\mathbf{k}_{\mathbf{x}}$*, and outputs the computation result* $\mathbf{y} \in \{0,1\}^{\ell_y}$*.*

Further, we require that $\mathsf{SG.InputKeyGen}(1^\lambda, 1^{\ell_x})$ *samples each of the* ℓ_x *input key pairs individually, i.e., its implementation only consists of running* $(\mathbf{K}[0,i], \mathbf{K}[1,i]) \leftarrow \mathsf{InputKeyGen}'(1^\lambda)$ *for some algorithm* $\mathsf{InputKeyGen}'$*.*

Because we do not require reusability for the standard garbling scheme SG*, we use the term* input privacy *to denote* 1-time input privacy *(Definition 5.14).*

Garbling schemes in the standard model as above are known to exist, with the current state-of-the-art producing a garbling whose size is 1.5λ times the number of AND gates.

Lemma 5.13 ([RR21])**.** *There is a standard model garbling scheme* SG *(with global key offsets), with garblings of size* $|\widehat{C}| = (1.5\lambda + 10)n$ *bits, where* n *denotes the number of AND gates in the circuit* C*.*

Correlation-robust hash functions. In our construction of preprocessing garbling, we will need to translate the output of a batch-select scheme (which is in $\mathcal{M}$, or concretely $\mathcal{M} = \mathbb{Z}_p^\ell$ for our instantiation in Section 5.4.3) into keys in $\mathcal{K}$ for the input wires of a circuit C_U garbled using a standard garbling scheme SG.

We do so by hiding a key in $\mathcal{K}$ using a hash function $H : \mathcal{M} \to \mathcal{K}$ applied to the corresponding batch-select output that is in $\mathcal{M}$. Furthermore, whenever our garbling scheme is reused ($T > 1$), we will need to ensure that H is correlation-robust in the following sense.

Definition 5.16 (Correlation-robust Hash Function)**.** *A hash function* $H : \mathcal{M} \to \mathcal{K}$ *(where* $\mathcal{M}$ *and* $\mathcal{K}$ *are parameterized by the security parameter* λ*) is* correlation-robust*, if for any polynomial* $T(\lambda)$ *and efficient adversaries* $\mathcal{A}$*, the following holds (where we suppress* λ*):*

$$\Bigg| \Pr[\mathcal{A}(\{\mathbf{l}^{(t)}, H(\mathbf{l}^{(t)} - \Delta), H(\mathbf{l}^{(t)} + \Delta)\}_{t\in[T]}) = 1 \mid \Delta, \mathbf{l}^{(1)}, \ldots, \mathbf{l}^{(T)} \leftarrow \mathcal{M}] -$$

$$\Pr\left[\mathcal{A}(\{\mathbf{l}^{(t)}, \mathbf{u}_-^{(t)}, \mathbf{u}_+^{(t)}\}_{t\in[T]}) = 1 \;\middle|\; \begin{array}{l} \mathbf{l}^{(1)}, \ldots, \mathbf{l}^{(T)} \leftarrow \mathcal{M}, \\ \mathbf{u}_-^{(1)}, \ldots, \mathbf{u}_-^{(T)}, \mathbf{u}_+^{(1)}, \ldots, \mathbf{u}_+^{(T)} \leftarrow \mathcal{K} \end{array}\right] \Bigg| \leq \mathrm{negl}(\lambda)$$

Due to $|\mathcal{M}| \geq 2^\lambda$, the correlation-robust hash function H can be instantiated using a random oracle. Also note that it is a *weaker notion* of the type of circular correlation-robustness that is already utilized by many existing garbling schemes involving e.g. the free-XOR optimization [KS08a].

5.7.4 Construction

We will construct a preprocessing garbling scheme for any computation class $\mathcal{U}$, s.t. for any universal function $U \in \mathcal{U}_{\ell_x,\ell_y}$, there is a universal circuit C_U that takes a function description $f \in \mathcal{F}_U$ whose bit-length is bounded by some s_U, an input $\mathbf{x} \in \{0,1\}^{\ell_x}$, and outputs $U(f, \mathbf{x})$.

For our construction, we assume a standard garbling scheme SG with key space $\mathcal{K}$ and a batch-select with message space $\mathcal{M} = \mathbb{Z}_p^\ell$. Further, we assume a correlation-robust hash function $H : \mathcal{M} \to \mathcal{K}$.

Construction 18 (Preprocessing Garbling)**.**

$\underline{\mathsf{RDGen}(U) \to \widehat{U}_{\mathsf{rd}}, \mathsf{st}_{\mathsf{rd}}}$**:** *(This algorithm creates reusable data, which can be used again for subsequent garbling sessions.)*

First setup the batch-select Sel scheme with dimension s_U, where s_U is the description length of functions in $\mathcal{F}_U$:

$$\mathsf{Sel.pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U}).$$

Next, sample the first message vector $\mathbf{l}_1$, and encrypt it using $\mathsf{Sel.Enc}_1$:

$$\mathbf{l}_1 \leftarrow \mathcal{M}^{s_U}, \quad (\mathsf{Sel.ct}_1, \mathsf{Sel.st}_1) \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1).$$

Output the reusable part of the garbling $\widehat{U}_{\mathsf{rd}} = (\mathsf{Sel.pp}, \mathsf{Sel.ct}_1)$, and the reusable state $\mathsf{st}_{\mathsf{rd}} = (\mathsf{Sel.pp}, \mathsf{Sel.st}_1, \mathbf{l}_1)$.

$\underline{\mathsf{InputKeyGen}(1^\lambda, 1^{\ell_x}) \to \mathbf{K}}$**:** Output input keys $\mathbf{K} \leftarrow \mathsf{SG.InputKeyGen}(1^\lambda, 1^{\ell_x})$.

$\underline{\mathsf{GarbleU}(\mathsf{st}_{\mathsf{rd}}, \mathbf{K}) \to \widehat{U}, \mathsf{st}}$**:** First, sample the input keys $\mathbf{K}^{\mathsf{Fn}}$ for the input wires of the universal circuit C_U that correspond to the function description, and use this to garble the universal circuit C_U:

$$\mathbf{K}^{\mathsf{Fn}} \leftarrow \mathsf{SG.InputKeyGen}(1^\lambda, 1^{s_U}), \quad \widehat{C}_U \leftarrow \mathsf{SG.Garble}(C_U, (\mathbf{K}^{\mathsf{Fn}}, \mathbf{K}))\ .$$

Next, sample the second message vector $\mathbf{l}_2$, and encrypt it using $\mathsf{Sel.Enc}_2$:

$$\mathbf{l}_2 \leftarrow \mathcal{M}^{s_U}, \quad (\mathsf{Sel.ct}_2, \mathsf{Sel.st}_2) \leftarrow \mathsf{Sel.Enc}_2(\mathbf{l}_2).$$

Then, for each $i \in [s_U]$, produce the ciphertexts $\mathsf{ct}'_{0,i}$ and $\mathsf{ct}'_{1,i}$ that the evaluator will use to translate the batch-select output $\mathbf{f}[i] \cdot \mathbf{l}_1[i] + \mathbf{l}_2[i]$ into the correct key $\mathbf{K}^{\mathsf{Fn}}[i, \mathbf{f}[i]]$:

$$\mathsf{ct}'_{0,i} := H(\mathbf{l}_2[i]) + \mathbf{K}^{\mathsf{Fn}}[i, 0]$$
$$\mathsf{ct}'_{1,i} := H(\mathbf{l}_1[i] + \mathbf{l}_2[i]) + \mathbf{K}^{\mathsf{Fn}}[i, 1]$$

Output function-independent garbling $\widehat{U} = (\widehat{C}_U, \mathsf{Sel.ct}_2, \{\mathsf{ct}'_{0,i}, \mathsf{ct}'_{1,i}\})$, and the state $\mathsf{st} = (\mathsf{Sel.pp}, \mathsf{Sel.st}_1, \mathsf{Sel.st}_2)$.

$\underline{\mathsf{GarbleFunc}(\mathsf{st}, f) \to \mathsf{hint}}$**:** Let $\mathbf{f} \in \{0,1\}^{s_U}$ denote the bit representation of f. Output a batch-select secret key for selecting $\mathbf{K}^{\mathsf{Fn}}[\mathbf{f}]$:

$$\mathsf{hint} = \mathsf{sk}_{\mathbf{f}} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{Sel.st}_1, \mathsf{Sel.st}_2, \mathbf{f}).$$

$\underline{\mathsf{Eval}(f, \widehat{U}_{\mathsf{rd}}, \widehat{U}, \mathsf{hint}, \mathbf{k_x}) \to \mathbf{y}}$**:** Parse the function-independent garbling as $\widehat{U}_{\mathsf{rd}} = (\mathsf{Sel.pp}, \mathsf{Sel.ct}_1)$ and $\widehat{U} = (\widehat{C}_U, \mathsf{Sel.ct}_2, \{ct'_{0,i}, \mathsf{ct}'_{1,i}\})$. Let $\mathbf{f} \in \{0,1\}^{s_U}$ denote the bit representation of f.

First decrypt the batch-select ciphertexts to recover the messages $\mathbf{l}_{\mathsf{res}}$ selected by $\mathbf{f}$:

$$\mathbf{l}_{\mathsf{res}} \leftarrow \mathsf{Sel.Dec}(\mathsf{sk}_{\mathbf{f}}, \mathsf{Sel.ct}_1, \mathsf{Sel.ct}_2, \mathbf{f}),$$
$$// \text{ s.t.} \quad \mathbf{l}_{\mathsf{res}} = \mathbf{l}_1 \odot \mathbf{f} + \mathbf{l}_2$$

Then, translate these messages into the appropriate input keys $\mathbf{k}_{\mathbf{f}}^{\mathsf{Fn}}$ by computing, for each $i \in [s_U]$:

$$\mathbf{k}_{\mathbf{f}}^{\mathsf{Fn}}[i] := \mathsf{ct}'_{\mathbf{f}[i],i} - H(\mathbf{l}_{\mathsf{res}}[i]) ,$$
$$// \text{ s.t.} \quad \mathbf{k}_{\mathbf{f}}^{\mathsf{Fn}}[i] = \mathbf{K}^{\mathsf{Fn}}[i, \mathbf{f}[i]] .$$

Then evaluate the garbled universal circuit as $\mathbf{y} \leftarrow \mathsf{SG.Eval}(\widehat{C}_U, (\mathbf{k}_{\mathbf{f}}^{\mathsf{Fn}}, \mathbf{k_x}))$, and output $\mathbf{y}$.

Correctness and security. Correctness follows straightforwardly from that of the standard model garbling scheme SG and that of the batch-select scheme Sel. We state and prove security formally below.

Lemma 5.14. *Assuming the standard garbling scheme* SG *satisfies input privacy, the batch-select scheme* Sel *satisfies* T*-times simulation security (Definition 5.5), and the hash function* H *is correlation-robust (Definition 5.16), the preprocessing garbling scheme in Construction 18 fulfills* T*-times input privacy (Definition 5.14).*

Proof. We describe the simulator $\mathsf{Sim}_{\mathsf{Priv}}$ required by Definition 5.14. It takes an offline function U, as well as T online descriptions $\{f^{(t)}\}$ and evaluation results $\{\mathbf{y}^{(t)}\}$. It simulates T garblings and input keys as described below.

- First run the standard model garbling simulator to simulate

$$\widetilde{C}_U^{(t)}, \widetilde{\mathbf{k}}_{\mathbf{f}}^{\mathsf{Fn},(t)}, \widetilde{\mathbf{k}}_{\mathbf{x}}^{(t)} \leftarrow \mathsf{SG.Sim}_{\mathsf{Priv}}(C_U, \mathbf{y}^{(t)}) \quad \forall t \in [T] .$$

- Next, simulate the batch-select output message vector randomly as

$$\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)} \leftarrow \mathcal{M}^{s_U} ,$$

and apply T-times security of Sel to simulate

$$\mathsf{Sel.pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U}),$$
$$\widetilde{\mathsf{Sel.ct}}_1, \{\widetilde{\mathsf{Sel.ct}}_2^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{f}^{(t)}}^{(t)}\} \leftarrow \mathsf{Sel.Sim}(\mathsf{Sel.pp}, \{\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}, \mathbf{f}^{(t)}\}).$$

- Finally, simulate the key translation ciphertexts in accordance with the batch-select output $\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}$: for the ciphertexts $\widetilde{\mathsf{ct}}'^{(t)}_{\mathbf{f}^{(t)}[i],i}$ that are decodable by the evaluator, we choose

$$\widetilde{\mathsf{ct}}'^{(t)}_{\mathbf{f}^{(t)}[i],i} := H(\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}[i]) + \widetilde{\mathbf{k}}_{\mathbf{f}}^{\mathsf{Fn},(t)}[i] ,$$

and we sample the remaining ciphertexts uniformly at random:

$$\widetilde{\mathsf{ct}}'^{(t)}_{1-\mathbf{f}^{(t)}[i],i} \leftarrow \mathcal{K} .$$

Output the simulated reusable garbling part $\widehat{U}_{\mathsf{rd}} = (\mathsf{Sel.pp}, \widetilde{\mathsf{Sel.ct}}_1)$, and for each iteration $t \in [T]$ the simulated offline garbling $\widehat{U}^{(t)} = (\widetilde{C}_U^{(t)}, \widetilde{\mathsf{Sel.ct}}_2^{(t)}, \{\widetilde{\mathsf{ct}}'^{(t)}_{0,i}, \widetilde{\mathsf{ct}}'^{(t)}_{1,i}\})$, online garbling $\mathsf{hint}^{(t)} = \widetilde{\mathsf{sk}}_{\mathbf{f}^{(t)}}^{(t)}$, and input keys $\widetilde{\mathbf{k}}_{\mathbf{x}}^{(t)}$.

We show a series of hybrids that transitions from $\mathsf{Hyb}_0 = \mathsf{Exp}^{\mathcal{A},0}_{\mathsf{Priv}}(\lambda)$ to $\mathsf{Hyb}_2 = \mathsf{Exp}^{\mathcal{A},1}_{\mathsf{Priv}}(\lambda)$. We abuse the notation to also write Hyb_i as the output distribution of the experiment.

Hyb_0: We briefly recall the game $\mathsf{Exp}^{\mathcal{A},0}_{\mathsf{Priv}}(\lambda)$. $\mathcal{A}$ selects $1^{\ell_x}, 1^{\ell_y}$, an offline function U, and T online descriptions $f^{(t)} \in \mathcal{F}_U$ (with bit representations $\mathbf{f}^{(t)} \in \{0,1\}^{s_U}$) and inputs $\mathbf{x}^{(t)} \in \{0,1\}^{\ell_x}$. The information received by the adversary is indicated through boxed terms.

$$\left.\begin{array}{r}\boxed{\mathsf{Sel.pp}} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U}) \\ \boxed{\mathsf{Sel.ct}_1}, \mathsf{Sel.st}_1 \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1)\end{array}\right| \mathbf{l}_1 \leftarrow \mathcal{M}^{s_U} \tag{5.13}$$

$$\text{for } t \in [T]:$$

$$\left.\begin{array}{r}\boxed{\mathsf{Sel.ct}_2^{(t)}}, \mathsf{Sel.st}_2^{(t)} \leftarrow \mathsf{Sel.Enc}_2(\mathbf{l}_2^{(t)}) \\ \boxed{\mathsf{sk}_{\mathbf{f}^{(t)}}^{(t)}} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{Sel.st}_1, \mathsf{Sel.st}_2^{(t)}, \mathbf{f}^{(t)})\end{array}\right| \mathbf{l}_2^{(t)} \leftarrow \mathcal{M}^{s_U} \tag{5.14}$$

$$\left.\begin{array}{r}\boxed{\widehat{C}_U^{(t)}} \leftarrow \mathsf{SG.Garble}(C_U, (\mathbf{K}^{\mathsf{Fn},(t)}, \mathbf{K}^{(t)})) \\ \boxed{\mathbf{k}_{\mathbf{x}}^{(t)}} = \mathbf{K}^{(t)}[\mathbf{x}^{(t)}].\end{array}\right| \begin{array}{r}\mathbf{K}^{\mathsf{Fn},(t)} \leftarrow \mathsf{SG.InputKeyGen}(1^\lambda, 1^{s_U}) \\ \mathbf{K}^{(t)} \leftarrow \mathsf{SG.InputKeyGen}(1^\lambda, 1^{\ell_x})\end{array} \tag{5.15}$$

$$\left.\begin{array}{r}\boxed{\mathsf{ct}'^{(t)}_{0,i}} := H(\mathbf{l}_2^{(t)}[i]) + \mathbf{K}^{\mathsf{Fn},(t)}[i,0] \\ \boxed{\mathsf{ct}'^{(t)}_{1,i}} := H(\mathbf{l}_1[i] + \mathbf{l}_2^{(t)}[i]) + \mathbf{K}^{\mathsf{Fn},(t)}[i,1]\end{array}\right| \tag{5.16}$$

Hyb_1: Instead of computing $\mathsf{Sel.pp}, \mathsf{Sel.ct}_1$ and $\{\mathsf{Sel.ct}_2^{(t)}, \mathsf{sk}_{\mathbf{f}}^{(t)}\}$ as above (Equation 5.13 and 5.14), Hyb_1 simulates them using $\mathsf{Sel.Sim}$:

$$\left.\begin{array}{r}\boxed{\mathsf{Sel.pp}} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U}) \\ \boxed{\widetilde{\mathsf{Sel.ct}}_1, \{\widetilde{\mathsf{Sel.ct}}_2^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{f}^{(t)}}^{(t)}\}} \\ \leftarrow \mathsf{Sel.Sim}(\mathsf{Sel.pp}, \{\mathbf{l}_{\mathsf{res}}^{(t)}, \mathbf{f}^{(t)}\}).\end{array}\right| \begin{array}{l}\mathbf{l}_1 \leftarrow \mathcal{M}^{s_U} \\ \mathbf{l}_2^{(t)} \leftarrow \mathcal{M}^{s_U} \\ \mathbf{l}_{\mathsf{res}}^{(t)} := \mathbf{l}_1 \odot \mathbf{f}^{(t)} + \mathbf{l}_2^{(t)}\end{array} \tag{5.17}$$

The T-times simulation security of Sel guarantees that the (boxed) simulated terms are indistinguishable from the correctly computed ones. We have $|\Pr[\mathsf{Hyb}_1(\lambda) = 1] - \Pr[\mathsf{Hyb}_0(\lambda)] = 1| \leq \mathrm{negl}(\lambda)$.

Hyb_2: Instead of generating $\mathbf{l}_2^{(t)}$ and then computing $\mathbf{l}_{\mathsf{res}}^{(t)}$ from it, we change Equation 5.17 in such a way that it generates both $\mathbf{l}_1 \leftarrow \mathcal{M}^{s_U}$ and $\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)} \leftarrow \mathcal{M}^{s_U}$ uniformly random. Then, we modify Equation 5.16 s.t. for any $t \in [T]$ and $i \in [s_U]$, the key translation ciphertexts are computed as

$$\boxed{\widetilde{\mathsf{ct}}'^{(t)}_{\mathbf{f}^{(t)}[i],i}} := \mathbf{k}_{\mathbf{f}}^{\mathsf{Fn},(t)} + H(\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}[i]) \quad \Bigg| \quad \mathbf{k}_{\mathbf{f}}^{\mathsf{Fn},(t)} := \mathbf{K}^{\mathsf{Fn},(t)}[i, \mathbf{f}^{(t)}[i]] \tag{5.18}$$

$$\boxed{\mathsf{ct}'^{(t)}_{1-\mathbf{f}^{(t)}[i],i}} := \mathbf{K}^{\mathsf{Fn},(t)}[i, 1-\mathbf{f}^{(t)}[i]] + \begin{cases} H(\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}[i] + \mathbf{l}_1[i]) & \text{if } \mathbf{f}^{(t)}[i] = 0 \\ H(\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}[i] - \mathbf{l}_1[i]) & \text{if } \mathbf{f}^{(t)}[i] = 1 \end{cases} \tag{5.19}$$

Note that the distribution is unchanged, and we have $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_1$.

Hyb_3: Now we can apply the correlation-robustness of H: Instead of computing the "unused" key translation ciphertexts as in Equation 5.19, we simulate it uniformly random:

$$\boxed{\widetilde{\mathsf{ct}}'^{(t)}_{1-\mathbf{f}^{(t)}[i],i}} \leftarrow \mathcal{K}$$

By correlation-robustness of H (which may be applied because $\mathbf{l}_1 \leftarrow \mathcal{M}^{s_U}$ is used nowhere but in the definition of Equation 5.19), the second summand in the definition of $\mathsf{ct}'^{(t)}_{1-\mathbf{f}^{(t)}[i],i}$ can be replaced by uniformly random. This allows us to equivalently replace $\mathsf{ct}'^{(t)}_{1-\mathbf{f}^{(t)}[i],i}$ itself by uniformly random, and therefore we have $|\Pr[\mathsf{Hyb}_3(\lambda) = 1] - \Pr[\mathsf{Hyb}_2(\lambda)] = 1| \leq \mathrm{negl}(\lambda)$.

Hyb_4: Instead of computing $\mathbf{k}_{\mathbf{f}}^{\mathsf{Fn},(t)}$ as in Equation 5.18 and $\widehat{C}_U^{(t)}, \mathbf{k}_{\mathbf{x}}^{(t)}$ as in Equation 5.15, Hyb_4 simulates them using $\mathsf{SG.Sim}_{\mathsf{Priv}}$:

$$\begin{array}{r} \boxed{\{\widetilde{C}_U^{(t)}, \widetilde{\mathbf{k}}_{\mathbf{f}}^{\mathsf{Fn},(t)}, \widetilde{\mathbf{k}}_{\mathbf{x}}^{(t)}\}} \\ \leftarrow \mathsf{SG.Sim}_{\mathsf{Priv}}(\{C_U, \mathbf{y}^{(t)}\}_{t \in [T]}). \end{array} \quad \Bigg| \quad \mathbf{y}^{(t)} = U(f^{(t)}, \mathbf{x}^{(t)})$$

By input privacy (Definition 5.14 simplified for standard garbling, i.e., $T = 1$ and without RDGen and $\mathsf{GarbleFunc}$), this hybrid is indistinguishable from the previous one: $|\Pr[\mathsf{Hyb}_4(\lambda) = 1] - \Pr[\mathsf{Hyb}_3(\lambda)] = 1| \leq \mathrm{negl}(\lambda)$.

By a hybrid argument, we conclude that $|\Pr[\mathsf{Hyb}_4(\lambda) = 1] - \Pr[\mathsf{Hyb}_0(\lambda)] = 1| \leq \mathrm{negl}(\lambda)$, which proves the theorem. □

Efficiency. With our instantiation of batch-select, the function-dependent phase (consisting only of sending $\mathsf{sk_f}$) is succinct: The $\mathsf{hint} = \mathsf{sk_f}$ consists of a single ring element in $\mathcal{R}_q$. The cost of $\widehat{U}$ essentially corresponds to the cost for garbling a universal circuit $\widehat{C}_U$ (e.g. $|\widehat{C}_U| = \lambda{\cdot}n \log n$ to support arbitrary Boolean circuits with at most n gates; see Lemmas 5.12 and 5.13). The *reusable* part $\widehat{U}_{\mathsf{rd}}$ may have size $O(\lambda \cdot n \log^2 n)$, but one factor of $\log n$ can be avoided by using our optimized batch-select constructions (i.e., reusability or *weak* batch-select).

Note regarding Key Translation. The reason for including key translation ciphertexts $\mathsf{ct}'_{0,i}, \mathsf{ct}'_{1,i}$ is the potential mismatch between the batch-select message space $\mathcal{M} = \mathbb{Z}_p^\ell$, and the SG key space $\mathcal{K}$ (which is typically of the form $\{0,1\}^\lambda$). If those two were equal (many existing garbling schemes could theoretically work with any sufficiently large key space, potentially at the cost of not supporting free-XOR anymore), then we may omit key translation ciphertexts. Orthogonally, if no reusability is needed ($T = 1$), we could also directly choose messages $\mathbf{l}_1 := \phi(\mathbf{K}^{\mathsf{Fn}}[1] - \mathbf{K}^{\mathsf{Fn}}[0])$ and $\mathbf{l}_2 := \phi(\mathbf{K}^{\mathsf{Fn}}[0])$ for any injective embedding $\phi : \mathcal{K} \to \mathcal{M}$, avoiding key translation completely despite $\mathcal{K} \neq \mathcal{M}$.

5.8 *Adaptive Security and Application to Malicious 2PC*

The purpose of this section is to formulate a stronger adaptive security for batch-select (Definition 5.19) and show that our batch-select scheme (Construction 14), unmodified, is adaptively secure if the two ingredients $\mathsf{LEnc}, \mathsf{LHE}$ satisfy approprate adaptive security notions (see Sections 5.8.1 and 5.8.2 for definitions and modified constructions). We then sketch how an adaptively secure batch select scheme (Section 5.8.3), together with an adaptive standard garbling scheme, results in an adaptive preprocess garbling scheme (Section 5.8.4).

Finally, in Section 5.8.5 we apply the adaptively secure batch select scheme to construct a T-session malicious 2PC protocol with a function independent preprocessing phase, and T *succinct* function dependent preprocessing phases. Our construction is a natural application of

batch select to the authenticated garbling framework [WRK17, KRRW18, DILO22, CWYY23] for malicious 2PC.

5.8.1 Adaptive Security of LEnc

Definition 5.17 (Adaptive T-noise Leakage Simulation Security of LEnc). *A* LEnc *scheme is* adaptive T-noise leakage simulation secure *w.r.t. a noise distribution* $\overline{\chi}_{\mathsf{LEnc}}$ *if there exist two efficient simulators* $\mathsf{LEnc.Sim}_1, \mathsf{LEnc.Sim}_2$ *such that for any efficient adversary* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{Exp}^{\mathcal{A},0}_{\mathsf{LEnc}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathcal{A},1}_{\mathsf{LEnc}}(\lambda) = 1]\right| \leq \mathrm{negl}(\lambda),$$

where the game $\mathsf{Exp}^{\mathcal{A},0}_{\mathsf{LEnc}}(\lambda)$ *is defined as follows.*

1. $\mathcal{A}(1^\lambda)$ *decides a message dimension* 1^w*, and receives* $\mathsf{pp} \leftarrow \mathsf{LEnc.Setup}(1^\lambda, w)$*.*

2. $\mathcal{A}$ *decides a message vector* $\mathbf{s} \in \mathcal{R}_q^w$ *and receives* ct*, where*

$$\begin{array}{ll} \text{If } b = 0 & (\mathbf{r}, \mathsf{ct}) \leftarrow \mathsf{LEnc.Enc}(\mathbf{s}) \\ \text{If } b = 1 & (\mathsf{ct}, \mathsf{st}) \leftarrow \mathsf{LEnc.Sim}_1(\mathsf{pp}). \end{array}$$

3. *Repeate the following for* $t = 1, \ldots, T$*. In the end* $\mathcal{A}$ *outputs a bit* b' *as the outcome of the game.*

   - $\mathcal{A}$ *decides a database* $\mathbf{a}^{(t)} \in \mathcal{R}_q^w$ *and receives a leakage* $\mathbf{e}^{(t)} + \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}}$*, where*

$$\begin{array}{lll} \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}} \leftarrow \overline{\chi}^w & \boldsymbol{\delta} = \mathsf{LEnc.Eval}(\mathsf{ct}, \mathbf{a}^{(t)}) & d^{(t)}_{\mathbf{a}} = \mathsf{LEnc.Digest}(\mathbf{a}^{(t)}) \\ \text{If } b = 0 & \mathbf{e}^{(t)} = \boldsymbol{\delta} - \mathbf{r} \cdot d^{(t)}_{\mathbf{a}} - \mathbf{s} \odot \mathbf{a} & \\ \text{If } b = 1 & \mathbf{e}^{(t)} \leftarrow \mathsf{LEnc.Sim}_2(\mathsf{st}, \mathbf{a}^{(t)}). & \end{array}$$

Our original LEnc construction already fulfills adaptivity, assuming the stronger *adaptive* error-leakage RingLWE assumption:

Lemma 5.15 (Adaptive security of Construction 15)**.** *Assuming* adaptive $\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi}_{\mathsf{LEnc}},T,\mathcal{L}_{1,2m}(g)}$*, Construction 15 (*$\mathsf{LEnc}$*) fulfills* adaptive simulation security with T-noise leakage *(Definition 5.17).*

Proof. The proof is similar to that of Lemma 5.9. We naturally split the simulator (previously Sim) into two separate simulators Sim_1 and Sim_2 in the following way:

- $\mathsf{Sim}_1(\mathsf{pp})$ samples all ciphertexts purely random

$$\widetilde{\mathbf{C}}_i \leftarrow \mathcal{R}_q^{w\times 2m} \quad \text{for } i = 0, \ldots, \ell-1 \ ,$$

  and outputs ciphertext $\widetilde{\mathsf{ct}} := (\widetilde{\mathbf{C}}_0, \ldots, \widetilde{\mathbf{C}}_{\ell-1})$ and state $\mathsf{st} := \mathsf{pp}$.
- $\mathsf{Sim}_2(\mathsf{st}, \mathbf{a}^{(t)})$ simulates the noise leakages as follows: sample

$$\mathbf{E}_i \leftarrow \chi^{w\times 2m} \quad \text{for } i = 0, \ldots, \ell-1 \ ,$$

  and for each $\mathsf{ind} \in \{0,1\}^\ell$, compute

$$\widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{i=0}^{\ell-1} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} \ .$$

  Return $\widetilde{\mathbf{e}}^{(t)}$.

The hybrids are the same as before, except that instead of $\mathsf{Hyb}_{i^*,j}$ outputting the values $((\mathbf{b}_0, \mathbf{b}_1), (\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1}), \{\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}\}_{[T]})$, we turn it into an interactive game $\mathsf{Hyb}^{\mathcal{D}}_{i^*,j}$ with an adversary $\mathcal{D}$. The adversary first receives $(\mathbf{b}_0, \mathbf{b}_1)$ as in Equation 5.6, then chooses a message vector $\mathbf{s}$ and receives ciphertexts $(\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1})$ as in Equation 5.7, and finally (for each $t \in [T]$) chooses a database $\mathbf{a}^{(t)}$ and receives $\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}$ as in Equation 5.8. In the end it outputs a decision bit.

We now show that $|\Pr[\mathsf{Hyb}^{\mathcal{D}}_{i^*,j} = 1] - \Pr[\mathsf{Hyb}^{\mathcal{D}}_{i^*,j+1} = 1]|$ is negligible for any $i^* \in [\ell]$ and $0 \le j < w$. To do so, we construct an adversary $\mathcal{A}$ for $\mathsf{a\text{-}elLWE}$ as follows.

- First, $\mathcal{A}$ receives public parameters $\mathbf{b}_0, \mathbf{b}_1 \in \mathcal{R}_q^m$ and a RingLWE sample $\mathbf{y} \in \mathcal{R}_q^{2m}$. Pass $\mathbf{b}_0, \mathbf{b}_1 \in \mathcal{R}_q^m$ on to $\mathcal{D}$ who outputs message vector $\mathbf{s} \in \mathcal{R}_q^w$.

Then, compute ciphertexts $\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1}$ as in $\mathsf{Hyb}_{i^*,j}$, except for the ind-th row of $\mathbf{C}_{i^*}$:

$$\mathbf{C}_{i^*}[\mathsf{ind}] := \mathbf{y}^T + \mathbf{r}_{i^*+1}[\mathsf{ind}] \cdot \left(\overline{\mathsf{ind}_{i^*}} \cdot \mathbf{g}^T \quad \mathsf{ind}_{i^*} \cdot \mathbf{g}^T\right)$$

Send ciphertexts $\mathbf{C}_0, \ldots, \mathbf{C}_{\ell-1}$ to $\mathcal{D}$.

- For each $t \in [T]$: $\mathcal{D}$ outputs a database $\mathbf{a}^{(t)}$. Use these to compute vectors $\mathbf{z}^{(t)}_{\mathsf{pre}}$ for any prefix $\mathsf{pre} \in \{0,1\}^\ell$.

  Submit the leakage matrix $(\mathbf{z}^{(t)}_{\mathsf{ind}_{:i^*}})^T$ to obtain leakage $\mathbf{l}[t] \in \mathcal{R}_q$. Compute errors $\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}$ as in $\mathsf{Hyb}_{i^*,j}$, except for

$$\widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] := \sum_{\substack{0 \le i < \ell \\ i \ne i^*}} \mathbf{E}_i[\mathsf{ind}] \cdot \mathbf{z}^{(t)}_{\mathsf{ind}_{:i}} \quad \text{and} \quad \widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}[\mathsf{ind}] := \widetilde{\mathbf{e}}^{(t)}[\mathsf{ind}] + \mathbf{l}[t]\ .$$

  Send $\widetilde{\mathbf{e}}^{(t)}_{\mathsf{res}}$ to $\mathcal{D}$.
- Output the same decision bit as $\mathcal{D}$.

As in the proof of Lemma 5.9, we get $\mathsf{a\text{-}elLWE}^{\mathcal{A},0}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}_{1,2m}(g)} \equiv \mathsf{Hyb}^{\mathcal{D}}_{i^*,j}$ and $\mathsf{a\text{-}elLWE}^{\mathcal{A},1}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}_{1,2m}(g)} \equiv \mathsf{Hyb}^{\mathcal{D}}_{i^*,j+1}$. Thus, by $\mathsf{a\text{-}elLWE}$, we get $\mathsf{Hyb}^{\mathcal{D}}_{i^*,j} \approx_{\mathrm{c}} \mathsf{Hyb}^{\mathcal{D}}_{i^*,j+1}$.

By adaptive leakage-error RingLWE and the hybrid argument, we get $\mathsf{Hyb}_0 \approx_{\mathrm{c}} \mathsf{Hyb}_\ell$, which concludes the security proof. □

5.8.2 Adaptive Security of LHE

Definition 5.18 (Adaptive T-times Simulation Security of LHE)**.** *An* LHE *scheme is* adaptive T-times simulation secure *if there exist three efficient simulators* $\mathsf{LHE.Sim}_1, \mathsf{LHE.Sim}_2, \mathsf{LHE.Sim}_3$ *such that for any efficient adversary* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{Exp}^{\mathcal{A},0}_{\mathsf{LHE}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathcal{A},1}_{\mathsf{LHE}}(\lambda) = 1]\right| \le \mathrm{negl}(\lambda),$$

where the game $\mathsf{Exp}^{\mathcal{A},0}_{\mathsf{LHE}}(\lambda)$ *is defined as follows.*

1. $\mathcal{A}(1^\lambda)$ *decides a message dimension* 1^w*, and receives* $\mathsf{pp} \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^w)$*.*

2. *$\mathcal{A}$ decides the first message vector* $\mathbf{l}_1 \in \mathcal{R}_q^w$ *and receives* ct_1*, where*

$$\text{If } b = 0 \qquad (\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{LHE.Enc}_1(\mathbf{m}_1)$$

$$\text{If } b = 1 \qquad (\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{LHE.Sim}_1(\mathsf{pp}).$$

3. *Repeated the following for* $t = 1, \ldots, T$*. In the end* $\mathcal{A}$ *outputs a bit* b' *as the outcome of the game.*

   - *$\mathcal{A}$ decides a second message vector* $\mathbf{m}_2^{(t)} \in \mathcal{R}_q^w$ *and receives* ct_2*, where*

   $$\text{If } b = 0 \qquad (\mathsf{ct}_2^{(t)}, \mathsf{st}_2^{(t)}) \leftarrow \mathsf{LHE.Enc}_2(\mathbf{m}_2)$$

   $$\text{If } b = 1 \qquad (\mathsf{ct}_2^{(t)}, \mathsf{st}_2^{(t)}) \leftarrow \mathsf{LHE.Sim}_2(\mathsf{st}_1).$$

   - *$\mathcal{A}$ decides an element* $y \in \mathcal{R}_q$ *and receives* $\mathsf{sk}_y^{(t)}$*, where*

   $$\text{If } b = 0 \qquad \mathsf{sk}_y^{(t)} \leftarrow \mathsf{LHE.KeyGen}(\mathsf{st}_1, \mathsf{st}_2^{(t)}, y^{(t)})$$

   $$\text{If } b = 1 \qquad \mathsf{sk}_y^{(t)} \leftarrow \mathsf{LHE.Sim}_3(\mathsf{st}_2^{(t)}, y^{(t)}, \mathbf{m}_{\mathsf{res}}^{(t)}), \quad \mathbf{m}_{\mathsf{res}}^{(t)} = \mathbf{m}_1 \odot y^{(t)} + \mathbf{m}_2^{(t)}.$$

We need to modify our original construction of LHE and add a random oracle H in order to obtain adaptivity:

Construction 19 (Adaptive LHE over rings LHE)**.** $\underline{\mathsf{Setup}(1^\lambda, 1^w) \to \mathsf{pp}}$**:** Output a public random matrix $\mathsf{pp} = \mathbf{a} \leftarrow \mathcal{R}_q^w$.

$\underline{\mathsf{Enc}_1(\mathbf{m}_1) \to \mathsf{ct}_1, \mathsf{st}_1}$**:** Sample RingLWE secrets $\mathbf{s}_1 \leftarrow \mathcal{R}_q^m$ and *truncated* noises $\mathbf{E} \leftarrow \overline{\mathcal{D}}_{\mathcal{R},s}^{w\times m}$, Output a ciphertext

$$\mathsf{ct}_1 := \mathbf{a} \cdot \mathbf{s}_1^T + \mathbf{m}_1 \cdot \mathbf{g}^T + \mathbf{E} \quad \in \mathcal{R}_q^{w\times m},$$

together with state $\mathsf{st}_1 = \mathbf{s}_1$.

$\underline{\mathsf{Enc}_2(\mathbf{m}_2) \to \mathsf{ct}_2, \mathsf{st}_2}$**:** Sample a RingLWE secret $s_2 \leftarrow \mathcal{R}_q$ and *truncated* noises $\bar{\mathbf{e}} \leftarrow \overline{\mathcal{D}}_{\mathcal{R},\bar{s}}^w$. Generate $r \leftarrow \{0,1\}^\lambda$. Output a ciphertext

$$\mathsf{ct}_2 := \mathbf{a} \cdot s_2 + \mathbf{m}_2 + \bar{\mathbf{e}} + H(r) \quad \in \mathcal{R}_q^w,$$

together with state $\mathsf{st}_2 = (\mathbf{s}_2, r)$.

$\underline{\mathsf{KeyGen}(\mathsf{st}_1, \mathsf{st}_2, y) \to \mathsf{sk}'_y}$**:** Parse the states $\mathsf{st}_1 = \mathbf{s}_1 \in \mathcal{R}_q^m$ and $\mathsf{st}_2 = (s_2 \in \mathcal{R}_q, r \in \{0,1\}^\lambda)$.

Output $\mathsf{sk}'_y := (\mathsf{sk}_y, r)$, where sk_y is the decryption key

$$\mathsf{sk}_y := \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y) + s_2 \quad \in \mathcal{R}_q \ .$$

$\underline{\mathsf{Dec}((\mathsf{sk}_y, r), \mathsf{ct}_1, \mathsf{ct}_2, y) \to \mathbf{m}_{\mathsf{res}}}$**:** First combine the ciphertexts into

$$\mathsf{ct}_{\mathsf{res}} := \mathsf{ct}_1 \cdot \mathbf{g}^{-1}(y) + \mathsf{ct}_2 \ - \ H(r) \quad \in \mathcal{R}_q^w \ ,$$
$$// \text{ s.t. } \mathsf{ct}_{\mathsf{res}} = \mathbf{a} \cdot (\mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y) + s_2) + (\mathbf{m}_1 \cdot \mathbf{g}^T \cdot \mathbf{g}^{-1}(y) + \mathbf{m}_2) + \text{noise}$$

Then, use sk_y to decrypt and return $\mathbf{m}_{\mathsf{res}} = \mathsf{ct}_{\mathsf{res}} - \mathbf{a} \cdot \mathsf{sk}_y$, which is supposed to equal $\mathbf{m}_{\mathsf{res}} = \mathbf{m}_1 \cdot y + \mathbf{m}_2 + \text{noise}$.

Lemma 5.16 (Security of Construction 19)**.** *Assuming* $\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}$, *Construction 17 (adaptive LHE) fulfills* adaptive T-times simulation security *(Definition 5.18) in the Random Oracle Model.*

Proof. The proof is analogous to that of Lemma 5.9. The main difference is that we need to additionally handle the term $H(r)$ used to mask the ciphertext ct_2.

The simulator manages the random oracle H. Whenever a query is being made (including queries made by the adversary), the simulator answers consistently, sampling random elements upon each query that was never seen before.

- $\mathsf{Sim}_1(\mathbf{a})$ samples ct_1 uniformly random:

$$\widetilde{\mathsf{ct}}_1 \leftarrow \mathcal{R}_q^{w \times m}$$

- $\mathsf{Sim}_2(\mathsf{st}_1)$ samples $\mathsf{ct}_2^{(t)}$ uniformly random:

$$\widetilde{\mathsf{ct}}_2^{(t)} \leftarrow \mathcal{R}_q^w$$

- $\mathsf{Sim}_3(\mathsf{st}_2^{(t)}, y^{(t)}, \mathbf{m}_{\mathsf{res}}^{(t)})$ first samples the key uniformly random:

$$\widetilde{\mathsf{sk}}_y^{(t)} \leftarrow \mathcal{R}_q$$

Then, it programs the random oracle by choosing $r^{(t)} \leftarrow \{0,1\}^\lambda$ and setting

$$\widetilde{\mathsf{ct}}^{(t)}_{\mathsf{res}} := \mathbf{a} \cdot \mathsf{sk}^{(t)}_y + \mathbf{m}^{(t)}_{\mathsf{res}} + (\mathbf{E} \cdot \mathbf{g}^{-1}(y^{(t)}) + \overline{\mathbf{e}}^{(t)}) \quad \forall t \in [T]$$
$$H(r^{(t)}) := \mathsf{ct}^{(t)}_2 - (\widetilde{\mathsf{ct}}^{(t)}_{\mathsf{res}} - \widetilde{\mathsf{ct}}_1 \cdot \mathbf{g}^{-1}(y^{(t)}))\ .$$

Return $\widetilde{\mathsf{sk}'_y}^{(t)} := (\widetilde{\mathsf{sk}}^{(t)}_y, r^{(t)})$.

Hyb_0 To recall, Hyb_0 generates ciphertexts $\mathsf{ct}_1, \{\mathsf{ct}^{(t)}_2\}$ and decryption keys $\{\mathsf{sk}^{(t)}_y\}$ in the following way:

$$\boxed{\mathbf{a}} \leftarrow \mathcal{R}^m_q \tag{5.20}$$

$$\boxed{\mathsf{ct}_1} := \mathbf{a} \cdot \mathbf{s}^T_1 + \mathbf{m}_1 \cdot \mathbf{g}^T + \mathbf{E} \quad \left| \quad \begin{array}{l} \mathbf{s}_1 \leftarrow \mathcal{R}^m_q \\ \mathbf{E} \leftarrow \overline{\mathcal{D}}^{w \times m}_{\mathcal{R},s} \end{array} \right. \tag{5.21}$$

For $t \in [T]$:

$$\boxed{\mathsf{ct}^{(t)}_2} := (\mathbf{a} \cdot s^{(t)}_2 + \mathbf{m}^{(t)}_2 + \overline{\mathbf{e}}^{(t)}) + H(r^{(t)}) \quad \left| \quad \begin{array}{l} s^{(t)}_2 \leftarrow \mathcal{R}_q \\ \overline{\mathbf{e}}^{(t)} \leftarrow \overline{\mathcal{D}}^{w}_{\mathcal{R},\overline{s}} \\ r^{(t)} \leftarrow \{0,1\}^\lambda \end{array} \right. \tag{5.22}$$

$$\boxed{\mathsf{sk}^{(t)}_y} := \mathbf{s}^T_1 \cdot \mathbf{g}^{-1}(y^{(t)}) + s^{(t)}_2, \boxed{r^{(t)}} \tag{5.23}$$

Hyb_1 As a first step, in Hyb_1 we sample the errors $\mathbf{E}$ and $\overline{\mathbf{e}}^{(t)}$ as Gaussians from $\chi^{w \times m} = \mathcal{D}^{w \times m}_{\mathcal{R},s}$ and $\overline{\chi}^w = \mathcal{D}^w_{\mathcal{R},\overline{s}}$, instead of truncated Gaussians from $\overline{\mathcal{D}}^{w \times m}_{\mathcal{R},s}$ and $\overline{\mathcal{D}}^w_{\mathcal{R},\overline{s}}$, respectively. By Lemma 5.2, we have $\mathsf{Hyb}_1 \approx_{\mathrm{s}} \mathsf{Hyb}_0$.

Hyb_2 Now, we defer choosing the "actual" value of $\mathsf{ct}^{(t)}_2$ (i.e., its value after removing the mask $H(r^{(t)})$) until the adversary has chosen $y^{(t)}$. In particular, we sample $\mathsf{ct}^{(t)}_2$ uniformly at random, and before revealing $r^{(t)}$, we program the random oracle H on input r in such

a way that $\mathsf{ct}_2^{(t)} - H(r^{(t)})$ has the value $\mathbf{a} \cdot s_2^{(t)} + \mathbf{m}_2^{(t)} + \overline{\mathbf{e}}^{(t)}$ as before. Formally,

$$\boxed{\mathsf{ct}_2^{(t)}} \leftarrow \mathcal{R}_q^w$$

$$\boxed{\mathsf{sk}_y^{(t)}} := \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)}, \boxed{r^{(t)}} \quad \left| \quad \begin{array}{l} s_2^{(t)} \leftarrow \mathcal{R}_q \\ r^{(t)} \leftarrow \{0,1\}^\lambda \end{array} \right.$$

Furthermore, before sending $\mathsf{sk}_y^{(t)}$ and $r^{(t)}$ to the adversary, $H(r^{(t)})$ is programmed as follows:

$$H(r^{(t)}) := \mathsf{ct}_2^{(t)} - (\mathbf{a} \cdot s_2^{(t)} + \mathbf{m}_2^{(t)} + \overline{\mathbf{e}}^{(t)}) \quad \left| \quad \overline{\mathbf{e}}^{(t)} \leftarrow \overline{\chi}^w \right.$$

Note that this hybrid Hyb_2 is statistically close to Hyb_1: conditioned on the adversary $\mathcal{A}$ not querying $H(r^{(t)})$ before the programming step is completed, the two worlds are identical. Furthermore, the adversary $\mathcal{A}$ will not query $H(r^{(t)})$ with more than negligible probability, because $r^{(t)}$ is freshly sampled right before programming $H(r^{(t)})$.

Hyb_3 Now we continue in a similar way as in the proof of Lemma 5.11: instead of computing $H(r^{(t)})$ directly from $\mathbf{m}_2^{(t)}$, we first define the *combined* ciphertext $\mathsf{ct}_{\mathsf{res}}^{(t)}$, and then simulate $H(r^{(t)})$ as a combination of $\mathsf{ct}_{\mathsf{res}}^{(t)}$ and ct_1.

More, precisely, $H(r^{(t)})$ is programmed as follows:

$$H(r^{(t)}) := \mathsf{ct}_2^{(t)} - (\widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} - \widetilde{\mathsf{ct}}_1 \cdot \mathbf{g}^{-1}(y^{(t)})) \quad \left| \quad \begin{array}{l} \overline{\mathbf{e}}^{(t)} \leftarrow \overline{\chi}^w \\ \widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} := \mathbf{a} \cdot \mathsf{sk}_y^{(t)} + \mathbf{m}_{\mathsf{res}}^{(t)} + (\mathbf{E} \cdot \mathbf{g}^{-1}(y^{(t)}) + \overline{\mathbf{e}}^{(t)}) \end{array} \right.$$

By definition of $\mathsf{sk}_y^{(t)} = \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)}$ and $\mathbf{m}_{\mathsf{res}}^{(t)} = \mathbf{m}_1 \cdot y^{(t)} + \mathbf{m}_2^{(t)}$, this way of choosing $H(r^{(t)})$ is identical to the previous one. Therefore, we have $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_2$.

Hyb_4 Note that in the previous hybrid, the uniformly random key $s_2^{(t)}$ is used nowhere but in the definition of $\mathsf{sk}_y^{(t)} = \mathbf{s}_1^T \cdot \mathbf{g}^{-1}(y^{(t)}) + s_2^{(t)}$. Therefore, in Hyb_2, we may equivalently generate

$$\boxed{\widetilde{\mathsf{sk}}_y^{(t)}} \leftarrow \mathcal{R}_q, \boxed{r^{(t)}} \leftarrow \{0,1\}^\lambda$$

uniformly random instead of computing $\widetilde{\mathsf{sk}}_y^{(t)}$ from $s_2^{(t)}$. We get $\mathsf{Hyb}_4 \equiv \mathsf{Hyb}_3$.

Hyb_5 In this hybrid, we replace the first ciphertext $\mathsf{ct}_1 = \mathbf{a} \cdot \mathbf{s}_1^T + \mathbf{m}_1 \cdot \mathbf{g}^T + \mathbf{E}$ by a uniformly generated vector

$$\widetilde{\mathsf{ct}}_1 \leftarrow \mathcal{R}_q^{w \times m} .$$

We define sub-hybrids $\mathsf{Hyb}_{4,i}$ for $i = 0, \dots, m$, where $\mathsf{Hyb}_{4,i}$ is identical to Hyb_4, except that the first i columns of ct_1 are sampled uniformly random. Note that $\mathsf{Hyb}_4 \equiv \mathsf{Hyb}_{4,0}$ and $\mathsf{Hyb}_{4,m} \equiv \mathsf{Hyb}_5$. Therefore, in order to prove $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_4$, it only remains to show $\mathsf{Hyb}_{3,i-1} \approx_c \mathsf{Hyb}_{3,i}$ for $i = 1, \dots, m$. To do so, assume there exists a distinguisher $\mathcal{D}$ for $\mathsf{Hyb}_{4,i-1}$ and $\mathsf{Hyb}_{4,i}$. We construct an adversary $\mathcal{A}$ for $\mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},T,\mathcal{L}_{1,1}^{\times w}(g)}$ in the following way (where H-queries by $\mathcal{D}$ are handled the same way as by our simulator):

- First, $\mathcal{A}$ receives public parameters $\mathbf{a} \in \mathcal{R}_q^w$ and a RingLWE sample $\mathbf{y} \in \mathcal{R}_q^w$. Pass $\mathsf{pp} := \mathbf{a}$ on to $\mathcal{D}$, who outputs the first message vector $\mathbf{m}_1 \in \mathcal{R}_q^w$.

  Then, compute the first ciphertext ct_1 as in $\mathsf{Hyb}_{4,i-1}$, except for the i-th column, which is chosen to be $\mathbf{y}$. Send ct_1 to $\mathcal{D}$.
- For each $t \in [T]$: $\mathcal{D}$ outputs the second message vector $\mathbf{m}_2^{(t)} \in \mathcal{R}_q^w$.

  Then, we sample $\mathsf{ct}_2^{(t)} \leftarrow \mathcal{R}_q^w$ randomly and send $\mathsf{ct}_2^{(t)}$ to $\mathcal{D}$, who outputs the element $y^{(t)} \in \mathcal{R}_q$.

  Now, submit the leakage matrix

  $$\mathrm{diag}(\mathbf{g}^{-1}(y^{(t)})[i], \dots, \mathbf{g}^{-1}(y^{(t)})[i])$$

  to the challenger, who outputs leakage $\mathbf{l}_t$ (*this is saying that* $\mathbf{l}_t$ *is a leakage of the noise* $\mathbf{E}[:, i]$ *when multiplied with* $\mathbf{g}^{-1}(y^{(t)})[i]$).

  Then, sample $\mathsf{sk}_y^{(t)} \leftarrow \mathcal{R}_q$ and $r^{(t)} \leftarrow \{0,1\}^\lambda$ randomly (as in Hyb_4), and program

  $$H(r^{(t)}) := \mathsf{ct}_2^{(t)} - (\widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} - \widetilde{\mathsf{ct}}_1 \cdot \mathbf{g}^{-1}(y^{(t)})) ,$$

  where $\mathsf{ct}_{\mathsf{res}}^{(t)}$ is computed as

  $$\widetilde{\mathsf{ct}}_{\mathsf{res}}^{(t)} := \mathbf{a} \cdot \mathsf{sk}_y^{(t)} + \mathbf{m}_{\mathsf{res}}^{(t)} + \left(\sum_{i' \in [m] \setminus \{i\}} \mathbf{E}[:, i'] \cdot \mathbf{g}^{-1}(y^{(t)})[i'] + \mathbf{l}_t \right) .$$

Send $(\mathsf{sk}_y^{(t)}, r)$ to $\mathcal{D}$.

- Output the same decision bit as $\mathcal{D}$.

As in the proof of Lemma 5.11, we get $\mathsf{Hyb}_{4,i-1}^{\mathcal{D}} \equiv \mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}^{\mathcal{A},0}$ and $\mathsf{Hyb}_{4,i}^{\mathcal{D}} \equiv \mathsf{a\text{-}elLWE}_{\mathcal{R},q,\chi,\overline{\chi},\mathcal{L}_{T,1}^{\times w}(g)}^{\mathcal{A},1}$. Thus, by $\mathsf{a\text{-}elLWE}$, we get $\mathsf{Hyb}_{4,i-1}^{\mathcal{D}} \approx_{\mathrm{c}} \mathsf{Hyb}_{4,i}^{\mathcal{D}}$.

By the hybrid argument, we get $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_5$. Because Hyb_5 is identical to the simulated world, adaptive T-times simulation security follows. □

5.8.3 Adaptive Security of Batch-Select

We now formalize adaptive T-times simulation security of the batch-select scheme below. In contrast to the selective version (Definition 5.5), where the challenge messages $\mathbf{l}_1, \mathbf{l}_2^{(1)}, \ldots, \mathbf{l}_2^{(T)}$ and selection vectors $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(T)}$ are decided in one-shot, we now allow an adversary to adaptively query messages and selection vectors, and receive corresponding ciphertexts and decryption keys immediately.

Definition 5.19 (Adaptive T-times Simulation Security of Sel)**.** *A* Sel *scheme is* adaptive T-times simulation secure *if there exist three efficient simulators* $\mathsf{Sel.Sim}_1, \mathsf{Sel.Sim}_2, \mathsf{Sel.Sim}_3$ *such that for any efficient adversary* $\mathcal{A}$*,*

$$\left|\Pr[\mathsf{Exp}_{\mathsf{Sel}}^{\mathcal{A},0}(\lambda) = 1] - \Pr[\mathsf{Exp}_{\mathsf{Sel}}^{\mathcal{A},1}(\lambda) = 1]\right| \leq \mathrm{negl}(\lambda),$$

where the game $\mathsf{Exp}_{\mathsf{Sel}}^{\mathcal{A},0}(\lambda)$ *is defined as follows.*

1. *$\mathcal{A}(1^\lambda)$ decides a batch size 1^w, and receives $\mathsf{pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^w)$.*

2. *$\mathcal{A}$ decides the first message vector $\mathbf{l}_1 \in \mathcal{M}^w$ and receives ct_1, where*

$$\begin{aligned} &\textit{If } b = 0 && (\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1) \\ &\textit{If } b = 1 && (\mathsf{ct}_1, \mathsf{st}_1) \leftarrow \mathsf{Sel.Sim}_1(\mathsf{pp}). \end{aligned}$$

3. *Repeated the following for $t = 1, \ldots, T$. In the end $\mathcal{A}$ outputs a bit b' as the outcome of the game.*

- $\mathcal{A}$ *decides a second message vector* $\mathbf{l}_2^{(t)} \in \mathcal{M}^w$ *and receives* ct_2*, where*

$$\text{If } b = 0 \qquad (\mathsf{ct}_2^{(t)}, \mathsf{st}_2^{(t)}) \leftarrow \mathsf{Sel.Enc}_2(\mathbf{l}_2)$$
$$\text{If } b = 1 \qquad (\mathsf{ct}_2^{(t)}, \mathsf{st}_2^{(t)}) \leftarrow \mathsf{Sel.Sim}_2(\mathsf{st}_1).$$

- $\mathcal{A}$ *decides a selection vector* $\mathbf{y}^{(t)} \in \{0,1\}^w$ *and receives* $\mathsf{sk}_{\mathbf{y}}^{(t)}$*, where*

$$\text{If } b = 0 \qquad \mathsf{sk}_{\mathbf{y}}^{(t)} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_1, \mathsf{st}_2^{(t)}, \mathbf{y}^{(t)})$$
$$\text{If } b = 1 \qquad \mathsf{sk}_{\mathbf{y}}^{(t)} \leftarrow \mathsf{Sel.Sim}_3(\mathsf{st}_2^{(t)}, \mathbf{y}^{(t)}, \mathbf{l}_{\mathsf{res}}^{(t)}), \quad \mathbf{l}_{\mathsf{res}}^{(t)} = \mathbf{l}_1 \odot \mathbf{y}^{(t)} + \mathbf{l}_2^{(t)}.$$

It turns out that our original construction, unmodified, is already adaptively secure if the two ingradients $\mathsf{LEnc}, \mathsf{LHE}$ satisfy their respective adaptive security notions (see Section 5.8.1, and 5.8.2).

The proof of the following lemma is analogous to Lemma 5.7 of selective security.

Lemma 5.17 (Adaptive Security of Construction 14)**.** *Assuming that the underlying* LHE *scheme is* adaptively T*-times simulation secure, and the underlying* LEnc *scheme is* adaptively *simulation secure with* T*-noise leakage, our earlier construction for batch select (Construction 14) fulfills* adaptive T*-times simulation security.*

Proof. The simulator is similar to that of Lemma 5.7, but we need to split it into the following three parts:

- $\mathsf{Sel.Sim}_1(\mathsf{pp})$: Run

$$\widetilde{\mathsf{LEnc.ct}}, \mathsf{LEnc.st} \leftarrow \mathsf{LEnc.Sim}_1(\mathsf{pp}_{\mathsf{LEnc}})$$
$$\widetilde{\mathsf{LHE.ct}}_1, \mathsf{LHE.st}_1 \leftarrow \mathsf{LHE.Sim}_1(\mathsf{pp}_{\mathsf{LHE}})$$

  and output ciphertext $\widetilde{\mathsf{ct}}_1 := (\widetilde{\mathsf{LEnc.ct}}, \widetilde{\mathsf{LHE.ct}}_1)$ and state $\mathsf{st}_1 = (\mathsf{LEnc.st}, \mathsf{LHE.st}_1)$.
- $\mathsf{Sel.Sim}_2(\mathsf{st}_1)$: Run

$$\widetilde{\mathsf{LHE.ct}}_2, \mathsf{LHE.st}_2 \leftarrow \mathsf{LHE.Sim}_2(\mathsf{LHE.st}_1)$$

  and output ciphertext $\widetilde{\mathsf{ct}}_2 := \widetilde{\mathsf{LHE.ct}}_2$ and state $\mathsf{st}_2 = (\mathsf{LEnc.st}, \mathsf{LHE.st}_2)$.

- $\mathsf{Sel.Sim}_3(\mathsf{st}_2, \mathbf{y}, \mathbf{l})$: Compute the encoded values $\widehat{\mathbf{y}} := \mathsf{Encode}(\mathbf{y})$ and $\widehat{\mathbf{l}} := \mathsf{Encode}(\mathbf{l})$. Simulate the LEnc noise and evaluate the laconic encryption $\boldsymbol{\delta}$:

$$\widetilde{\mathbf{e}} \leftarrow \mathsf{LEnc.Sim}_2(\mathsf{LEnc.st}, \widehat{\mathbf{y}})$$
$$\boldsymbol{\delta} \leftarrow \mathsf{LEnc.Eval}(\widetilde{\mathsf{LEnc.ct}}, \widehat{\mathbf{y}})$$

Then, simulate the output $\widetilde{\mathsf{res}}'$ of the LHE and accordingly its key $\widetilde{\mathsf{sk}}_{\mathbf{y}}$ as follows, where the digest is computed as $d_{\widehat{\mathbf{y}}} \leftarrow \mathsf{LEnc.Digest}(\widehat{\mathbf{y}})$:

$$\overline{\mathbf{e}} \leftarrow \overline{\chi}_{\mathsf{LEnc}}^{w'}$$
$$\widetilde{\mathsf{res}}' := \boldsymbol{\delta} - \widetilde{\mathbf{e}} + \widehat{\mathbf{l}} \cdot \Delta - \overline{\mathbf{e}}$$
$$\widetilde{\mathsf{sk}}_{\mathbf{y}} \leftarrow \mathsf{LHE.Sim}_3(\mathsf{LHE.st}_2, \widetilde{\mathsf{res}}', d_{\widehat{\mathbf{y}}})$$

Output $\widetilde{\mathsf{sk}}_{\mathbf{y}}$.

We use an analogous series of hybrid experiments that transitions from Hyb_0 (the "real" distribution as defined in Definition 5.5) to Hyb_3 (the "simulated" distribution). We abuse notation to also write Hyb_i as the output of the distribution of the corresponding experiment.

Hyb_0 To recall, Hyb_0 (omitting generation of public parameters pp) computes in the first phase $\mathsf{ct}_1 = (\mathsf{LEnc.ct}, \mathsf{LHE.ct}_1)$, and in the second phase, for each $t \in [T]$, (1) $\mathsf{LHE.ct}_2^{(t)}$, and (2) $\mathsf{sk}_{\mathbf{y}^{(t)}}^{(t)}$, in the following way (where, as in the construction, we define the digests $d_{\widehat{\mathbf{y}}} := \mathsf{LEnc.Digest}(\widehat{\mathbf{y}}^{(t)})$, and encoded values $\widehat{\mathbf{l}}_1 := \mathsf{Encode}(\mathbf{l}_1)$ and $\widehat{\mathbf{l}}_2 := \mathsf{Encode}(\mathbf{l}_2)$):

$$\mathbf{r}, \boxed{\mathsf{LEnc.ct}} \leftarrow \mathsf{LEnc.Enc}(\widehat{\mathbf{l}}_1 \cdot \Delta)$$
$$\boxed{\mathsf{LHE.ct}_1}, \mathsf{st}_1 \leftarrow \mathsf{LHE.Enc}_1(\mathbf{r})$$
$$\text{For } t \in [T]:$$
$$\boxed{\mathsf{LHE.ct}_2^{(t)}}, \mathsf{sk}_2^{(t)} \leftarrow \mathsf{LHE.Enc}_2(\widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)}) \quad \Big| \quad \overline{\mathbf{e}}^{(t)} \leftarrow \overline{\chi}_{\mathsf{LEnc}}^{w'}$$
$$\boxed{\mathsf{sk}_{\mathbf{y}^{(t)}}^{(t)}} \leftarrow \mathsf{LHE.KeyGen}(\mathsf{sk}_1, \mathsf{sk}_2^{(t)}, d_{\widehat{\mathbf{y}}}^{(t)})$$

Hyb_1 We now use the *adaptive* simulation security of LHE to replace generation of $\widetilde{\mathsf{LHE.ct}}_1$, $\{\widetilde{\mathsf{LHE.ct}}_2^{(t)}, \widetilde{\mathsf{sk}}_{\mathbf{y}^{(t)}}^{(t)}\}_{[T]}$ (i.e., the final three lines in Hyb_0) by the following:

$$\boxed{\widetilde{\mathsf{LHE.ct}}_1}, \mathsf{LHE.st}_1 \leftarrow \mathsf{LHE.Sim}_1(\mathsf{LHE.pp})$$

For $t \in [T]$:

$$\boxed{\widetilde{\mathsf{LHE.ct}}_2^{(t)}}, \mathsf{LHE.st}_2^{(t)} \leftarrow \mathsf{LHE.Sim}_2(\mathsf{LHE.st}_1)$$

$$\boxed{\widetilde{\mathsf{sk}}_{\mathbf{y}^{(t)}}^{(t)}} \leftarrow \mathsf{LHE.Sim}(\mathsf{LHE.st}_2^{(t)}, d_{\widehat{\mathbf{y}}}^{(t)}, \mathsf{res}'^{(t)}) \quad \left| \begin{array}{l} \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} \leftarrow \overline{\chi}_{\mathsf{LEnc}}^{w'} \\ \mathsf{res}'^{(t)} \leftarrow \mathbf{r} \cdot d_{\widehat{\mathbf{y}}}^{(t)} + \widehat{\mathbf{l}}_2^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} \end{array} \right.$$

The indistinguishability $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$ follows directly from the adaptive T-times simulation security of LHE.

Hyb_2 In Hyb_2, we now rewrite $\mathsf{res}'^{(t)}$ as

$$\boldsymbol{\delta}^{(t)} \leftarrow \mathsf{LEnc.Eval}(\mathsf{LEnc.ct}, \widehat{\mathbf{y}}^{(t)}) \quad \forall i \in [T]$$
$$\mathbf{e}^{(t)} := \boldsymbol{\delta}^{(t)} - (\mathbf{r} \cdot d_{\widehat{\mathbf{y}}}^{(t)} - (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)})$$
$$\mathsf{res}'^{(t)} \leftarrow \boldsymbol{\delta}^{(t)} - \mathbf{e}^{(t)} + \widehat{\mathbf{l}}^{(t)} \cdot \Delta - \overline{\mathbf{e}}_{\mathsf{LEnc}}^{(t)} \quad \forall t \in [T]$$

Note that $\mathbf{e}^{(t)}$ denotes the LEnc "evaluation error", i.e., the difference between the expected result $\mathbf{r} \cdot d_{\widehat{\mathbf{y}}}^{(t)} - (\widehat{\mathbf{l}}_1 \cdot \Delta) \odot \widehat{\mathbf{y}}^{(t)}$ and the noisy outcome $\boldsymbol{\delta}^{(t)}$.

As in the proof of Lemma 5.7, we have $\mathsf{Hyb}_1 \equiv \mathsf{Hyb}_2$.

Hyb_3 Note that Hyb_2 does not make use of $\widehat{\mathbf{l}}_2^{(t)}$ anymore, but only $\widehat{\mathbf{l}}_1$ (in $\mathsf{LEnc.ct}$ and when computing the LEnc error $\mathbf{e}^{(t)}$). We now use the simulation security of LEnc in order to eliminate the final usages of $\widehat{\mathbf{l}}_1$, by simulating $\mathsf{LEnc.ct}$ and $\mathbf{e}^{(t)}$. Specifically, in Hyb_3, we replace the computation of $\mathbf{r}$ and $\mathsf{LEnc.ct}$ by the simulation

$$\mathsf{LEnc.st}, \widetilde{\mathsf{LEnc.ct}} \leftarrow \mathsf{LEnc.Sim}_1(\mathsf{pp}_{\mathsf{LEnc}}) ,$$

and the computation of the noise $\mathbf{e}^{(t)}$ by the simulation

$$\widetilde{\mathbf{e}}^{(t)} \leftarrow \mathsf{LEnc.Sim}_2(\mathsf{LEnc.st}, \widehat{\mathbf{y}}^{(t)})$$

Because $\mathbf{e}^{(t)}$ is only used as part of the quantity $\mathbf{e}^{(t)} + \overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}}$ (in the definition of $\mathsf{res}'^{(t)}$) with $\overline{\mathbf{e}}^{(t)}_{\mathsf{LEnc}}$ being a fresh noise generated from $\overline{\chi}^{w'}$, we get the indistinguishability $\mathsf{Hyb}_2 \approx_{\mathrm{c}} \mathsf{Hyb}_3$.

Observe that Hyb_3 proceeds identically as the simulated world. By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_{\mathrm{c}} \mathsf{Hyb}_3$, which proves the security. □

5.8.4 Adaptive Security of Preprocessing Garbling

In this section, we sketch a simple way of achieving *adaptive* T-times input privacy for preprocessing garbling in the RO model, assuming a batch-select that fulfills adaptive security as above, and an adaptively secure garbling scheme in the standard model.

First, we need to modify the preprocessing garbling interface (Definition 5.13) slightly: $\mathsf{GarbleFunc}(\mathsf{st}, f) \to \mathsf{hint}, d$ will output some output decoding information d in addition to the hint. Furthermore, $\mathsf{Eval}(f, \widehat{U}_{\mathsf{rd}}, \widehat{U}, \mathsf{hint}, \mathbf{k}_{\mathbf{x}}, d)$ utilizes this decoding information for evaluating the circuit.

Defining Adaptive T-times Input Privacy. In contrast to standard T-times input privacy (Definition 5.14), the adversary can now choose the online descriptions $f^{(t)}$ and inputs $\mathbf{x} \in \{0,1\}^{\ell_x}$ one after another:

1. The adversary only selects $1^{\ell_x}, 1^{\ell_y}$ and the offline function $U \in \mathcal{U}_{\ell_x,\ell_y}$. It receives $\widehat{U}_{\mathsf{rd}}$ and $\{\widehat{U}^{(t)}\}$, generated in the real resp. simulated world as follows:

$$\begin{array}{r} (\widehat{U}_{\mathsf{rd}}, \mathsf{st}_{\mathsf{rd}}) \leftarrow \mathsf{RDGen}(U) \\ \mathbf{K}^{(t)} \leftarrow \mathsf{InputKeyGen}(1^\lambda, 1^{\ell_x}) \\ (\widehat{U}^{(t)}, \mathsf{st}^{(t)}) \leftarrow \mathsf{GarbleU}(\mathsf{st}_{\mathsf{rd}}, \mathbf{K}^{(t)}) \end{array} \quad \text{or} \quad \widehat{U}_{\mathsf{rd}}, \{\widehat{U}^{(t)}\} \leftarrow \mathsf{Sim}_{\mathsf{Priv}}(U)\ .$$

2. Then, for each $t \in [T]$:

   - The adversary chooses the online description $f^{(t)}$, and receives the $\mathsf{hint}^{(t)}$, generated

in the real resp. simulated world as follows:

$$\mathsf{hint}^{(t)}, d^{(t)} \leftarrow \mathsf{GarbleFunc}(\mathsf{st}^{(t)}, f^{(t)}) \quad \text{or} \quad \mathsf{hint}^{(t)} \leftarrow \mathsf{Sim}_{\mathsf{Priv}}(f^{(t)}) \ .$$

- The adversary chooses the input $\mathbf{x}^{(t)} \in \{0,1\}^{\ell_x}$, and receives the input labels $\mathbf{k}_{\mathbf{x}}^{(t)}$ and input decoding d generated in the real resp. simulated world as follows:

$$\mathbf{k}_{\mathbf{x}}^{(t)} = \mathbf{K}^{(t)}[\mathbf{x}^{(t)}] \quad \text{or} \quad \mathbf{k}_{\mathbf{x}}^{(t)}, d^{(t)} \leftarrow \mathsf{Sim}_{\mathsf{Priv}}(U(f^{(t)}, \mathbf{x}^{(t)})) \ .$$

For a *standard* garbling scheme (without separate preprocessing phase or T-reusability) to fulfill adaptive input privacy, the definition above is simplified towards $T = 1$ and skipping the second stage (since the online description $f^{(t)}$ does not contain any information).

Modifying the Preprocessing Garbling Scheme. In the adaptive security game above, note that with our plain preprocessing garbling scheme (Construction 18), the adversary would be able to view the function's input labels $\mathbf{k}_{\mathbf{f}}^{\mathsf{Fn}}$ to the garbling $\widehat{C}_U$ already after selecting the online description $f^{(t)}$. However, it may adaptively choose the input $\mathbf{x}^{(t)}$ after this step. Therefore, we cannot apply (adaptive) SG input privacy to show our scheme secure.

Instead, we will modify Construction 18 slightly. We account for the issue above by hiding $\mathbf{k}_{\mathbf{f}}^{\mathsf{Fn}}$ until the adversary has received the decoding information d in the final step. Specifically, we require the hash function H used for key translation (which previously just needed to fulfill correlation-robustness) to be a Random Oracle, and modify the scheme's algorithms in the following way.

- In GarbleU, we generate an additional $\mathsf{seed} \leftarrow \mathcal{M}^{s_U}$, which is used for computing the key translation ciphertexts

$$\mathsf{ct}'_{0,i} := H(\mathbf{l}_2[i] \ + \ \mathsf{seed}) + \mathbf{K}^{\mathsf{Fn}}[i,0]$$
$$\mathsf{ct}'_{1,i} := H(\mathbf{l}_1[i] + \mathbf{l}_2[i] \ + \ \mathsf{seed}) + \mathbf{K}^{\mathsf{Fn}}[i,1]$$

- GarbleFunc additionally outputs the decoding information, which is exactly the seed: $d := \mathsf{seed}$.

- Eval replaces its computation of the function's input labels by

$$\mathbf{k}_{\mathbf{f}}^{\mathsf{Fn}}[i] := \mathsf{ct}'_{\mathbf{f}[i],i} - H(\mathbf{l}_{\mathsf{res}}[i] \; + \; \mathsf{seed}) \; .$$

Proving Adaptive T-times Input Privacy. In the security game for the scheme described above, the function's input labels $\mathbf{k}_{\mathbf{f}}^{\mathsf{Fn}}$ to the garbling $\widehat{C}_U$ of the universal circuit are effectively hidden by seed until the adversary also receives the remaining input labels $\mathbf{k}_{\mathbf{x}}$.

We now sketch how to show security, assuming that the underlying scheme SG satisfies adaptive privacy, the batch-select scheme Sel satisfies adaptive security, and H is a programmable random oracle.

The three stages of our simulator will be as follows:

- $\mathsf{Sim}_{\mathsf{Priv}}(U)$ simulates the reusable part of the garbling as $\widehat{U}_{\mathsf{rd}} = (\mathsf{Sel.pp}, \mathsf{Sel.ct}_1)$, where

$$\mathsf{Sel.pp} \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U}), \quad (\mathsf{Sel.st}_1, \widetilde{\mathsf{Sel.ct}_1}) \leftarrow \mathsf{Sel.Sim}_1(\mathsf{Sel.pp}) \; .$$

  It also simulates the non-reusable offline garblings $\widehat{U}^{(t)} := (\widetilde{C}_U^{(t)}, \widetilde{\mathsf{Sel.ct}_2}, \{\widetilde{\mathsf{ct}}_{0,i}'^{(t)}, \widetilde{\mathsf{ct}}_{1,i}'^{(t)}\})$, where

$$\widehat{C}_U^{(t)} \leftarrow \mathsf{Sim}_{\mathsf{Priv}}^{\mathsf{SG}}.\mathsf{Sim}(C_U), \quad (\mathsf{Sel.st}_2^{(t)}, \widetilde{\mathsf{Sel.ct}_2}^{(t)}) \leftarrow \mathsf{Sel.Sim}_2(\mathsf{Sel.st}_1), \quad \widetilde{\mathsf{ct}}_{0,i}'^{(t)}, \widetilde{\mathsf{ct}}_{1,i}'^{(t)} \leftarrow \mathcal{K}$$

- $\mathsf{Sim}_{\mathsf{Priv}}(f^{(t)})$ then uses the final batch-select simulator to simulate the $\mathsf{hint} := \widetilde{\mathsf{sk}}_{\mathbf{f}^{(t)}}^{(t)}$ as

$$\widetilde{\mathsf{sk}}_{\mathbf{f}^{(t)}}^{(t)} \leftarrow \mathsf{Sel.Sim}_3(\mathsf{Sel.st}_2^{(t)}, \mathbf{f}^{(t)}, \widetilde{\mathbf{l}_{\mathsf{res}}}^{(t)}) \; ,$$

  where $\widetilde{\mathbf{l}_{\mathsf{res}}}^{(t)} \leftarrow \mathcal{M}^{s_U}$ is uniformly random.
- $\mathsf{Sim}_{\mathsf{Priv}}(\mathbf{y})$ then runs the standard garbling simulator to obtain the input keys

$$\mathbf{k}_{\mathbf{f}^{(t)}}^{\mathsf{Fn},(t)}, \mathbf{k}_{\mathbf{x}}^{(t)} \leftarrow \mathsf{Sim}_{\mathsf{Priv}}^{\mathsf{SG}}.\mathsf{Sim}(\mathbf{y}) \; .$$

  It programs the random oracle to "fix" the decodable key translation ciphertext:

$$H(\widetilde{\mathbf{l}}_{\mathsf{res}}^{(t)}[i] + \mathsf{seed}) := \mathsf{ct}'^{(t)}_{\mathbf{f}^{\mathsf{Fn},(t)}[i],i} - \mathbf{k}_{\mathbf{f}^{(t)}}^{(t)}[i] \quad \forall i \in [s_U]$$

  for some random $\mathsf{seed} \leftarrow \mathcal{M}^{s_U}$. Then, it outputs input keys $\mathbf{k}_{\mathbf{x}}^{(t)}$ and decoding information $d := \mathsf{seed}$.

The hybrids are very similar to that in the proof of Lemma 5.14.

- First, starting from the real world, we replace the outputs of the batch-select scheme Sel by its simulations. This eliminates the need to know messages $\mathbf{l}_1$ and $\mathbf{l}_2^{(t)}$: simulation of $\mathsf{Sel.ct}_1$ and $\mathsf{Sel.ct}_2$ happens in the first stage of the game, and simulation of $\mathsf{sk}^{(t)}_{\mathbf{f}^{(t)}}$ happens in the second stage, for which only the "visible" messages $\mathbf{l}_{\mathsf{res}} := \mathbf{l}_1 \odot \mathbf{f}^{(t)} + \mathbf{l}_2^{(t)}$ are required.
- Second, we also modify generation of the key translation ciphertexts, in such a way that it only depends on a uniformly random $\mathbf{l}_{\mathsf{res}} \leftarrow \mathcal{M}^{s_U}$ (there is no usage of individual messages $\mathbf{l}_1$ and $\mathbf{l}_2^{(t)}$ anymore):

$$\widetilde{\mathsf{ct}}'^{(t)}_{\mathbf{f}^{(t)}[i],i} := H(\widetilde{\mathbf{l}}^{(t)}_{\mathsf{res}}[i] + \mathsf{seed}) + \widetilde{\mathbf{k}}^{\mathsf{Fn},(t)}_{\mathbf{f}}[i]$$
$$\widetilde{\mathsf{ct}}'^{(t)}_{1-\mathbf{f}^{(t)}[i],i} \leftarrow \mathcal{K}$$

  We can do so because H is a random oracle, and therefore it is unlikely that the adversary will ever query H on two inputs that differ in exactly $\mathbf{l}_1$.
- Now we can use the RO properties to delay choosing the key translation ciphertexts until the final phase where the output $\mathbf{y}$ is known: we generate $\widetilde{\mathsf{ct}}'^{(t)}_{\mathbf{f}^{(t)}[i],i} \leftarrow \mathcal{K}$ uniformly random, and in the final phase we program

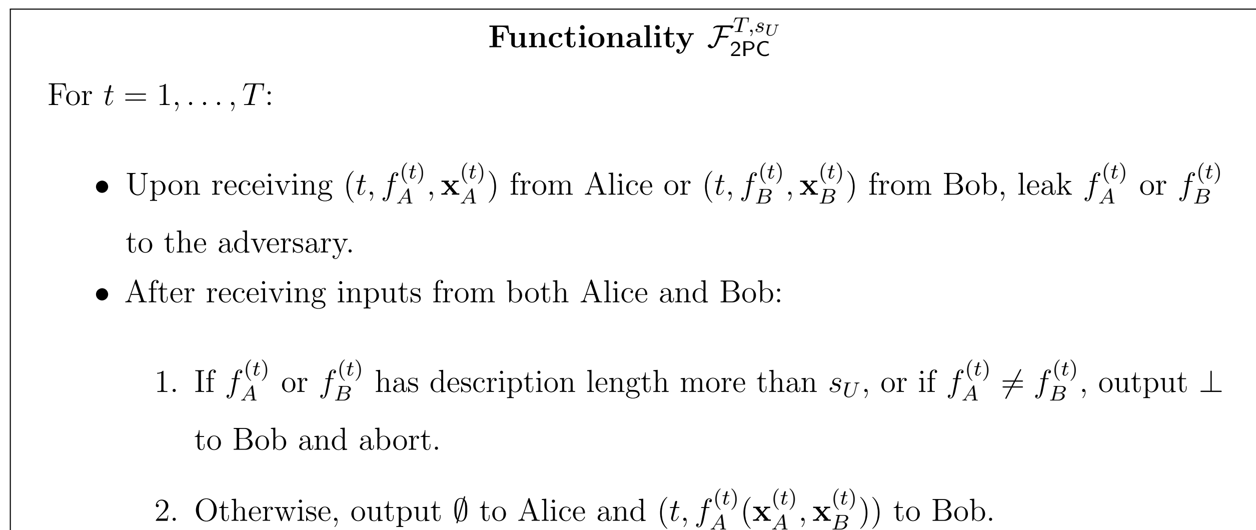
$$H(\widetilde{\mathbf{l}}^{(t)}_{\mathsf{res}}[i] + \mathsf{seed}) := \widetilde{\mathsf{ct}}'^{(t)}_{\mathbf{f}^{(t)}[i],i} - \widetilde{\mathbf{k}}^{\mathsf{Fn},(t)}_{\mathbf{f}}[i]\ .$$

  This is indistinguishable from the previous hybrid, because seed is completely hidden from the adversary, so it is unlikely that it will query $\widetilde{\mathbf{l}}^{(t)}_{\mathsf{res}}[i]+\mathsf{seed}$ before this programming step.
- Finally, the function's input labels $\mathbf{k}^{\mathsf{Fn},(t)}_{\mathbf{f}^{(t)}}$ are not used until the final stage (where the output $\mathbf{y}$ is known) to program H. Therefore, we may use adaptive input privacy of the garbling of the universal circuit to replace $\widehat{C}_U^{(t)}$ (computed in the first stage) and input labels $\mathbf{k}^{\mathsf{Fn},(t)}_{\mathbf{f}^{(t)}}, \mathbf{k}^{(t)}_{\mathbf{x}}$ (computed in the final stage) by

$$\widehat{C}_U^{(t)} \leftarrow \mathsf{Sim}^{\mathsf{SG}}_{\mathsf{Priv}}.\mathsf{Sim}(C_U)$$
$$\mathbf{k}^{\mathsf{Fn},(t)}_{\mathbf{f}^{(t)}}, \mathbf{k}^{(t)}_{\mathbf{x}} \leftarrow \mathsf{Sim}^{\mathsf{SG}}_{\mathsf{Priv}}.\mathsf{Sim}(\mathbf{y})$$

5.8.5 *Preprocessing T-Session Malicious 2PC*

In this section, we construct a protocol, in the random oracle model, realizing the T-session 2PC functionality (Figure 5.1) against malicious adversaries. The protocol has a instance-independent preprocessing phase, where Alice and Bob know only an upper bound s_U of the function discription length, with amortized $O(\lambda \cdot s_U)$ bits of communication. In each session, it has a succinct function dependent preprocessing phase with $\mathrm{poly}(\lambda)$ bits of communication, and an online phase with with $\ell_{xA} + \ell_{xB} + \mathrm{poly}(\lambda)$ bits of communication.



Functionality $\mathcal{F}_{\mathsf{2PC}}^{T,s_U}$

For $t = 1, \ldots, T$:

- Upon receiving $(t, f_A^{(t)}, \mathbf{x}_A^{(t)})$ from Alice or $(t, f_B^{(t)}, \mathbf{x}_B^{(t)})$ from Bob, leak $f_A^{(t)}$ or $f_B^{(t)}$ to the adversary.
- After receiving inputs from both Alice and Bob:

  1. If $f_A^{(t)}$ or $f_B^{(t)}$ has description length more than s_U, or if $f_A^{(t)} \neq f_B^{(t)}$, output $\perp$ to Bob and abort.
  2. Otherwise, output $\emptyset$ to Alice and $(t, f_A^{(t)}(\mathbf{x}_A^{(t)}, \mathbf{x}_B^{(t)}))$ to Bob.

Figure 5.1: The T-session 2PC functionality.

We will first recall and summarize the authenticated garbling framework of [WRK17, DILO22] for malicious 2PC protocols, and then describe how to achieve succinct function dependent and online phases with our *adaptively* secure batch select scheme in the framework.

The Authenticated Garbling Framework. The framework has three main steps, which we summarize below. While the original framework considers evaluating a target circuit C with two private inputs $\mathbf{x}_A, \mathbf{x}_B$ from Alice and Bob, it can be easily extended to support circuits with an additional *public* input $\mathbf{x}_P$ that's only known at the input step together with $\mathbf{x}_A, \mathbf{x}_B$.

Preprocessing: Alice and Bob jointly run a sub-protocol $\mathsf{AuthGarb}^C$ (Figure 5.4, 5.5), parameterized by the target circuit C.

- Alice obtains input keys $\mathbf{K}_P, \mathbf{K}_A, \mathbf{K}_B$, an input mask $\mathbf{a}_I$, and decryption data D.
- Bob obtains a garbled circuit $\overline{C}$ and an input mask $\mathbf{b}_I$.

Input:

1. Bob masks his input $\overline{\mathbf{x}}_B = \mathbf{x}_B \oplus \mathbf{b}_I$, and sends $\overline{\mathbf{x}}_B$ to Alice.
2. Alice masks her input $\overline{\mathbf{x}}_A = \mathbf{x}_A \oplus \mathbf{a}_I$ and selects input labels according to the inputs, $\mathbf{k}_P = \mathbf{K}_P[\mathbf{x}_P]$, $\mathbf{k}_A = \mathbf{K}_A[\overline{\mathbf{x}}_A]$ $\mathbf{k}_B = \mathbf{K}_B[\overline{\mathbf{x}}_B]$. Alice sends $D, \mathbf{k}_P, \mathbf{k}_A, \mathbf{k}_B, \overline{\mathbf{x}}_A$ to Bob.

Output: Bob locally runs an evaluation algorithm $\mathsf{AuthEval}$ (Figure 5.6) and outputs the result $\mathbf{y} \leftarrow \mathsf{AuthEval}^C(\overline{C}, D, \mathbf{k}_P, \mathbf{k}_A, \mathbf{k}_B, \mathbf{x}_P, \overline{\mathbf{x}}_A, \overline{\mathbf{x}}_B)$.

For completeness, we recreate the sub-protocol $\mathsf{AuthGarb}^C$ and the evaluation algorithm $\mathsf{AuthEval}^C$ in Figures 5.4, 5.5 and 5.6.

In [DILO22], the authors showed that the preprocessing step, i.e. the $\mathsf{AuthGarb}$ sub-protocol, can be realized in constant rounds and with $O(|C| \cdot (\lambda + \kappa))$ bits of communication using pseudorandom correlation generators (PCG) for vector oblivious linear evaluation [BCGI18, CRR21] (VOLE) and multiplication triples [BCG+20] (MT) correlations. We refer to [DILO22] for more details.

Minimizing Function Dependent Communication Using Batch Select. The idea is analoguous to how we minimize the function dependent garbling size in our preprocessing garbling construction.

In an instance *independent* offline phase, Alice and Bob run the $\mathsf{AuthGarb}$ sub-protocol on a universal circuit U that takes a function description f (of bounded length s_U) as the public input, and two inputs $\mathbf{x}_A$, $\mathbf{x}_B$ from Alice and Bob.

In the function *dependent* offline phase, it remains for Alice to transmit the input labels selected by the target function descriptoin f to Bob. Our batch select scheme lets Alice achieve this succinctly, with poly(λ) bits. Applying the same idea for the input labels selected by Alice's and Bob's inputs also reduces the online phase communication to poly(λ) bits.

We illustrate the modified protocol below. For simplicity, we assume here a batch select scheme Sel with matching message space to encrypt the input labels.

Instance Independent Offline:

1. Alice and Bob jointly run the sub-protocol $\mathsf{AuthGarb}^U$, w.r.t. a universal circuit U.
   - Alice gets input keys $\mathbf{K}_P, \mathbf{K}_A, \mathbf{K}_B$, an input mask $\mathbf{a}_I$, and decryption information D.
   - Bob gets a garbled circuit $\overline{U}$ and an input mask $\mathbf{b}_I$.
2. Alice encrypts the input keys using batch select as follows, and the decryption information using a random oracle $\mathsf{ct}_O = D \oplus H(\mathsf{seed})$,

$$(\mathsf{st}_{P,1}, \mathsf{Sel.ct}_{P,1}) \leftarrow \mathsf{Sel.Enc}(\mathbf{K}_P[\mathbf{1}] - \mathbf{K}_P[\mathbf{0}]), \quad (\mathsf{st}_{P,2}, \mathsf{Sel.ct}_{P,2}) \leftarrow \mathsf{Sel.Enc}(\mathbf{K}_P[\mathbf{0}]),$$
$$(\mathsf{st}_{I,1}, \mathsf{Sel.ct}_{I,1}) \leftarrow \mathsf{Sel.Enc}(\mathbf{K}_I[\mathbf{1}] - \mathbf{K}_I[\mathbf{0}]), \quad (\mathsf{st}_{I,2}, \mathsf{Sel.ct}_{I,2}) \leftarrow \mathsf{Sel.Enc}(\mathbf{K}_I[\mathbf{0}]),$$

   where $\mathbf{K}_I := \mathbf{K}_A \,\|\, \mathbf{K}_B$, and $\mathsf{seed} \leftarrow \{0,1\}^\lambda$. Alice sends the ciphertexts $\mathsf{Sel.ct}_{P,1}$, $\mathsf{Sel.ct}_{P,2}$, $\mathsf{Sel.ct}_{I,1}$, $\mathsf{Sel.ct}_{I,2}$, ct_O to Bob.

Function Dependent Offline:

1. Alice computes and sends a decryption key $\mathsf{sk}_P \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_{P,1}, \mathsf{st}_{P,2}, f)$ to Bob.
2. Bob locally decrypts input labels for f, $\mathbf{k}_P \leftarrow \mathsf{Sel.Dec}(\mathsf{sk}_P, \mathsf{Sel.ct}_{P,1}, \mathsf{Sel.ct}_{P,2}, f)$.

Online:

1. Bob masks his masked input $\overline{\mathbf{x}}_B = \mathbf{x}_B \oplus \mathbf{b}_I$ to Alice.

2. Alice masks her input $\overline{\mathbf{x}}_A = \mathbf{x}_A \oplus \mathbf{a}_I$ and computes a decryption key accordingly: $\mathsf{sk}_I \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_{I,1}, \mathsf{st}_{I,2}, \overline{\mathbf{x}}_A \,\|\, \overline{\mathbf{x}}_B)$. Alice sends $\overline{\mathbf{x}}_A$, sk_I, seed to Bob.
3. Bob locally decrypts input labels for $\mathbf{k}_A, \mathbf{k}_B$, output information D, and runs the evaluation algorithm to recover the result as in the original framework.

Amortizing the Instance Independent Offline Phase. While achieving succinct instance dependent and online phases, the above solution incurs an overhead to the instance independent phase. First, Alice and Bob runs the sub-protocol $\mathsf{AuthGarb}$ on a universal circuit U, which is at least $O(\log|f|)$ times larger than the actual target circuit f. Second, the first batch select ciphertext $\mathsf{Sel.ct}_{P,1}$ is concretely much larger than the authenticated garbling size of the universal circuit $\overline{U}$. (The cost of sending $\mathsf{Sel.ct}_{P,2}$ and the public parameters $\mathsf{Sel.pp}$ are concretely similar to sending $\overline{U}$.)

Fortunately our batch select scheme allows re-using $\mathsf{Sel.ct}_{P,1}$ upto T times. We leverage this reusability to amortize the communication overhead of sending $\mathsf{Sel.ct}_{P,1}$, and construct a T-session 2PC protocol. When $T = \Omega(\log|f|)$, the amortized function independent preprocessing takes $O(\lambda \cdot |f| \log|f|) = O(\lambda \cdot s_U)$ bits of communication.

We describe our T-session 2PC protocol in Figure 5.2, 5.3. The ingradients to the protocol are:

- the $\mathsf{AuthGarb}$ sub-protocol (Figure 5.4, 5.5), and the evaluation algorithm $\mathsf{AuthEval}$ (Figure 5.6) from the authenticated garbling framework.
- a batch select scheme Sel with message space $\mathcal{M}$ and adaptive T-times simulation security;
- two random oracles $H : \{0,1\}^\lambda \to \{0,1\}^*$, $H' : \mathcal{M} \to \{0,1\}^\lambda$.

The communication costs of the protocol are:

- $O(T \cdot \lambda \cdot s_U)$ bits in the instance independent offline phase;
- $\mathrm{poly}(\lambda)$ in each function dependent offline phase;
- $\ell_{xA} + \ell_{xB} + \mathrm{poly}(\lambda)$ bits in each online phase.

Theorem 5.5. *Let λ be the computational security parameter, and $s_U = \text{poly}(\lambda)$ be an upper bound on target circuits' description length. Assuming the underlying batch select scheme* Sel *has adaptive T-times simulation security, the protocol* $\mathsf{2PC}^{T,s_U}$ *UC-realizes the T-session 2PC functionality $\mathcal{F}_{\mathsf{2PC}}^{T,s_U}$ in the $\mathcal{F}_{\mathsf{pre}}$-hybrid model and in the random oracel (RO) model, the presence of malicious adversaries who statically corrupt one of the participants.*

Before proving Theorem 5.5, we briefly recap the UC framework.

Overview of the Universal Composibility (UC) Framework

The UC framework [Can01] captures the security of the protocol $\mathsf{2PC}^{T,s_U}$ with an ideal functionality $\mathcal{F}_{\mathsf{2PC}}^{T,s_U}$. The functionality defines an ideal protocol execution, where an environment $\mathcal{Z}$ provides inputs to and reads outputs from the two parties. And the parties simply forward their inputs to and receive outputs from $\mathcal{F}_{\mathsf{2PC}}$ as specified in Figure 5.1.

The ideal adversary/simulator Sim lets $\mathcal{F}_{\mathsf{2PC}}$ know of a corrupted party in the beginning, and then interacts with $\mathcal{F}_{\mathsf{2PC}}$ according to the corresponding adversarial interface. In additional to explicitly specified adversarial interfaces, Sim receives all messages sent to and determins all outgoing messages from the corrupted party. The environment $\mathcal{Z}$ communicates with Sim freely throughout the protocol execution.

To prove the security of the protocol $\mathsf{2PC}$, we show that the ideal protocol specified above *emulates* a real protocol execution with an adversary $\mathcal{A}$ controlling a corrutped party, the honest party, and the environment $\mathcal{Z}$. We describe the real protocol execution, and the meaning of emulation below.

In the real protocol execution, the environment $\mathcal{Z}$ provides inputs to and reads outputs from the actual protocol participants. An adversary $\mathcal{A}$ decides a corrupted party in the beginning, and controls them throughout the protocol. The environment $\mathcal{Z}$ also communicates with $\mathcal{A}$ freely throughout the protocol execution.

We say that an ideal protocol execution with an ideal adversary Sim emulates a real protocol execution with an adversary $\mathcal{A}$, if no environment $\mathcal{Z}$ can tell whether it's interacting

$\mathsf{2PC}^{T,s_U}$

Function Independent Offline Phase:

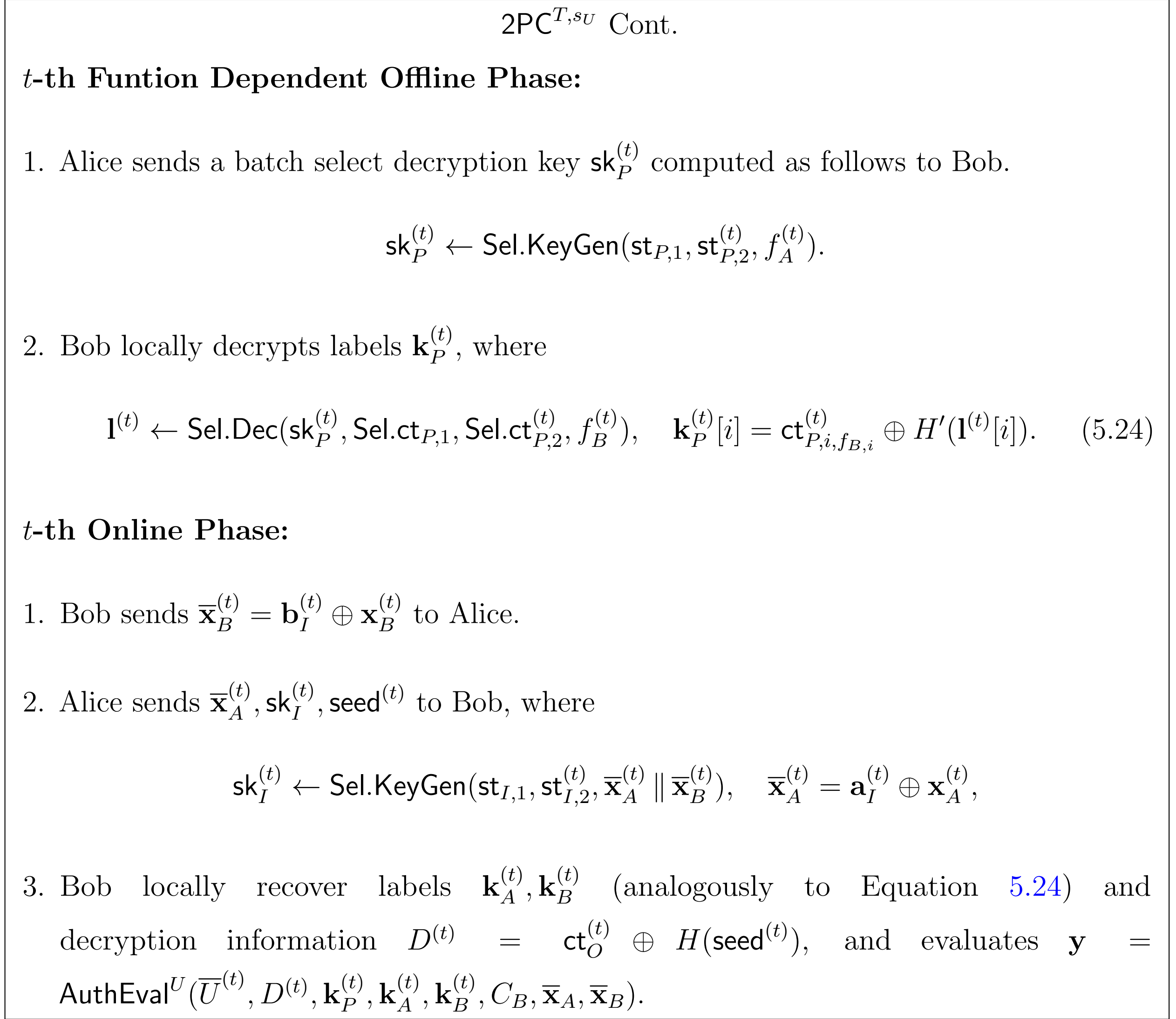
1. Alice and Bob run T instances of the sub-protocol $\mathsf{AuthGarb}^U$ in parallel, where $U : \{0,1\}^{s_U} \times \{0,1\}^{\ell_{xA}} \times \{0,1\}^{\ell_{xB}} \to \{0,1\}^{\ell_y}$ is a universal circuit accepting function descriptions of size bounded by s_U. For each $t \in [T]$:
   - Alice obtains input labels $\mathbf{K}_P^{(t)}, \mathbf{K}_A^{(t)}, \mathbf{K}_B^{(t)}$ masks $\mathbf{a}_I^{(t)}$, and decryption information $D^{(t)}$.
   - Bob obtains authenticated garblings $\overline{U}^{(t)}$ and masks $\mathbf{b}_I^{(t)}$.

2. Alice first encrypts the labels for public inputs into $\mathsf{ct}_P = (\mathsf{Sel.pp}_P,\ \mathsf{ct}_{P,1},\ \{\mathsf{ct}_{P,2}^{(t)}\},\ \{\mathsf{ct}_{P,i,b}^{(t)}\})$:
   - Sample $\mathbf{l}_1 \leftarrow \mathcal{M}^{s_U}$, and $\mathbf{l}_2^{(t)} \leftarrow \mathcal{M}^{s_U}$ for $t \in [T]$, and compute
   $$\forall i \in [s_U], t \in [T] \quad \mathsf{ct}_{P,i,b}^{(t)} = H'(\mathbf{l}_1[i] \cdot b + \mathbf{l}_2^{(t)}[i]) \oplus \mathbf{K}_P^{(t)}[i,b],$$
   - Run $\mathsf{Sel.pp}_P \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U})$ and compute
   $$(\mathsf{st}_{P,1}, \mathsf{Sel.ct}_{P,1}) \leftarrow \mathsf{Sel.Enc}_1(\mathbf{l}_1), \quad (\mathsf{st}_{P,2}^{(t)}, \mathsf{Sel.ct}_{P,2}^{(t)}) \leftarrow \mathsf{Sel.Enc}_2(\mathbf{l}_2^{(t)}) \quad \forall t \in [T].$$

   Alice then analogously encrypts the labels for private inputs into ct_I (with another instance of batch select), and the decryption information into $\mathsf{ct}_O^{(t)} = D^{(t)} \oplus H(\mathsf{seed}^{(t)})$ using random seeds. Alice sends $\mathsf{ct}_P, \mathsf{ct}_I, \{\mathsf{ct}_O^{(t)}\}$ to Bob.

Figure 5.2: Our preprocessing T-session malicious 2PC protocol.

in the ideal or the real protocol. We say the protocol $\mathsf{2PC}^{T,s_U}$ UC-realizes the functionality $\mathcal{F}_{\mathsf{2PC}}^{T,s_U}$ if for all efficient adversary $\mathcal{A}$, there exists an efficient ideal adversary Sim such that the ideal protocol with Sim emulates the real protocol with $\mathcal{A}$.



$\mathsf{2PC}^{T,s_U}$ Cont.

t-th Funtion Dependent Offline Phase:

1. Alice sends a batch select decryption key $\mathsf{sk}_P^{(t)}$ computed as follows to Bob.

$$\mathsf{sk}_P^{(t)} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_{P,1}, \mathsf{st}_{P,2}^{(t)}, f_A^{(t)}).$$

2. Bob locally decrypts labels $\mathbf{k}_P^{(t)}$, where

$$\mathbf{l}^{(t)} \leftarrow \mathsf{Sel.Dec}(\mathsf{sk}_P^{(t)}, \mathsf{Sel.ct}_{P,1}, \mathsf{Sel.ct}_{P,2}^{(t)}, f_B^{(t)}), \quad \mathbf{k}_P^{(t)}[i] = \mathsf{ct}_{P,i,f_{B,i}}^{(t)} \oplus H'(\mathbf{l}^{(t)}[i]). \tag{5.24}$$

t-th Online Phase:

1. Bob sends $\overline{\mathbf{x}}_B^{(t)} = \mathbf{b}_I^{(t)} \oplus \mathbf{x}_B^{(t)}$ to Alice.

2. Alice sends $\overline{\mathbf{x}}_A^{(t)}, \mathsf{sk}_I^{(t)}, \mathsf{seed}^{(t)}$ to Bob, where

$$\mathsf{sk}_I^{(t)} \leftarrow \mathsf{Sel.KeyGen}(\mathsf{st}_{I,1}, \mathsf{st}_{I,2}^{(t)}, \overline{\mathbf{x}}_A^{(t)} \,\|\, \overline{\mathbf{x}}_B^{(t)}), \quad \overline{\mathbf{x}}_A^{(t)} = \mathbf{a}_I^{(t)} \oplus \mathbf{x}_A^{(t)},$$

3. Bob locally recover labels $\mathbf{k}_A^{(t)}, \mathbf{k}_B^{(t)}$ (analogously to Equation 5.24) and decryption information $D^{(t)} = \mathsf{ct}_O^{(t)} \oplus H(\mathsf{seed}^{(t)})$, and evaluates $\mathbf{y} = \mathsf{AuthEval}^U(\overline{U}^{(t)}, D^{(t)}, \mathbf{k}_P^{(t)}, \mathbf{k}_A^{(t)}, \mathbf{k}_B^{(t)}, C_B, \overline{\mathbf{x}}_A, \overline{\mathbf{x}}_B)$.

Figure 5.3: Our preprocessing T-session malicious 2PC protocol, continued.

Formally, let $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{2PC}}^{T,s_U},\mathsf{Sim},\mathcal{Z}}$ and $\mathsf{Real}_{\mathsf{2PC}^{T,s_U},\mathcal{A},\mathcal{Z}}$ denotes the output of an environment after interacting in the ideal and the real protocol. We require that for all efficient $\mathcal{A}$, there exists an efficient Sim such that for all efficient $\mathcal{Z}$:

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{2PC}}^{T,s_U},\mathsf{Sim},\mathcal{Z}} \approx_c \mathsf{Real}_{\mathsf{2PC}^{T,s_U},\mathcal{A},\mathcal{Z}}.$$

The UC framework allows for a modular presentation of protocols, thanks to the universal

composition theorem. Consider an inner protocol π_{in} that UC-realizes an inner functionality $\mathcal{F}_{\mathsf{in}}$, and an outer protocol π_{out} that has access to copies of $\mathcal{F}_{\mathsf{in}}$, i.e., in a $\mathcal{F}_{\mathsf{in}}$-hybrid model, and UC-realizes another outer functionality $\mathcal{F}_{\mathsf{out}}$. The composition theorem ensures the composition of π_{out} and π_{in}, i.e., replacing each copy of $\mathcal{F}_{\mathsf{in}}$ with an instance of π_{i}, still UC-realizes $\mathcal{F}_{\mathsf{out}}$.

Proof of Theorem 5.5

We describe an ideal adversary Sim that externally interacts with the functionality $\mathcal{F}_{\mathsf{2PC}}$ and the environment $\mathcal{Z}$, while internally simulates a protocol execution with an instance of the adversary $\mathcal{A}$. When interacting with $\mathcal{Z}$, Sim simply forwards all communication between $\mathcal{A}$ and $\mathcal{Z}$. We consider separately the case when Alice or Bob is corrupted.

When Bob is Corrupted. Sim proceed as follows.

Instance Independent Offline Phase:

- Sim plays the role of $\mathcal{F}_{\mathsf{pre}}$ with Bob following its description (Figure 5.7). In the process, Sim samples its own global MAC key $\Delta_A \leftarrow \{0,1\}^\lambda$, input masks $\mathbf{a}_I^{(t)}$, and output masks $\{a^{(w,t)}\}_{w\in O}$ and tags $\{M_A^{(w,t)}\}_{w\in O}$. It also receives Bob's global MAC key $\Delta_B \in \{0,1\}^\kappa$ and input masks $\mathbf{b}_I^{(t)}$.
- Sim follows the description of $\mathsf{AuthGarb}$ (Figure 5.4, 5.5) to compute and send garbled tables to Bob. In the process, Sim samples and stores input labels $\mathbf{K}_P^{(t)}, \mathbf{K}_A^{(t)}, \mathbf{K}_B^{(t)}$.
- Sim runs the (stateful) simulator $\mathsf{Sel.Sim}$ to simulate the batch select ciphertexts $\mathsf{ct}_P = (\mathsf{Sel.pp}_P, \mathsf{Sel.ct}_{P,1}, \{\mathsf{Sel.ct}_{P,2}^{(t)}\}, \{\mathsf{ct}_{P,i,b}^{(t)}\})$ for public inputs:

$$\mathsf{Sel.pp}_P \leftarrow \mathsf{Sel.Setup}(1^\lambda, 1^{s_U}), \quad \mathsf{Sel.ct}_{P,1} \leftarrow \mathsf{Sel.Sim}(\mathsf{Sel.pp}_P),$$
$$\mathsf{Sel.ct}_{P,2}^{(t)} \leftarrow \mathsf{Sel.Sim}() \quad \text{for } t \in [T],$$

and then compute the ciphertexts $\{\mathsf{ct}_{i,b}^{(t)}\}$ honestly:

$$\mathbf{l}_{P,1} \leftarrow \mathcal{M}^{s_U}, \quad \mathbf{l}_{P,2}^{(t)} \leftarrow \mathcal{M}^{s_U} \quad \forall t \in [T]$$
$$\mathsf{ct}_{P,i,b}^{(t)} = H(\mathbf{l}_{P,1}[i] \cdot b + \mathbf{l}_{P,2}^{(t)}[i]) \oplus \mathbf{L}_P^{(t)}[i,b], \quad \forall i \in [s_U], t \in [T]$$

It analogously simulates the ciphertexts ct_I for private inputs, and then the ciphertext for decryption information at random $\mathsf{ct}_O^{(t)} \leftarrow \$$.

It sends honestly computed public parameters $\mathsf{Sel.pp}$, ciphertexts $\{\mathsf{ct}_{i,b}^{(t)}\}$ and the simulated ciphertexts $\mathsf{Sel.ct}_1$, $\{\mathsf{Sel.ct}_2^{(t)}\}$ to Bob.

t-th Function Dependent Offline Phase: In order to simulate the t-th batch select decryption key, Sim waits for the functionality $\mathcal{F}_{\mathsf{2PC}}$ to leak the target circuit $f_A^{(t)}$, and runs the (stateful) simulator $\mathsf{Sel.Sim}$:

$$\mathsf{sk}_P^{(t)} \leftarrow \mathsf{Sel.Sim}(\mathbf{l}_{P,1} \odot f_A^{(t)} + \mathbf{l}_{P,2}^{(t)}).$$

It sends the simulated decryption key $\mathsf{sk}_P^{(t)}$ to Bob.

t-th Online Phase:

- Sim receives masked inputs $\overline{\mathbf{x}}_B$ from Bob, and extracts the actual input $\mathbf{x}_B^{(t)} = \overline{\mathbf{x}}_B \oplus \mathbf{b}_I^{(t)}$. Sim then sends a message $(t, f_A^{(t)}, \mathbf{x}_B^{(t)})$ to the functionality $\mathcal{F}_{\mathsf{2PC}}$.
- Sim simulates the masked inputs from Alice as $\overline{\mathbf{x}}_A = \mathbf{a}_I^{(t)} \oplus \mathbf{0}$, and the batch select decryption key $\mathsf{sk}_I^{(t)}$ as

$$\mathsf{sk}_I^{(t)} \leftarrow \mathsf{Sel.Sim}(\mathbf{l}_{I,1} \odot \overline{\mathbf{x}}_A \,\|\, \overline{\mathbf{x}}_B + \mathbf{l}_{I,2}^{(t)}).$$

- In order to simulate the decryption information $D^{(t)}$, Sim waits for the functionality to reveal the evaluation result $\mathbf{y}^{(t)}$, and program $D^{(t)}$ so that evaluation on the simulated input labels correctly reveals $\mathbf{y}^{(t)}$. Sim first computes a difference vector

$$\delta = f_A(\mathbf{0}, \mathbf{x}_B^{(t)}) \oplus \mathbf{y}^{(t)},$$

and then adjusts Alice's bits $a^{(w,t)}$ and tags $M_A^{(w,t)}$ on output wires: (We abuse notations to write $\delta[w]$ to mean the difference bit in δ corresponding to the output wire $w \in O$.)

$$\forall w \in O, \text{ if } \delta[w] = 1, \quad a^{(w,t)} \leftarrow a^{(w,t)} \oplus 1, \quad M_A^{(w,t)} \leftarrow M_A^{(w,t)} \oplus \Delta_B.$$

Sim sets $D^{(t)} = \{a^{(w,t)}, M_A^{(w,t)}\}_{w \in O}$, samples a random $\mathsf{seed}^{(t)}$, and programs the random oracle $H(\mathsf{seed}^{(t)}) \leftarrow \mathsf{ct}_O^{(t)} \oplus D^{(t)}$. Finally, Sim sends $\overline{\mathbf{x}}_A^{(t)}$, $\mathsf{sk}_I^{(t)}$, and $\mathsf{seed}^{(t)}$ to Bob.

It remains to show the output of any environment $\mathcal{Z}$ in the a real protocol execution is indistinguishable from that in an ideal execution, i.e. $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{2PC}}^{T,s_U},\mathsf{Sim},\mathcal{Z}} \approx_c \mathsf{Real}_{\mathsf{2PC}^{T,s_U},\mathcal{A},\mathcal{Z}}$. We describe a series of hybrid experiments that transitions from $\mathsf{Hyb}_0 = \mathsf{Real}_{\mathsf{2PC}^{T,s_U},\mathcal{A},\mathcal{Z}}$ to $\mathsf{Hyb}_3 = \mathsf{IDEAL}_{\mathcal{F}_{\mathsf{2PC}}^{T,s_U},\mathsf{Sim},\mathcal{Z}}$. We abuse the notation to also write Hyb_i as the output distribution of the environment.

Hyb_0**:** This is the real protocol execution between an honest Alice and a corrupted Bob in the $\mathcal{F}_{\mathsf{pre}}$-hybrid and random oracle model.

Hyb_1**:** Instead of honestly computing the batch select ciphertexts $\mathsf{Sel.ct}_{P,1}, \{\mathsf{Sel.ct}_{P,2}^{(t)}\}$ and decryption keys $\mathsf{Sel.sk}_P^{(t)}$ for public inputs, Alice runs the (stateful) batch select simulator as follows.

- In the instance independent offline phase, run

$$\mathsf{Sel.ct}_{P,1} \leftarrow \mathsf{Sel.Sim}(\mathsf{Sel.pp}_P), \quad \text{for } t \in [T], \mathsf{Sel.ct}_{P,2}^{(t)} \leftarrow \mathsf{Sel.Sim}().$$

- In the function dependent offline phase, run

$$\mathsf{sk}_P^{(t)} \leftarrow \mathsf{Sel.Sim}(\mathbf{I}_{P,1} \odot f_A^{(t)} + \mathbf{I}_{P,2}^{(t)}).$$

The adaptive T-times simulation security of Sel guarantees that the Hyb_1 is computationally indistinguishable from Hyb_0.

Hyb_2**:** Simulate the batch select ciphertexts and decryption keys for private inputs analogously to the previous hybrid:

$$\mathsf{Sel.ct}_{I,1} \leftarrow \mathsf{Sel.Sim}(\mathsf{Sel.pp}_I), \quad \text{for } t \in [T], \mathsf{Sel.ct}_{I,2}^{(t)} \leftarrow \mathsf{Sel.Sim}().$$
$$\mathsf{sk}_I^{(t)} \leftarrow \mathsf{Sel.Sim}(\mathbf{I}_{I,1} \odot \overline{\mathbf{x}}_A \,\|\, \overline{\mathbf{x}}_B + \mathbf{I}_{I,2}^{(t)}).$$

The adaptive T-times simulation security of Sel guarantees that the Hyb_2 is computationally indistinguishable from Hyb_1.

Hyb_3**:** In the instance independent offline phase, instead of computing $\mathsf{ct}_O^{(t)}$ honestly as $\mathsf{ct}_O^{(t)} = D^{(t)} \oplus H(\mathsf{seed}^{(t)})$, sample $\mathsf{ct}_O^{(t)}$ directly at random. Then in the online phase, program the random oracle phase as $H(\mathsf{seed}^{(t)}) \leftarrow \mathsf{ct}_O^{(t)} \oplus D^{(t)}$.

Since $\mathsf{seed}^{(t)}$ is sampled at random, the adversary has only negligable probability of querying $H(\mathsf{seed}^{(t)})$ before obtaining $\mathsf{seed}^{(t)}$ in the online phase. Therefore, Hyb_3 is statistically indistinguishable from Hyb_2.

Hyb_4**:** Alice additionally plays the role of $\mathcal{F}_{\mathsf{pre}}$ which allows her to learn Bob's global MAC key Δ_B and input masks $\mathbf{b}_I^{(t)}$ submitted to $\mathcal{F}_{\mathsf{pre}}$ during the instance independent offline phase.

Then during the t-th online phase, Alice receives a masked input $\overline{\mathbf{x}}_B^{(t)}$ from Bob, and extracts an input $\mathbf{x}_B^{(t)} = \overline{\mathbf{x}}_B^{(t)} \oplus \mathbf{b}_I^{(t)}$. Alice computes $\mathbf{y}^{(t)} = f_A^{(t)}(\mathbf{x}_A, \mathbf{x}_B)$, and program the decryption information $D^{(t)}$ such that the evaluation result of $\mathsf{AuthEval}$ equals $\mathbf{y}^{(t)}$.

Note that Hyb_4 is distributed identically to Hyb_3 because Alice is still computing her masked inputs honestly as $\overline{\mathbf{x}}_A^{(t)} = \mathbf{x}_A^{(t)} \oplus \mathbf{a}_I^{(t)}$, and the programmed decryption information $D^{(t)}$ is the same as the honestly computed.

Hyb_5**:** In the input phase, Alice simulates her masked inputs as $\overline{\mathbf{x}}_A^{(t)} = \mathbf{a}_I^{(t)} \oplus \mathbf{0}$, independent of her actual inputs $\mathbf{x}_A^{(t)}$, and then program the decryption information $D^{(t)}$ such that $\mathsf{AuthEval}$ still evaluates to the correct result $\mathbf{y}^{(t)} = f_A^{(t)}(\mathbf{x}_A, \mathbf{x}_B)$.

Note that this hybrid is distributed identically to $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{2PC}}^{T,s_U},\mathsf{Sim},\mathcal{Z}}$. The fact that Hyb_5 is statistically indistinguishable from Hyb_4 (in the random oracle model) follows from the same proof as Theorem 5.1 in [WRK17]. Hence we omit details here.

When Alice is Corrupted. Sim proceed as follows.

Instance Independent Offline Phase:

- Sim plays the role of $\mathcal{F}_{\mathsf{pre}}$ with Alice following its description. In the process, Sim samples its own input masks $\mathbf{b}_I$, and receives Alice's input masks $\mathbf{a}_I^{(t)}$.
- Sim receives garbled tables and batch select ciphertexts from Alice and store them.

t-th Function Dependent Offline Phase: Sim receives a batch select decryption key $\mathsf{sk}_P^{(t)}$ from Alice, and use it to recover input labels $\mathbf{k}_P^{(t)}$ following the description of 2PC.

t-th Online Phase:

- Sim simulates its masked inputs as $\overline{\mathbf{x}}_B^{(t)} = \mathbf{b}_I \oplus \mathbf{0}$, and sends $\overline{\mathbf{x}}_B$ to Alice.
- Sim receives, besides a masked input $\overline{\mathbf{x}}_A$, a batch select decryption key $\mathsf{sk}_I^{(t)}$ and a $\mathsf{seed}^{(t)}$, and use them to recover input labels $\mathbf{k}_A^{(t)}, \mathbf{k}_B^{(t)}$ and decryption information $D^{(t)}$.
- Sim waits for the functionality $\mathcal{F}_{\mathsf{2PC}}$ to leak the target circuit $f_B^{(t)}$, and then runs AuthEval to check whether the evaluation prcedure aborts.
  - If AuthEval doesn't abort, then extract Alice's input as $\mathbf{x}_A^{(t)} = \overline{\mathbf{x}}_A^{(t)} \oplus \mathbf{a}_I^{(t)}$, and send a message $(t, f_B^{(t)}, \mathbf{x}_A^{(t)})$ to the functionality $\mathcal{F}_{\mathsf{2PC}}$.
  - If AuthEval aborts, then trigger an abort for the functionality $\mathcal{F}_{\mathsf{2PC}}$ by sending a message $(t, \emptyset, \emptyset)$.

The fact that $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{2PC}}^{T,s_U},\mathsf{Sim},\mathcal{Z}}$ is statistically indistinguishable from $\mathsf{Real}_{\mathsf{2PC}^{T,s_U},\mathcal{A},\mathcal{Z}}$ follows from the same proof as Theorem 5.1 in [WRK17]. Hence we omit details here.

5.9 Evaluation

To demonstrate concrete efficiency, we have implemented our batch-select scheme [4] (Construction 14). Here, our primary focus is *computational efficiency* instead of optimal (near-0) offline communication cost that would requires exponential modulus (as described e.g. in Theorem 5.3). Nevertheless, our (amortized) offline cost will be just ≈ 6 bits per 128-bit value transferred to a receiver.

Parameter Setting. We consider the ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^{4096}+1)$ of degree $n = 4096$, and as required by our batch-select scheme, we choose the modulus $q = p \cdot \Delta$ as a product of two primes p, Δ. In order to be able to perform efficient computations, we choose the moduli p and Δ as two different NTT-friendly primes (i.e., $p \equiv \Delta \equiv 1 \bmod 8192$). Using the Chinese Remainder Theorem, this allows us to represent an element of the ring $\mathcal{R}_q$ as one $\mathcal{R}_p$-element and one $\mathcal{R}_\Delta$-element. Multiplication and addition over $\mathcal{R}_q$ can then be performed separately over $\mathcal{R}_p$ and $\mathcal{R}_\Delta$. We choose p as a 50-bit prime (then, three elements of the plaintext space $\mathbb{Z}_p$ will be enough to store one 128-bit key for our garbling application), and Δ as a 59-bit prime (this is the maximum bitlength supported by the SEAL framework [SEA23] we build upon).

For concreteness, suppose we apply Sel to vectors of $w' = 512$ ring elements, i.e., vectors of $w = w' \cdot n = 512 \cdot n = 2^{21}$ elements in the message space $\mathcal{M} = \mathbb{Z}_p$. This means that $\lfloor \frac{2^{21}}{3} \rfloor = 699\,050$ elements in $\mathbb{Z}_p^3$ are supported. (Recall that we are interested in $\mathbb{Z}_p^3$, because its size is larger than 2^λ, and hence we can encrypt 128-bit input labels in it.)

We are now aiming to provide $\lambda = 128$-bit security. As recommended by the homomorphic encryption standard [ACC+19], given the degree-4096 ring $\mathcal{R}_q$ (with $\log q \leq 111$) we assume that the RingLWE assumption $\mathsf{LWE}_{\mathcal{R},\infty,q,\chi^*}$ holds with Gaussian noise of standard deviation $\sigma^* = 8/\sqrt{2\pi} \approx 3.2$, i.e., with error distribution $\chi^* = \mathcal{D}_{\mathcal{R},s^*}$ of parameter $s^* = 8$.

Note that the security of LEnc and LHE (as invoked by our T-reusable batch-select scheme) relies on the elLWE assumption with two different types of leakage sets: $\mathcal{L}_{T,2m}(g)$ and $\mathcal{L}_{1,1}^{\times w'}(g)$.

---

[4]The source code can be found under `https://github.com/MarianDietz/tinylabels`.

To ensure that this follows from our assumption $\mathsf{LWE}_{\mathcal{R},\infty,q,\chi^*}$, we combine Theorem 5.2 with Lemmas 5.3 and 5.4. It guarantees that it suffices to choose parameters s and $\overline{s}$ as follows, where s is used for distributions $\chi = \mathcal{D}_{\mathcal{R},s}$ inside both LHE and LEnc, and $\overline{s}$ is used for both distributions $\overline{\chi}_{\mathsf{LEnc}} = \overline{\mathcal{D}}_{\mathcal{R},\overline{s}}$ (used to hide LEnc noise leakage) and $\overline{\chi} = \mathcal{D}_{\mathcal{R},\overline{s}}$ (used inside LHE):

$$s = 2s^* + 1 + \eta_\epsilon(\mathcal{R}) \quad \text{and} \quad \overline{s} = (s+1) \cdot g \cdot n \cdot \sqrt{2m \cdot T} \, .$$

In the above, there are some tunable parameters: m (used inside our LHE and LEnc constructions, where the gadget vector is determined through its *base* g and *dimension* m with $g^m < q$) affects the ciphertext size and therefore the amount of communication, and T denotes how often ct_1 may be reused. We can freely choose these parameters, as long as correctness is guaranteed, i.e., $\Delta \geq 2\sqrt{\lambda}(g \cdot m \cdot d \cdot s \cdot (\lceil \log_2 w' \rceil + 1) + 2\overline{s})$ still holds given the choices of s and $\overline{s}$ above. For our evaluation, we choose

$$T = 2^{15} = 32\,768 \quad \text{and} \quad m = 4 \, ,$$

and therefore also $g = 2^{28}$.

Communication. A single ring element can be represented with 56 KB (4096 coefficients with 109 bits each). Therefore, our basic batch-select construction (without RO-optimization) achieves the following efficiencies:

- Public parameters pp: $w' + 2m$ ring elements ($\approx$ 29 MB)
- Reusable ciphertext ct_1: $m \cdot w' + 2m \cdot w' \lceil \log_2 w' \rceil$ ring elements ($\approx$ 2.2 GB, or when amortized over $T = 2^{15}$ iterations: 67 KB)
- Non-reusable ciphertext ct_2: w' ring elements ($\approx$ 29 MB)
- Key $\mathsf{sk}_{\mathbf{y}}$: 1 ring element ($\approx$ 56 KB)

The following two optimizations regarding *communication size* may be applied:

- *Weak* laconic encryption would reduce the size of ct_1 to $m \cdot w' + 4m \cdot w'$ ring elements ($\approx$ 571 MB instead of $\approx$ 2.2 GB). However, this would simultaneously require the noise $\overline{s}$ to increase by another factor of $\sqrt{w'}$; hence reducing the maximum number T of uses for ct_1 by a factor of $w' = 512$ (i.e., $T \leq 2^6$).

- The RO optimization (see Section 5.4.4) allows reducing the size of ct_2: Instead of sending w' ring elements, it suffices to send a λ-bit seed, plus $2w'n = 2w$ rejection sampling-bits. This reduces the size of ct_2 to just ≈ 524 KB in our setting. Combining this with amortization over $T = 2^{15}$ iterations, the *total communication cost* per iteration, $|\mathsf{ct}_1|/T + |\mathsf{ct}_2|$, will be only 591 KB (i.e., only ≈ 6.4 bits per transferred element in $\mathbb{Z}_p^3$).

Computation optimizations. Note that naive multiplication of two polynomials stored in coefficient form is very expensive: first, both polynomials need to be converted to NTT form, then they are multiplied component-wise, and then another inverse-NTT needs to be performed. To optimize this, our implementation keeps all polynomials in NTT form whenever possible. Then, each multiplication requires only one component-wise multiplication.

NTT transformations are now only required when generating or removing noise, or when converting the LEnc database from $\mathbf{a} \in \mathcal{R}_p^{w'}$ to the larger modulus $\mathcal{R}_q^{w'}$. In addition—this is the dominating use of NTT transformations in the input-dependent phase consisting of $\mathsf{Sel.KeyGen}$ and $\mathsf{Sel.Dec}$—computing the hash-tree inside $\mathsf{LEnc.Digest}$ requires $4mw'$ forward-NTT and $2w'$ inverse-NTT.

Hardware setup. We implemented our batch-select scheme on top of the SEAL library [SEA23] (which allows us to utilize existing implementations of ring operations), and tested it on a server with an Intel Xeon Platinum 8160M Processor, 2.10GHz. The implementation utilizes only a single core.

We achieve an additional speedup using the Intel HEXL framework [BKS+21], which utilizes the Intel AVX512 instruction set. However, the server we tested on only supports AVX512-DQ instructions, and therefore we can expect that a CPU with the AVX512-IFMA52 instruction set would be able to improve the computation time of NTT operations by another factor of 3.

Evaluation results. Table 5.4 displays the amount of computation time and the communication size on the discussed parameters (while our implementation does not directly support

the RO-optimization, we do not expect the running time of Enc_2 to increase significantly when it generates its output from RO). In Table 5.5, we further split the computation time to demonstrate which operations are causing the highest cost.

Putting these results into context, suppose the batch-select scheme is used for transmitting input keys (of bitlength $\lambda = 128$) for a garbled circuit. Naively sending input labels without batch-select requires sending more than 11 MB in the online phase. On the other hand, batch-select allows a tradeoff: only 56 KB need to be sent (once the input is known), in exchange for $1.87s$ of computation time. Thus, our scheme could outperform the naive baseline whenever the bandwidth is limited by about 45 Mbps.

Also note that our scheme allows for parallelization (because most computation happens independently for each of the $w' = 512$ components). For example, when using 8 cores instead of 1 core, computation time in the input-dependent phase could be as small as $0.24s$, outperforming the naive baseline whenever the bandwidth is limited by 350 Mbps.

	communication	computation
Naive	$\lvert\mathbf{x}\rvert \cdot \lambda$	0
DDH ([AIKW13])	$\lvert\mathbf{x}\rvert + \lvert\mathbf{x}\rvert/\lambda$ $+$ offline: $\lvert\mathbf{x}\rvert \cdot \lambda \cdot \log p$	$(\lvert\mathbf{x}\rvert \cdot \lambda \cdot \log p)$ Mult_G
LWE ([AIKW13])	$\lvert\mathbf{x}\rvert + \lvert\mathbf{x}\rvert/\lambda$ $+$ offline: $\lvert\mathbf{x}\rvert \cdot \text{poly}(\lambda, n, \log q)$	$(\lvert\mathbf{x}\rvert \cdot \lambda \cdot n \cdot \log q)$ Add_q
bilinear DDH ([ABI+23])	$\lvert\mathbf{x}\rvert + \lvert\mathbf{x}\rvert/\lambda$ ★	$(\lvert\mathbf{x}\rvert \cdot \lambda \cdot \log p)$ Mult_G $+ \lvert\mathbf{x}\rvert$ Bilinear ops
iO+SSBH ([ABI+23])	$\lvert\mathbf{x}\rvert + \text{poly}(\log \lvert\mathbf{x}\rvert, \lambda)$	$(\lvert\mathbf{x}\rvert \cdot \text{poly}(\lambda))$ iO-evals
RSA ([ABI+23])	$\lvert\mathbf{x}\rvert + \text{poly}(\lambda)$	$(\lvert\mathbf{x}\rvert \cdot \log \lvert\mathbf{x}\rvert \cdot \log N)$ Mult_N
RingLWE (this work)	$\lvert\mathbf{x}\rvert + \text{poly}(\lambda)$ ★	$(\frac{\lvert\mathbf{x}\rvert \log \lvert\mathbf{x}\rvert}{n})$ $\mathsf{Mult}_{\mathcal{R}_q}$

Constant factors are omitted. $|\mathbf{x}|$ denotes the input length, p is the DDH group order, N is the RSA modulus, n is the RingLWE degree / LWE dimension and q is the (Ring)LWE modulus (which is exponentially large, i.e., $q = 2^{\Theta(\lambda)}$). For computation time, we write Mult to denote the cost of a single multiplication within the ring or modulus indicated in the subscript. When indicated, the scheme requires an additional *offline* step which can be performed without knowing $\mathbf{x}$. We use ★ to denote schemes that require an offline step that only needs to be executed *once*, whose communication therefore vanishes after a sufficiently large number of instances.

We further remark that the costs of the (bilinear) DDH and LWE schemes shown here applies an optimization that splits the input into smaller blocks. See [AIKW13] for details.

Table 5.1: Comparison between techniques for input label compression, in terms of *online communication* and *online computation time*.

amortized $|\mathsf{ct}_1|$	amortization	modulus q
$\lambda \cdot |\mathbf{x}| \log |\mathbf{x}|$	$T = 1$	$2^{O(\lambda)}$
$o(1) \cdot |\mathbf{x}|$	$T = \sqrt{q}$	$\mathrm{poly}(\lambda)$
$\mathrm{negl}(\lambda)$	$T = \sqrt{q}$	$\lambda^{\omega(1)}$

(a) Size of ct_1

size $|\mathsf{ct}_2|$	modulus q	$\mathbf{K}[0]$
$\lambda \cdot |\mathbf{x}|$	$2^{O(\lambda)}$	any
$o(1) \cdot |\mathbf{x}|$	$\mathrm{poly}(\lambda)$	random
0	$2^{2(1+\epsilon')\lambda}$	random

(b) Size of ct_2

We omit constant factors. On the left-hand side, we show the size of $|\mathsf{ct}_1|$ amortized across T instances (with the same offset $\mathbf{K}[1] - \mathbf{K}[0]$), assuming that q fulfills the stated constraint. On the right-hand side, we show the size of $|\mathsf{ct}_2|$ assuming that q fulfills the stated constraint. In the bottom two rows, the encrypted keys $\mathbf{K}[0]$ cannot be chosen arbitrarily but are generated as random within our batch-select scheme (this requires the ROM). Everywhere we assume the RingLWE dimension satisfies $n > \lambda$ and $|\mathbf{x}| = \Omega(n)$.

Table 5.2: The amount of offline communication (split into $|\mathsf{ct}_1|$ and $|\mathsf{ct}_2|$).

	#NTT	#Mult	Time (μs)
garbler (ours)	0.02	0.01	0.90
evaluator (ours)	0.02	0.12	1.77
Yao-style GC (1 Mbps)	sending 16 bytes		128.0
Yao-style GC (10 Mbps)	sending 16 bytes		12.8
Yao-style GC (50 Mbps)	sending 16 bytes		2.56

The computational costs are dominated by the number of NTT and component-wise vector multiplications (both in a ring of dimension 4096 and < 60-bit modulus). We compare these numbers with the amount of time that would be required for sending *uncompressed* input labels $\mathbf{K}[\mathbf{x}]$ in standard Yao-style garbled circuits.

Table 5.3: Concrete online costs (per input bit) of for input length $|\mathbf{x}| \approx 700$K.

AuthGarb^C

The protocol assumes the $\mathcal{F}_{\mathsf{pre}}^{C,\lambda,\kappa}$ functionality defined in Figre 5.7.

Let W be the set of indices of all wires in C, and I_P, I_A, I_B, O be those of public inputs, Alice and Bob's inputs, and output wires. Let $W_\wedge$ be the output wires of all AND gates.

1. Alice and Bob invoke $\mathcal{F}_{\mathsf{pre}}^{(C,\lambda,\kappa)}$.
   - Alice samples a global MAC key $\Delta_A \leftarrow \{0,1\}^\lambda$, bits $a^{(w)} \leftarrow \{0,1\}$, tags $M_A^{(w)} \leftarrow \{0,1\}^\kappa$ for $w \in W$, and sets $a^{(w)} = 0$ for $w \in I_B \cup I_P$.
   - Bob analogously samples $\Delta_B, \{b^{(w)}\}, \{M_A^{(w)}\}$, and sets $b^{(w)} = 0$ for $w \in I_A \cup I_P$.

   They send $(\Delta_A, \{a^{(w)}, M_A^{(w)}\})$ and $(\Delta_B, \{b^{(w)}, M_B^{(w)}\})$ to $\mathcal{F}_{\mathsf{pre}}$ and receive back $(\{K_A^{(w)}\}, \{\widehat{a}^{(w)}, \widehat{M}_A^{(w)}, \widehat{K}_A^{(w)}\})$ and $(\{K_B^{(w)}\}, \{\widehat{b}^{(w)}, \widehat{M}_B^{(w)}, \widehat{K}_B^{(w)}\})$.

2. Alice and Bob locally computes intermediate bits, tags, and MAC keys for each AND gate $(i, j, k, \wedge)$ as follows:
   - Alice computes

$$
\begin{aligned}
\overline{a}_0^{(k)} &= a^{(k)} \oplus \widehat{a}^{(k)}, & \overline{a}_1^{(k)} &= a^{(k)} \oplus \widehat{a}^{(k)} \oplus a^{(i)}, \\
\overline{a}_2^{(k)} &= a^{(k)} \oplus \widehat{a}^{(k)} \oplus a^{(j)}, & \overline{a}_3^{(k)} &= a^{(k)} \oplus \widehat{a}^{(k)} \oplus a^{(i)} \oplus a^{(j)} \oplus 1, \\
\overline{M}_{A,0}^{(k)} &= M_A^{(k)} \oplus \widehat{M}_A^{(k)}, & \overline{M}_{A,1}^{(k)} &= M_A^{(k)} \oplus \widehat{M}_A^{(k)} \oplus M_A^{(i)} \\
\overline{M}_{A,2}^{(k)} &= M_A^{(k)} \oplus \widehat{M}_A^{(k)} \oplus M_A^{(j)}, & \overline{M}_{A,3}^{(k)} &= M_A^{(k)} \oplus \widehat{M}_A^{(k)} \oplus M_A^{(i)} \oplus M_A^{(j)}, \\
\overline{K}_{A,0}^{(k)} &= K_A^{(k)} \oplus \widehat{K}_A^{(k)}, & \overline{K}_{A,1}^{(k)} &= K_A^{(k)} \oplus \widehat{K}_A^{(k)} \oplus K_A^{(i)} \\
\overline{K}_{A,2}^{(k)} &= K_A^{(k)} \oplus \widehat{K}_A^{(k)} \oplus K_A^{(j)}, & \overline{K}_{A,3}^{(k)} &= K_A^{(k)} \oplus \widehat{K}_A^{(k)} \oplus K_A^{(i)} \oplus K_A^{(j)} \oplus \Delta_A.
\end{aligned}
$$

   - Bob analoguously computes $\overline{b}_d^{(k)}, \overline{M}_{B,d}^{(k)}, \overline{K}_{B,d}^{(k)}$ for $d \in [3]$.

Figure 5.4: Sub-protocol AuthGarb, adapted from [WRK17, DILO22].

$\mathsf{AuthGarb}^C$ Cont.

3. Alice compute garbled tables as follows and sends them to Bob.
   - For each wire $w \in W$, sample a random key $L_0^{(w)} \leftarrow \{0,1\}^\lambda$, and set $L_1^{(w)} = L_0^{(w)} \oplus \Delta_A$.
   - For each AND gate $(i, j, k, \wedge)$, compute a garbled table $\mathsf{tb}^{(k)} = (\mathsf{ct}_0^{(k)}, \mathsf{ct}_1^{(k)}, \mathsf{ct}_2^{(k)}, \mathsf{ct}_3^{(k)})$ where for $d \in [3]$

$$\mathsf{ct}_d^{(k)} = H(L_{d_0}^{(i)} \| L_{d_1}^{(j)} \| d) \oplus \left(\overline{a}_d^{(k)} \| \overline{M}_{A,d}^{(k)} \| L_0^{(k)} \oplus \overline{a}_d^{(k)} \cdot \Delta_A \oplus \overline{K}_{A,d}^{(k)} \right)$$

4. Alice outputs
   - the input keys $L_0^{(w)}, L_1^{(w)}$ for $w \in I_P \cup I_A \cup I_B$, organized into $\mathbf{K}_P, \mathbf{K}_A, \mathbf{K}_B$;
   - the bits $a^{(w)}$ for $w \in I_A$ organized as a vector $\mathbf{a}_I$;
   - decryption information $D = \{a^{(w)}, M_A^{(w)}\}_{w \in O}$.

   Bob outputs
   - the authenticated garbling $\widehat{C} = (\Delta_B, \{\mathsf{tb}^{(k)}\}, \{\overline{b}_d^{(k)}, \overline{M}_{B,d}^{(k)}, \overline{K}_{B,d}^{(k)}\})$;
   - the bits $b^{(w)}$ for $w \in I_B$ organized as a vector $\mathbf{b}_I$.

Figure 5.5: Sub-protocol $\mathsf{AuthGarb}$, adapted from [WRK17, DILO22], continued.

$\mathsf{AuthEval}^C(\overline{C}, D, \mathbf{k}_P, \mathbf{k}_A, \mathbf{k}_B, \overline{\mathbf{x}}_P, \overline{\mathbf{x}}_A, \overline{\mathbf{x}}_B)$

Let W be the set of indices of all wires in C, and I_P, I_A, I_B, O be those of public inputs, Alice and Bob's inputs, and output wires. Let $W_\wedge$ be the output wires of all AND gates.

1. Parse the inputs as:

$$\overline{C} = (\Delta_B, \{\mathsf{tb}^{(k)}\}_{k\in W_\wedge}, \{\overline{b}_d^{(k)}, \overline{M}_{B,d}^{(k)}, \overline{K}_{B,d}^{(k)}\}_{k\in W_\wedge, d\in[3]}),$$
$$\mathsf{tb}^{(k)} = \{\mathsf{ct}_d^{(k)}\}_{d\in[3]}, \quad D = \{a^{(w)}, M_A^{(w)}\}_{w\in O}.$$

2. In topological order, for every gate (i, j, k, Type) the algorithm holds $L^{(i)}, L^{(j)}$ and bits $\overline{x}^{(i)}, \overline{x}^{(j)}$, and proceed as follows:

   - If Type is $\oplus$, then compute

   $$L^{(k)} = L^{(i)} \oplus L^{(j)}, \quad \overline{x}^{(k)} = \overline{x}^{(i)} \oplus \overline{x}^{(j)}.$$

   - If Type is $\wedge$, first decrypt the ciphertext $\mathsf{ct}_d^{(k)}$ where $d = 2 \cdot \overline{x}^{(i)} + \overline{x}^{(j)}$:

   $$(\overline{a}_d^{(k)} \| \overline{M}_d^{(k)} \| L_A^{(k)}) = \mathsf{ct}_d^{(k)} \oplus H(L^{(i)} \| L^{(j)} \| d),$$

   Next verify that $\overline{M}_d^{(k)} = \overline{a}_d^{(k)} \cdot \Delta_B \oplus \overline{K}_{B,d}^{(k)}$, and then compute

   $$\overline{x}^{(k)} = \overline{a}_d^{(k)} \oplus \overline{b}_d^{(k)}, \quad L^{(k)} = L_A^{(k)} \oplus \overline{M}_{B,d}^{(k)}.$$

3. In the end, the algorithm holds $\overline{x}^{(w)}$ for every $w \in O$, and proceeds as follows.

   - For every $w \in O$, verify that $M_A^{(w)} = a^{(w)} \cdot \Delta_B \oplus K_B^{(w)}$, and compute $x^{(w)} = \overline{x}^{(w)} \oplus a^{(w)} \oplus b^{(w)}$.
   - Output the values on the output wires as $\mathbf{y}$.

Figure 5.6: Algorithm $\mathsf{AuthEval}$, adapted from [WRK17, DILO22].

Functionality $\mathcal{F}_{\mathsf{pre}}^{(C,\lambda,\kappa)}$

Let W be the set of indices of all wires in C, $I, O \subset W$ be the indices of input and outptu wires, and $W_\wedge$, output wires of AND gates. Further divide I input indices for public inputs I_P, Alice's inputs I_A and Bob's inputs I_B.

- On receiving $(\Delta_A \in \{0,1\}^\lambda, \{a^{(w)} \in \{0,1\}, M_A^{(w)} \in \{0,1\}^\kappa\}_{w \in W})$ from Alice, or $(\Delta_B \in \{0,1\}^\kappa, \{b^{(w)} \in \{0,1\}, M_B^{(w)} \in \{0,1\}^\lambda\}_{w \in W})$ from Bob, store them.
- Once received messages from both Alice and Bob:

1. For every input wire $w \in I_A$, adjust Bob's bit $b^{(w)} = 0$. For every input wire $w \in I_B$, adjust Alice's bit $a^{(w)} = 0$.
2. For every XOR gate $(i, j, k, \oplus)$, adjust the bits and tags on the output wire k:

$$a^{(k)} = a^{(i)} \oplus a^{(j)}, \qquad M_A^{(k)} = M_A^{(i)} \oplus M_A^{(j)},$$
$$b^{(k)} = b^{(i)} \oplus b^{(j)}, \qquad M_B^{(k)} = M_B^{(i)} \oplus M_B^{(j)}.$$

3. For every AND gate $(i, j, k, \wedge)$, sample random additive shares $\widehat{a}^{(k)}, \widehat{b}^{(k)} \in \{0,1\}$ such that

$$\widehat{a}^{(k)} \oplus \widehat{a}^{(k)} = (a^{(i)} \oplus b^{(i)}) \cdot (a^{(j)} \oplus b^{(j)}),$$

and random tags $\widehat{M}_A^{(k)} \leftarrow \{0,1\}^\kappa, \widehat{M}_B^{(k)} \leftarrow \{0,1\}^\lambda$.

4. For every tag, compute the corresponding mac key:

$$\forall w \in W, \qquad K_B^{(w)} = M_A^{(w)} \oplus a^{(w)} \cdot \Delta_B \quad K_A^{(w)} = M_B^{(w)} \oplus b^{(w)} \cdot \Delta_A$$
$$\forall k \in W_\wedge, \qquad \widehat{K}_B^{(k)} = \widehat{M}_A^{(k)} \oplus \widehat{a}^{(k)} \cdot \Delta_B \quad \widehat{K}_A^{(k)} = \widehat{M}_B^{(k)} \oplus \widehat{b}^{(k)} \cdot \Delta_A.$$

Output $(\{K_A^w\}, \{\widehat{a}^{(w)}, \widehat{M}_A^{(w)}, \widehat{K}_A^w\})$ and $(\{K_B^w\}, \{\widehat{b}^{(k)}, \widehat{M}_B^{(k)}, \widehat{K}_B^k\})$ to Alice and Bob respectively.

Figure 5.7: The preprocessing functionality adapted from [WRK17, DILO22].

	Reusable		One-time	Input-dependent	
	Setup	Enc_1	Enc_2	KeyGen	Dec
Conjectured time	$0.13s$	$14.93s$	$0.17s$	$0.44s$	$0.71s$
Time	$0.13s$	$16.65s$	$0.25s$	$0.63s$	$1.24s$
Time / msg	$0.18\mu s$	$23.83\mu s$	$0.35\mu s$	$0.90\mu s$	$1.77\mu s$
Time ($T = 2^{15}$)	$3.8\mu s$	$0.0005s$	"	"	"
	pp	ct_1	ct_2	$\mathsf{sk}_{\mathbf{y}}$	n/a
Size	29MB	2.2GB	29MB	56KB	
Size / msg	42 byte	3.1KB	41 byte	0.08 byte	
Size ($T = 2^{15}$)	885 byte	67KB	"	"	
Size / msg ($T = 2^{15}$)	$<$ 0.01 byte	0.1 byte	"	"	
Size w/ RO opt.			524 KB		
Size / msg w/ RO opt.			0.75 byte		

All *time* rows (except the first one) show the actual times required by our implementation, where *per msg* is the time required on average by each of the $w = 699\,050$ messages (i.e., input labels) encrypted by the batch-select scheme, and $T = 2^{15}$ indicates that we amortize the reusable parts across 2^{15} iterations. The first row shows the computation time that we conjecture for a CPU that has AVX512-IFMA52 instructions. These numbers are estimated using the benchmarks from [BKS+21], where one multiplication requires $1.08\mu s$, one forward NTT requires $5.81\mu s$, and one inverse NTT requires $5.72\mu s$.

Table 5.4: Time cost of the algorithms of batch-select, and sizes of the outputs.

	Add. (per msg)	Mult. (per msg)	NTT (per msg)	CRT (per msg)
Setup	0	0	0	0
Enc_1	$0.46s$ $(0.66\mu s)$	$0.79s$ $(1.13\mu s)$	$1.30s$ $(1.86\mu s)$	0
	118 784 (0.17)	77 824 (0.11)	77 824 (0.11)	
Enc_2	$0.01s$ $(0.01\mu s)$	$0.01s$ $(0.02\mu s)$	$0.03s$ $(0.04\mu s)$	0
	3072 (0.004)	1024 (0.001)	2024 (0.003)	
KeyGen	$0.01s$ $(0.01\mu s)$	$0.05s$ $(0.07\mu s)$	$0.17s$ $(0.24\mu s)$	$0.28s$ $(0.40\mu s)$
	8184 (0.01)	8184 (0.01)	11 254 (0.02)	4092 (0.01)
Dec	$0.09s$ $(0.125\mu s)$	$0.56s$ $(0.801\mu s)$	$0.20s$ $(0.288\mu s)$	$0.28s$ $(0.396\mu s)$
	97 776 (0.14)	87 024 (0.12)	12 278 (0.02)	4092 (0.01)

In parentheses, the time and number required on average for each of the $w = 699\,050$ messages, for the most time-intensive components: (1) the number of polynomial additions (also including subtractions), (2) the number of component-wise multiplications in NTT form, (3) the number of NTT transformations (including both forward and inverse transformations), and (4) the number of CRT decompositions (the number of CRT compositions, ommitted from the table, is lower by a factor of m). In Enc_1, the majority of time is spent on generating noise polynomials.

Table 5.5: Time cost and number of invocations of batch-select.

Part II

HOMOMORPHIC SECRET SHARING (HSS) TECHNIQUES FOR PARTIAL AND STANDARD GARBLING

Chapter 6

ALGEBRAIC HOMOMORPHIC MAC FROM GROUPS ASSUMPTIONS

This chapter is adapted from preprint work [ILL24a].

6.1 Introduction

Diversifying assumptions is a central theme in cryptography. While almost all cryptographic primitives can be built from powerful tools such as indistinguishability obfuscation (iO), in most cases this is an overkill. Excluding iO, lattice-based assumptions have also shown great versatility in implying a wide range of primitives. For many of these primitives, we still do not have constructions from cryptographic groups or other number-theoretic assumptions. In particular, primitives with homomorphic evaluation capabilities, such as homomorphic encryption, attribute-based encryption, homomorphic MACs/signatures, and succinct garbling schemes have state-of-the-art constructions using lattices, while group-based or number-theoretic constructions typically only suffice for more limited functionalities or efficiency features.

This work is motivated by the challenge of narrowing the gap between the provable consequences of iO or lattices and group-based assumptions. Besides the direct benefit of diversifying assumptions, efforts in this direction have often led to other major developments and inspired new techniques that found unexpected applications.

Expanding group-based cryptography. In this work, we provide the first group-based constructions for central cryptographic primitives that were previously only known using iO or lattices. This includes the following:

- Succinct protocols for Conditional Disclosure of Secrets (CDS) [GIKM00] for general

circuits, improving over a previous one-way function based construction for truth tables [ABI+23];

- Succinct protocols for Private Simultaneous Messages (PSM) [FKN94a] and garbling schemes [Yao86] for truth tables and shallow branching programs, providing the first fully succinct group-based constructions for any kind of programs;
- Constrained Pseudo-Random Functions (CPRF) [BW13, KPTZ13, BGI14] for general constraint circuits, overcoming the NC^1 barrier in previous group-based constructions [CMPR23].

For all these primitives, we present the first group-based constructions, under (variants of) standard assumptions. Our new constructions are enabled using Homomorphic MAC (HMAC) with natural and useful algebraic properties – called algebraic HMAC (aHMAC) – that we construct using groups. Along the way, we also significantly improve over previous group-based HMAC schemes without the algebraic property, in particular, removing depth constraint, dispensing the need for sub-exponential hardness and non-black-box usage of cryptography. Let us explain more.

Main Tool: Algebraic Homomorphic MAC. A standard homomorphic MAC, introduced by Gennaro and Wichs [GW13] following Agrawal and Boneh [AB09], enables users to authenticate their data using a (shared) secret key. After receiving data $(m_1, \cdots, m_n)$ together with their MAC tags $(\sigma_1, \cdots, \sigma_n)$, anyone can homomorphically execute a program P over the data and tags to generate a succinct tag that authenticates the output of $P(m_1, \cdots, m_n)$; verification of the output tag can be done without even knowing the original inputs. Similar to homomorphic encryption, HMAC is *composable* in the sense that one can incrementally combine authenticated outputs of partial computations to derive an authenticated output of a larger computation. The key feature of homomorphic MAC is that the tags are *succinct*, of size poly(λ), independent of the computation complexity or even the input length. As such, they enable verifying computations over outsourced data in a communication-succinct way (though verification may take as long as the verified computation).

In this work, we propose an algebraic enhancement, called aHMAC, requiring that the tags of aHMAC (both input and evaluated ones) have the most widely used algebraic form $\mathbf{\Delta} \cdot m + \mathbf{K}$, as in a standard information-theoretic MAC. Here $\mathbf{\Delta}$ is a λ-bit global authentication key and K is a pseudorandom blinding term, that can be derived from the secret MAC key. In particular, given input tags $\{\mathbf{\Delta} \cdot x_i + \mathbf{K}_i\}_i$ and the inputs $\{x_i\}$, homomorphic evaluation of a function f yields output tag $\mathbf{\Delta} \cdot y + \mathbf{K}_f$, where $y = f(\{x_i\})$, and the key $\mathbf{K}_f$ can be computed from the original keys $\{\mathbf{K}_i\}$ according to the function f, independent of the inputs $\{x_i\}$. We show that aHMAC readily implies HMAC (see Section 6.4.6), and the additional algebraic properties are very useful for applications.

Constructions of HMAC: There have been a large body of works on constructing HMAC and its stronger public key variant homomorphic signatures [BF11]; see Table 6.1 (and references in [ACG24]). Not surprisingly (by now), using lattice assumptions or iO together with other standard tools, one can obtain fully composable homomorphic MACs and signatures for all polynomial-sized circuits with succinct tags/signatures [GW13, GVW15b, GU24]. On the "low end," assuming just one-way functions, partially succinct HMACs are known for computing low degree polynomials [CF13], which can further be made fully succinct using groups [CF13, BFR13, CFGN14]. In a more recent work [ACG24], using non-interactive Batch Arguments (BARGs), one can even obtain succinct HMAC for bounded-depth circuits assuming subexponential k-Lin over pairing groups or DDH over non-pairing groups. However, the way BARG is used makes the construction non-black-box and inefficient. The weakness of supporting only bounded-depth circuits and reliance on sub-exponential hardness and non-black-box usage of cryptography highlight gaps between the best known group-based HMAC and constructions using lattices or iO. Another line of works [CFT22, KLVW23, WW24], leveraging pairing-group, circumvents these drawbacks but does not achieve the full functionality of HMAC, since they only permit very limited composability. This leaves open the following questions:

Is there a black-box construction of (non-algebraic) HMACs for circuits from

polynomial hardness of standard group assumptions? Or any construction of aHMAC from groups?

In this work, we answer both questions affirmatively for the stronger notion of aHMAC. More specifically, we present *black-box* constructions of 1) aHMAC for bounded depth circuits based on DDH in prime-order or Paillier groups, or DCR in Paillier groups [Pai99, DJ01], and 2) aHMAC for all circuits, based on their circular variants. This immediately implies similar HMAC constructions under the same assumptions. These constructions can also be instantiated using prime-order groups, assuming DDH for bounded depth circuits or circular power-DDH for all circuits, albeit with (an arbitrarily small) inverse-polynomial error that can be eliminated in the context of applications.

Applications to CDS, PSM, and CPRF. Leveraging the algebraic property of our aHMAC, we are able to significantly extend the reach of group-based assumptions, obtaining succinct CDS for circuits, succinct PSM for truth tables and other simple programs, as well as CPRF for circuits, from the same assumptions enabling our aHMAC. All of these primitives were previously only known under lattices or iO.

6.1.1 Applications

We now give a more detailed account of the above applications.

Conditional Disclosure of Secrets. A k-party Conditional Disclosure of Secret (CDS) protocol [GIKM00] considers a setting where k parties want to disclose a common secret $s \in \{0, 1\}$ to a referee if and only if their respective inputs $x_1, \cdots, x_k$ (known to the referee) satisfy some fixed (public) predicate $P : [N]^k \to \{0, 1\}$. Using randomness, the parties achieve this by sending simultaneous messages to the referee. This notion is equivalent to secret sharing for partite functions and partial garbling schemes [IW14]. CDS has found many interesting applications, including to symmetric private information retrieval protocols [GIKM00], attribute based encryption [GVW15a, Att14, Wee14, IW14], (priced) oblivious transfer [AIR01], and secret sharing [LV18, ABF+19, ABNP20, AN21, ABI+23].

Understanding the minimal communication complexity of CDS has been an active research direction. This question is qualitatively interesting because without security requirements, one bit of communication suffices by simply sending the secret to the referee. This motivates the challenge of *fully succinct CDS*, where the total communication is independent of the description size of the predicate P. For the simplest case of a *truth-table* representation of P, in the information theoretic two-party setting, partially succinct schemes are known with communication $2^{O(\sqrt{n \log n})}$ [LVW18], where $n = \log N$. In the computational setting, it is known that one-way functions are sufficient for $\mathrm{poly}(\lambda, k, \log N)$ communication (poly-logarithmic in the truth-table size N^k) [HK20, ABI+23, Hea24], but again with computational cost that scales polynomially with the input domain size, or equivalently the truth-table representation. For general predicates P expressed as Boolean circuits over the bit-representation of the inputs, succinct and efficient CDS is only known using lattices or iO [BGG+14, HLL23].

We present the first group-based construction of succinct CDS for circuits based on the same assumptions underlying our aHMAC. Interestingly, while aHMACs instantiated using prime-order groups have inverse-polynomial errors, these errors can be eliminated in CDS through correctness and security amplification, giving fully correct and secure instantiation from prime-order groups.

Private Simultaneous Messages. A Private Simultaneous Messages (PSM) protocol [FKN94a, IK97] enables k parties with shared common randomness to securely evaluate a public predicate $P : [N]^k \to \{0, 1\}$ on their *private* inputs $x_1, \cdots, x_k$, by simultaneously sending messages to a referee, revealing only the output $P(x_1, \ldots, x_k)$. For $N = 2$, it is equivalent to decomposable randomized encoding [IK00] and garbling schemes [Yao86, BHR12]. Similar to CDS, the PSM model has been a test bed for understanding the communication overhead of secure computation in a simple non-interactive setting, as well as for yielding secure computation protocols with efficient online communication [AIKW13].

The additional requirement of hiding the inputs from the referee makes PSM more challenging to realize than CDS. Even for the simplest truth-table representation of P, the only known succinct constructions uses lattice [GKP+13b, BGG+14, HLL23] or iO [KLW15,

BCG+18]. Without lattices or iO, the best known protocols are information theoretic and have communication complexity grows polynomially with N: $O(N^{1/2})$ for $k = 2$ [BIKK14] and $O_k(N^{(k-1)/2})$ for infinitely many k [BKN18, AL19]. We obtain a (computationally secure) PSM protocol and garbling schemes with communication complexity $\mathrm{poly}(\lambda, k, \log N)$, polylogarithmic in the truth-table size N^k and computation complexity $\mathrm{poly}(\lambda, N^k)$, based on DDH in prime order groups, or DCR in Paillier groups. This yields exponential improvement in communication compared to the previous state-of-the-art without using lattices or iO. More generally, our PSM protocol can handle functions P represented by shallow branching programs, including DFAs and decision trees, with computational cost that scales polynomially with the program size.

Constrained PRFs. Constrained pseudorandom function (CPRFs) [BW13, KPTZ13, BGI14] allow for delegating "constrained" secret keys that enable evaluating the function at certain authorized inputs – as specified by a constraint predicate $P : \{0,1\}^n \to \{0,1\}$ – while keeping the function value at unauthorized inputs pseudorandom. Here we consider the common scenario where a single constrained key is published for a selectively chosen function–referred to as 1-key selective CPRF. The stronger variants supporting multiple keys and/or achieving adaptive security are only known using heavy tools such as iO.

Assuming one-way functions, the works of [BW13, KPTZ13, BGI14] showed how to construct CPRFs for simple function classes that include prefix-fixing functions and range functions, which through the "puncturing" technique of Sahai and Waters [SW14] become a crucial tool in many iO applications. CPRFs for general functions have other applications, such as, broadcast encryption schemes and identity-based key exchange [BW13].

We obtain CPRFs for general constraint circuits, based on DCR, or DDH plus the small exponent assumption in Paillier groups. Previously, CPRFs for general circuits were only known relying either on lattices [BV15] or multilinear maps [BW13] or iO [HKKW19, BLW17, DKN+20] plus other tools. Group-based CPRFs were limited to NC^1 circuits [GGM84, AMN+18, DKN+20, CMPR23], and our new construction overcomes this NC^1 barrier.

In summary, in this work, we significantly extend the reach of group-based cryptography, by providing the first group-based constructions of several important primitives including succinct CDS for circuits, PSM for truth table, and CPRFs for circuits. We obtain these results through the new aHMAC primitive, an algebraic enhancement of HMAC that is likely to find other applications. Our group-based construction of aHMAC also represents significant improvement over previous group-based HMACs, dispensing with depth constraint, sub-exponential hardness, and non-black-box use of cryptography.

In the body of the paper, we present our succinct PSM and CDS protocols using a unified framework of *garbling schemes* and *partial garbling* schemes, respectively.

6.1.2 Concurrent Work

In an independent and concurrent work, Liu, Wang, Yang, and Yu [LWYY25] constructed a garbling scheme for Boolean circuits that uses 1 bit per gate, based on RLWE or NTRU. There are several major differences between their work and ours. First, we achieve full succinctness, i.e., garbling size independent of the number of gates, whereas their garbled circuits still have linear size dependency. However, they achieve standard input privacy guarantees for general circuits, whereas our schemes either ensure only partial hiding, or only support simple classes of programs. Since succinct garbling was already known under LWE, a major focus of their work is improving concrete efficiency. In contrast, our schemes are based on group assumptions, diversifying the assumptions underlying succinct garbling.

6.2 Technical Overview

6.2.1 Overview of aHMAC Construction

Our Assumptions. The set of assumptions we use in this work includes the standard assumption of DDH in prime-order groups and its adaptation in Paillier groups, DCR in Paillier groups, as well as their circular security variants. We assume familiarity with the DDH assumptions and give a short description of the circular variants of DDH and DCR.

The Circular Power Decisional Diffie Hellman (CP-DDH) assumption (Definition 7.3) asserts that a circular encryption of bits of the secret key s using powers of the secret key is pseudorandom. More precisely, for appropriately sampled group elements g, f and random exponents s and $\{a_i, b_i, c_i\}_i$, the following computational indistinguishability holds:

$$g, g^s, (g^{a_i}, g^{sa_i}, g^{s^2 a_i} \cdot f^{s[i]})_{i \in [\log s]} \qquad \approx_c \qquad g, g^s, (g^{a_i}, g^{b_i}, g^{c_i})_{i \in [\log s]}.$$

The CP-DDH assumption can be postulated over prime-order groups, as well as the Paillier/Damgård-Jurik and class groups, and our constructions can be instantiated using any of these groups. For prime-order groups, CP-DDH holds in the standard generic group model (GGM) [Sho97]; see Section 6.3.4. In particular, our CDS and succinct partial garbling scheme is unconditionally secure in the GGM.

Alternatively, we can replace CP-DDH in the Paillier/Damgård-Jurik groups by assuming key-dependent message (KDM) security of the Damgård-Jurik cryptosystem (Definition 6.10) [BRS03, BHHO08]. The semantic security of this cryptosystem relies on the Decisional Composite Residuosity (DCR) assumption [DJ01]. KDM variants of this assumption are commonly used in prior Paillier-group based HSS works, for example [FGJS17, OSY21, MORS24].[1] We write KDM-DCR as a shorthand.

Theorem 6.1 (Algebraic Homomorphic MAC, Informal; see Theorem 6.5, 6.6)**.** *Assume DDH over prime-order or Paillier groups. There is an aHMAC (with tags of the form $\mathbf{\Delta} \cdot y + \mathbf{K}$) for bounded-depth circuits, where the evaluation key size scales linearly with the circuit depth.*

Under the CP-DDH assumption over prime-order or Paillier groups, or alternatively KDM-DCR over Paillier groups, the aHMAC can support general circuits with fixed $\mathrm{poly}(\lambda)$*-size evaluation key.*

Instantiations using prime-order groups have inverse polynomial $1/poly(\lambda)$ correctness errors, while instantiations using Paillier groups have negligible correctness errors.

---

[1]More concretely, our notion of KDM security assumes that encryptions (under sk) of sk^{-1} are secure. This is strictly weaker than [MORS24], which assumes encryptions of any affine function of $\mathsf{sk}, \mathsf{sk}^{-1}$ are secure, and incomparable with [FGJS17, OSY21], which assume encryptions of bits of sk are secure.

Technical Overview

We provide an overview of our aHMAC construction, which builds heavily on techniques from the homomorphic secret sharing (HSS) literature. HSS [BGI16] is a 2-party analogue of FHE, allowing a local mapping from shares of an input x to *additive* shares of an output $f(x)$. However, group-based HSS cannot be applied directly to construct an aHMAC for two reasons: *i)* known group-based HSS schemes only support limited classes of circuits captured by so-called "RMS programs" and *ii)* in standard group-based HSS schemes, *both* shares depend on the input. In aHMAC, there is an "asymmetry of information": The homomorphic evaluation over tags can depend on inputs to compute the output tag $\Delta \cdot y + \mathbf{K}_f$, but not the derivation of $\mathbf{K}_f$ from the original keys $\{\mathbf{K}_i\}$ (which only depends on f).

The HSS limitation to RMS programs stems from the fact that if two parties hold only additive shares of two values x and y, it is not clear how to "multiply" these shares to obtain shares of the product xy. A key idea in HSS is that if x is an input and the two parties additionally have an appropriate encryption $\mathsf{Enc}_{\mathbf{s}}(x \cdot \mathbf{s})$ of x multiplied by the secret key $\mathbf{s}$, x can be "multiplied" with shares of $(y \cdot \mathbf{s})$ to obtain shares of $(xy \cdot \mathbf{s})$. As a result, the computation of both parties depends on x.

The difficulty with handling general circuits is that there is no encryption for intermediate computation values, only shares of $(x \cdot \mathbf{s})$ and $(y \cdot \mathbf{s})$. A key technical contribution of this work is a method for "multiplying" these shares, when x and y are known to just one party (e.g., the evaluator) while the computation of the other party (e.g., the authenticator) is independent of x, y. Our technique is inspired by the recent work of Meyer, Orlandi, Roy, and Scholl [MORS24], but with crucial differences. The work of [MORS24] designed a way of "multiplying" shares of $(x \cdot \mathbf{s})$ and $(y \cdot \mathbf{s})$, without knowledge of x, y, by revealing to one party auxiliary information that depends on the other party's shares. Our technique eliminate the need for auxiliary information, by leveraging that x, y are known to one party.

We now describe our aHMAC based on the CP-DDH assumption that supports all circuits. This construction can be easily modified to obtain a leveled variant for bounded depth circuits,

from power-DDH (without circular security). To obtain a construction based on standard DDH, we rely on a technique from the recent work [MORS25].

More technically, in aHMAC, a KeyGen algorithm generates a secret key sk and an evaluation key evk. They are distributed to an authenticator and an evaluator respectively. The authenticator can use the secret key sk to compute tags σ_x for inputs x (with an arbitrary unique id). An evaluator can use the evaluation key evk to homomorphically evaluate any Boolean circuit C over the tags σ_x.

$$\begin{array}{ll} \underline{\text{Authenticator}}(\mathsf{sk}): & \underline{\text{Evaluator}}(\mathsf{evk}): \\ \sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id}), & \sigma_z \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \{\sigma_x^{(i)}\}). \end{array}$$

Correctness requires the evaluated tag σ_z to have an algebraic form (hence the name) $\sigma_z = \Delta \cdot z + k_C$ over $\mathbb{Z}$, where $z = C(\mathbf{x})$, Δ is a global secret specified at key generation time, and k_C is a MAC key that can be derived without knowing the authenticated inputs $\mathbf{x}$:

$$k_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \{\mathsf{id}^{(i)}\}), // \text{ s.t. } \sigma_z = \Delta \cdot C(x) + k_C \text{ over } \mathbb{Z}.$$

Security requires the global secret Δ remains hidden to the evaluator. Succinctness requires the tags, including the evaluated ones, have bit-lengths independent of the evaluation circuit C.

Constructing aHMACs from DDLog. The authentication algorithm $\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id})$ is simple. It evaluates a PRF on the id to produce a one-time pad $k_x \leftarrow \mathsf{F}(\mathsf{key}, \mathsf{id})$, and outputs $\sigma_x := \Delta \cdot x + k_x$ over $\mathbb{Z}$. The PRF key key and the global secret Δ are included in sk.

The core of our construction is an evaluation procedure to derive from two algebraic tags σ_x, σ_y to another σ_z while maintaining their algebraic forms:

$$\begin{array}{ll} \underline{\text{Authenticator}}(\mathsf{sk}): & \underline{\text{Evaluator}}(\mathsf{evk}): \\ \text{from } k_x, k_y & \text{from } \sigma_x = \Delta \cdot x + k_x,\ \sigma_y = \Delta \cdot y + k_y, \\ \text{to } k_z, & \text{to } \sigma_z = \Delta \cdot z + k_z, \end{array}$$

for the cases $z = x + y$ and $z = x \cdot y$ over $\mathbb{Z}$. This would give us the algorithms $\mathsf{EvalTag}$ and $\mathsf{EvalKey}$ respectively for evaluating circuits consisting of Add and Mult gates. As we will see,

our evaluation procedure requires the input values x, y to be polynomially bounded. But this still suffices for evaluating any Boolean circuit, which can be implemented via Add, Mult gates while keeping intermediate values bounded by 1.

We first note the easy case, $z = x + y$: setting $\sigma_z := \sigma_x + \sigma_y$, and $k_z := k_x + k_y$ over $\mathbb{Z}$ suffices. We focus on the case of $z = x \cdot y$. Our starting point is the following identity

$$\sigma_x \cdot \sigma_y - \Delta(x \cdot \sigma_y + y \cdot \sigma_x) + (\Delta^2 + \Delta)z = \Delta \cdot z + k_x \cdot k_y,$$

where the RHS would be a tag for z of the desired form, with $k_z = k_x \cdot k_y$. While the terms $\sigma_x \cdot \sigma_y$, $(x \cdot \sigma_y + y \cdot \sigma_x)$, and z are computable by the evaluator, the apparent challenge is to allow the evaluator to multiply Δ, and $(\Delta^2 + \Delta)$ with those terms without leaking Δ. For this, we apply the natural idea of putting Δ in the exponents of a random group (with a generator g) element h, and compute the identity *in the exponents*:

$$\begin{aligned} r \leftarrow \$, \quad h = g^r, \quad h_1 = h^{\Delta}, \quad h_2 = h^{\Delta^2} g^{\Delta}, \\ \implies h^{\sigma_x \cdot \sigma_y} / h_1^{x \cdot \sigma_y + y \cdot \sigma_x} \cdot h_2^z = g^{\Delta \cdot z} \cdot h^{k_x \cdot k_y}. \end{aligned} \tag{6.1}$$

The evaluation key evk consists exactly of those group elements h, h_1, h_2. By a DDH-like assumption, which we call circular-power-DDH (CP-DDH), the terms h, h_1, h_2 together don't leak Δ.

It remains to recover the exponents into an integer satisfying the desired algebraic from. The rich literature of HSS constructions provides two methods. (a) The work of [BGI16, DKK18] present an algorithm, DDLog that works for any cyclic group (of order p) and small exponents $m <$ poly, but fails with 1/poly probability over the choice of a common public randomness R:

$$\forall a \in \langle g \rangle, \quad \Pr_R[\mathsf{DDLog}_g(a \cdot g^m; R) = m + \mathsf{DDLog}_g(a; R) \bmod p] \geq 1 - 1/\text{poly}. \tag{6.2}$$

This will lead to an aHMAC scheme with overall 1/poly correctness error. (b) The work of [ADOS22] introduces a framework of groups (including the Damgård-Jurik groups and class groups) with efficient and perfectly correct DDLog algorithms for certain "easy" subgroups. This will lead to an aHMAC scheme with perfect correctness.

For simplicity, we assume method (a) in this overview. We obtain the following almost correct evaluation procedures:

$$\begin{array}{ll} \underline{\text{Authenticator}}(\mathsf{sk} \ni (h, R), k_x, k_y): & \underline{\text{Evaluator}}(\mathsf{evk} = (h, h_1, h_2, R), \sigma_x, \sigma_y): \\ k_z^* \leftarrow \mathsf{DDLog}_g(h^{k_x \cdot k_y}; R), & a := h^{\sigma_x \cdot \sigma_y} / h_1^{x \cdot \sigma_y + y \cdot \sigma_x} \cdot h_2^z, \\ & \sigma_z^* \leftarrow \mathsf{DDLog}_g(a; R). \end{array}$$

Assuming the term $\Delta \cdot z$ is small, we can apply the DDLog guarantee (although with 1/poly failure probability):

$$\begin{aligned} \sigma^* &= \mathsf{DDLog}_g(a; R) \\ (\text{Eq } 6.1) &= \mathsf{DDLog}_g(g^{\Delta \cdot z} \cdot h^{k_x \cdot k_y}; R) \\ (\text{Eq } 6.2) &\equiv \Delta \cdot z + \mathsf{DDLog}_g(h^{k_x \cdot k_y}; R) \equiv \Delta \cdot z + k_z^* \bmod p. \end{aligned}$$

The procedure as described has two more challenges to resolve:

- The DDLog algorithm from (a) requires $\Delta \cdot z$ to be polynomially bounded, while the global secret Δ needs to be exponentially large in CP-DDH. We resolve this by decomposing the large secret Δ into a bit-vector, so that each entry is small. (See Section 6.4 for details.)
- The DDLog algorithm only ensures $\sigma_z^* = \Delta \cdot z + k_z^* \bmod p$, while we need the equality to hold over $\mathbb{Z}$. A standard trick is to add a common random shift to both σ_z^* and k_z^* mod p. If p is sufficiently larger than $\Delta \cdot z$, then the equality holds over $\mathbb{Z}$ except with negligible probability. The common random shifts and the randomness R for DDLog can be derived from a public PRF seed included in both sk and evk.

What we obtain now is an aHMAC scheme with 1/poly correctness, and where the global secret Δ is sampled as a secret group exponent. In our applications, it will be convenient if the final evaluated tags σ_z can have a user-supplied global secret Δ' instead. Furthermore, for evaluating a multi-output circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$, we would like the user-supplied secrets to be different for each output bit of $\mathbf{z} = C(\mathbf{x})$. [2] We implement those features in our

[2]This can be trivially achieved by running the single-output scheme ℓ_z times. But in the multi-output version, we require the tags sizes to be independent of ℓ_z.

final constructions in Section 6.4, which also contains a construction with negl correctness error based on method (b), a construction based on the KDM security of Damgård-Jurik encryption, and leveled variants of the constructions.

We summarize the syntax of our final constructions.

$$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathbf{\Delta})$$

$$\begin{array}{ll} \underline{\text{Authenticator}}(\mathsf{sk}): & \underline{\text{Evaluator}}(\mathsf{evk}): \\ \sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id}), & \sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \{\sigma_x^{(i)}\}), \\ \mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \{\mathsf{id}^{(i)}\}), & \quad // \text{ s.t. } \sigma_{\mathbf{z}} = \mathbf{\Delta} \odot \mathbf{z} + \mathbf{k}_C \text{ over } \mathbb{Z}, \end{array}$$

where $\mathbf{z} = C(\mathbf{x})$, and $\odot$ denotes component-wise multiplication between vectors.

6.2.2 *Overview of Succinct Partial Garbling and CDS*

Using aHMAC, we first obtain a succinct partial garbling scheme for general circuits, which implies succinct CDS for circuits.

Garbling schemes [Yao86] play a critical role in modern cryptography, enabling non-interactive secure computation in a variety of applications. A garbling scheme is a randomized algorithm transforming a "program" $C : \{0,1\}^n \to \{0,1\}^m$, such as a circuit or a branching program, into a garbled program $\hat{C}$ along with a pair of short keys $(k_{i,0}, k_{i,1})$ for each input bit x_i. Given the program C, the garbled program $\hat{C}$ and the input keys $k_x = (k_{i,x_i})_{i \in [n]}$ for a private input x, anyone can compute $C(x)$ while learning nothing else about x, in the sense that $(\hat{C}, k_x)$ can be simulated (up to computational indistinguishability) given C and $C(x)$ alone.

Motivated by potential efficiency benefits in applications, Ishai and Wee [IW14] extended this notion to *partial garbling*, where part of the input is public. In partial garbling, the program is decomposed into a public part $C_{\text{pub}}(x)$, which depends only on a public input x, and a private part $C_{\text{priv}}(C_{\text{pub}}(x), y)$, which also involves a private input y. Partial garbling generalizes standard garbling, privacy-free garbling [FNO15], and CDS [GIKM00, AARV17].

The latter can be view as a special instance of partial garbling, which in turn also implies partial garbling when combined with standard garbling; see Section 6.2.2.

Yao's original garbling scheme for circuits [Yao86] and its optimization [BMR90, NPS99, KS08a, PSSW09, KMR14, GLNP15, ZRE15, RR21] have garbled circuit size growing linearly with the size $|C|$ of the original circuit, creating a communication bottleneck for large-scale computations. *Succinct garbling*, where the size of the garbled program $\hat{C}$ does not grow with C, holds the promise to resolve the bottleneck. Specifically, the garbled circuit size $|\hat{C}|$ should only depend (polynomially) on the security parameter, and possibly also on second-order parameters such as the input and output length. However, existing succinct garbling schemes rely on "high-end" primitives, such as iO [KLW15, BCG+18] or combinations of FHE and ABE [GKP+13b, BGG+14, HLL23]. Without iO or lattices, even for the weaker notion of partial garbling (or CDS), succinct schemes where the garbled circuit size is only independent of the size $|C_{\text{pub}}|$ of the public computation, are only known for highly restricted program classes, such as truth tables [HK21, Hea24, ABI+23].

Theorem 6.2 (Succinct Partial Garbling for Circuits, Informal; see Theorems 6.10, 6.11)**.** *Assume DDH over prime-order groups, or Paillier groups, or class groups. Let $\mathcal{C}$ be the class of two-input circuits of the form $C(x, y) = C_{\text{priv}}(C_{\text{pub}}(x), y)$. Then, there is a succinct partial garbling scheme for $\mathcal{C}$ with garbled circuit of size $(|C_{\text{priv}}| + D_{\text{pub}}) \cdot \text{poly}(\lambda)$, where D_{pub} is the depth of the public circuit C_{pub}. Under the CP-DDH assumption, or alternatively KDM-DCR, the garbled circuit size is reduced to $|C_{\text{priv}}| \cdot \text{poly}(\lambda)$.*

Technical Overview. As a first step, we compose an aHMAC scheme with any symmetric encryption E to *succinctly* implement a simple case of partial garbling $\mathsf{CDS}(\mathbf{x}, y) = f(\mathbf{x}) \cdot y$.

At high level, the garbler use aHMAC tags for $\mathbf{x}$ as their labels, and prepares an encryption of y under the global secret Δ. To help decryption by the evaluator, the garbler also releases

an evaluated MAC key k_f.

$\underline{\mathsf{CDSGb}}(f, y)$:	$\underline{\mathsf{CDSEv}}(f, \mathsf{gb}, \mathbf{x}, \{L_x^{(i)} = \sigma_{\mathbf{x}[i]}^{(i)}\})$:
$\Delta \leftarrow \$,\ (\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \Delta)$	Output 0 if $f(\mathbf{x}) = 0$. O/w:
$\sigma_b^{(i)} \leftarrow \mathsf{Auth}(\mathsf{sk}, b, \mathsf{id}^{(i)})$,	$\sigma_z \leftarrow \mathsf{EvalTag}(\mathsf{evk}, f, \mathbf{x}, \{\sigma_{\mathbf{x}[i]}^{(i)}\})$,
$k_f \leftarrow \mathsf{EvalKey}(\mathsf{sk}, f, \{\mathsf{id}^{(i)}\})$,	$\Delta = \sigma_z - k_f$,
$\mathsf{ct}_y \leftarrow \mathsf{E.Enc}(\Delta, y)$,	$y = \mathsf{E.Dec}(\Delta, \mathsf{ct}_y)$,
Output $\{L_{x,b}^{(i)} = \sigma_b^{(i)}\}$,	Output y.
and $\mathsf{gb} = (\mathsf{evk}, k_f, \mathsf{ct}_y)$.	

The evaluator, given aHMAC tags for $\mathbf{x}$ can evaluate f on them to obtain a tag $\sigma_z = \Delta \cdot f(\mathbf{x}) + k_f$. (For simplicity, we assume the aHMAC scheme has negl correctness errors in this overview. See Section 6.5.2 for how to handle 1/poly correctness errors.) When $f(\mathbf{x}) = 1$, the evaluators use k_f to recover Δ, and decrypts y correctly. When $f(\mathbf{x}) = 0$, $k_f = \sigma_z$ is completely redundant. The aHMAC security then guarantees that Δ remains hidden to the evaluator, and hence the ciphertext ct_y under Δ securely hides y.

We can extend the above construction to a slightly more general case, where $\mathsf{CDS}(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}) \odot \mathbf{y}$. Here $f : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_y}$ is a multi-output function, and $\odot$ denotes component-wise multiplication. An evaluator can learn the j-th components of the secret input $\mathbf{y}$ if and only if $f(\mathbf{x})[j] = 1$.

Finally, we compose the simple-case succinct partial garbling $\mathsf{CDSGb}, \mathsf{CSDEv}$ with any standard Boolean garbling $\mathsf{BG.Garb}, \mathsf{BG.Eval}$ to handle general cases, $C(\mathbf{x}, \mathbf{y}) = C_{\text{priv}}(C_{\text{pub}}(\mathbf{x}), \mathbf{y})$, where the public computation P_{pub} outputs intermediate values $\mathbf{w} = C_{\text{pub}}(\mathbf{x})$, and the private computation C_{priv} outputs final results $\mathbf{z} = C_{\text{priv}}(\mathbf{w}, \mathbf{y})$.

At a high level, the construction runs BG to garble the private computation C_{priv}, producing labels for possible values of $\mathbf{w}$ and $\mathbf{y}$, and runs $\mathsf{CDSGb}, \mathsf{CDSEv}$ to release only the labels

corresponding to $\mathbf{w} = C_{\mathrm{pub}}(\mathbf{x})$.

$\underline{\mathsf{Garb}}(C = (C_{\mathrm{pub}}, C_{\mathrm{priv}}))$:	$\underline{\mathsf{Eval}}(C, \widehat{C}, \mathbf{x}, \{L_x^{(i)}, \overline{L}_x^{(i)}\}, \{L_y^{(i)}\})$:
$(\widehat{C_{\mathrm{priv}}}, \{L_{w,b}^{(i)}\}, \{L_{y,b}^{(i)}\}) \leftarrow \mathsf{BG.Garb}(C_{\mathrm{priv}})$,	$\mathbf{w} = C_{\mathrm{pub}}(\mathbf{x})$,
$(\{L_{x,b}^{(i)}\}, \mathsf{gb}) \leftarrow \mathsf{CDSGb}(C_{\mathrm{pub}}, \{L_{w,1}^{(j)}\}))$,	$\{L_w^{(j)}\}_{\mathbf{w}[j]=1} \leftarrow \mathsf{CDSEv}(C_{\mathrm{pub}}, \mathsf{gb}, \mathbf{x}, \{L_x^{(i)}\})$,
$(\{\overline{L}_{x,b}^{(i)}\}, \overline{\mathsf{gb}}) \leftarrow \mathsf{CDSGb}(\overline{C_{\mathrm{pub}}}, \{L_{w,0}^{(i)}\}))$,	$\{L_w^{(j)}\}_{\mathbf{w}[j]=0} \leftarrow \mathsf{CDSEv}(\overline{C_{\mathrm{pub}}}, \mathsf{gb}, \mathbf{x}, \{\overline{L}_x^{(i)}\})$,
Output $\{L_{x,b}^{(i)}, \overline{L}_{x,b}^{(i)}\}, \{L_{y,b}^{(i)}\}$,	$\mathbf{z} \leftarrow \mathsf{BG.Eval}(C_{\mathrm{priv}}, \widehat{C_{\mathrm{priv}}}, \{L_w^{(j)}\}, \{L_y^{(i)}\})$,
and $\widehat{C} = (\widehat{C_{\mathrm{priv}}}, \mathsf{gb}, \overline{\mathsf{gb}})$.	Output $\mathbf{z}$.

6.2.3 From Partial to Full Garbling and PSM

We present a generic transformation that combines a succinct partial garbling scheme and an HE scheme for a program class $\mathcal{P}$ to obtain a succinct (fully hiding) garbling scheme for the same program class, which implies PSM for the same class. The transformation is extremely simple, and follows the blueprint of constructing succinct garbled circuits from FHE and ABE [GKP+13b, BGG+14], replacing the ABE ingredient by partial garbling. To garble a program P, use the partial garbling scheme to garble the circuit $C(\hat{x}, \mathsf{sk}) = \mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, P, \hat{x})) = P(x)$ that performs homomorphic evaluation of P over a public HE ciphertext $\hat{x}$ of the actual input x, followed by HE decryption using the secret key sk. The succinctness of partial garbling ensures that the garbled circuit $\hat{C}$ grows only with the complexity of HE decryption (i.e., $C_{\mathrm{priv}}(\star, \star) = \mathsf{Dec}(\star, \star)$) which in turn depends only on the output length and the security parameter, if assuming circular security. Without circular security, the size additionally depends on the depth of the homomorphic evaluation of P (i.e., $C_{\mathrm{pub}}(\star) = \mathsf{Eval}(\mathsf{pk}, P, \star)$). For the types of simple programs we consider here, the homomorphic evaluation depth is bounded by a fixed polynomial in the security parameter, eliminating the need for circular security.

We note that while using HE to ensure the privacy of the input x is a standard technique, our key observation here is that partial garbling suffices for ensuring the correctness / integrity of the homomorphic evaluation, replacing the stronger ABE techniques (with succinct secret

keys) used in [GKP+13b, BGG+14, QWW21]. This approach, though simple in retrospect, is crucial for moving away from lattice-based assumptions and iO.

Lemma 6.1 (From Partial to Full Garbling, Informal)**.** *Any homomorphic encryption scheme for a class of programs $\mathcal{P}$ can be transformed into a succinct garbling scheme for $\mathcal{P}$, using a succinct partial garbling scheme able to support the homomorphic evaluation of the homomorphic encryption scheme for $\mathcal{P}$. The size of the garbled program is* $\mathrm{poly}(\lambda) \cdot m$ *if assuming circular security, or* $\mathrm{poly}(\lambda) \cdot (m + D_{\mathsf{Eval}})$ *without circular security, where m is the output length and D_{Eval} is the maximal circuit depth of the homomorphic evaluation of programs in $\mathcal{P}$.*

Succinct Garbling for Simple Programs and Implications for General Circuits. General-purpose FHE schemes are currently only known lattice assumptions or iO/FE, which this work tries to avoid. However, for several simple but useful classes of programs, there are HE schemes relying on group-based assumptions. For instance, private information retrieval (PIR) schemes with polylogarithmic communication, which can be based on a variety of assumptions [KO97, Ste98, OS07, Lip05, GR05, DGI+19], can be viewed as a (compact) HE scheme for *truth-table* programs. This was generalized in [IP07, DGI+19] to yield, under similar assumptions, an HE scheme for *bounded-length branching programs* of unbounded size, where only the length bound impacts the encryption size.[3] As special cases, this enables the compact evaluation of decision trees and DFAs of an arbitrary size on an encrypted input. In a different direction, the Boneh-Goh-Nissim cryptosystem [BGN05] implies an HE scheme for quadratic polynomials over a finite field under an assumption on bilinear groups. Here the program size is at most quadratic in the input size.

Combining the above HE schemes with our succinct partial garbling, we obtain the following.

[3]This is analogous to LWE-based HE for circuits, where the ciphertext size depends on the circuit depth but not on its size.

Corollary 6.1 (Succinct Garbling for Simple Programs)**.** *Assuming DDH over prime-order groups, or Paillier/Damgård-Jurik groups, or class groups, there are succinct garbling schemes for the following classes:*

- *bounded-length (unbounded size) branching programs, assuming additionally the DDH assumption over prime order groups, or the DCR assumption over Paillier groups. This implies succinct garbling for truth tables, deterministic finite automata (DFAs), and decision trees of an arbitrary size.*
- *quadratic polynomials, assuming additionally the hardness of the subgroup decision problem in composite-order bilinear groups.*

We further show a generic composition theorem that leverages succinct garbling for simple programs, to garble circuits consisting of general gates computing these simple programs. The cost of garbling scales with the number of wires in circuits over general gates, without growing with the number of Boolean gates (such as AND gates) required to implement a general gate.

Corollary 6.2 (Implication for Garbling General Circuits)**.** *Under the same assumptions as in Corollary 6.1, there is a garbling scheme for circuits over general gates, each computing one of the simple programs in Corollary 6.1, where the garbling size is* $\mathrm{poly}(\lambda) \cdot \#wires$*, where* $\#wires$ *is the number of wires in the circuit.*

Implication for PSM protocols. Our succinct garbling schemes imply the first group-based succinct PSM protocols for simple programs. In the case of truth tables, we get PSM protocols for arbitrary functions with polylogarithmic communication and polynomial computation in the input domain size.

Corollary 6.3 (Computationally Secure PSM for Truth Tables)**.** *Assuming DDH over prime-order groups, or DCR plus DDH over Paillier groups, any* k*-party function* $f : [N]^k \to \{0, 1\}$ *has a computationally secure PSM protocol with* $\mathrm{poly}(\lambda, k, \log N)$ *communication and* $\mathrm{poly}(\lambda, N^k)$ *computation.*

PSM vs. generalized secret sharing. Finally, it is interesting to compare our succinct computational PSM result for truth tables from Corollary 6.3 with a recent succinct computational secret sharing (SCSS) scheme from [ABI+23], which applies to *general* access structures represented by a truth table. One might a-priori expect the SCSS question to be easier because of its one-sided hiding requirement and the relation with partial garbling and CDS, for which we have strong positive results for truth tables even in the information-theoretic setting. However, the current state of the art on the two problems is incomparable. The group-based construction of SCSS from [ABI+23] specifically relies on the RSA assumption, and it is open whether the same conclusion holds from other group-based assumptions, such as the ones we use in this work. On the other hand, using LWE-based succinct garbling [BGG+14], succinct PSM can be based on LWE, which is still open for SCSS. The known group-based PSM and SCSS solutions are also incomparable in terms of the class of programs they efficiently support beyond truth tables: branching programs with bounded length in the PSM case and monotone CNF formulas of unbounded size in the SCSS case.

6.2.4 Overview of Constrained Pseudorandom Functions

By combining an aHMAC with HSS schemes, and following the HSS-based blueprint of Couteau, Meyer, Passelègue and Riahinia [CMPR23], we obtain the first CPRF for general constraint circuits from group-based assumptions. Note that the "bounded-depth" variant of aHMAC suffices for this application, and hence no circular security assumption is needed.

Theorem 6.3 (CPRFs for Circuits, Informal; see Theorem 6.14)**.** *There is a 1-key, selectively-secure CPRF supporting general constraint circuits based on DCR, or DDH and small-exponent assumptions, over Paillier groups.*

Technical Overview. In a constrained PRF (CPRF), there are two evaluation algorithms $\mathsf{Eval}, \mathsf{CEval}$, one with a normal key msk, and one with a constrained key sk_C with respect to a circuit C. Correctness requires evaluations (on $\mathbf{x}$) using both keys should equal if $C(\mathbf{x}) = 0$. Security requires evaluations using msk remain pseudorandom if $C(\mathbf{x}) = 1$, even given the

constrained key sk_C.

$$\mathsf{msk} \leftarrow \mathsf{KeyGen}(1^\lambda), \quad \mathsf{evk} \leftarrow \mathsf{Constrain}(\mathsf{msk}, C).$$

$$\begin{array}{lll} \text{Correctness:} & \mathsf{Eval}(\mathsf{msk}, \mathbf{x}) = \mathsf{CEval}(\mathsf{sk}_C, \mathbf{x}), & \text{if } C(\mathbf{x}) = 0, \\ \text{Security:} & \mathsf{Eval}(\mathsf{msk}, \mathbf{x}) \text{ pseudorandom given } \mathsf{sk}_C & \text{o/w.} \end{array} \tag{6.3}$$

The Blueprint of [CMPR23]. The observation of [CMPR23] is that common homomorphic secret sharing schemes (HSS) for evaluating restricted multiplication straight-line programs (RMS) allows for an extended evaluation algorithm. Normally, an HSS evaluation for a function f locally transforms a pair of input shares I_0, I_1 for $\mathbf{x}$ into additive shares z_0, z_1 of $f(\mathbf{x})$:

$$I_0, I_1 \leftarrow \mathsf{HSS.Input}(\mathbf{x}), \quad (\mathsf{evk}_0, \mathsf{evk}_1) \leftarrow \mathsf{HSS.Setup}(1^\lambda),$$
$$z_b \leftarrow \mathsf{HSS.Eval}(b, \mathsf{evk}_b, I_b, f), \quad \text{s.t. } z_1 = z_0 + f(\mathbf{x}).$$

The extended evaluation takes an additional pair of shares Δ_0, Δ_1 of the form $\Delta_1 = \Delta \cdot w + \Delta_0$ for some secret value Δ, and output additive shares z_0, z_1 of $w \cdot f(\mathbf{x})$. (See Section 6.6 for details of this extension.)

$$I_0, I_1 \leftarrow \mathsf{HSS.Input}(\mathbf{x}), \quad (\mathsf{evk}_0, \mathsf{evk}_1, \Delta) \leftarrow \mathsf{HSS.Setup}(1^\lambda),$$
$$\text{any } \Delta_0, \Delta_1 \text{ s.t. } \Delta_1 = \Delta_0 + \Delta \cdot w$$
$$z_b \leftarrow \mathsf{HSS.ExtEval}(b, \mathsf{evk}_b, I_b, \Delta_b, f), \quad \text{s.t. } z_1 = z_0 + w \cdot f(\mathbf{x}).$$

In order to construct a CPRF scheme, it now suffices to design a mechanism that let Eval and CEval respectively derive shares Δ_0, Δ_1 such that $\Delta_1 = \Delta \cdot C(\mathbf{x}) + \Delta_0$, and then run an extended HSS evaluation of the function $\mathsf{F}_\mathbf{x}(\cdot) = \mathsf{F}(\cdot, \mathbf{x})$, on input shares I_0, I_1 of a secret key key for a PRF F.

Blueprint:

$\mathsf{msk} = (\mathsf{evk}_0, I_0, \ldots,), \mathsf{sk}_C = (\mathsf{evk}_1, I_1, \ldots,),$

$\mathsf{Eval}:$ derive Δ_0, $z_0 \leftarrow \mathsf{HSS.ExtEval}(0, \mathsf{evk}_0, I_0, \Delta_0, \mathsf{F}_\mathbf{x}),$

$\mathsf{CEval}:$ derive Δ_1 $z_1 \leftarrow \mathsf{HSS.ExtEval}(1, \mathsf{evk}_1, I_1, \Delta_1, \mathsf{F}_\mathbf{x}).$

By the extended evaluation correctness of HSS, we have $z_1 = z_0 + C(\mathbf{x}) \cdot \mathsf{F}(\mathsf{key}, \mathbf{x})$, which satisfy both CPRF correctness and security (Equation 6.3). The work of [CMPR23] uses another special HSS to let Eval, CEval derive the desired shares Δ_0, Δ_1. We instead use an aHMAC scheme for this.

In our construction of CPRF, we generate a secret key $\mathsf{aHMAC.sk}$ and evaluation key $\mathsf{aHMAC.evk}$ with respect to a user-supplied global secret Δ, which is exactly the secret for extended HSS evaluations. We view the constrained circuit C as a bit string, and compute tags $\{\sigma_C^{(i)}\}$ authenticating the bits of C.

$$\begin{aligned}
\text{KeyGen}(1^\lambda) :& (\mathsf{evk}_0, \mathsf{evk}_1, \Delta) \leftarrow \mathsf{HSS.Setup}(1^\lambda), \\
& \mathsf{key} \leftarrow \$, \quad (I_0, I_1) \leftarrow \mathsf{HSS.Input}(\mathsf{key}), \\
& (\mathsf{aHMAC.sk}, \mathsf{aHMAC.evk}) \leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, \Delta). \\
& \text{Outputs } \mathsf{msk} = (I_0, \mathsf{evk}_0, I_1, \mathsf{evk}_1, \mathsf{aHMAC.sk}, \mathsf{aHMAC.evk}). \\
\text{Constrain}(\mathsf{msk}, C) :& \sigma_C^{(i)} \leftarrow \mathsf{aHMAC.Auth}(\mathsf{aHMAC.sk}, C[i], \mathsf{id}^{(i)}). \\
& \text{Outputs } \mathsf{sk}_C = (I_1, \mathsf{evk}_1, C, \{\sigma_C^{(i)}\}, \mathsf{aHMAC.evk}).
\end{aligned}$$

Then, following the blueprint Eval and CEval can respectively run $\mathsf{aHMAC.EvalKey}$ and $\mathsf{aHMAC.EvalTag}$ with a universal function $U_{\mathbf{x}}(C) = C(\mathbf{x})$ to derive k_w and σ_w as the shares Δ_0 and Δ_1.

$$\begin{aligned}
\text{Eval}(\mathsf{msk}, \mathbf{x}) :& \Delta_0 = k_U \leftarrow \mathsf{aHMAC.EvalKey}(\mathsf{aHMAC.sk}, U_{\mathbf{x}}, \{\mathsf{id}^{(i)}\}), \\
& z_0 \leftarrow \mathsf{HSS.ExtEval}(0, \mathsf{evk}_0, I_0, \Delta_0, \mathsf{F}_{\mathbf{x}}), \\
\text{CEval}(\mathsf{evk}, \mathbf{x}) :& \Delta_1 = \sigma_U \leftarrow \mathsf{aHMAC.EvalTag}(\mathsf{aHMAC.evk}, U_{\mathbf{x}}, C, \{\sigma_C^{(i)}\}), \\
& z_1 \leftarrow \mathsf{HSS.ExtEval}(1, \mathsf{evk}_1, I_1, \Delta_1, \mathsf{F}_{\mathbf{x}}).
\end{aligned}$$

By the correctness of aHMAC, we indeed have $\Delta_1 = \Delta \cdot C(\mathbf{x}) + \Delta_0$.

6.3 *Preliminaries*

6.3.1 *Definition of Partial Garbling Schemes*

In this chapter, we consider a generalization to standard garbling schemes (Definition 2.1), called partial garbling. In a standard garbling scheme, a program P and Boolean inputs $\mathbf{x}$ are encoded respectively into a garbled program $\widehat{P}$ and input labels $\{L_x^{(i)}\}$. The standard security requires that $\widehat{P}, \{L_x^{(i)}\}$ together reveals nothing about the input $\mathbf{x}$ beyond $P(\mathbf{x})$. The notion of partial garbling, first proposed in [IW14], relaxes the security requirement to allow part of the input $\mathbf{x}$ to be leaked. We refer to this part as the pub

The program can then be decomposed into two parts: $P(\mathbf{x}, \mathbf{y}) = P_{\text{priv}}(P_{\text{pub}}(\mathbf{x}), \mathbf{y})$, where P_{pub} represents the computation that depends on the public input $\mathbf{x}$ alone, and P_{priv} represents the rest that also depends on the private input $\mathbf{y}$. We call $P_{\text{pub}}, P_{\text{priv}}$ the public and private computations of P, respectively. We give two useful examples suitable for partial garbling:

- $P(\mathbf{x}, s) = f(\mathbf{x}) \cdot s$ implements a conditional disclosure of secret (CDS) functionality. The output of P reveals the secret input s if and only if the public input satisfy $f(\mathbf{x}) = 1$.
- $P(\mathsf{ct}, \mathsf{sk}) = \mathsf{HE.Dec}(\mathsf{HE.Eval}(f, \mathsf{ct}), \mathsf{sk})$ implements a homomorphic evaluation of HE ciphertexts ct, followed by decryption using a secret key sk. Assuming the ciphertexts correctly encrypts some input $\mathbf{x}$, then a partial garbling of P implements a (fully private) standard garbling of f.

In exchange for the relaxed security, we expect more efficient constructions. In this work, we consider the strong efficiency requirement, *succinctness with respect to public computation*, that the garbled program size $|\widehat{P}|$ should be independent of the complexity of the public computation. More precisely, we consider garbling families of programs $\mathcal{P} = \{\mathcal{P}_\lambda\}$ indexed by a security parameter λ. Each family $\mathcal{P}_\lambda$ satisfies some restrictions on its parameters, e.g. computation depth, but importantly has no polynomial bound on the *size* of the program. In this work, we obtain succinct partial garbling schemes for two classes:

- General Boolean circuits: $\mathcal{C} = \{\mathcal{C}_\lambda\}$, where $\mathcal{C}_\lambda$ consists of all two-input Boolean circuits,

of form $C(\mathbf{x}, \mathbf{y}) = C_{\text{priv}}(C_{\text{pub}}(\mathbf{x}), \mathbf{y})$, based on circular-security assumptions, CP-DDH or KDM-DCR (Definition 7.3, 6.10);

- Bounded-depth Boolean circuits: $\mathcal{C}^d = \{\mathcal{C}^d_\lambda\}$, where $\mathcal{C}^d_\lambda$ consists of all two-input Boolean circuits with depth bounded by some fixed polynomial $d(\lambda)$, based on DDH (Definition 6.7).

As applications of our succinct partial garbling for bounded-depth circuits, we also obtain (fully private) succinct standard garbling (Definition 2.1, extended to supporting families of programs in the natural way,) schemes for two classes. Here succinctness requires the garbled program size $|\widehat{P}|$ to be independent of the complexity of the *entire* computation.

- Bounded-length branching programs: $\mathcal{P} = \{\mathcal{P}^\ell_\lambda\}$, where $\mathcal{P}^\ell_\lambda$ consists of branching programs with length bounded by some fixed polynomial $\ell(\lambda)$ and size below $2^{\text{poly}(\lambda)}$.
- Quadratic polynomials (mod 2): $\mathcal{Q} = \{\mathcal{Q}_\lambda\}$, where $\mathcal{Q}_\lambda$ consists of quadratic polynomials with below $2^{\text{poly}(\lambda)}$ number of monomials.

Besides succinctness, we also impose composability of garbled circuits at the syntax level: the garbled program evaluation should output arbitrary *target* key functions $K^{(i)}_z$ (specified at garbling time) applied to the output bits $\mathbf{z} = P(\mathbf{x}, \mathbf{y})$.

Definition 6.1 (Partial Garbling)**.** *Let $\mathcal{P} = \{\mathcal{P}_\lambda\}$ be a class of programs with Boolean inputs and of the form $P(\mathbf{x}, \mathbf{y}) = P_{\text{priv}}(P_{\text{pub}}(\mathbf{x}), \mathbf{y})$, and $\mathcal{L} = \{\mathcal{L}_\lambda\}_\lambda$ be a label space of sizes $|\mathcal{L}_\lambda| \leq 2^{\text{poly}(\lambda)}$. A partial garbling scheme for $\mathcal{P}$ with label space $\mathcal{L}$ consists of two efficient algorithms:*

- $\mathsf{Garb}(1^\lambda, P \in \mathcal{P}_\lambda, \{K^{(i)}_z\}_{i\in[\ell_z]})$ *takes a program $P : \{0,1\}^{\ell_x} \times \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_z}$, and target key functions $\{K^{(i)}_z\}$ (mapping output bits to labels in $\mathcal{L}_\lambda$). It outputs a garbling $\widehat{P}$, and input key functions $\{K^{(i)}_x\}_{i\in[\ell_x]}$, and $\{K^{(i)}_y\}_{i\in[\ell_y]}$.*
- $\mathsf{Eval}(P, \widehat{P}, \{x^{(i)}, L^{(i)}_x\}_{i\in[\ell_x]}, \{L^{(i)}_y\}_{i\in[\ell_y]})$ *takes a program P, a garbling $\widehat{P}$, public inputs $x^{(i)}$, their labels $L^{(i)}_x$, and labels for private inputs $L^{(i)}_y$. It outputs labels $L^{(i)}_z$ for $i \in [\ell_z]$.*

Correctness: *For every polynomial* $p(\lambda)$, *there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$, *programs* $P \in \mathcal{P}_\lambda$ *with size* $|P| \leq p(\lambda)$, *inputs* $\mathbf{x}$, $\mathbf{y}$, *and target key functions* $\{K_z^{(i)}\}$ *the following holds:*

$$\Pr\left[\begin{array}{l|r} \mathsf{Eval}(P, \widehat{P}, \{x^{(i)}, L_x^{(i)}\}, \{L_y^{(i)}\}) & (\widehat{P}, \{K_x^{(i)}\}, \{K_y^{(i)}\}) \leftarrow \mathsf{Garb}(1^\lambda, P, \{K_z^{(i)}\}), \\ = \{L_z^{(i)}\} & L_x^{(i)} = K_x^{(i)}(x^{(i)}),\ L_y^{(i)} = K_y^{(i)}(y^{(i)}), \\ & \mathbf{z} = P(\mathbf{x}, \mathbf{y}),\ L_z^{(i)} = K_y^{(i)}(z^{(i)}). \end{array}\right]$$
$$\geq 1 - \mathrm{negl}(\lambda).$$

(Computational) Security: *There exists an efficient simulator* Sim *such that for every polynomial* $p(\lambda)$, *sequence of programs* $\{P_\lambda\}$ *from* $\mathcal{P}$ *such that* $|P_\lambda| < p(\lambda)$, *sequence of inputs* $\{\mathbf{x}_\lambda, \mathbf{y}_\lambda\}$, *and sequence of target key functions* $\{K_{z,\lambda}^{(i)}\}_{i,\lambda}$, *the following holds (suppressing the subscript* λ *for brevity):*

$$\left\{\mathsf{Sim}(1^\lambda, P, \mathbf{x}, \{L_z^{(i)}\}) \,|\, \mathbf{z} = P(\mathbf{x}, \mathbf{y}),\ L_z^{(i)} = K_z^{(i)}(z^{(i)})\right\}_\lambda$$
$$\approx_c \left\{ \widehat{P}, \{L_x^{(i)}\}, \{L_y^{(i)}\} \;\middle|\; \begin{array}{l} (\widehat{P}, \{K_x^{(i)}\}, \{K_y^{(i)}\}) \leftarrow \mathsf{Garb}(1^\lambda, P), \\ L_x^{(i)} = K_x^{(i)}(x^{(i)}),\ L_y^{(i)} = K_y^{(i)}(y^{(i)}), \end{array} \right\}_\lambda$$

We now define the default succinctness requirement for partial garbling.

Definition 6.2 (Succinctness w.r.t. Public Computation)**.** *We say a partial garbling scheme for a class* $\mathcal{P}$ *is* succinct w.r.t. public computation *if there exists a* $\mathrm{poly}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$ *and* $P \in \mathcal{P}_\lambda$, *the garbling size* $\widehat{P}$ *is bounded by* $\mathrm{poly}(\lambda, |P_{\mathrm{priv}}|)$, *where* $|P_{\mathrm{priv}}|$ *denotes the complexity of the private computation* P_{priv}.

Note that a standard Boolean garbling scheme, e.g. Yao's garbling, can be viewed as a special case of our definition of partial garbling, where the public input is $\emptyset$. For standard garbling schemes, we define succinctness with respect to the entire (private) computation.

Definition 6.3 (Succinctness (w.r.t. Entire Computation))**.** *We say a (standard) garbling scheme for a class* $\mathcal{P}$ *is* succinct *if there exists a* $\mathrm{poly}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$ *and* $P \in \mathcal{P}_\lambda$, *the garbling size* $\widehat{P}$ *is bounded by* $\mathrm{poly}(\lambda, \ell_z)$.

6.3.2 Definitions of PSM and CDS

Definition 6.4 (Private Simultaneous Message (PSM) Schemes [FKN94a, IK97])**.** *Let $\mathcal{P} = \{\mathcal{P}_\lambda\}$ be a class of programs with Boolean inputs and outputs. A PSM scheme for $\mathcal{P}$ consists of two efficient algorithms.*

- $\mathsf{Encode}(1^\lambda, P \in \mathcal{P}_\lambda, i \in [\ell_x], x \in \{0,1\}; \mathbf{r})$ *takes a program $P : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$, an input position i, an input bit x, and a shared randomness $\mathbf{r} \in \{0,1\}^\lambda$. It outputs a message msg_i.*
- $\mathsf{Recon}(1^\lambda, P, \{\mathsf{msg}_i\}_{[\ell_x]})$ *takes a program P and one message msg_i for each input position $i \in [\ell_x]$. It outputs an evaluation result $\mathbf{z} \in \{0,1\}^{\ell_z}$.*

Correctness. *For every polynomial $p(\lambda)$, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$, programs $P \in \mathcal{P}_\lambda$ with ℓ_x inputs and of size $|P| \le p(\lambda)$, and inputs $\mathbf{x} \in \{0,1\}^{\ell_x}$, the following holds:*

$$\Pr\left[\begin{array}{l|c} \mathsf{Recon}(1^\lambda, P, \{\mathsf{msg}_i\}) & \mathbf{r} \leftarrow \{0,1\}^\lambda,\ \textit{and}\ \forall i \in [\ell_x] \\ = P(\mathbf{x}) & \mathsf{msg}_i \leftarrow \mathsf{Encode}(1^\lambda, P, i, \mathbf{x}[i]; \mathbf{r}) \end{array}\right] \ge 1 - \mathrm{negl}(\lambda).$$

(Computational) Privacy. *For every polynomial $p(\lambda)$, sequence of programs $\{P_\lambda\}$ from $\mathcal{P}$ such that $|P_\lambda| \le p(\lambda)$, sequences of inputs $\{\mathbf{x}_\lambda\}$, $\{\mathbf{x}'_\lambda\}$ satsifying $P_\lambda(\mathbf{x}_\lambda) = P_\lambda(\mathbf{x}'_\lambda)$, the following computational indistinguishability holds:*

$$\left\{ \{\mathsf{msg}_i\} \;\middle|\; \begin{array}{c} \mathbf{r} \leftarrow \{0,1\}^\lambda,\ \textit{and}\ \forall i \in [\ell_x] \\ \mathsf{msg}_i \leftarrow \mathsf{Encode}(1^\lambda, P, i, \mathbf{x}[i]; \mathbf{r}) \end{array} \right\}_\lambda$$

$$\approx_c \left\{ \{\mathsf{msg}'_i\} \;\middle|\; \begin{array}{c} \mathbf{r}' \leftarrow \{0,1\}^\lambda,\ \textit{and}\ \forall i \in [\ell_x] \\ \mathsf{msg}'_i \leftarrow \mathsf{Encode}(1^\lambda, P, i, \mathbf{x}'[i]; \mathbf{r}') \end{array} \right\}_\lambda$$

Succinctness. *We say a PSM scheme for a class $\mathcal{P}$ is* succint *if there exists a fixed* $\mathrm{poly}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$ and $P \in \mathcal{P}_\lambda$, the total message bit-lengths $\sum_i |\mathsf{msg}_i|$ is bounded by* $\mathrm{poly}(\lambda, \ell_z)$.

Remark. As outlined in [FKN94a], a garbling scheme implements a multi-party private simultaneous messages (PSM) protocol as follows.

- Each party i running Encode computes a garbled circuit $\widehat{P}$ using shared randomness, and also a input label $L_x^{(i)}$. The message from this party consists of $\mathsf{msg}_i := (\widehat{P}, L_x^{(i)})$. As an optimization, we can also require only server 1 to include the garbled circuit $\widehat{P}$.
- The referee running Recon evaluates the garbled circuit $\widehat{P}$ with all input labels $\{L_x^{(i)}\}$ from each party to obtain the result $\mathbf{z} = P(\mathbf{x})$.

Therefore, a succinct standard garbling scheme for $\mathcal{P}$ directly implies a succinct PSM protocol.

Definition 6.5 (Disclosure of Secret (CDS) Schemes [GIKM00])**.** *Let $\mathcal{P} = \{\mathcal{P}_\lambda\}$ be a class of predicats with Boolean inputs and (1-bit) outputs. A CDS scheme for $\mathcal{P}$ consists of two efficient algorithms.*

- $\mathsf{Encode}(1^\lambda, P \in \mathcal{P}_\lambda, i \in [\ell_x], x \in \{0,1\}, \mathbf{z} \in \{0,1\}^{\ell_z}; \mathbf{r})$ *takes a predicate $P : \{0,1\}^{\ell_x} \to \{0,1\}$, an input position i, an input bit x, a shared secret $\mathbf{z} \in \{0,1\}^{\ell_z}$, and a shared randomness $\mathbf{r} \in \{0,1\}^\lambda$. It outputs a message msg_i.*
- $\mathsf{Recon}(1^\lambda, P, \mathbf{x} \in \{0,1\}^{\ell_x}, \{\mathsf{msg}_i\}_{[\ell_x]})$ *takes a predicate P, a public input $\mathbf{x}$, and one message msg_i for each input position $i \in [\ell_x]$. It outputs either $\perp$ or the secret $\mathbf{z}$.*

Correctness. *For every polynomial $p(\lambda)$, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$, predicates $P \in \mathcal{P}_\lambda$ with ℓ_x inputs and of size $|P| \le p(\lambda)$, inputs $\mathbf{x} \in \{0,1\}^{\ell_x}$, and secrets $\mathbf{z} \in \{0,1\}^{\ell_z}$, the following holds:*

$$\Pr\left[\begin{array}{l} \mathsf{Recon}(1^\lambda, P, \{\mathsf{msg}_i\}) \\ = \mathbf{z} \end{array} \middle| \begin{array}{c} \mathbf{r} \leftarrow \{0,1\}^\lambda, \ \textit{and } \forall i \in [\ell_x] \\ \mathsf{msg}_i \leftarrow \mathsf{Encode}(1^\lambda, P, i, \mathbf{x}[i], \mathbf{z}; \mathbf{r}) \end{array}\right] \ge 1 - \mathrm{negl}(\lambda).$$

(Computational) Privacy. *For every polynomial $p(\lambda)$, sequence of predicates $\{P_\lambda\}$ from $\mathcal{P}$ such that $|P_\lambda| \le p(\lambda)$, sequence of inputs $\{\mathbf{x}_\lambda\}$, satsifying $P_\lambda(\mathbf{x}_\lambda) = 0$, and sequence of secrets*

$\{\mathbf{z}_\lambda\}$, *the following computational indistinguishability holds:*

$$\left\{ \{\mathsf{msg}_i\} \;\middle|\; \begin{array}{c} \mathbf{r} \leftarrow \{0,1\}^\lambda, \text{ and } \forall i \in [\ell_x] \\ \mathsf{msg}_i \leftarrow \mathsf{Encode}(1^\lambda, P, i, \mathbf{x}[i], \mathbf{z}; \mathbf{r}) \end{array} \right\}_\lambda$$

$$\approx_c \left\{ \{\mathsf{msg}'_i\} \;\middle|\; \begin{array}{c} \mathbf{r}' \leftarrow \{0,1\}^\lambda, \text{ and } \forall i \in [\ell_x] \\ \mathsf{msg}'_i \leftarrow \mathsf{Encode}(1^\lambda, P, i, \mathbf{x}'[i], \mathbf{0}; \mathbf{r}') \end{array} \right\}_\lambda .$$

Succinctness. *We say a CDS scheme for a class $\mathcal{P}$ is* succint *if there exists a fixed* $\mathrm{poly}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$, $P \in \mathcal{P}_\lambda$, and secret $\mathbf{z} \in \{0,1\}^{\ell_z}$, the total message bit-lengths $\sum_i |\mathsf{msg}_i|$ is bounded by* $\mathrm{poly}(\lambda, \ell_z)$.

Remark. In the analogous way as using a standard garbling scheme to implement PSM, we can use a partial garbling scheme to implement CDS:

- Define $P'(\mathbf{x}, \mathbf{z}) := \mathbf{z} \cdot P(\mathbf{x})$.
- Each party i running Encode computes a garbled circuit $\widehat{P'}$ using shared randomness, an input label for its input $L_x^{(i)}$, and labels for the secret input $\{L_z^{(j)}\}$. The message from this party consists of $\mathsf{msg}_i := (\widehat{P'}, L_x^{(i)}, \{L_{\mathbf{z}}^{(j)}\})$. As an optimization, we can also require only server 1 to include the garbled circuit $\widehat{P'}$ and labels $\{L_z^{(j)}\}$ for the secret.
- The referee running Recon evaluates the garbled circuit $\widehat{P'}$ with the public input $\mathbf{x}$, corresponding public input labels $\{L_x^{(i)}\}$, and the secret input labels $\{L_z^{(j)}\}$ to obtain the result $\mathbf{z} \cdot P(\mathbf{x})$.

Therefore, a succinct partial garbling scheme for $\mathcal{P}$ directly implies a succinct CDS protocol.

6.3.3 Hardness Assumptions

In this work, we will use two types of groups: (1) groups satisfying the non-interactive distributed log sharing (NIDLS) framework [ADOS22], which have distributed discrete log (DDLog) algorithms with perfect correctness, and (2) prime-order groups (Definition 2.6), which have DDLog algorithms with a 1/poly correctness error [BGI16, DKK18].

Definition 6.6 (NIDLS Framework [ADOS22]). *Let $\mathcal{G} = \{\mathcal{G}_\lambda\}$ be a sequence of families of groups (with efficient group operations). We say $\mathcal{G}$ is an instantiation of the NIDLS framework if the following three efficient algorithms exist:*

- $\mathsf{Gen}(1^\lambda)$ *outputs public parameters* $\mathsf{pp} = (G, F, H, f, t, \ell)$ *where*
  - $G \in \mathcal{G}_\lambda$ *is a finite Abelian group with subgroups F, H s.t. $G = F \times H$;*
  - F *is a cyclic group of order $t > 2^\lambda$, and f is a generator of F;*
  - ℓ *is an upper-bound on the order of H.*
- $\mathsf{Samp}(\mathsf{pp})$ *samples an element $g \in G$ with the guarantee that $f \in \langle g \rangle$, and that the following statistical indistinguishability holds:*

$$\left\{ \mathsf{pp}, \rho, g^s \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ s \leftarrow [\ell]. \end{array} \right\}$$
$$\approx \left\{ \mathsf{pp}, \rho, g' \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}), g' \leftarrow \langle g \rangle. \end{array} \right\}.$$

  It outputs g and the sampling randomness ρ.
- $\mathsf{DDLog}(\mathsf{pp}, a \in G)$ *takes an element a and outputs a value $\alpha \in \mathbb{Z}_t$ with the guarantee that for all $a \in G$, $m \in \mathbb{Z}_t$:*

$$\mathsf{DDLog}(\mathsf{pp}, a \cdot f^m) = \mathsf{DDLog}(\mathsf{pp}, a) + m \bmod t.$$

Remark. Compared to the description in [ADOS22], we additionally require the subgroups F have large orders $t > 2^\lambda$. This is needed in our application to (non-interactively) convert additive shares mod t of $0, 1$ values into shares over $\mathbb{Z}$.

Known instantiations of the framework, with large subgroups F, include (the ciphertext spaces of) Damgård-Jurik encryption, a variant of Joye-Libert encryption described in [ADOS22], and class groups.

Lemma 6.2 (Distributed Discrete Log with Error [BGI16, DKK18]). *For any cyclic group G with order p and a generator g, there exists an algorithm* $\mathsf{DDLog}_{G,g}$*:*

- $\mathsf{DDLog}_{G,g}(\delta \in (0,1], B \in [p], \phi : G \to \{0,1\}^{\lceil \log(2B/\delta) \rceil}, a \in G)$ *takes an error bound* δ*, a message bound* B*, a function* ϕ *mapping group elements to bit strings, and an element* a*. It outputs a value* $\alpha \in \mathbb{Z}_p$*.*

The algorithm requires $O(\sqrt{B/\delta})$ *group operations, and has the guarantee that for all* $0 < \delta \leq 1$*,* $B < p$*,* $a \in G$*, and* $m \leq B$*:*

$$\Pr\left[\begin{array}{c} \mathsf{DDLog}_{G,g}(\delta, B, \phi, a \cdot g^m) \\ = \mathsf{DDLog}_{G,g}(\delta, B, \phi, a) + m \bmod p \end{array} \middle| \ \phi \leftarrow \$ \right] \geq 1 - \delta,$$

where $\phi \leftarrow \$$ *means sampling at random from all possible mappings.*

We state the standard DDH and power-DDH assumptions below, followed by our new circular-power-DDH and a proof that it holds in GGM.

Definition 6.7 (DDH Assumption)**.** *We say the DDH assumption holds in the NIDLS framework if the following holds:*

$$\left\{ \mathsf{pp}, (\rho, g), g^a, g^b, g^{ab} \ \middle| \ \begin{array}{r} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ a, b \leftarrow [\ell]. \end{array} \right\}$$
$$\approx_c \left\{ \mathsf{pp}, (\rho, g), g^a, g^b, g^c \ \middle| \ \begin{array}{r} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ a, b, c \leftarrow [\ell]. \end{array} \right\}.$$

We say the DDH assumption holds in prime-order groups if the following holds:

$$\left\{ G, g, g^a, g^b, g^{ab} \ \middle| \ \begin{array}{c} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ a, b \leftarrow \mathbb{Z}_p. \end{array} \right\} \approx_c \left\{ G, g, g^a, g^b, g^c \ \middle| \ \begin{array}{c} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ a, b, c \leftarrow \mathbb{Z}_p. \end{array} \right\}.$$

Via a standard hybrid argument, we obtain DDH in matrix form:

Lemma 6.3 (DDH in Matrix Form)**.** *Assuming DDH in the NIDLS framework, for any*

polynomials $m(\lambda)$, $n(\lambda)$, the following holds:

$$\left\{ \mathsf{pp}, (\rho, g), g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{a}\cdot\mathbf{b}^T} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ \mathbf{a} \leftarrow [\ell]^m,\ \mathbf{b} \leftarrow [\ell]^n. \end{array} \right\}$$

$$\approx_c \left\{ \mathsf{pp}, (\rho, g), g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{C}} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}), \mathbf{a} \leftarrow [\ell]^m, \mathbf{b} \leftarrow [\ell]^n, \mathbf{C} \leftarrow [\ell]^{m\times n}. \end{array} \right\}.$$

Similarly, assuming DDH in prime-order groups, for any polynomials $m(\lambda)$, $n(\lambda)$, the following holds:

$$\left\{ G, g, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{a}\cdot\mathbf{b}^T} \;\middle|\; \begin{array}{r} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ \mathbf{a} \leftarrow \mathbb{Z}_p^m,\ \mathbf{b} \leftarrow \mathbb{Z}_p^n. \end{array} \right\}$$

$$\approx_c \left\{ G, g, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{C}} \;\middle|\; \begin{array}{r} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ \mathbf{a} \leftarrow \mathbb{Z}_p^m,\ \mathbf{b} \leftarrow \mathbb{Z}_p^n, \mathbf{C} \leftarrow \mathbb{Z}_p^{m\times n}. \end{array} \right\}.$$

Definition 6.8 (Circular-Power-DDH Assumption)**.** *We say the circular-power-DDH assumption holds in the NIDLS framework if the following holds:*

$$\left\{ \begin{array}{c} \mathsf{pp}, (\rho, g), g^s, g^{a_i}, g^{sa_i}, g^{s^2 a_i} \cdot f^{s[i]} \\ (\text{for } i \in \lceil \log \ell \rceil) \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ s, \{a_i\} \leftarrow [\ell]. \end{array} \right\}$$

$$\approx_c \left\{ \begin{array}{c} \mathsf{pp}, (\rho, g), g^s, g^{a_i}, g^{b_i}, g^{c_i} \\ (\text{for } i \in \lceil \log \ell \rceil) \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ s, \{a_i, b_i, c_i\} \leftarrow [\ell]. \end{array} \right\}.$$

We say the circular-power-DDH assumption holds in prime-order groups if the following holds:

$$\left\{ \begin{array}{c} G, g, g^s, g^{a_i}, g^{sa_i}, g^{s^2 a_i + s[i]} \\ (\text{for } i \in \lceil \log p \rceil) \end{array} \;\middle|\; \begin{array}{l} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ s, \{a_i\} \leftarrow \mathbb{Z}_p. \end{array} \right\}$$

$$\approx_c \left\{ \begin{array}{c} G, g, g^s, g^{a_i}, g^{b_i}, g^{c_i} \\ (\text{for } i \in \lceil \log p \rceil) \end{array} \;\middle|\; \begin{array}{l} (G, g) \leftarrow \mathsf{Gen}(1^\lambda), \\ s, \{a_i, b_i, c_i\} \leftarrow \mathbb{Z}_p. \end{array} \right\}.$$

Remark. CP-DDH implies DDH, which just requires indistinguishability of the first 3 terms in the above. We also show in Theorem 6.4 that CP-DDH in prime-order groups holds in the generic group model (GGM) as formulated in [Sho97].

The following small-exponent assumption is commonly assumed in the NIDLS framework, and is requires for obtaining HSS (for NC1 circuits) in prior work [ADOS22]. We don't rely on this assumption except when using existing HSS schemes as a black box.

Definition 6.9 (Small Exponent Assumption)**.** *We say the small exponent assumption holds in the NIDLS framework if the following holds:*

$$\left\{ \mathsf{pp}, \rho, g^{s} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^{\lambda}), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ s \leftarrow [\ell]. \end{array} \right\}$$
$$\approx_c \left\{ \mathsf{pp}, \rho, g^{s'} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^{\lambda}), \\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ s' \leftarrow [2^{\lambda}]. \end{array} \right\}.$$

While the Damgård-Jurik encryption scheme [Pai99, DJ01] satisfy the NIDLS framework, our applications can alternatively rely directly on the KDM security of this scheme, rather than CP-DDH defined generically for the NIDLS framework. We give preliminaries for Damgård-Jurik below.

Construction 20 (Damgård-Jurik Encryption [Pai99, DJ01])**.** Let $B' = B'(\lambda) \leq 2^{\mathrm{poly}(\lambda)}$ be a bound on message magnitude. The Damgård-Jurik encryption scheme for integer messages consists of the following algorithms.

- $\mathsf{KeyGen}(1^{\lambda})$: sample two λ-bit primes p, q, set $N = p \cdot q$, and choose the smallest integer ζ such that $N^{\zeta} > B'$. Output $\mathsf{pk} = (N, \zeta)$ and $\mathsf{sk} = \varphi(N)$, where $\varphi(\cdot)$ is the Euler's totient function.
- $\mathsf{Enc}(\mathsf{pk}, m \in \mathbb{Z})$: sample $r \leftarrow \mathbb{Z}^*_{N^{\zeta+1}}$, and output a ciphertext

$$c = r^{N^{\zeta}} \cdot (1+N)^{m} \bmod N^{\zeta+1}.$$

- $\mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, c)$: compute and output

$$m = \mathsf{DLog}_{(1+N)}(c^{\mathsf{sk}})/\mathsf{sk} \bmod N^{\zeta+1},$$

  where $\mathsf{DLog}_{(1+N)}$ efficiently recovers x from $(1+N)^{x} \bmod N^{\zeta+1}$.

Definition 6.10 (KDM Security [BRS03, BHHO08])**.** *A public key encryption scheme is KDM secure w.r.t. a class of functions* $\mathcal{F}$ *if for every efficient adversary* $\mathcal{A}$ *there exists a negligible function* $\text{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$*:*

$$\left|\Pr\left[\text{Exp}_{\text{KDM}}^{\mathcal{A},\mathcal{F},0}(\lambda) = 1\right] - \Pr\left[\text{Exp}_{\text{KDM}}^{\mathcal{A},\mathcal{F},1}(\lambda) = 1\right]\right| \leq \text{negl}(\lambda),$$

where the experiment $\text{Exp}_{\text{KDM}}^{\mathcal{A},\mathcal{F},b}(\lambda)$ *is as follows:*

1. *Sample public and secret keys* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and launch* $\mathcal{A}(1^\lambda, \mathsf{pk})$*.*

2. *Answer arbitrary number of queries* $f \in \mathcal{F}$ *from* $\mathcal{A}$ *with*

$$\begin{aligned} c_f &= \mathsf{Enc}(\mathsf{pk}, f(\mathsf{sk})) && \textit{if } b = 0 \\ c_f &= \mathsf{Enc}(\mathsf{pk}, 0^{|f(\mathsf{sk})|}) && \textit{if } b = 1. \end{aligned}$$

3. *In the end,* $\mathcal{A}$ *outputs a bit* b' *as the experiment result.*

Remark. We will need the class $\mathcal{F}$ to contain constant functions $f_C(\mathsf{sk}) = C$, and inverse functions $f'_M(\mathsf{sk}) = \mathsf{sk}^{-1} \bmod M$ for all $C, M \in \mathbb{N}$.

Note that standard semantic security can be viewed as a special case of KDM security where $\mathcal{F}$ contains only constant functions. The semantic security of Damgård-Jurik encryption is also known as the DCR assumption (Definition 2.9). We write KDM-DCR as a shorthand for the KDM security of Damgård-Jurik encryption.

Lemma 6.4 (DDLog algorithm for Damgård-Jurik [RS21])**.** *Let* p, q *be any distinct primes,* $\zeta \geq 1$ *be any positive integer,* $N = p \cdot q$*,* $\mathsf{pk} = (N, \zeta)$*, and* $\mathsf{sk} = \varphi(N)$*. There exists an efficient algorithm* $\mathsf{DDLog}_{N,\zeta}(\cdot)$*, such that for all* $x, y, z \in \mathbb{Z}$ *and* $c \in \text{Supp}(\mathsf{Enc}(\mathsf{pk}, y))$*, the following holds:*

$$\mathsf{DDLog}_{N,\zeta}(c^{\mathsf{sk}\cdot x+z}) \equiv \mathsf{sk} \cdot x \cdot y + \mathsf{DDLog}_{N,\zeta}(c^z) \mod N^\zeta.$$

6.3.4 CP-DDH in Generic Group Model

We first recall the definition of (Shoup's) generic group model [Sho97] as formulated in [Zha22], and then prove that our new assumption, CP-DDH in prime order groups, holds in this model (Theorem 6.4).

Definition 6.11 (GGM [Sho97, Zha22])**.** *Let $p \in \mathbb{Z}$ be a positive integer and $S \subseteq \{0,1\}^*$ be a set of strings of length bounded by some B, and cardinality at least p. In the generic group model for a cyclic group of order p, a random injective* labeling function *$L : \mathbb{Z}_p \to S$ is chosen, whose outputs $L(x)$ represents group elements g^x with respect to a fixed generator g. All parties – including the adversary and the challenger – are allowed the following queries (incurring unit cost) to a group oracle:*

- Labeling queries: *The party submits $x \in \mathbb{Z}_p$, and receives $L(x)$.*
- Group operations: *The party submits $(l_1, l_2, a_1, a_2) \in S^2 \times \mathbb{Z}_p^2$. If l_1, l_2 are valid labels for $x_1, x_2 \in \mathbb{Z}_p$, i.e. $L(x_1) = l_1, L(x_2) = l_2$, then the party receives the label $L(a_1x_1 + a_2x_2)$. Otherwise, the party receives $\perp$.*

Theorem 6.4 (CP-DDH in GGM)**.** *For every sequence of prime orders $\{p_\lambda\}_\lambda$ where $p_\lambda > 2^\lambda$, and every adversary $\mathcal{A}$ with polynomial number of queries in GGM (for groups of orders $\{p_\lambda\}$), the following holds:*

$$|\Pr[\mathcal{A}(\mathsf{Labels}_{\lambda,\mathrm{CPDDH}}) = 1] - \Pr[\mathcal{A}(\mathsf{Labels}_{\lambda,\mathrm{Rand}}) = 1]| \leq \mathrm{negl}(\lambda),$$

where the labels provided to $\mathcal{A}$ are sampled as follows:

$$\mathsf{Labels}_{\lambda,\mathrm{CPDDH}} = \left\{ \begin{array}{c|c} L(1), L(s), L(a_i), L(sa_i), L(s^2a_i + s[i]) & \multirow{2}{*}{$s, \{a_i\} \leftarrow \mathbb{Z}_{p_\lambda}$} \\ (\text{for } i \in \lceil \log p_\lambda \rceil) & \end{array} \right\},$$

$$\mathsf{Labels}_{\lambda,\mathrm{Rand}} = \left\{ \begin{array}{c|c} L(1), L(s), L(a_i), L(b_i), L(c_i) & \multirow{2}{*}{$s, \{a_i, b_i, c_i\} \leftarrow \mathbb{Z}_{p_\lambda}$} \\ (\text{for } i \in \lceil \log p_\lambda \rceil) & \end{array} \right\}.$$

Proof. We show a series of hybrid experiments whose output distribution transition from $\mathcal{A}(\mathsf{Labels}_{\lambda,\mathrm{CPDDH}})$ to $\mathcal{A}(\mathsf{Labels}_{\lambda,\mathrm{Rand}})$.

Hyb_0 : In this experiment, the challenger uniformly samples exponents $s, \{a_i\}$, queries the group oracle to obtain $\mathsf{Labels}_{\lambda,\mathrm{CPDDH}}$, and provides them to $\mathcal{A}$. The queries of $\mathcal{A}$ to the group oracle are answered by the oracle. The output of $\mathcal{A}$ is also the output of this experiment, i.e. $\mathsf{Hyb}_0 \equiv \mathcal{A}(\mathsf{Labels}_{\lambda,\mathrm{CPDDH}})$.

Hyb_1 : Instead of relying on the actual group oracle, the challenger simulates all its answers (to queries of both the challenger and the adversary) by lazily sampling a random label table L:

- Upon a labeling query of $x \in \mathbb{Z}_{p_\lambda}$: If it's not answered before, sample a random label $L(x) \leftarrow S$ and remember it. Otherwise, return $L(x)$.
- Upon a group operation query of (l_1, l_2, a_1, a_2): If either l_1 or l_2 is not in S, then return $\perp$. Otherwise, for an unqueried label, say l_1, randomly sample a previously un-queried $x_1 \in \mathbb{Z}_p$ and set $L(x_1) = l_1$. Finally, compute $x_3 = a_1 x_1 + a_2 x_2 \bmod p$ and return $L(x_3)$.

As the challenger perfectly simulates the group oracle, we have $\mathsf{Hyb}_1 \equiv \mathsf{Hyb}_0$.

Hyb_2 : Instead of simulating the group oracle as in Hyb_1, the challenger first translates every query into an affine function α over the values $s, \{a_i, sa_i, s^2 a_i + s[i]\}$, and then answers the query as $L'(\alpha)$ with a random table L' mapping distinct translated affine functions to random labels in S.

The challenger's own labeling queries of exponents among $s, \{a_i, sa_i, s^2 a_i + s[i]\}$ are translated to the affine functions that "select" the correct input variables. $\mathcal{A}$'s queries are handled as follows.

- Upon a labeling query of x: Translate it to the constant function $\alpha(\cdot) = x$.
- Upon a group operation query of (l_1, l_2, a_1, a_2): If either l_1 or l_2 is not in S, then return $\perp$. Otherwise, for an unqueried label, say l_1, randomly sample a previously un-queried constant function α_1, and set $L'(\alpha_1) = l_1$. Finally, translate the query to $\alpha_3 := a_1 \cdot \alpha_1 + a_2 \cdot \alpha_2$, which is another affine function.

We observe that the simulated answers in Hyb_2 and Hyb_1 are equivalent, unless *there exists queried affine functions* $\alpha \neq \alpha'$ *such that* $\alpha(s, \{a_i, sa_i, s^2a_i + s[i]\}) = \alpha'(s, \{a_i, sa_i, s^2a_i + s[i]\})$. We claim (and prove in the end) that this "bad event" happens with negligiable probability, because it implies a non-zero affine function $\alpha^* = \alpha - \alpha'$ satisfying $\alpha^*(s, \{a_i, sa_i, s^2a_i + s[i]\}) \equiv 0 \bmod p$:

Claim 6.1. *For every prime* p*, every non-zero affine function* α *over* $3\lceil \log p \rceil + 1$ *inputs the following holds:*

$$\Pr[\alpha(s, \{a_i, sa_i, s^2a_i + s[i]\}) \equiv 0 \bmod p \,|\, s, \{a_i\} \leftarrow \mathbb{Z}_p] \leq 3/p.$$

Therefore, we have $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_3$. We also note that in Hyb_2 the labels provided to $\mathcal{A}$, both as inputs $\mathsf{Labels}_{\lambda,\mathrm{CPDDH}}$ and as answers to its queries, are computed independent of the exponents $s, \{a_i\}$ sampled by the challenger.

Hyb_3 In this experiment, the challenger uniformly samples exponents $s, \{a_i, b_i, c_i\}$, queries the group oracle to obtain $\mathsf{Labels}_{\lambda,\mathrm{Rand}}$, and provides them to $\mathcal{A}$. The queries of $\mathcal{A}$ are answered by the oracle.

By exactly the same arguments as used to argue $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_2$ above, we have $\mathcal{A}(\mathsf{Labels}_{\lambda,\mathrm{Rand}}) \equiv \mathsf{Hyb}_3 \approx \mathsf{Hyb}_2$.

By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_3$, which proves the theorem. We prove the claim now.

of Claim. Denote the coefficients of α as $c, d, e_i, f_i, g_i \in \mathbb{Z}_p$ for $i \in \lceil \log p \rceil$:

$$\begin{aligned} &\alpha(s, \{a_i, sa_i, s^2a_i + s[i]\}) \\ :=& c + d \cdot s + \sum_i e_i \cdot a_i + \sum_i f_i \cdot sa_i + \sum_i g_i \cdot (s^2a_i + s[i]) \\ =& \underbrace{c + sd + \sum_i s[i]g_i}_{\gamma} + \sum_i a_i \underbrace{(e_i + sf_i + s^2g_i)}_{\beta_i}. \end{aligned}$$

Let "Target" denote the event $\alpha(s, \{a_i, sa_i, s^2 a_i + s[i]\}) \equiv 0 \bmod p$. We analyze two possible cases:

Case A: $\gamma, \{\beta_i\}$ don't all evaluate to zero (mod p). By Schwartz-Zippel lemma, viewing $\{a_i\}$ as variables, we have

$$\Pr[\text{Target} \,|\, \text{Case A}] \leq 1/p$$

Case B: $\gamma, \{\beta_i\}$ all evaluate to zero (mod p). As we assume α is a non-zero function, at least one of the following are true:

- Exists i^* such $e_{i^*}, f_{i^*}, g_{i^*}$ are not all zero. By Schwartz-Zippel lemma, viewing s as the variable, we have $\Pr[\text{Case B}] \leq \Pr[\beta_{i^*} = 0] \leq 2/p$.
- $\{e_i, f_i, g_i\}$ are zero for all i, but c, d are not all zero. By Schwartz-Zipple lemma, viewing s as the variable, we have $\Pr[\text{Case B}] \leq \Pr[\gamma = 0] \leq 1/p$.

In both cases, we have

$$\Pr[\text{Case B}] \leq 2/p.$$

We conclude the proof by the following identity:

$$\begin{aligned}\Pr[\text{Target}] =& \Pr[\text{Case A}]\Pr[\text{Target} \,|\, \text{Case A}] + \Pr[\text{Case B}]\Pr[\text{Target} \,|\, \text{Case B}] \\ \leq& 1 \cdot \Pr[\text{Target} \,|\, \text{Case A}] + \Pr[\text{Case B}] \cdot 1 \leq 3/p.\end{aligned}$$

□

6.4 Construction of Main Tool: Algebraic Homomorphic MACs

Our notion of algebraic homomorphic MACs (aHMACs) can be roughly viewed as a refinement of existing homomorphic MACs (HMACs [AB09, GW13, CF13]). In both notions, there are Auth, and EvalTag algorithms that respectively produce authentication tags σ_x for some input x, and homomorphically evaluate some circuit C over them. The resulting tags σ_z should be verifiable with respect to the circuit C and unforgeable.

The main difference in our definition is the requirement that evaluated tags have the the form $\sigma_z = \Delta \cdot C(\mathbf{x}) + k_C$ (over $\mathbb{Z}$), where Δ is a global secret specified at key generation time, and k_C is computable from only the secret key and the circuit C, without knowing $\mathbf{x}$. This format is known as information-theoretic MACs. As our applications in this work doesn't require explicitely verifying the evaluated tags, we omit the Verify algorithm, verifiability, and unforgeability from our main Definition 6.12.

In Section 6.4.6, we provide the standard HMAC definition and show that our aHMAC definition (Definition 6.12) implies a HMAC scheme with verifiability and a weaker variant of unforgeability than commonly required in the literature. We also show how to generically amplify a scheme with weak unforgeability to achieve the usual unforgeability. (The amplified scheme doesn't satisfy the algebraic format anymore.)

Definition 6.12 (Algebraic Homomorphic MACs)**.** *An algebraic homomorphic MAC scheme (for arbitrary Boolean circuits) consists of four efficient algorithms:*

- $\mathsf{KeyGen}(1^\lambda, \mathbf{\Delta} \in [2^\lambda]^{\ell_z})$ *takes a vector of global secrets* $\mathbf{\Delta}$*, (one for each evaluation output bit,) and outputs a secret key* sk *and evaluation key* evk*.*
- $\mathsf{Auth}(\mathsf{sk}, x \in \{0,1\}, \mathsf{id} \in \{0,1\}^\lambda)$ *takes as inputs the secret key* sk*, a bit* x *to be authenticated, and an* id *associated with the bit. It outputs a tag* σ_x*. We will write* $\sigma_{\mathbf{x}} = (\ldots, \sigma_x^{(i)}, \ldots)$ *to mean a vector of such tags.*
- $\mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$ *takes as inputs the evaluation key* evk*, a Boolean circuit* $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$*, input bits* $\mathbf{x}$*, and their associated tags* $\sigma_{\mathbf{x}}$*. It outputs tags* $\sigma_{\mathbf{z}} \in \mathbb{Z}^{\ell_z}$ *authenticating the outputs of* $C(\mathbf{x})$*.*
- $\mathsf{EvalKey}(\mathsf{sk}, C, \mathbf{id})$ *takes as inputs the secret key* sk*, a Boolean circuit* $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$ *and the* id*s associated with its inputs. It outputs MAC keys* $\mathbf{k}_C \in \mathbb{Z}^{\ell_z}$ *for the outputs of* C*.*

δ-Correctness: *Let* $\delta = \delta(\lambda)$ *be an error bound. For every polynomial* $p(\lambda)$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$*, Boolean circuits* $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$

where $|C| \leq p(\lambda)$*, global secrets* $\mathbf{\Delta} \in [2^\lambda]^{\ell_z}$*, inputs* $\mathbf{x} \in \{0,1\}^{\ell_x}$*, and ids* $\textsf{\textbf{id}} \in \{0,1\}^{\ell_x \times \lambda}$*, the following holds:*

$$\Pr\left[\begin{array}{l|l} & (\mathsf{sk},\mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathbf{\Delta}) \\ \sigma_{\mathbf{z}} = \mathbf{\Delta} \odot C(\mathbf{x}) + \mathbf{k}_C & \sigma^{(i)} \leftarrow \mathsf{Auth}(\mathsf{sk}, \mathbf{x}[i], \textsf{\textbf{id}}[i]) \\ (\text{over } \mathbb{Z}^{\ell_z}, \odot \text{ means} & \sigma_{\mathbf{x}} := (\sigma^{(0)}, \ldots, \sigma^{(\ell_x - 1)}) \\ \text{component-wise mult}) & \sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}}) \\ & \mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \textsf{\textbf{id}}) \end{array}\right] \geq 1 - \delta(\lambda) - \text{negl}(\lambda).$$

(Adaptive) Security: *There exists a pair of efficient simulators* $\mathsf{Sim}_1, \mathsf{Sim}_2$ *such that for every efficient adversary* $\mathcal{A}$*, there exists a negligible function* $\text{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$*, the following holds*

$$\left|\Pr[\text{Exp}_{\text{priv}}^{\mathcal{A},0}(\lambda) = 1] - \Pr[\text{Exp}_{\text{priv}}^{\mathcal{A},1}(\lambda) = 1]\right| \leq \text{negl}(\lambda),$$

where the experiment $\text{Exp}_{\text{priv}}^{\mathcal{A},b}$ *is as follows:*

1. *Launch* $\mathcal{A}(1^\lambda)$*. Receive from* $\mathcal{A}$ *a vector* $\mathbf{\Delta} \in [2^\lambda]^{\ell_z}$*, compute* evk *as follows and send it to* $\mathcal{A}$*.*

$$\begin{cases} \mathsf{evk}, \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathbf{\Delta}) & \textit{if } b = 0, \\ \mathsf{evk}, \mathsf{st} \leftarrow \mathsf{Sim}_1(1^{\ell_z}) & \textit{if } b = 1, \end{cases}$$

2. *Receive any number of adaptively chosen queries* $(x \in \{0,1\}, \mathsf{id} \in \{0,1\}^\lambda)$ *with distinct* id*s from* $\mathcal{A}$*, and answer each with a tag* σ *computed as follows.*

$$\begin{cases} \sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id}) & \textit{if } b = 0, \\ \sigma, \mathsf{st} \leftarrow \mathsf{Sim}_2(\mathsf{st}) & \textit{if } b = 1. \end{cases}$$

   (Queries with previously used id*s are ignored.)*

3. $\mathcal{A}$ *outputs a bit* b' *as the output of the experiment.*

Succinctness: *The tags produced by* Auth *and* $\mathsf{EvalTag}$ *have bounded sizes by some fixed* $\mathrm{poly}(\lambda)$.

Remark. The adaptive security definition implies the following weaker, but simpler, selective security that suffices for our applications. *There exists an efficient simulator* Sim *such that for every sequence of global MAC keys* $\{\mathbf{\Delta}_\lambda\}_\lambda$ *and inputs* $\{\mathbf{x}_\lambda\}_\lambda$ *(of polynomial lengths* $\ell_z(\lambda)$ *and* $\ell_x(\lambda)$*), and* distinct *ids* $\{\mathbf{id}_\lambda\}_\lambda$*, the following holds*

$$\left\{\mathsf{Sim}(1^\lambda, 1^{\ell_z})\right\}_\lambda \approx_c \left\{\mathsf{evk}, \sigma_{\mathbf{x}} := (\sigma^{(i)}, \ldots, \sigma^{(\ell_x-1)}) \;\middle|\; \begin{array}{l} (\mathsf{sk},\mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathbf{\Delta}) \\ \sigma^{(i)} \leftarrow \mathsf{Auth}(\mathsf{sk}, \mathbf{x}[i], \mathbf{id}[i]) \end{array}\right\}_\lambda .$$

We prove the stronger adaptive version for our constructions this section, but use the simpler version in our applications to succinct partial garbling (Section 6.5) and constrained PRF (Section 6.6).

Remark. We define a leveled variant analogously to Definition 6.12 with the following differences.

- KeyGen additionally takes a depth-bound D on evaluation circuits as 1^D.
- $\mathsf{EvalTag}$ and $\mathsf{EvalKey}$ take circuits of depth less than D, and correctness only holds w.r.t. those circuits. (We don't require the circuits to be leveled.)
- Succinctness requires the bit-lengths of tags produced by Auth and $\mathsf{EvalTag}$ to be bounded by some fixed $\mathrm{poly}(\lambda, D)$.

In Section 6.4.1, 6.4.2, we show constructions of aHMACs assuming CP-DDH in the NIDLS framework or in prime-order groups. In Section 6.4.3, we describe a construction assuming KDM-DCR (in Damgård-Jurik groups). In Section 6.4.4, we describe leveled variants of these constructions assuming DDH in the NIDLS framework or in prime-order groups. (A leveled variant assuming DCR is described in a recent concurrent work [MORS25], we only include this in our theorem statement for completeness, but don't claim any credit. Our leveled constructions assuming DDH are also inspired by the techniques from [MORS25].) In Section 6.4.6 we show that our aHMAC definition implies a HMAC scheme with verifiability

and (a weaker variant of) unforgeability. We also show how to generically amplify weak unforgeability to standard unforgeability. Finally, in Appendix 6.4.5 we include a construction based on lattice assumptions from subsequent work [ILL25], and a leveled variant analogous to our group-based constructions for completeness.

Our results on aHMACs are summarized as follows. (See Theorem 6.8, 6.9 in Section 6.4.6 for the implied results on HMACs.)

Theorem 6.5 (aHMACs)**.** *We have the following constructions:*

1. *Assuming CP-DDH in the NIDLS framework (e.g. Damgård-Jurik groups and class groups) (Definition 6.6, 7.3), or KDM-DCR (Definition 6.10), there exists an aHMAC scheme achieving* negl*-correctness.*

2. *Assuming CP-DDH in prime-order groups, for every polynomial $p(\lambda)$, there exists an aHMAC scheme achieving $1/p$-correctness.*

In the above, evk *costs* $\ell_z \cdot \mathrm{poly}(\lambda)$ *bits.*

Theorem 6.6 (Leveled aHMACs)**.** *We have the following constructions (besides those from Theorem 6.5):*

1. *Assuming DDH in the NIDLS framework (Definition 6.7), or DCR, [4] there exists a leveled aHMAC scheme achieving* negl*-correctness.*

2. *Assuming DDH in prime-order groups, for every polynomial $p(\lambda)$, there exists a leveled aHMAC scheme achieving $1/p$-correctness.*

In the above, evk *costs* $(\ell_z + D) \cdot \mathrm{poly}(\lambda)$ *bits.*

---

[4] As noted, the leveled contruction under DCR is adapted from [MORS25].

6.4.1 aHMACs from the NIDLS Framework

In this section, we construct an aHMAC scheme in the NIDLS framework (Definition 6.6). See Section 6.2.1 for an overview and intuitions.

Construction 21 (aHMACs from the NIDLS Framework)**.** Ingredients:

- An instance $\mathcal{G} = \{\mathcal{G}_\lambda\}$ of the NIDLS framework with large order F, i.e., the subgroup F of each $G \in \mathcal{G}_\lambda$ has order at least $t > 2^\lambda$.
- Two PRFs $\mathsf{F}_1 : \mathcal{K}_1 \times \{0,1\}^* \to [t]$ and $\mathsf{F}_2 : \mathcal{K}_2 \times \{0,1\}^* \to [2^\lambda]$.

Note that every Boolean circuit C can be implemented via an arithmetic circuit C' over $\mathbb{Z}$ as follows:

$$\forall x, y \in \{0,1\}, \quad x \,\mathrm{AND}\, y = x \cdot y, \quad x \,\mathrm{OR}\, y = x + y - x \cdot y, \quad \mathrm{Not}\, x = 1 - x.$$

The wire values in C' are 0 or 1. In the following construction of $\mathsf{EvalTag}$ and $\mathsf{EvalKey}$, we will evaluate C' instead of C.

For an integer vector $\mathbf{r}$, we write $g^{\mathbf{r}} = (\ldots, g^{\mathbf{r}[i]}, \ldots)$, denote component-wise multiplication by $\odot$, and define $\mathsf{BC}(\mathbf{r}) = \sum_i \mathbf{r}[i] \cdot 2^i$ over $\mathbb{Z}$. When using a PRF to generate n values from a single input, we write $\mathsf{F}^n(s, x) = (\ldots, \mathsf{F}(s, x\|i), \ldots)_{i \in [n]}$ for vectorized operations.

$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \boldsymbol{\Delta})$:

Generate public parameters $\mathsf{pp} = (G, F, H, f, t, \ell) \leftarrow \mathsf{Gen}(1^\lambda)$, a group element $(g, \rho) \leftarrow \mathsf{Samp}(\mathsf{pp})$, and a secret exponent $s \leftarrow [\ell]$. Then compute ciphertexts $\mathsf{ct}_s, \mathsf{ct}_\Delta$ encrypting the bits of s (as a bit vector $\bar{\mathsf{s}} := \mathsf{Bits}(s)$ of length $\ell_s := \lceil \log \ell \rceil$, such that $\mathsf{BC}(\bar{\mathsf{s}}) = s$) and of $\boldsymbol{\Delta}$ (as a bit matrix in $\{0,1\}^{\ell_z \times \lambda}$):

$$\begin{aligned} \mathbf{h} = g^{\mathbf{r}}, \mathbf{r} \leftarrow [\ell]^{\ell_s}, \qquad & \mathbf{H} = g^{\mathbf{R}}, \mathbf{R} \leftarrow [\ell]^{\ell_z \times \lambda}, \\ \mathsf{ct}_s = (\mathbf{h}, \mathbf{h}^s, \mathbf{h}^{s^2} \cdot f^{\bar{\mathsf{s}}}), \quad & \mathsf{ct}_\Delta = (\mathbf{H}, \mathbf{H}^{-s} \cdot f^{\boldsymbol{\Delta}}). \end{aligned} \tag{6.4}$$

Finally, sample PRF keys $\mathsf{key}_1 \leftarrow \mathcal{K}_1$, $\mathsf{key}_2 \leftarrow \mathcal{K}_2$. Output $\mathsf{sk} = (\mathsf{pp}, \mathbf{h}, \mathbf{H}, \bar{\mathsf{s}}, \mathsf{key}_1, \mathsf{key}_2)$, and $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$.

$\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id})$:

Parse the secret exponent $\bar{\mathbf{s}}$ and the (secret) PRF key key_2 from sk. Then compute an authentication tag

$$\sigma_x = \bar{\mathbf{s}} \cdot x + \mathsf{F}_2^{\ell_s}(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathbb{Z}^{\ell_s}.$$

$\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$:

Parse pp, ciphertexts $\mathsf{ct}_s = \{\mathbf{h}, \mathbf{h}_1, \mathbf{h}_2\}, \mathsf{ct}_\Delta = \{\mathbf{H}, \mathbf{H}_1\}$, and a PRF key key_1 from evk.

1. Assign the tags $\sigma_{\mathbf{x}}$ to corresponding input wires of C', and then a tag $\sigma^{(w)}$ to every output wire w of some gate in C' (with input wires w_1, w_2 and values x_1, x_2) following the topological order:

   - For Add gates, set $\sigma^{(w)} := \sigma^{(w_1)} + \sigma^{(w_2)}$ over $\mathbb{Z}^{\ell_s}$.
   - For Mult gates, compute the output tag $\sigma^{(w)}$ as follows:

$$\mathbf{a}^{(w)} := \mathbf{h}^{\mathsf{BC}(\sigma^{(w_1)})\cdot\mathsf{BC}(\sigma^{(w_2)})} \odot \mathbf{h}_1^{-\mathsf{BC}(\sigma^{(w_1)})\cdot x_2 - \mathsf{BC}(\sigma^{(w_2)})\cdot x_1} \odot \mathbf{h}_2^{x_1 \cdot x_2},$$

$$\sigma^{(w)} := \mathsf{DDLog}(\mathsf{pp}, \mathbf{a}^{(w)}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, w) \mod t.$$

We note the following invariant: if the input tags have the form $\sigma^{(w_1)} = \bar{\mathbf{s}} \cdot x_1 + \mathbf{k}^{(w_1)}$, and $\sigma^{(w_2)} = \bar{\mathbf{s}} \cdot x_2 + \mathbf{k}^{(w_2)}$, over $\mathbb{Z}$, then the computed tag also has the form $\sigma^{(w)} = \bar{\mathbf{s}} \cdot z + \mathbf{k}^{(w)}$ over $\mathbb{Z}$. For Add gates, the invariant is immediate, with $\mathbf{k}^{(w)} := \mathbf{k}^{(w_1)} + \mathbf{k}^{(w_2)}$ over $\mathbb{Z}$. For Mult gates, we note the following core identity:

$$\begin{aligned}&\mathsf{BC}(\sigma^{(w_1)})\mathsf{BC}(\sigma^{(w_2)}) - s \cdot \left(\mathsf{BC}(\sigma^{(w_1)})x_2 + \mathsf{BC}(\sigma^{(w_2)})x_1\right) + s^2 z \\ =&\mathsf{BC}(\mathbf{k}^{(w_1)})\mathsf{BC}(\mathbf{k}^{(w_2)}) \quad \text{over } \mathbb{Z}.\end{aligned}$$

Plugging in the fact that $\mathbf{h}_1 = \mathbf{h}^s$, $\mathbf{h}_2 = \mathbf{h}^{s^2} \cdot f^{\overline{\mathbf{s}}}$, we obtain

$$\begin{aligned}
&\mathbf{a}^{(w)} = f^{\overline{\mathbf{s}} \cdot z} \cdot \mathbf{h}^{\mathsf{BC}(\mathbf{k}^{(w_1)})\mathsf{BC}(\mathbf{k}^{w_2})} \\
\Rightarrow &\mathsf{DDLog}(\mathsf{pp}, \mathbf{a}^{(w)}) = \overline{\mathbf{s}} \cdot z + \mathsf{DDLog}(\mathsf{pp}, \mathbf{h}^{\mathsf{BC}(\mathbf{k}^{(w_1)})\mathsf{BC}(\mathbf{k}^{(w_2)})}) \bmod t. \\
\Rightarrow &\sigma^{(w)} = \overline{\mathbf{s}} \cdot z + \mathbf{k}^{(w)} \bmod t, \\
&\text{w/ } \mathbf{k}^{(w)} := \mathsf{DDLog}(\mathsf{pp}, \mathbf{h}^{\mathsf{BC}(\mathbf{k}^{(w_1)})\mathsf{BC}(\mathbf{k}^{(w_2)})}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, w) \bmod t.
\end{aligned}$$

We have obtained the desired invariant mod t, and now argue it also holds over $\mathbb{Z}$. For each coordinate i, there are at most $\|\overline{\mathbf{s}} \cdot z\|_\infty \leq 1$ possible values of $\mathbf{k}^{(w)}[i]$ to break the invariant over $\mathbb{Z}$. Since $\mathbf{k}^{(w)}$ is distributed pseudorandomly mod t, due to the offset by F_1, the probability of it breaking the invariant is $\leq (1/t)^{\ell_s} = \mathrm{negl}(\lambda)$.

2. Compute the final output tags $\sigma_{\mathbf{z}} = (\ldots, \mathsf{BC}(\sigma'^{(o_j)}), \ldots)_{j \in [\ell_z]}$, where $\{o_j\}$ are the output wires of C' (with values $\{z_j\}$):

$$\begin{aligned}
\mathbf{a}'^{(o_j)} &:= \mathbf{H}[j]^{\mathsf{BC}(\sigma^{(o_j)})} \odot \mathbf{H}_1[j]^{z_j}, \\
\sigma'^{(o_j)} &= \mathsf{DDLog}(\mathsf{pp}, \mathbf{a}'^{(o_j)}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, o_j) \quad \bmod t.
\end{aligned}$$

Similarly, we note if the tags $\sigma^{(o_j)}$ have the form $\sigma^{(o_j)} = \overline{\mathbf{s}} \cdot z_j + \mathbf{k}^{(o_j)}$, then we have

$$\begin{aligned}
&\sigma'^{(o_j)} = \mathbf{\Delta}[j] \cdot z_j + \mathbf{k}'^{(o_j)} \quad \text{over } \mathbb{Z}, \\
&\text{w/ } \mathbf{k}'^{(o_j)} := \mathsf{DDLog}(\mathsf{pp}, \mathbf{H}[j]^{\mathsf{BC}(\mathbf{k}^{(o_j)})}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, o_j) \bmod t. \\
\Rightarrow &\sigma_{\mathbf{z}} = \mathbf{\Delta} \odot z_j + \mathbf{k}_C \quad \text{over } \mathbb{Z} \quad \text{w/ } \mathbf{k}_C := \mathsf{BC}(\mathbf{k}'^{(o_j)}),
\end{aligned}$$

where in the last line we abuse notations to write $\mathbf{\Delta}$ as a vector in $\mathbb{Z}^{\ell_z}$.

$\mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \mathbf{id})$:

Parse the PRF keys $\mathsf{key}_1, \mathsf{key}_2$ and group elements $\mathbf{h}, \mathbf{H}$ from sk. Then compute MAC

keys $\mathbf{k}^{(w_j)}$ associated with each input wire w_j of C':

$$\mathbf{k}^{(w_j)} = \mathsf{F}_2^{\ell_s}(\mathsf{key}_2, \mathbf{id}[j]).$$

1. Assign a MAC key to every output wire w of some gate in C' (with input wires w_1, w_2) following the topological order:

   - For Add gates, set $\mathbf{k}^{(w)} := \mathbf{k}^{(w_1)} + \mathbf{k}^{(w_2)}$ over $\mathbb{Z}^{\ell_s}$.
   - For Mult gates, compute the output MAC key $\mathbf{k}^{(w)}$ as follows:

$$\mathbf{b}^{(w)} := \mathbf{h}^{\mathsf{BC}(\mathbf{k}^{(w_1)})\cdot\mathsf{BC}(\mathbf{k}^{(w_2)})},$$

$$\mathbf{k}^{(w)} := \mathsf{DDLog}(\mathsf{pp}, \mathbf{b}^{(w)}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, w) \mod t.$$

   As noted before, we have $\sigma^{(w)} = \bar{\mathbf{s}} \cdot z + \mathbf{k}^{(w)}$ over $\mathbb{Z}$.

2. Compute the final output MAC keys $\mathbf{k}_C = (\ldots, \mathsf{BC}(\mathbf{k'}^{(o_j)}), \ldots)_{j\in\ell_z}$, where $\{o_j\}$ are the output wires of C':

$$\mathbf{b'}^{(o_j)} := \mathbf{H}[j]^{\mathsf{BC}(\mathbf{k}^{(o_j)})},$$

$$\mathbf{k'}^{(o_j)} = \mathsf{DDLog}(\mathsf{pp}, \mathbf{b'}^{(o_j)}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, o_j) \mod t.$$

   As noted before, we have $\sigma_{\mathbf{z}} = \mathbf{\Delta} \odot \mathbf{z} + \mathbf{k}_C$ over $\mathbb{Z}$ as desired.

Correctness and Efficiency: To help digest the construction, we have broken up and embedded correctness analysis as notes in the above. We note that the tags output by Auth and the $\mathsf{EvalTag}$ all have bounded magnitude by $O(t) \le O(2^\lambda)$. Hence they have bit-lengths bounded by $O(\lambda \cdot \ell_s) = \mathrm{poly}(\lambda)$, and satisfy succinctness. The evaluation key evk contains mainly the ciphertexts ct_s, ct_Δ, which are $O(\ell_s + \ell_z \times \lambda)$ group elements. In total, evk has bit-length $\ell_z \cdot \mathrm{poly}(\lambda)$.

Security: We state and prove the following security lemma.

Lemma 6.5. *Under CP-DDH in the NIDLS framework, Construction 21 is secure.*

of Lemma 6.7. The security of an aHMAC scheme (Definition 6.12) requires simulators $\mathsf{Sim}_1, \mathsf{Sim}_2$ to simulate an evaluation key evk and adaptively queried authentication tags σ.

- Sim_1 samples all components of the simulated $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$ at random. In more detail, it samples a random PRF key $\mathsf{key}_1 \leftarrow \mathcal{K}_1$, public parameters of a NIDLS group $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$, and a random group element $(g, \rho) \leftarrow \mathsf{Samp}(\mathsf{pp})$. It then samples random ciphertexts $\mathsf{ct}_s = (\widetilde{\mathbf{h}}, \widetilde{\mathbf{h}}_1, \widetilde{\mathbf{h}}_2)$, and $\mathsf{ct}_\Delta = (\widetilde{\mathbf{H}}, \widetilde{\mathbf{H}}_1)$:

$$\widetilde{\mathbf{h}} = g^{\mathbf{r}}, \mathbf{r} \leftarrow [\ell]^{\ell_s}, \quad \widetilde{\mathbf{h}}_1 = g^{\mathbf{r}_1}, \mathbf{r}_1 \leftarrow [\ell]^{\ell_s}, \quad \widetilde{\mathbf{h}}_2 = g^{\mathbf{r}_2}, \mathbf{r}_2 \leftarrow [\ell]^{\ell_s},$$
$$\widetilde{\mathbf{H}} = g^{\mathbf{R}}, \mathbf{R} \leftarrow [\ell]^{\ell_z \times \lambda}, \quad \widetilde{\mathbf{H}}_1 = g^{\mathbf{R}_1}, \mathbf{R}_1 \leftarrow [\ell]^{\ell_z \times \lambda}.$$

- Sim_2 samples the authentication tag at random $\widetilde{\sigma} \leftarrow [2^\lambda]^{\ell_s}$.

We show a series of hybrids that transitions from the real-world experiment $\mathsf{Hyb}_0 = \mathrm{Exp}^0_{\mathrm{priv}}$ in Definition 6.12 to the simulation-world experiment $\mathsf{Hyb}_5 = \mathrm{Exp}^1_{\mathrm{priv}}$.

Hyb_0 : We summarize the real-world distribution of the evaluation key $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$, where $\mathsf{ct}_s = (\mathbf{h}, \mathbf{h}_1, \mathbf{h}_2)$, and $\mathsf{ct}_\Delta = (\mathbf{H}, \mathbf{H}_1)$, and of the authentication tag σ for some query (x, id).

$$\begin{aligned}
&\mathsf{key}_1 \leftarrow \mathcal{K}_1, \quad \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda), && \\
&\mathbf{h} = g^{\mathbf{r}},\ \mathbf{h}_1 = g^{s\cdot\mathbf{r}},\ \mathbf{h}_2 = g^{s^2\cdot\mathbf{r}} \cdot f^{\overline{\mathbf{s}}}, && \big|(\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}), \mathbf{r} \leftarrow [\ell]^{\ell_s}, \\
&\mathbf{H} = g^{\mathbf{R}},\ \mathbf{H}_1 = g^{s\cdot\mathbf{R}} \cdot f^{\mathbf{\Delta}}, && \big|\mathbf{R} \leftarrow [\ell]^{\ell_z \times \lambda}, s \leftarrow [\ell]. \qquad (6.5) \\
&\sigma = \overline{\mathbf{s}} \cdot x + \mathsf{F}_2^{\ell_s}(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathbb{Z} && \big|\mathsf{key}_2 \leftarrow \mathcal{K}_2, \overline{\mathbf{s}} := \mathsf{Bits}(s). \qquad (6.6)
\end{aligned}$$

Hyb_1 : Instead of computing each tag σ as in Equation 6.6, Hyb_1 simulates it as $\widetilde{\sigma} \leftarrow [2^\lambda]^{\ell_s}$. The PRF security of F_2 ensures that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

Hyb_2 : Instead of sampling the random exponents $\mathbf{R} \leftarrow [\ell]^{\ell_z \times \lambda}$ as in Equation 6.5, Hyb_2 simulate it as $\widetilde{\mathbf{R}} = \mathbf{r}' \cdot \mathbf{r}^T$, where $\mathbf{r}' \leftarrow [\ell]^{\ell_z}$. [5] The matrix form of DDH (Lemma 6.3) in the NIDLS framework ensures that $\mathsf{Hyb}_2 \approx_c \mathsf{Hyb}_1$.

---

[5]An omitted detail (for brevity) here is the mismatch of dimensions: $\widetilde{\mathbf{R}}$ should have dimension $\ell_z \times \lambda$, but $\mathbf{r}' \cdot \mathbf{r}^T$ has dimension $\ell_z \times \ell_s$, where $\ell_s = \lceil \log \ell \rceil \geq \lambda$. We simply take $\mathbf{R}$ to be the first λ columns of $\mathbf{r}' \cdot \mathbf{r}^T$.

To summarize, in Hyb_2 the terms $\mathbf{h}$, $\mathbf{h}_1$, $\mathbf{h}_2$, $\mathbf{H}$, $\mathbf{H}_1$ are computed as:

$$\begin{array}{l|l} \mathbf{h} = g^{\mathbf{r}},\ \mathbf{h}_1 = g^{s\cdot\mathbf{r}},\ \mathbf{h}_2 = g^{s^2\cdot\mathbf{r}} \cdot f^{\overline{\mathbf{s}}}, & (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}), \mathbf{r} \leftarrow [\ell]^{\ell_s}, \\ \mathbf{H} = g^{\mathbf{r}'\cdot\mathbf{r}^T},\ \mathbf{H}_1 = g^{\mathbf{r}'\cdot(s\cdot\mathbf{r})^T} \cdot f^{\mathbf{\Delta}}, & \mathbf{r}' \leftarrow [\ell]^{\ell_z}, s \leftarrow [\ell]. \end{array}$$

In particular, $\mathbf{H}$ and $\mathbf{H}_1$ can be derived from $\mathbf{h}$, $\mathbf{h}_1$, $\mathbf{r}'$ and $\mathbf{\Delta}$.

Hyb_3 : Instead of computing $\mathbf{h}$, $\mathbf{h}_1$, $\mathbf{h}_2$ as above, Hyb_3 simulates:

$$\widetilde{\mathbf{h}} = g^{\mathbf{a}}, \quad \widetilde{\mathbf{h}}_1 = g^{\mathbf{b}}, \quad \widetilde{\mathbf{h}}_2 = g^{\mathbf{c}}, \mid (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow [\ell]^{\ell_s}.$$

CP-DDH in the NIDLS framework ensures that $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_2$.

In Hyb_3, the terms $\mathbf{H}$, $\mathbf{H}_1$ (derived from $\widetilde{\mathbf{h}}$ and $\widetilde{\mathbf{h}}_1$) becomes

$$\mathbf{H} = g^{\mathbf{r}'\cdot\mathbf{a}^T},\ \mathbf{H}_1 = g^{\mathbf{r}'\cdot\mathbf{b}^T} \cdot f^{\mathbf{\Delta}}, \mid \mathbf{r}' \leftarrow [\ell]^{\ell_z}.$$

Hyb_4 : Instead of computing $\mathbf{H}$, $\mathbf{H}_1$ as above, Hyb_4 simulates them as

$$\widetilde{\mathbf{H}} = g^{\mathbf{R}},\ \widetilde{\mathbf{H}}_1 = g^{\mathbf{R}_1} \cdot f^{\mathbf{\Delta}} \mid \mathbf{R}, \mathbf{R}_1 \leftarrow [\ell]^{\ell_z \times \lambda}.$$

The matrix form of DDH in the NIDLS framework ensures $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_3$.

Hyb_5 : Instead of computing $\widetilde{\mathbf{H}}_1$ as above, Hyb_5 simulates $\widetilde{\mathbf{H}}_1 = g^{\mathbf{R}_1}$, i.e., independent of $\mathbf{\Delta}$. The Samp algorithm (Definition 6.6) ensures

$$g^{\mathbf{R}_1} \cdot f^{\mathbf{\Delta}} \approx \mathsf{Uniform}(\langle g \rangle) \cdot f^{\mathbf{\Delta}} \equiv \mathsf{Uniform}(\langle g \rangle) \approx g^{\mathbf{R}_1},$$

where the first and last indistinguishabilities are statistical. Hence we have $\mathsf{Hyb}_5 \approx \mathsf{Hyb}_4$.

By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_5$, which proves the lemma. $\square$

6.4.2 aHMACs from Prime-Order Groups

In this section, we construct an aHMAC scheme in prime-order groups. It follows the same blue-print as Construction 21 in the NIDLS framework, but will have a 1/poly-correctness error due to the imperfect DDLog algorithm in prime-order groups.

Construction 22 (aHMACs from Prime-Order Groups)**.** Ingredients:

- Prime-order groups $\mathcal{G} = \{\mathcal{G}_\lambda\}$ with an algorithm Gen and orders $p > 2^\lambda$.
- A compatible PRF $\mathsf{F}_3 : \mathcal{K}_3 \times G \to \{0,1\}^\lambda$ used for the DDLog algorithm.
- Two PRFs $\mathsf{F}_1 : \mathcal{K}_1 \times \{0,1\}^* \to [p]$ and $\mathsf{F}_2 : \mathcal{K}_2 \times \{0,1\}^* \to [2^\lambda]$.

Compared to Construction 21, the Auth, $\mathsf{EvalTag}$, $\mathsf{EvalKey}$ algorithms are the same except DDLog now requires three additional parameters δ', B, ϕ (Lemma 7.27). We set $\delta' = \delta/O(|C|)$ so that running DDLog $O(|C|)$ times has an overall error probability bounded by δ, and $B = 1$ which equals the wire value bound when implementing C as an arithmetic circuit (as explained in Construction 21). We sample a public PRF key key_3 during KeyGen which specifies the function $\phi(\cdot) := \mathsf{F}_3(\mathsf{key}_3, \cdot)$. [6] The remaining differences are in KeyGen, where we compute $\mathsf{ct}_s, \mathsf{ct}_\Delta$ as part of evk differently:

$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \Delta)$:

Generate public parameters $\mathsf{pp} = (G, g) \leftarrow \mathsf{Gen}(1^\lambda)$ and a secret exponent $s \leftarrow \mathbb{Z}_p$. Then compute ciphertexts $\mathsf{ct}_s, \mathsf{ct}_\Delta$ encrypting the bits of s (as a bit vector $\bar{\mathsf{s}} \in \{0,1\}^{\ell_s}$) and $\mathbf{\Delta}$ (as a bit matrix in $\{0,1\}^{\ell_z \times \lambda}$):

$$\mathbf{h} = g^{\mathbf{r}}, \mathbf{r} \leftarrow \mathbb{Z}_p^{\ell_s}, \qquad \mathbf{H}, = g^{\mathbf{R}}, \mathbf{R} \leftarrow \mathbb{Z}_p^{\ell_z \times \lambda}$$

$$\mathsf{ct}_s = (\mathbf{h}, \mathbf{h}^s, \mathbf{h}^{s^2} \cdot g^{\bar{\mathsf{s}}}), \qquad \mathsf{ct}_\Delta = (\mathbf{H}, \mathbf{H}^{-s} \cdot g^{\mathbf{\Delta}}).$$

Finally, sample PRF keys $\mathsf{key}_1 \leftarrow \mathcal{K}_1$, $\mathsf{key}_2 \leftarrow \mathcal{K}_2$, $\mathsf{key}_3 \leftarrow \mathcal{K}_3$. Output $\mathsf{sk} = (\mathsf{pp}, \mathbf{h}, \mathbf{H}, \bar{\mathsf{s}}, \mathsf{key}_1, \mathsf{key}_2, \mathsf{key}_3)$, and $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1, \mathsf{key}_3)$.

---

[6]The output length of F_3 is truncated to $\lceil \log(2B/\delta') \rceil$ as required by ϕ.

Correctness, Efficiency, and Security. Correctness arguments are the same as Construction 21, except that every invokation of DDLog has a δ' correctness error. We have set $\delta' = \delta/O(|C|)$ such that the overall error probability from running DDLog $O(|C|)$ times is below δ as required. The scheme has the same asymptotic efficiency as Construction 21, i.e., with $|\mathsf{evk}| \leq \ell_z \cdot \mathrm{poly}(\lambda)$.

We state the following security lemma, whose proof is completely analogous to Lemma 6.7.

Lemma 6.6. *Under CP-DDH in prime-order groups, Construction 22 is secure.*

6.4.3 aHMACs From Damgård-Jurik

While the Damgård-Jurik encryption scheme (Construction 20) can be viewed as an instantiation of the NIDLS framework, we note that its particular structure allows a more convenient DDLog algorithm (Lemma 6.4):

$$\mathsf{DDLog}_{N,\zeta}(c^{\mathsf{sk}\cdot x+z}) \equiv \mathsf{sk}\cdot x\cdot y + \mathsf{DDLog}_{N,\zeta}(c^z) \mod N^\zeta,$$

where c is any ciphertext that decrypts to some value y, and sk is not a random exponent, but a fixed secret value $\mathsf{sk} = \varphi(N)$. We will use this DDLog variant on ciphertexts encrypting the inverse of the secret key $\mathsf{sk}^{-1} \bmod N^\zeta$, which can effectively remove a factor of sk from any tag of the form $\sigma = \mathsf{sk}\cdot x + k$ over $\mathbb{Z}$: [7]

$$\begin{aligned} &\mathsf{ct}_s \leftarrow \mathsf{DJ.Enc}(\mathsf{pk}, 1/\mathsf{sk} \bmod N^\zeta), \\ \Longrightarrow\ &\mathsf{DDLog}_{N,\zeta}(\mathsf{ct}_s^{\mathsf{sk}\cdot x+k}) \equiv \mathsf{sk}\cdot x\cdot \mathsf{sk}^{-1} + \mathsf{DDLog}_{N,\zeta}(\mathsf{ct}_s^k) \mod N^\zeta, \\ &\qquad \equiv x + \mathsf{DDLog}_{N,\zeta}(\mathsf{ct}_s^k) \mod N^\zeta. \end{aligned}$$

[7] This usage of DDLog is inspired by the technique from [MORS24].

This leads to the following evaluation procedures for a Mult gate in $\mathsf{EvalTag}$ and $\mathsf{EvalKey}$ respectively:

$\mathsf{EvalTag}$ given σ_x, σ_y computes:

$$a = \mathsf{ct}_x^{\sigma_x \cdot \sigma_y}, \quad \sigma_z^* := -\mathsf{DDLog}_{N,\zeta}(a) + \sigma_x \cdot y + \sigma_y \cdot x \bmod N^\zeta$$

$\mathsf{EvalKey}$ given k_x, k_y computes:

$$b = \mathsf{ct}_x^{k_x \cdot k_y}, \quad k_z^* := \mathsf{DDLog}_{N,\zeta}(b) \bmod N^\zeta$$

The DDLog algorithm ensures $\sigma_z^* = \mathsf{sk} \cdot x \cdot y + k_z^* \bmod N^\zeta$. The $\mathsf{EvalTag}$ and $\mathsf{EvalKey}$ algorithms then apply a common random shift to σ_z^* and k_z^* respectively to make the equality holds also over $\mathbb{Z}$. We omit further details for this construction, which are analogous to Construction 21. Security of this construction relies on the KDM security of Damgård-Jurik encryption, which ensures ct_s does not leak anything about sk.

We note that in a recent concurrent work [MORS25], very similar techniques to the above are used to achieve a primitive called semi-private offline HSS (also assuming the KDM security of Damgård-Jurik encryption). The authors also give a leveled variant of their construction assuming only the standard DCR assumption. Adapting their leveled construction leads to a leveled aHMAC scheme assuming DCR. We omit re-creating this leveled construction here, and refer readers to [MORS25] for more details. Inspired by their leveled construction based on DCR, we present leveled constructions based on DDH in the NIDLS framework and in prime-order groups in Section 6.4.4.

6.4.4 Leveled aHMACs without Circular Security Assumptions

In this section, we show leveled variants that avoid the circular security assumption CP-DDH in Construction 21 and 22, at the cost of a larger evaluation key evk with size growing linearly with the depth bound D of evaluation circuits: $|\mathsf{evk}| = (\ell_z + D) \cdot \mathrm{poly}(\lambda)$. The leveled variants assume DDH in the NIDLS framework and prime-order groups respectively.

In the following, we focus on explaining the modifications to Construction 21 and prove it secure. The modifications to Construction 22 and the security proof are analogous.

Construction 23 (Leveled aHMACs from the NIDLS Framework)**.** This construction relies on the same ingradients as Construction 21.

$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^D, \mathbf{\Delta})$: Compared to Construction 21, the only difference is that the ciphertexts $\mathsf{ct}_s, \mathsf{ct}_\Delta$ (Equation 6.4) are replaced with per-level ciphertexts $\mathsf{ct}^{(j)}$ for $j \in [D]$, and a final one ct_Δ computed as follows. First, sample *two* secret exponents per level

$$\forall j = 0, \dots, D, \quad s_L^{(j)}, s_R^{(j)} \leftarrow [\ell].$$

Then compute the ciphertexts $\{\mathsf{ct}^{(j)}\}$ and ct_Δ.

$$\begin{aligned} \forall j \in [D], \qquad & \mathbf{h} = g^{\mathbf{r}}, \text{ for fresh } \mathbf{r} \leftarrow [\ell]^{2\ell_s}, \\ & \mathsf{ct}^{(j)} := (\mathbf{h}, \mathbf{h}^{s_L^{(j)}}, \mathbf{h}^{s_R^{(j)}}, \mathbf{h}^{s_L^{(j)} \cdot s_R^{(j)}} \cdot f^{\mathsf{Bits}(s_L^{(j+1)}, s_R^{(j+1)})}), \\ \text{for } j = D, \qquad & \mathbf{H} = g^{\mathbf{R}}, \text{ for fresh } \mathbf{R} \leftarrow [\ell]^{\ell_z \times \lambda}, \\ & \mathsf{ct}_\Delta := (\mathbf{H}, \mathbf{H}^{-s_L^{(D)}} \cdot f^{\mathbf{\Delta}}). \end{aligned}$$

$\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id})$: Compared to Construction 21, the only difference is that the secret vector $\bar{\mathbf{s}}$ used to be the bits of a global secret s, but now is the bits of the level-0 secrets:

$$\bar{\mathbf{s}} = \bar{\mathbf{s}}^{(0)} := \mathsf{Bits}(s_L^{(0)}, s_R^{(0)}).$$

$\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$: Compared to Construction 21, there are two overall differences:

- Each tag assigned to an intermediate wire w, of depth j and with value x, used to have the form $\bar{\mathbf{s}} \cdot x + \mathbf{k}^{(w)}$ for a global secret vector $\bar{\mathbf{s}}$, but now will have the form $\bar{\mathbf{s}}^{(j)} \cdot x + \mathbf{k}^{(w)}$ for a per-level secret vector $\bar{\mathbf{s}}^{(j)} := \mathsf{Bits}(s_L^{(j)}, s_R^{(j)})$.
- Before evaluating a gate, an additional step is required to ensure the tags on both input wires have the same-level secret vector.

We first present evaluation procedures for Add and Mult gates assuming both input tags $\sigma^{(w_1)}, \sigma^{(w_2)}$ have the same level-j secret vector.

- For Add gates, set $\sigma^{(w)} := \sigma^{(w_1)} + \sigma^{(w_2)}$ over $\mathbb{Z}^{2\ell_s}$.

Note that if the input tags have the form $\sigma^{(w_1)} = \bar{\mathbf{s}}^{(j)} \cdot x_1 + \mathbf{k}^{(w_1)}$, and $\sigma^{(w_2)} = \bar{\mathbf{s}}^{(j)} \cdot x_2 + \mathbf{k}^{(w_2)}$ over $\mathbb{Z}$, then the computed tag also has the form $\sigma^{(w)} = \bar{\mathbf{s}}^{(j)} \cdot z + \mathbf{k}^{(w)}$ over $\mathbb{Z}$, where $\mathbf{k}^{(w)} = \mathbf{k}^{(w_1)} + \mathbf{k}^{(w_2)}$.

- For Mult gates, parse $\sigma^{(w_1)} = (\sigma_L^{(w_1)}, \sigma_R^{(w_1)})$, $\sigma^{(w_2)} = (\sigma_L^{(w_2)}, \sigma_R^{(w_2)})$, and $\mathsf{ct}^{(j)} = (\mathbf{h}, \mathbf{h}_{1,L}, \mathbf{h}_{1,R}, \mathbf{h}_2)$. Compute

$$\mathbf{a}^{(w)} := \mathbf{h}^{\mathsf{BC}(\sigma_L^{(w_1)}) \cdot \mathsf{BC}(\sigma_R^{(w_2)})} \odot \mathbf{h}_{1,R}^{-\mathsf{BC}(\sigma_L^{(w_1)}) \cdot x_2} \odot \mathbf{h}_{1,L}^{-\mathsf{BC}(\sigma_R^{(w_2)}) \cdot x_1} \odot \mathbf{h}_2^{x_1 \cdot x_2},$$
$$\sigma^{(w)} := \mathsf{DDLog}(\mathsf{pp}, \mathbf{a}^{(w)}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, w) \mod t.$$

We show if the input tags have the form $\sigma^{(w_1)} = \bar{\mathbf{s}}^{(j)} \cdot x_1 + \mathbf{k}^{(w_1)}$, and $\sigma^{(w_2)} = \bar{\mathbf{s}}^{(j)} \cdot x_2 + \mathbf{k}^{(w_2)}$ over $\mathbb{Z}$, then the computed tag has the form $\sigma^{(w)} = \bar{\mathbf{s}}^{(j+1)} \cdot z + \mathbf{k}^{(w)}$ over $\mathbb{Z}$, with the level-$(j+1)$ secret vector. We rely on the following core identity:

$$\begin{aligned} &\mathsf{BC}(\sigma_L^{(w_1)})\mathsf{BC}(\sigma_R^{(w_2)}) - s_R^{(j)} \cdot \big(\mathsf{BC}(\sigma^{(w_1)})x_2\big) - s_L^{(j)} \cdot \big(\mathsf{BC}(\sigma^{(w_2)})x_1\big) + s_L^{(j)} s_R^{(j)} z \\ =&\mathsf{BC}(\mathbf{k}_L^{(w_1)})\mathsf{BC}(\mathbf{k}_R^{(w_2)}) \quad \text{over } \mathbb{Z}. \end{aligned}$$

Plugging in $\mathbf{h}_{1,L} = \mathbf{h}^{s_L^{(j)}}$, $\mathbf{h}_{1,R} = \mathbf{h}^{s_R^{(j)}}$, and $\mathbf{h}_2 = \mathbf{h}^{s_L^{(j)} s_R^{(j)}} \cdot f^{\bar{\mathbf{s}}^{(j+1)}}$, we obtain

$$\begin{aligned} &\mathbf{a}^{(w)} = f^{\bar{\mathbf{s}} \cdot z} \cdot \mathbf{h}^{\mathsf{BC}(\mathbf{k}_L^{(w_1)})\mathsf{BC}(\mathbf{k}_R^{(w_2)})} \\ \Rightarrow &\mathsf{DDLog}(\mathsf{pp}, \mathbf{a}^{(w)}) = \bar{\mathbf{s}} \cdot z + \mathsf{DDLog}(\mathsf{pp}, \mathbf{h}^{\mathsf{BC}(\mathbf{k}_L^{(w_1)})\mathsf{BC}(\mathbf{k}_R^{(w_2)})}) \bmod t. \\ \Rightarrow &\sigma^{(w)} = \bar{\mathbf{s}}^{(j+1)} \cdot z + \mathbf{k}^{(w)} \bmod t, \\ &\text{w/ } \mathbf{k}^{(w)} := \mathsf{DDLog}(\mathsf{pp}, \mathbf{h}^{\mathsf{BC}(\mathbf{k}_L^{(w_1)})\mathsf{BC}(\mathbf{k}_R^{(w_2)})}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, w) \bmod t. \end{aligned}$$

We have showed the desired invariant mod t. By the same argument as in Construction 21, it also holds over $\mathbb{Z}$ except with negligible probabiblity.

We can now transform any level-j tag to a level-$(j+1)$ tag of the same value by applying the described Mult procedure with another level-j tag of the constant value 1. We can

obtain level-j tag of 1 for all levels, by starting from an arbitray input tag (of level-0) and squaring it j times.

$\mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \textbf{\textsf{id}})$: As in Construction 21, perform matching evaluations over MAC keys in the same order as $\mathsf{EvalTag}$. We present evaluation procedures for Add and Mult gates assuming the input MAC keys are $\mathbf{k}^{(w_1)}, \mathbf{k}^{(w_2)}$, and the output wire w has depth j.

- For Add gates, set $\mathbf{k}^{(w)} := \mathbf{k}^{(w_1)} + \mathbf{k}^{(w_2)}$ over $\mathbb{Z}^{2\ell_s}$.

  As noted in $\mathsf{EvalTag}$, the matching evaluated tag equals $\sigma^{(w)} = \bar{\mathbf{s}}^{(j)} z + \mathbf{k}^{(w)}$ over $\mathbb{Z}$.

- For Mult gates, parse $\mathbf{k}^{(w_1)} = (\mathbf{k}_L^{(w_1)}, \mathbf{k}_R^{(w_1)})$, and $\mathbf{k}^{(w_2)} = (\mathbf{k}_L^{(w_2)}, \mathbf{k}_R^{(w_2)})$. Read $\mathbf{h}^{(j)}$ from sk, and compute

$$\mathbf{b}^{(w)} := \left(\mathbf{h}^{(j)}\right)^{\mathsf{BC}(\mathbf{k}_L^{(w_1)}) \cdot \mathsf{BC}(\mathbf{k}^{(w_2)}{}_R)},$$
$$\mathbf{k}^{(w)} := \mathsf{DDLog}(\mathsf{pp}, \mathbf{b}^{(w)}) + \mathsf{F}_1^{\ell_s}(\mathsf{key}_1, w) \quad \bmod t.$$

  As noted in $\mathsf{EvalTag}$, the matching evaluated tag equals $\sigma^{(w)} = \bar{\mathbf{s}}^{(j+1)} z + \mathbf{k}^{(w)}$ over $\mathbb{Z}$.

Correctness, Efficiency, and Security. As before, we have broken up and embedded correctness analysis as notes in the above. Compared to Construction 21, the leveled construction has a larger evk consisting of per-level ciphertexts $\{\mathsf{ct}^{(j)}\}_{[D]}$ of poly(λ) bits each, and a final one ct_Δ of $\ell_z \cdot$ poly(λ) bits. In total, the bit-length of evk is bounded by $(D + \ell_z) \cdot$ poly(λ). Finally, we state and prove the following security lemma.

Lemma 6.7. *Under DDH in the NIDLS framework, Construction 23 is secure.*

Proof. The security of an aHMAC scheme (Definition 6.12) requires a pair of simulators $\mathsf{Sim}_1, \mathsf{Sim}_2$ to simulate an evaluation key evk and adaptively queried authentication tags σ.

- Sim_1 samples all components of the simulated $\mathsf{evk} = (\mathsf{pp}, \{\mathsf{ct}^{(j)}\}_{[D]}, \mathsf{ct}_\Delta, \mathsf{key}_1)$ at random. In more detail, it samples a random PRF key $\mathsf{key}_1 \leftarrow \mathcal{K}_1$, public parameters of a NIDLS group $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$, and a random group element $(g, \rho) \leftarrow \mathsf{Samp}(\mathsf{pp})$. It then samples random ciphertexts $\mathsf{ct}^{(j)} = (\widetilde{\mathbf{h}}^{(j)}, \widetilde{\mathbf{h}}^{(j)}_{1,L}, \widetilde{\mathbf{h}}^{(j)}_{1,R}, \widetilde{\mathbf{h}}^{(j)}_2)$, and $\mathsf{ct}_\Delta = (\widetilde{\mathbf{H}}, \widetilde{\mathbf{H}}_1)$:

$$\forall j \in [D],\ \widetilde{\mathbf{h}}^{(j)} = g^{\mathbf{r}^{(j)}},\ \widetilde{\mathbf{h}}^{(j)}_{1,L} = g^{\mathbf{r}^{(j)}_{1,L}},\ \widetilde{\mathbf{h}}^{(j)}_{1,R} = g^{\mathbf{r}^{(j)}_{1,R}},\ \widetilde{\mathbf{h}}^{(j)}_2 = g^{\mathbf{r}^{(j)}_2}, \quad \mathbf{r}^{(j)}, \mathbf{r}^{(j)}_{1,L}, \mathbf{r}^{(j)}_{1,R}, \mathbf{r}^{(j)}_2 \leftarrow [\ell]^{2\ell_s},$$
$$\widetilde{\mathbf{H}} = g^{\mathbf{R}},\ \widetilde{\mathbf{H}}_1 = g^{\mathbf{R}_1} \qquad \mathbf{R}, \mathbf{R}_1 \leftarrow [\ell]^{\ell_z \times \lambda}.$$

- Sim_2 samples the authentication tag at random $\widetilde{\sigma} \leftarrow [2^\lambda]^{2\ell_s}$.

We show a series of hybrids that transitions from the real-world experiment $\mathsf{Hyb}_0 = \mathrm{Exp}^0_{\mathrm{priv}}$ in Definition 6.12 to the simulation-world experiment $\mathsf{Hyb}_3 = \mathrm{Exp}^1_{\mathrm{priv}}$.

Hyb_0 : We summarize the real-world distribution of the evaluation key $\mathsf{evk} = (\mathsf{pp}, \{\mathsf{ct}^{(j)}\}, \mathsf{ct}_\Delta, \mathsf{key}_1)$, where $\mathsf{ct}^{(j)} = (\mathbf{h}^{(j)}, \mathbf{h}^{(j)}_{1,L}, \mathbf{h}^{(j)}_{1,R}, \mathbf{h}^{(j)}_2)$, and $\mathsf{ct}_\Delta = (\mathbf{H}, \mathbf{H}_1)$, and of the authentication tag σ for some query (x, id).

$$\mathsf{key}_1 \leftarrow \mathcal{K}_1, \quad \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda), \qquad \Big|\ (\rho, g) \leftarrow \mathsf{Samp}(\mathsf{pp}),\ s^{(j)}_L, s^{(j)}_R \leftarrow [\ell],\ \forall j = 0, \ldots, D,$$

$$\forall j \in [D] : \mathbf{h}^{(j)} = g^{\mathbf{r}^{(j)}},\ \mathbf{h}^{(j)}_{1,L} = g^{s^{(j)}_L \cdot \mathbf{r}^{(j)}},\ \mathbf{h}^{(j)}_{1,R} = g^{s^{(j)}_R \cdot \mathbf{r}^{(j)}},\ \mathbf{h}^{(j)}_2 = g^{s^{(j)}_L s^{(j)}_R \cdot \mathbf{r}^{(j)}} \cdot f^{\bar{\mathbf{s}}^{(j+1)}}, \qquad \Big|\ \mathbf{r}^{(j)} \leftarrow [\ell]^{2\ell_s},\ \bar{\mathbf{s}}^{(j+1)} := \mathsf{Bits}(s^{(j+1)}_L, s^{(j+1)}_R), \tag{6.7}$$

$$\mathbf{H} = g^{\mathbf{R}},\ \mathbf{H}_1 = g^{s^{(D)}_L \cdot \mathbf{R}} \cdot f^{\mathbf{\Delta}}, \qquad \Big|\ \mathbf{R} \leftarrow [\ell]^{\ell_z \times \lambda}, \tag{6.8}$$

$$\sigma = \bar{\mathbf{s}}^{(0)} \cdot x + \mathsf{F}^{\ell_s}_2(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathbb{Z} \qquad \Big|\ \mathsf{key}_2 \leftarrow \mathcal{K}_2, \tag{6.9}$$

Hyb_1 : Instead of computing each tag σ as in Equation 6.9, Hyb_1 simulates it as $\widetilde{\sigma} \leftarrow [2^\lambda]^{\ell_s}$. The PRF security of F_2 ensures that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

$\mathsf{Hyb}_{2,0,0}$: Instead of computing $\mathbf{h}^{(0)}, \mathbf{h}^{(0)}_{1,L}, \mathbf{h}^{(0)}_{1,R}, \mathbf{h}^{(0)}_2$ as in Equation 6.7, compute them from independent random exponents as follows:

$$\mathbf{h}^{(0)} = g^{\mathbf{r}^{(0)}},\ \mathbf{h}^{(0)}_{1,L} = g^{\mathbf{r}^{(0)}_{1,L}},\ \mathbf{h}^{(0)}_{1,R} = g^{\mathbf{r}^{(0)}_{1,R}},\ \mathbf{h}^{(0)}_2 = g^{\mathbf{r}^{(0)}_2} \cdot f^{\bar{\mathbf{s}}^{(1)}}, \qquad \Big|\ \mathbf{r}^{(0)}, \mathbf{r}^{(0)}_{1,L}, \mathbf{r}^{(0)}_{1,R}, \mathbf{r}^{(0)}_2 \leftarrow [\ell]^{2\ell_s}.$$

By DDH in the NIDLS framework (Definition 6.7), we have $\mathsf{Hyb}_{2,0,0} \approx_c \mathsf{Hyb}_1$.

$\mathsf{Hyb}_{2,0,1}$**:** Instead of computing $\mathbf{h}_2^{(0)}$ as the previous hybrid, directly compute it as

$$\mathbf{h}_2^{(0)} = g^{\mathbf{r}_2^{(0)}}, \Big| \mathbf{r}_2^{(0)} \leftarrow [\ell]^{2\ell_s}.$$

without depending on the secret vector $\bar{\mathbf{s}}^{(1)}$. The Samp algorithm (Definition 6.6) ensures

$$g^{\mathbf{r}_2^{(0)}} \cdot f^{\bar{\mathbf{s}}^{(1)}} \approx \mathsf{Uniform}(\langle g \rangle) \cdot f^{\bar{\mathbf{s}}^{(1)}} \equiv \mathsf{Uniform}(\langle g \rangle) \approx g^{\mathbf{r}_2^{(0)}}.$$

Hence we have $\mathsf{Hyb}_{2,0,1} \approx \mathsf{Hyb}_{2,0,0}$.

$\mathsf{Hyb}_{2,j}$: for $j = 1, \ldots D-1$, instead of computing $\mathbf{h}^{(j)}, \mathbf{h}_{1,L}^{(j)}, \mathbf{h}_{1,R}^{(j)}, \mathbf{h}_2^{(j)}$ as in Equation 6.7, compute them from independent random exponents as follows:

$$\left.\begin{array}{l} \mathbf{h}^{(j)} = g^{\mathbf{r}^{(j)}},\ \mathbf{h}_{1,L}^{(j)} = g^{\mathbf{r}_{1,L}^{(j)}},\ \mathbf{h}_{1,R}^{(j)} = g^{\mathbf{r}_{1,R}^{(j)}}, \\ \mathbf{h}_2^{(j)} = g^{\mathbf{r}_2^{(j)}} \end{array}\right| \mathbf{r}^{(j)}, \mathbf{r}_{1,L}^{(j)}, \mathbf{r}_{1,R}^{(j)}, \mathbf{r}_2^{(j)} \leftarrow [\ell]^{2\ell_s}.$$

By analogous arguments from $\mathsf{Hyb}_{2,0,0}$ and $\mathsf{Hyb}_{2,0,1}$, we have $\mathsf{Hyb}_{2,j} \approx_c \mathsf{Hyb}_{2,j-1}$.

Hyb_3 Instead of computing $\mathbf{H}, \mathbf{H}_1$ as in Equation 6.8, compute them from independent random exponents as follows:

$$\mathbf{H} = g^{\mathbf{R}}, \mathbf{H} = g^{\mathbf{R}_1} \, \big| \mathbf{R}, \mathbf{R}_1 \leftarrow [\ell]^{\ell_z \times \lambda}.$$

By analogous arguments from $\mathsf{Hyb}_{2,0,0}$ and $\mathsf{Hyb}_{2,0,1}$, we have $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_{2,D-1}$.

By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_3$, which proves the lemma. □

6.4.5 aHMAC from Lattices

While the focus of this work are group-based constructions of algebraic homomorphic MACs, in this section we include a lattice based construction as well as a leveled variant for completeness.

The lattice construction was first presented in the subsequent work of [ILL25], based on a circular variant of the RingLWE assumption called circular-power-RingLWE (CP-RLWE), or

alternatively a leveled variant based on power-RingLWE (P-RLWE), without the circular assumption. Here we present a different leveled construction based on the more standard RingLWE assumption with small secrets (i.e. with coefficients of polynomial magnitudes). This construction is analogous to the leveled group-based constructions in Section 6.4.4, and is inspired by the techniques from the recent work of [MORS25].

Theorem 6.7 (aHMAC from Lattices). *Let* $\mathsf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ *be the public parameters specified in Construction 24. We have the following constructions:*

1. *Assuming CP-RLWE (Definition 7.5) with respect to* pp*, then there exists an aHMAC scheme for arbitrary Boolean circuits.*

2. *Assuming RLWE (Definition 6.13) with respect to* pp*, then there exists a leveled aHMAC scheme for bounded-depth Boolean circuits.*

In the above, an authentication tag (evaluated or not) costs $\mathrm{poly}(\lambda)$ *bits. In the non-leveled scheme, an* evk *costs* $\ell_z \cdot \mathrm{poly}(\lambda)$ *bits. And in the leveled scheme for bounded depth circuits by* D*, an* evk *costs* $(\ell_z + D) \cdot \mathrm{poly}(\lambda)$ *bits.*

Definition 6.13 (RingLWE). *We say the RingLWE assumption holds with respect to the ring* $\mathcal{R}(\lambda)$*, a modulus* $q(\lambda)$*, error and secret distributions* $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda)$ *if the following holds for every polynomial* $m(\lambda)$*:*

$$\left\{ \begin{array}{l|l} \mathbf{a},\ s \cdot \mathbf{a} + \mathbf{e}, & s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e}, \leftarrow \mathcal{D}_{\mathsf{err}}^m, \\ (\text{over } \mathcal{R}_{\mathrm{q}}) & \mathbf{a} \leftarrow \mathcal{R}_q^m \end{array} \right\}_\lambda \approx_c \left\{ \mathbf{a}, \mathbf{b} \leftarrow \mathcal{R}_q^m \right\}_\lambda$$

Definition 6.14 (Circular-Power-RingLWE). *We say the circular power RingLWE (CP-RLWE) assumption holds with respect to the ring* $\mathcal{R}(\lambda)$*, two modulus* $p(\lambda), q(\lambda)$ *such that* $q = p \cdot \alpha$*, error and secret distributions* $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda)$ *if the following holds for every polynomial* $m(\lambda)$*:*

$$\left\{ \begin{array}{l|l} \mathbf{a},\ s \cdot \mathbf{a} + \mathbf{e}_1,\ s^2 \cdot \mathbf{a} + \mathbf{e}_2 + s \cdot \alpha & s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e}_1, \mathbf{e}_2 \leftarrow \mathcal{D}_{\mathsf{err}}^m, \\ (\text{over } \mathcal{R}_{\mathrm{q}}) & \mathbf{a} \leftarrow \mathcal{R}_q^m \end{array} \right\}_\lambda \approx_c \left\{ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow \mathcal{R}_q^m \right\}_\lambda$$

Construction 24 (aHMAC from CP-RLWE)**.** The construction is with respect to the following RingLWE public parameters $\mathsf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$:

- a polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where $n \leq \text{poly}(\lambda)$ is a power-of-two;
- two modulus $p > \lambda^{\omega(1)}$, and $q = p \cdot \alpha$, where $\alpha > p \cdot \lambda^{\omega(1)}$.
- error and secret distributions $\mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}} \subseteq \mathcal{R}$ with coefficients bounded by poly(λ).

Additionally, it relies on a PRF $\mathsf{F} : \mathcal{K} \times \{0,1\}^* \to \mathcal{R}_p$. In the following, for a ring element $r \in \mathcal{R}$ we abuse notation to write $\mathsf{BC}(r)$ to mean viewing the coefficients of r as an integer vector over $\mathbb{Z}^n$ and applying bit-composition $\mathsf{BC}(\cdot)$ it.

$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^D, \mathbf{\Delta})$: Sample a secret ring element $s \leftarrow \mathcal{D}_{\mathsf{sk}}$, and compute ciphertexts $\mathsf{ct}_s, \mathsf{ct}_\Delta$ as follows (we abuse notations to write $\mathbf{\Delta} \in \mathcal{R}_p^{\ell_z}$ as the vector of ring elements whose coefficients encode the input $\mathbf{\Delta} \in \{0,1\}^{\ell_z \times \lambda}$):

$$\begin{aligned} & a, \leftarrow \mathcal{R}_q,\ \mathbf{a}' \leftarrow \mathcal{R}_q^{\ell_z},\ e_1, e_2 \leftarrow \mathcal{D}_{\mathsf{err}},\ \mathbf{e}' \leftarrow \mathcal{D}_{\mathsf{err}}^{\ell_z}, \\ & \mathsf{ct}_s := (a, \underbrace{s \cdot a + e_1}_{b}, \underbrace{s^2 \cdot a + e_2 - s \cdot \alpha}_{c}),\ \mathsf{ct}_\Delta := (\mathbf{a}', \underbrace{s \cdot \mathbf{a}' + \mathbf{e}' - \mathbf{\Delta} \cdot \alpha}_{\mathbf{b}'}). \end{aligned} \tag{6.10}$$

Finally, sample PRF keys $\mathsf{key}_1, \mathsf{key}_2 \leftarrow \mathcal{K}$. Output $\mathsf{sk} = (\mathsf{pp}, a, \mathbf{a}, s, \mathsf{key}_1, \mathsf{key}_2)$, and $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$.

$\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id})$: Parse the secret element s and the (secret) PRF key key_2 from sk. Then compute and output an authentication tag (technically viewing the coefficients as an integer vector over $\mathbb{Z}^n$).

$$\sigma_x = s \cdot x + \mathsf{F}(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathcal{R}.$$

$\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$: Parse pp, ciphertexts $\mathsf{ct}_s = \{a, b, c\}$, $\mathsf{ct}_\Delta = \{\mathbf{a}', \mathbf{b}'\}$, and a PRF key key_1 from evk.

1. Assign the tags $\sigma_{\mathbf{x}}$ to corresponding input wires of C', and then a tag $\sigma^{(w)}$ to every output wire w of some gate in C' (with input wires w_1, w_2 and values x_1, x_2) following the topological order:

- For Add gates, set $\sigma^{(w)} := \sigma^{(w_1)} + \sigma^{(w_2)}$ over $\mathcal{R}$.
- For Mult gates, compute the output tag $\sigma^{(w)}$ as follows:

$$d^{(w)} := a \cdot \sigma^{(w_1)} \cdot \sigma^{(w_2)} - b \cdot (\sigma^{(w_1)} \cdot x_2 + \sigma^{(w_2)} \cdot x_1) + c \cdot x_1 \cdot x_2 \text{ over } \mathcal{R}_q,$$
$$\sigma^{(w)} := \lfloor d/\alpha \rceil + \mathsf{F}(\mathsf{key}_1, w) \text{ over } \mathcal{R}_p.$$

We note the following invariant: if the input tags have the form $\sigma^{(w_1)} = s \cdot x_1 + k^{(w_1)}$, and $\sigma^{(w_2)} = s \cdot x_2 + k^{(w_2)}$, over $\mathcal{R}$, then the computed tag also has the form $\sigma^{(w)} = s \cdot z + k^{(w)}$ over $\mathcal{R}$. For Add gates, the invariant is immediate, with $k^{(w)} := k^{(w_1)} + k^{(w_2)}$ over $\mathcal{R}$. For Mult gates, we note the following core identity:

$$\sigma^{(w_1)} \cdot \sigma^{(w_2)} - s \cdot \left(\sigma^{(w_1)} x_2 + \sigma^{(w_2)} x_1\right) + s^2 z = k^{(w_1)} \cdot k^{(w_2)} \quad \text{over } \mathcal{R}_q.$$

Plugging in the fact that $b = s \cdot a + e_1$, $c = s^2 \cdot a + e_2 - s \cdot \alpha$, we obtain

$$d^{(w)} = s \cdot z \cdot \alpha + a \cdot k^{(w_1)} \cdot k^{(w_2)} + \text{error}, \text{ where}$$
$$\text{error } = e_1(\sigma^{(w_1)} x_2 + \sigma^{(w_2)} x_1) + e_2 z, \text{ and } \|\text{error}\|_\infty \leq p \cdot \text{poly}(\lambda) \ll \alpha.$$

We have shown that the error term from $d^{(w)}$ is much smaller than α. Hence the rounding step removes it, except with negligible probability. (See Lemma 1 in [BKS19].)

$$\sigma^{(w)} = \lfloor d^{(w)}/\alpha \rceil + \mathsf{F}(\mathsf{key}_1, w) = sz + \underbrace{\lfloor a k^{(w_1)} k^{(w_2)}/\alpha \rceil + \mathsf{F}(\mathsf{key}_1, w)}_{k^{(w)}} \text{ over } \mathcal{R}_p.$$

Shifting by the (pseudo-)random factor $\mathsf{F}(\mathsf{key}_1, w)$ ensures that $\sigma^{(w)} = sz + k^{(w)}$ holds over $\mathcal{R}$ except with negligible probability, as long as $\|sz\|_\infty \ll p$.

2. Compute the final output tags $\sigma_{\mathbf{z}} = (\ldots, \mathsf{BC}(\sigma'^{(o_j)}), \ldots)_{j \in [\ell_z]}$, where $\{o_j\}$ are output wires of C' (with values $\{z_j\}$):

$$d'^{(o_j)} := \mathbf{a}'[j] \cdot \sigma^{(o_j)} - \mathbf{b}'[j] \cdot z_j,$$
$$\sigma'^{(o_j)} = \lfloor d'^{(o_j)}/\alpha \rceil + \mathsf{F}(\mathsf{key}_1, o_j) \text{ over } \mathcal{R}_p$$

Similarly, we note if the tags $\sigma^{(o_j)}$ have the form $\sigma^{(o_j)} = s \cdot z_j + k^{(o_j)}$, then we have

$$\sigma'^{(o_j)} = \mathbf{\Delta}[j] \cdot z_j + k'^{(o_j)} \text{ over } \mathcal{R},$$
$$\text{w/ } k'^{(o_j)} := \lfloor \mathbf{a}'[j] \cdot k^{(o_j)}/\alpha \rfloor + \mathsf{F}(\mathsf{key}_1, o_j) \text{ over } \mathcal{R}_p$$
$$\Rightarrow \sigma_{\mathbf{z}} = \mathbf{\Delta} \odot z_j + k_C \quad \text{over } \mathcal{R} \quad \text{w/ } k_C := \mathsf{BC}(k'^{(o_j)}),$$

where in the last line we abuse notations to write $\mathbf{\Delta}$ as a vector in $\mathbb{Z}^{\ell_z}$.

$\mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \mathbf{id})$: Parse the PRF keys $\mathsf{key}_1, \mathsf{key}_2$ and ring elements $a, \mathbf{a}'$ from sk. Then compute MAC keys $k^{(w_j)}$ associated with each input wire w_j of C':

$$k^{(w_j)} = \mathsf{F}(\mathsf{key}_2, \mathbf{id}[j]).$$

1. Assign a MAC key to every output wire w of some gate in C' (with input wires w_1, w_2) following the topological order:
   - For Add gates, set $k^{(w)} := k^{(w_1)} + k^{(w_2)}$ over $\mathcal{R}$.
   - For Mult gates, compute the output MAC key $k^{(w)}$ as follows:

$$k^{(w)} = \lfloor a k^{(w_1)} k^{(w_2)}/\alpha \rfloor + \mathsf{F}(\mathsf{key}_1, w) \text{ over } \mathcal{R}_p.$$

As noted before, we have $\sigma^{(w)} = s \cdot z + k^{(w)}$ over $\mathcal{R}$.

2. Compute the final output MAC keys $\mathbf{k}_C = (\ldots, \mathsf{BC}(k'^{(o_j)}), \ldots)_{j \in \ell_z}$, where $\{o_j\}$ are the output wires of C':

$$k'^{(o_j)} = \lfloor \mathbf{a}'[j] \cdot k^{(o_j)}/\alpha \rfloor + \mathsf{F}(\mathsf{key}_1, o_j) \text{ over } \mathcal{R}_p.$$

As noted before, we have $\sigma_{\mathbf{z}} = \mathbf{\Delta} \odot \mathbf{z} + \mathbf{k}_C$ over $\mathcal{R}$ as desired.

Correctness, Efficiency, and Security. As in the group-based constructions, we have broken up and embedded correctness analysis as notes in the above. We note that the tags

output by Auth and the $\mathsf{EvalTag}$ all have bounded coefficients by $O(2^\lambda)$. Hence they have bit-lengths bounded by $O(\lambda \cdot n) = \mathrm{poly}(\lambda)$, and satisfy succinctness. The evaluation key evk contains mainly the ciphertexts ct_s, ct_Δ, which are $O(\ell_z)$ ring elements. In total, evk has bit-length $\ell_z \cdot \mathrm{poly}(\lambda)$.

We state and prove the following security lemma.

Lemma 6.8. *Under CP-RLWE with respect to the public parameters* $\mathsf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ *in Construction 24, then Construction 24 is a secure aHMAC scheme.*

Proof. The security of an aHMAC scheme (Definition 6.12) requires simulators $\mathsf{Sim}_1, \mathsf{Sim}_2$ to simulate an evaluation key evk and adaptively queried authentication tags σ.

- Sim_1 samples all components of the simulated $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$ at random. In more detail, it samples a random PRF key $\mathsf{key}_1 \leftarrow \mathcal{K}_1$, and random ciphertexts $\mathsf{ct}_s = (a, b, c)$, and $\mathsf{ct}_\Delta = (\mathbf{a}', \mathbf{b}')$:

$$a, b, c \leftarrow \mathcal{R}_q, \quad \mathbf{a}', \mathbf{b}' \leftarrow \mathcal{R}_q^{\ell_z}.$$

- Sim_2 samples the authentication tag at random $\widetilde{\sigma} \leftarrow [2^\lambda]^n$.

We show a series of hybrids that transitions from the real-world experiment $\mathsf{Hyb}_0 = \mathrm{Exp}^0_{\mathrm{priv}}$ in Definition 6.12 to the simulation-world experiment $\mathsf{Hyb}_3 = \mathrm{Exp}^1_{\mathrm{priv}}$.

Hyb_0 : We summarize the real-world distribution of the evaluation key $\mathsf{evk} = (\mathsf{pp}, \mathsf{ct}_s, \mathsf{ct}_\Delta, \mathsf{key}_1)$, where $\mathsf{ct}_s = (\mathbf{h}, \mathbf{h}_1, \mathbf{h}_2)$, and $\mathsf{ct}_\Delta = (\mathbf{H}, \mathbf{H}_1)$, and of the authentication tag σ for some query (x, id).

$$\mathsf{key}_1 \leftarrow \mathcal{K}_1,$$

$$\begin{aligned} & a \leftarrow \mathcal{R}_q,\ b = s \cdot a + e_1,\ c = s^2 \cdot a + e_2 + s \cdot \alpha && \Big|\, s \leftarrow \mathcal{D}_{\mathsf{sk}}, e_1, e_2 \leftarrow \mathcal{D}_{\mathsf{err}}, \\ & \mathbf{a}' \leftarrow \mathcal{R}_q^{\ell_z},\ \mathbf{b}' = s \cdot \mathbf{a}' + \mathbf{e}' - \mathbf{\Delta} \cdot \alpha, && \Big|\, \mathbf{e}' \leftarrow \mathcal{D}_{\mathsf{err}}^{\ell_z}, \end{aligned} \tag{6.11}$$

$$\sigma = s \cdot x + \mathsf{F}(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathcal{R} \qquad \big|\, \mathsf{key}_2 \leftarrow \mathcal{K}_2, \tag{6.12}$$

Hyb_1 : Instead of computing each tag σ as in Equation 6.12, Hyb_1 simulates it as $\widetilde{\sigma} \leftarrow [2^\lambda]^{\ell_s}$.
The PRF security of F_2 ensures that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

Hyb_2 : Instead of computing $b, c, \mathbf{b}'$ as in Equation 6.11, sample them directly at random:

$$a \leftarrow \mathcal{R}_q,\ b \leftarrow \mathcal{R}_q,\ c \leftarrow \mathcal{R}_q$$
$$\mathbf{a}' \leftarrow \mathcal{R}_q^{\ell_z},\ \mathbf{b}' \leftarrow \mathcal{R}_q^{\ell_z}.$$

CP-RLWE ensures that $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_2$.

By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_3$, which proves the lemma. □

Construction 25 (Leveled aHMAC from RLWE)**.** This construction relies on the same ingradients as Construction 24.

$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^D, \boldsymbol{\Delta})$: Compared to Construction 24, the only difference is that the ciphertexts $\mathsf{ct}_s, \mathsf{ct}_\Delta$ (Equation 6.10) are replaced with per-level ciphertexts $\mathsf{ct}^{(j)}$ for $j \in [D]$, and a final one ct_Δ computed as follows. First, sample *two* secret exponents per level

$$\forall j = 0, \ldots, D, \quad s_L^{(j)}, s_R^{(j)} \leftarrow \mathcal{D}_{\mathsf{sk}}.$$

Then compute the ciphertexts $\{\mathsf{ct}^{(j)}\}$ and ct_Δ.

$$\begin{aligned}
\forall j \in [D], \quad & \mathbf{a}^{(j)} \leftarrow \mathcal{R}_q^2,\ \mathbf{e}_{1,L}^{(j)}, \mathbf{e}_{1,R}^{(j)} \leftarrow \mathcal{D}_{\mathsf{err}}^2,\ \mathbf{e}_2^{(j)} \leftarrow \mathcal{R}_p \\
& \mathsf{ct}^{(j)} := \big(\mathbf{a}^{(j)}, s_L^{(j)}\mathbf{a}^{(j)} + \mathbf{e}_{1,L}^{(j)}, s_R^{(j)}\mathbf{a}^{(j)} + \mathbf{e}_{1,R}^{(j)}, \\
& \qquad s_L^{(j)} \cdot s_R^{(j)}\mathbf{a} + \mathbf{e}_2^{(j)} + (s_L^{(j+1)}, s_R^{(j+1)}) \cdot \alpha\big), \\
\text{for } j = D, \quad & \mathbf{a}' \leftarrow \mathcal{R}_q^{\ell_z},\ \mathbf{e}' \leftarrow \mathcal{D}_{\mathsf{err}}^{\ell_z}, \\
& \mathsf{ct}_\Delta := (\mathbf{a}', s_L^{(D)} \cdot \mathbf{a}' + \mathbf{e}' + \boldsymbol{\Delta} \cdot \alpha).
\end{aligned}$$

$\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id})$: Compared to Construction 24, the only difference is that the global secret s now becomes the level-0 secrets:

$$\sigma_x = (s_L^{(0)}, s_R^{(0)}) \cdot x + \mathsf{F}^2(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathcal{R}^2.$$

$\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$: Compared to Construction 24, there are two overall differences:

- Each tag assigned to an intermediate wire w, of depth j and with value x, used to have the form $s \cdot x + k^{(w)}$ for a global secret s, but now will have the form $\mathbf{s}^{(j)} \cdot x + \mathbf{k}^{(w)}$ for a per-level secret vector $\mathbf{s}^{(j)} := (s_L^{(j)}, s_R^{(j)})$.
- Before evaluating a gate, an additional step is required to ensure the tags on both input wires have the same-level secret vector.

We first present evaluation procedures for Add and Mult gates assuming both input tags $\sigma^{(w_1)}, \sigma^{(w_2)}$ have the same level-j secret vector.

- For Add gates, set $\sigma^{(w)} := \sigma^{(w_1)} + \sigma^{(w_2)}$ over $\mathcal{R}^2$.

Note that if the input tags have the form $\sigma^{(w_1)} = \mathbf{s}^{(j)} \cdot x_1 + \mathbf{k}^{(w_1)}$, and $\sigma^{(w_2)} = \mathbf{s}^{(j)} \cdot x_2 + \mathbf{k}^{(w_2)}$ over $\mathcal{R}$, then the computed tag also has the form $\sigma^{(w)} = \mathbf{s}^{(j)} \cdot z + \mathbf{k}^{(w)}$ over $\mathcal{R}^2$, where $\mathbf{k}^{(w)} = \mathbf{k}^{(w_1)} + \mathbf{k}^{(w_2)}$.

- For Mult gates, parse $\sigma^{(w_1)} = (\sigma_L^{(w_1)}, \sigma_R^{(w_1)})$, $\sigma^{(w_2)} = (\sigma_L^{(w_2)}, \sigma_R^{(w_2)})$, and $\mathsf{ct}^{(j)} = (\mathbf{a}^{(j)}, \mathbf{b}_L^{(j)}, \mathbf{b}_R^{(j)}, \mathbf{c}^{(j)})$. Compute

$$\mathbf{d}^{(w)} := \mathbf{a}^{(j)} \cdot \sigma_L^{(w_1)} \cdot \sigma_R^{(w_2)} - \mathbf{b}_R^{(j)} \cdot \sigma_L^{(w_1)} \cdot x_2 - \mathbf{b}_L^{(j)} \cdot \sigma_R^{(w_2)} \cdot x_1 + \mathbf{c}^{(j)} \cdot x_1 \cdot x_2 \text{ over } \mathcal{R}_q^2,$$

$$\sigma^{(w)} := \lfloor \mathbf{d}^{(w)} / \alpha \rceil + \mathsf{F}^2(\mathsf{key}_1, w) \text{ over } \mathcal{R}_p^2.$$

We show if the input tags have the form $\sigma^{(w_1)} = \mathbf{s}^{(j)} \cdot x_1 + \mathbf{k}^{(w_1)}$, and $\sigma^{(w_2)} = \mathbf{s}^{(j)} \cdot x_2 + \mathbf{k}^{(w_2)}$ over $\mathcal{R}^2$, then the computed tag has the form $\sigma^{(w)} = \mathbf{s}^{(j+1)} \cdot z + \mathbf{k}^{(w)}$ over $\mathcal{R}^2$, with the level-$(j+1)$ secret vector. We rely on the following core identity:

$$\sigma_L^{(w_1)} \sigma_R^{(w_2)} - s_R^{(j)} \cdot \left(\sigma_L^{(w_1)} x_2\right) - s_L^{(j)} \cdot \left(\sigma_R^{(w_2)} x_1\right) + s_L^{(j)} s_R^{(j)} z = \mathbf{k}_L^{(w_1)} \mathbf{k}_R^{(w_2)} \quad \text{over } \mathcal{R}_q.$$

Plugging in $\mathbf{b}_L^{(j)} = s_L^{(j)} \mathbf{a}^{(j)} + \mathbf{e}_{1,L}^{(j)}$, $\mathbf{b}_R^{(j)} = s_R^{(j)} \mathbf{a}^{(j)} + \mathbf{e}_{1,R}^{(j)}$, $\mathbf{c} = s_L^{(j)} s_R^{(j)} \mathbf{a}^{(j)} + \mathbf{e}_2^{(j)} +$

$\mathbf{s}^{(j+1)} \cdot \alpha$, we obtain

$$\mathbf{d}^{(w)} = \mathbf{s}^{(j+1)} \cdot z \cdot \alpha + \mathbf{a} \cdot \mathbf{k}_L^{(w_1)} \mathbf{k}_R^{(w_2)} + \text{error, where}$$

$$\text{error} = \mathbf{e}_{1,R}^{(j)} \sigma_L^{(w_1)} x_2 + \mathbf{e}_{1,L}^{(j)} \sigma_R^{(w_2)} x_1 + \mathbf{e}_2^{(j)} z \text{ and } \|\text{error}\|_\infty \leq p \cdot \text{poly}(\lambda) \ll \alpha,$$

$$\Rightarrow \lfloor \mathbf{d}^{(w)}/\alpha \rfloor + \mathsf{F}^2(\mathsf{key}_1, w) = \mathbf{s}^{(j+1)} \cdot z + \underbrace{\lfloor \mathbf{a} \cdot \mathbf{k}_L^{(w_1)} \mathbf{k}_R^{(w_2)}/\alpha \rfloor + \mathsf{F}^2(\mathsf{key}_1, w)}_{\mathbf{k}^{(w)}} \text{ over } \mathcal{R}_p.$$

$$\Rightarrow \sigma^{(w)} = \mathbf{s}^{(j+1)} \cdot z + \mathbf{k}^{(w)} \text{ over } \mathcal{R},$$

where the last two equalities hold except with negligible probability, by the same arguments as in Construction 24.

We can now transform any level-j tag to a level-$(j+1)$ tag of the same value by applying the described Mult procedure with another level-j tag of the constant value 1. We can obtain level-j tag of 1 for all levels, by starting from an arbitray input tag (of level-0) and squaring it j times.

$\mathbf{k}_C \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C, \mathbf{id})$: As in Construction 24, perform matching evaluations over MAC keys in the same order as $\mathsf{EvalTag}$. We present evaluation procedures for Add and Mult gates assuming the input MAC keys are $\mathbf{k}^{(w_1)}, \mathbf{k}^{(w_2)}$, and the output wire w has depth j.

- For Add gates, set $\mathbf{k}^{(w)} := \mathbf{k}^{(w_1)} + \mathbf{k}^{(w_2)}$ over $\mathcal{R}^2$.

  As noted in $\mathsf{EvalTag}$, the matching evaluated tag equals $\sigma^{(w)} = \mathbf{s}^{(j)} z + \mathbf{k}^{(w)}$ over $\mathcal{R}^2$.

- For Mult gates, parse $\mathbf{k}^{(w_1)} = (\mathbf{k}_L^{(w_1)}, \mathbf{k}_R^{(w_1)})$, and $\mathbf{k}^{(w_2)} = (\mathbf{k}_L^{(w_2)}, \mathbf{k}_R^{(w_2)})$. Read $\mathbf{a}^{(j)}$ from sk, and compute

$$\mathbf{k}^{(w)} := \lfloor \mathbf{a}^{(j)} \cdot \mathbf{k}_L^{(w_1)} \mathbf{k}_R^{(w_2)}/\alpha \rfloor + \mathsf{F}^2(\mathsf{key}_1, w) \text{ over } \mathcal{R}_p.$$

As noted in $\mathsf{EvalTag}$, the matching evaluated tag equals $\sigma^{(w)} = \mathbf{s}^{(j+1)} z + \mathbf{k}^{(w)}$ over $\mathcal{R}^2$.

Correctness, Efficiency, and Security. As before, we have broken up and embedded correctness analysis as notes in the above. Compared to Construction 24, the leveled construction has a larger evk consisting of per-level ciphertexts $\{\mathsf{ct}^{(j)}\}_{[D]}$ of $\mathrm{poly}(\lambda)$ bits each, and a final one ct_Δ of $\ell_z \cdot \mathrm{poly}(\lambda)$ bits. In total, the bit-length of evk is bounded by $(D + \ell_z) \cdot \mathrm{poly}(\lambda)$. Finally, we state and prove the following security lemma.

Lemma 6.9. *Under RingLWE with respect to the public parameters* $\mathsf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ *in Construction 24, Construction 25, is a secure leveled aHMAC scheme.*

Proof. The security of an aHMAC scheme (Definition 6.12) requires a pair of simulators $\mathsf{Sim}_1, \mathsf{Sim}_2$ to simulate an evaluation key evk and adaptively queried authentication tags σ.

- Sim_1 samples all components of the simulated $\mathsf{evk} = (\mathsf{pp}, \{\mathsf{ct}^{(j)}\}_{[D]}, \mathsf{ct}_\Delta, \mathsf{key}_1)$ at random. In more detail, it samples a random PRF key $\mathsf{key}_1 \leftarrow \mathcal{K}$, and random ciphertexts $\mathsf{ct}^{(j)} = (\mathbf{a}^{(j)}, \mathbf{b}_L^{(j)}, \mathbf{b}_R^{(j)}, \mathbf{c}^{(j)})$, and $\mathsf{ct}_\Delta = (\mathbf{a}', \mathbf{b}')$:

$$\forall j \in [D],\ \mathbf{a}^{(j)}, \mathbf{b}_L^{(j)}, \mathbf{b}_R^{(j)}, \mathbf{c}^{(j)} \leftarrow \mathcal{R}_q^2, \quad \mathbf{a}', \mathbf{b}' \leftarrow \mathcal{R}_q^{\ell_z}.$$

- Sim_2 samples the authentication tag at random $\widetilde{\sigma} \leftarrow [p]^{2n}$.

We show a series of hybrids that transitions from the real-world experiment $\mathsf{Hyb}_0 = \mathrm{Exp}^0_{\mathrm{priv}}$ in Definition 6.12 to the simulation-world experiment $\mathsf{Hyb}_3 = \mathrm{Exp}^1_{\mathrm{priv}}$.

Hyb_0 : We summarize the real-world distribution of the evaluation key $\mathsf{evk} = (\mathsf{pp}, \{\mathsf{ct}^{(j)}\}, \mathsf{ct}_\Delta, \mathsf{key}_1)$, where $\mathsf{ct}^{(j)} = (\mathbf{a}^{(j)}, \mathbf{b}_L^{(j)}, \mathbf{b}_R^{(j)}, \mathbf{c}^{(j)})$, and $\mathsf{ct}_\Delta = (\mathbf{a}', \mathbf{b}')$, and of the authentication tag σ

for some query (x, id).

$$\begin{array}{rll} & \mathsf{key}_1 \leftarrow \mathcal{K}, & \Big| s_L^{(j)}, s_R^{(j)} \leftarrow \mathcal{D}_{\mathsf{sk}}, \forall j = 0, \ldots, D, \\ \forall j \in [D]: & \mathbf{a}^{(j)} \leftarrow \mathcal{R}_q^2,\ \mathbf{b}_L^{(j)} = s_L^{(j)} \mathbf{a}^{(j)} + \mathbf{e}_{1,L}^{(j)}, & \Big| \mathbf{e}_{1,L}^{(j)}, \mathbf{e}_{2,L}^{(j)} \leftarrow \mathcal{D}_{\mathsf{err}}^2, \\ & \mathbf{b}_R^{(j)} = s_R^{(j)} \mathbf{a}^{(j)} + \mathbf{e}_{1,R}^{(j)}, & \Big| \mathbf{e}_2^{(j)} \leftarrow \mathcal{R}_p^2, \\ & \mathbf{c}^{(j)} = s_L^{(j)} s_R^{(j)} \mathbf{a}^{(j)} + \mathbf{e}_2^{(j)} - \mathbf{s}^{(j+1)} \cdot \alpha, & \Big| \mathbf{s}^{(j+1)} := (s_L^{(j+1)}, s_R^{(j+1)}), \end{array} \tag{6.13}$$

$$\mathbf{a}' \leftarrow \mathcal{R}_q^{\ell_z},\ \mathbf{b}' = s_L^{(D)} \mathbf{a}' + \mathbf{e}' - \mathbf{\Delta} \cdot \alpha, \quad \big| \mathbf{e}' \leftarrow \mathcal{D}_{\mathsf{err}}^{\ell_z}. \tag{6.14}$$

$$\sigma = \mathbf{s}^{(0)} \cdot x + \mathsf{F}^2(\mathsf{key}_2, \mathsf{id}) \text{ over } \mathcal{R} \quad \big| \mathsf{key}_2 \leftarrow \mathcal{K}, \tag{6.15}$$

Hyb_1 : Instead of computing each tag σ as in Equation 6.15, Hyb_1 simulates it as $\widetilde{\sigma} \leftarrow [p]^{2n}$. The PRF security of F ensures that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

$\mathsf{Hyb}_{2,0,0}$: Instead of computing $\mathbf{c}^{(0)}$ as in Equation 6.13, compute it in terms of $\mathbf{b}_R^{(0)}$:

$$\mathbf{c}^{(0)} = s_L^{(0)} \underbrace{(s_R^{(0)} \mathbf{a}^{(0)} + \mathbf{e}_{1,R}^{(0)})}_{\mathbf{b}_R^{(0)}} + \mathbf{e'}_2^{(0)} + \mathbf{e}_2^{(0)} - \mathbf{s}^{(1)} \cdot \alpha \Big| \mathbf{e'}_2^{(0)} \leftarrow \mathcal{D}_{\mathsf{err}}^2,\ \mathbf{e}_2^{(0)} \leftarrow \mathcal{R}_p^2.$$

By our setting, we have $\|s_L^{(0)} \mathbf{e}_{1,R}^{(0)}\|_\infty \ll p$ and $\|\mathbf{e'}_2^{(0)}\|_\infty \ll p$. Hence $\mathsf{Hyb}_{2,0,0} \approx \mathsf{Hyb}_1$.

$\mathsf{Hyb}_{2,0,1}$: Instead of computing $\mathbf{b}_L^{(0)}, \mathbf{b}_R^{(0)}$ as in Equation 6.13, sample them at random:

$$\left.\begin{array}{l} \mathbf{a}^{(0)}, \mathbf{b}_L^{(0)}, \mathbf{b}_R^{(0)} \leftarrow \mathcal{R}_q^2, \\ \mathbf{c}^{(0)} = s_L^{(0)} \mathbf{b}_R^{(0)} + \mathbf{e}_2' + \mathbf{e}_2 - \mathbf{s}^{(1)} \cdot \alpha, \end{array}\right| \mathbf{e}_2' \leftarrow \mathcal{D}_{\mathsf{err}}^2,\ \mathbf{e}_2 \leftarrow \mathcal{R}_p^2.$$

By RingLWE, we have $\mathsf{Hyb}_{2,0,1} \approx_c \mathsf{Hyb}_{2,0,0}$.

$\mathsf{Hyb}_{2,0,2}$**:** Instead of computing $\mathbf{c}^{(0)}$ as the previous hybrid, directly sample it at random:

$$\mathbf{c}^{(0)} \leftarrow \mathcal{R}_q^2,$$

without depending on the secret vector $\mathbf{s}^{(1)}$. By RingLWE, we have $\mathsf{Hyb}_{2,0,2} \approx \mathsf{Hyb}_{2,0,1}$.

$\mathsf{Hyb}_{2,j}$: for $j = 1, \ldots D-1$, instead of computing $\mathbf{a}^{(j)}, \mathbf{b}_L^{(j)}, \mathbf{b}_R^{(j)}, \mathbf{c}^{(j)}$ as in Equation 6.7, sample them at random:

$$\mathbf{a}^{(j)}, \mathbf{b}_L^{(j)}, \mathbf{b}_R^{(j)}, \mathbf{c}^{(j)} \leftarrow \mathcal{R}_q^2.$$

By analogous arguments from $\mathsf{Hyb}_{2,0,0}$ and $\mathsf{Hyb}_{2,0,2}$, we have $\mathsf{Hyb}_{2,j} \approx_c \mathsf{Hyb}_{2,j-1}$.

Hyb_3 Instead of computing $\mathbf{a}', \mathbf{b}'$ as in Equation 6.8, sample them at random:

$$\mathbf{a}', \mathbf{b}' \leftarrow \mathcal{R}_q^{\ell_z}.$$

By RingLWE, we have $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_{2,D-1}$.

By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_3$, which proves the lemma. □

6.4.6 Standard HMAC from aHMAC

In this section, we explain the connection between our notion of aHMAC and standard HMAC schemes [AB09, GW13, CF13] (Definition 6.15). First, we show that with minor syntactical changes, we can adapt an aHMAC scheme into a HMAC scheme satisfying 1-hop verifiability (Definition 6.16) and a weak variant of unforgeability (Definition 6.17). Then, we show that our constructions of aHMAC can actually be modified to satsify multi-hop verifiability (Definition 6.19). (We keep the main construction for 1-hop evaluation as it's more convenient for the applications considered in this paper.) Finally, we show how to amplify weak unforgeability to standard unforgeability generically. The amplified scheme don't not have the algebraic form anymore.

Definition 6.15 (Homomorphic MAC (HMAC))**.** *A homomorphic MAC scheme has the a similar syntax to an aHMAC scheme (Definition 6.12), except with a slightly different* KeyGen *algorithm, and with a* Verify *instead of* EvalKey *algorithm.*

- $\mathsf{KeyGen}(1^\lambda, 1^{\ell_z})$ *takes in an upperbound* ℓ_z *on the output length of supported evaluation circuits, and outputs a secret key* sk *and evaluation key* evk*.*
- Auth, EvalTag *has the same syntax as Definition 6.12.*

- $\mathsf{Verify}(\mathsf{sk}, C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$: *takes as inputs the secret key* sk*, a Boolean circuit* $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$*, the* id*s associated with the inputs, output bits* $\mathbf{z} \in \{0,1\}^{\ell_z}$*, and evaluated tags* $\sigma_{\mathbf{z}} \in \mathbb{Z}^{\ell_z}$ *authenticating* $\mathbf{z}$*. It outputs either* $\top$*, indicating accept, or* $\bot$*, indicating reject.*

Remark. We define a leveled variant where KeyGen additionally takes a depth-bound D on evaluation circuits as 1^D, and require $\mathsf{EvalTag}$ to only take circuits of depth less than D.

Definition 6.16 ((1-hop) δ-Verifiability)**.** *Let* $\delta = \delta(\lambda)$ *be an error bound. For every polynomial* $p(\lambda)$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every Boolean circuit* $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$ *where* $|C| \le p(\lambda)$*, inputs* $\mathbf{x} \in \{0,1\}^{\ell_x}$*, and ids* $\mathbf{id} \in \{0,1\}^{\ell_x \times \lambda}$*, the following holds*

$$\Pr\left[\begin{array}{l|c} \mathsf{Verify}(\mathsf{sk}, C, \mathbf{id}, & (\mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_z}) \\ \qquad C(\mathbf{x}), \sigma_{\mathbf{z}}) = \top & \sigma^{(i)} \leftarrow \mathsf{Auth}(\mathsf{sk}, \mathbf{x}[i], \mathbf{id}[i]) \\ & \sigma_{\mathbf{x}} := (\ldots, \sigma^{(i)}, \ldots) \\ & \sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}}) \end{array}\right] \ge 1 - \delta(\lambda) - \mathrm{negl}(\lambda).$$

Definition 6.17 (Unforgeability)**.** *For every efficient adversary* $\mathcal{A}$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$*, the following holds*

$$\Pr[\mathrm{Exp}_{\mathrm{UF}}^{\mathcal{A}}(\lambda) = \mathrm{Win}] \le \mathrm{negl}(\lambda),$$

where the experiment $\mathrm{Exp}_{\mathrm{UF}}$ *is as follows:*

1. *Launch* $\mathcal{A}(1^\lambda)$*. Receive from* $\mathcal{A}$ *an output length* 1^{ℓ_z}*, compute* $(\mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_z})$*, and send* evk *to* $\mathcal{A}$*.*

2. *Receive from* $\mathcal{A}$ *any number of adaptively chosen queries in the following forms.*

   - *Authentication queries* $\{x^{(i)}, \mathsf{id}^{(i)}\}$ *with distinct* id*s.*
   - *Verification queries* $\{C^{(i)}, \mathbf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma_{\mathbf{z}}^{(i)}\}$*.*

In the weak version, we require $\mathsf{id}^{(i)}$ to contain only previously queried id*s.*

Answer them using $\mathsf{sk}, \mathbf{\Delta}$ *and running* $\mathsf{Auth}, \mathsf{Verify}$*.*

3. *Receive from $\mathcal{A}$ a forgery $(C, \mathsf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$, and output* Win *if* $\mathsf{Verify}(\mathsf{sk}, C, \mathsf{id}, \mathbf{z}, \sigma_{\mathbf{z}}) = \top$*, and either of the following is true.*

   - *All $id^{(i)} \in \mathsf{id}$ are queried in Step 2. Let $\mathbf{x}$ be the queried inputs corresponding to* id*. We have $C(\mathbf{x}) \neq \mathbf{z}$.*
   - *There exist i such that $\mathsf{id}[i]$ is not queried in Step 2.*

In the weak version, we require C to not be determined by the already queried (partial) inputs. More specifically, we require the remaining positions can be efficiently assigned in two ways to cause differen evaluations of C.

Remark. We embeded the definition of *weak* unforgeability in the above as boxed notes. We write $\mathrm{Exp}_{\mathrm{wUF}}$ to denote the weak unforgeability experiment. We will also consider the following slight variants to the experiments $\mathrm{Exp}_{\mathrm{UF}}, \mathrm{Exp}_{\mathrm{wUF}}$:

- We can enforce a bounded $Q(\lambda) \leq \mathrm{poly}(\lambda)$ number of verification queries from the adversary.
- We can allow a $\delta = 1/\mathrm{poly}(\lambda)$ chance of the adversary winning. We call it soundness error.

Theorem 6.8 (HMACs)**.** *We have the following constructions:*

1. *Assuming CP-DDH in the NIDLS framework (e.g. Damgård-Jurik groups and class groups) or the KDM security of Damgård-Jurik encryption, there exists an HMAC scheme for arbitrary Boolean circuits, achieving 0-hop and multi-hop* negl*-verifiability (Definition 6.18, 6.19), and unforgeability (Definition 6.19).*

2. *Assuming CP-DDH in prime-order groups, for every polynomials* $p_1(\lambda), p_2(\lambda), p_3(\lambda)$, *there exists an HMAC scheme for arbitrary Boolean circuits, achieving 0-hop and multi-hop* $1/p_1$*-verifiability, and unforgeability assuming* $\leq p_2(\lambda)$ *verification queries and with* $1/p_3$ *soundness errors.*

In the above, an authentication tag (evaluated or not) costs $\mathrm{poly}(\lambda)$ *bits, and an* evk *costs* $\ell_z \cdot \mathrm{poly}(\lambda)$ *bits.*

Theorem 6.9 (Leveled HMACs)**.** *We have the following constructions:*

1. *Assuming DDH in the NIDLS framework or the semantic security of Damgård-Jurik encryption (i.e. the DCR assumption), there exists a leveled HMAC scheme for bounded-depth Boolean circuits, achieving 0-hop and multi-hop* $\mathrm{negl}(\lambda)$*-verifiability and unforgeability.*

2. *Assuming DDH in prime-order groups, for every polynomials* $p_1(\lambda), p_2(\lambda), p_3(\lambda)$*, there exists a leveled aHMAC scheme for bounded-depth Boolean circuits, achieving 0-hop and multi-hop* $1/p_1$*-verifiability, and unforgeability assuming* $\leq p_2(\lambda)$ *verification queries and with* $1/p_3$ *soundness errors.*

In the above, an authentication tag (evaluated or not) costs $\mathrm{poly}(\lambda)$ *bits, and an* evk *supporting bounded-depth circuits by* D *costs* $(\ell_z + D) \cdot \mathrm{poly}(\lambda)$ *bits.*

In the remaining of this section, we focus on explaining the outlined steps to obtain non-leveled HMACs (Theorem 6.8) from aHMACs. The steps to obtain leveled HMACs (Theorem 6.9) from aHMACs are analogous.

Syntactical Changes from aHMAC to HMAC. The syntax of Auth, EvalTag are the same for both aHMAC and HMAC. The differences are (1) in HMAC, the KeyGen algorithm doesn't take any user-supplied vector $\mathbf{\Delta}$ and (2) in HMAC a Verify algorithm replaces the EvalKey algorithm. To obatin a standard HMAC scheme from aHMAC, we modify KeyGen, and implement Verify as follows. (Auth, EvalTag and are the same as aHMAC.Auth, aHMAC.EvalTag.)

Construction 26 (HMAC from aHMAC).

- $(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_z})$: Sample a global secret $\mathbf{\Delta} \leftarrow [2^\lambda]^{\ell_z}$, and run $(\mathsf{sk}', \mathsf{evk}') \leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, \mathbf{\Delta})$. Output $\mathsf{sk} := (\mathsf{sk}', \mathbf{\Delta})$ and $\mathsf{evk} = \mathsf{evk}'$.
- $b \leftarrow \mathsf{Verify}(\mathsf{sk}, C, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$: First parse $\mathsf{sk} = (\mathsf{sk}', \mathbf{\Delta})$ and compute the evaluated MAC key $\mathbf{k}_C \leftarrow \mathsf{aHMAC.EvalKey}(\mathsf{sk}', C, \mathbf{id})$. Then check whether the evaluated tags satisfy the correct form: $\sigma_{\mathbf{z}} \stackrel{?}{=} \mathbf{\Delta} \odot \mathbf{z} + \mathbf{k}_C$ (over $\mathbb{Z}$). If yes, output $b := \top$. Otherwise, output $b := \bot$.

It's clear (hence we omit the proof) that if the underlying aHMAC scheme has δ-correctness, then the obtained HMAC scheme satisfies δ-verifiability.

Proposition 6.1. *Assuming the underlying aHMAC scheme has δ-correctness per Definition 6.12, then Construction 26 satisfies δ-verifiability per Definition 6.16.*

We show that the straightforwardly adapted HMAC scheme from any aHMAC scheme with negl correctness error satisfy *weak* unforgeability.

Proposition 6.2. *Assuming the underlying aHMAC scheme has* $\mathrm{negl}(\lambda)$*-correctness per Definition 6.12, then Construction 26 satisfy* weak *unforgeability per Definition 6.17.*

Proof. We show a series of hybrids that transitions from $\mathsf{Hyb}_0 := \mathrm{Exp}^{\mathcal{A}}_{\mathrm{wUF}}(\lambda)$ to Hyb_3, where the the queries of $\mathcal{A}$ are all answered without depending on the global secret $\mathbf{\Delta}$. We then argue that a forgery is impossible in Hyb_3 due to the randomness of $\mathbf{\Delta}$.

Hyb_0**:** This is the experiment $\mathrm{Exp}^{\mathcal{A}}_{\mathrm{wUF}}(\lambda)$.

Hyb_1**:** Instead of answering each verification query $(C^{(i)}, \mathbf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma^{(i)}_{\mathbf{z}})$ using $\mathsf{sk}, \mathbf{\Delta}$ and running Verify, proceed as follows.

- Let $\mathbf{x}, \sigma_{\mathbf{x}}$ be the already queried inputs and tags associated with $\mathbf{id}$. (By assumption, all $\mathsf{id} \in \mathbf{id}$ have been queried.)
- Compute $\sigma^*_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C^{(i)}, \mathbf{x}, \sigma_{\mathbf{x}})$, and answer according to the check

$$\sigma^*_{\mathbf{z}} + (\mathbf{z}^{(i)} - C^{(i)}(\mathbf{x})) \odot \mathbf{\Delta} = \sigma^{(i)}_{\mathbf{z}}. \tag{6.16}$$

By negl(λ)-correctness, $\sigma_{\mathbf{z}}$ satisfy $\sigma^*_{\mathbf{z}} = \mathbf{\Delta} \odot C^{(i)}(\mathbf{x}) + \mathbf{k}_C$, while by construction, Verify only passes if $\sigma^{(i)}_{\mathbf{z}} = \mathbf{\Delta} \cdot \mathbf{z}^{(i)} + \mathbf{k}_C = \sigma^*_{\mathbf{z}} + (\mathbf{z}^{(i)} - C^{(i)}(\mathbf{x})) \odot \mathbf{\Delta}$. Therefore, we have $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_0$.

Hyb_2: Instead of checking the forgery $(C, \mathsf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$ by running Verify, proceed as follows.

- If every $id \in \mathsf{id}$ has been queried, with associated input $\mathbf{x}$, and $C(\mathbf{x}) = \mathbf{z}$, output "Loss";
- If exists an $id \in \mathsf{id}$ (at i-th position) not queried in Step 2, but C is already determined by the queried inputs, then directly output "Loss";
- Otherwise, for every un-queried input position j, one can assign assign a bit x_j and compute an associated tag $\sigma \leftarrow \mathsf{Auth}(\mathsf{sk}, x_j, \mathsf{id}[j])$ such that the "completed" input $\mathbf{x}$ associated with id satsify $C(\mathbf{x}) \neq \mathbf{x}$. Let $\mathbf{x}$ be the completed inputs corresponding to id, and $\sigma_{\mathbf{x}}$ be associated tags. Then compute $\sigma^*_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}})$, and check

$$\sigma^*_{\mathbf{z}} + (\mathbf{z} - C(\mathbf{x})) \odot \mathbf{\Delta} \stackrel{?}{=} \sigma_{\mathbf{z}}.$$

  If not, output "Loss". Otherwise, output "Win".

Similarly to the previous hybrid, by negl(λ)-correctness we have $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_1$.

Hyb_3: Instead of answering authentication queries using $\mathbf{\Delta}, \mathsf{sk}$ and running Auth, the experiment Hyb_3 runs $\mathsf{Sim}_1, \mathsf{Sim}_2$ as guaranteed by Definition 6.12 to simulate the evaluation key evk, the answers to authentication queries, and the tags associated with un-queried "fake" inputs. The aHMAC security guarantees $\mathsf{Hyb}_3 \approx_c \mathsf{Hyb}_2$.

Note that the global secret $\mathbf{\Delta}$ is now only used for answering verification queries as in Equation 6.16 and in the final check for forgery.

Hyb_4: Instead of answering verification queries using $\mathbf{\Delta}$ as in Equation 6.16, directly check if $\mathbf{z}^{(i)} - C^{(i)}(\mathbf{x}) \stackrel{?}{=} 0$. Due to the randomness of $\mathbf{\Delta}$, we have $\mathsf{Hyb}_4 \approx \mathsf{Hyb}_3$.

Note that the global secret $\mathbf{\Delta}$ is now only used for checking for forgery.

By a hybrid argument, we conclude that $\mathrm{Exp}^{\mathcal{A}}_{\mathrm{wUF}}(\lambda) \equiv \mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_4$. In Hyb_4, the adversary $\mathcal{A}$ wins only if it outputs a forgery $(C, \textbf{id}, \mathbf{z}, \sigma_{\mathbf{z}})$ such that $\sigma_{\mathbf{z}} = \sigma^*_{\mathbf{z}} + (C(\mathbf{x}) - \mathbf{z}) \odot \boldsymbol{\Delta}$. As noted, the adversary $\mathcal{A}$'s view is entirely independent of $\boldsymbol{\Delta}$. Hence in the case of $C(\mathbf{x}) - \mathbf{z} \neq \mathbf{0}$, the forgery has negligible chance of passing the checks due to the randomness of $\boldsymbol{\Delta}$. □

The above proof of weak unforgeability relies on $\mathrm{negl}(\lambda)$-correctness of the underlying aHMAC scheme to transition from Hyb_0 to Hyb_1 and then from Hyb_1 to Hyb_2. In general, if the underlying aHMAC has δ-correctness, then the transition from Hyb_0 to Hyb_2 incurrs a $Q \cdot \delta$ soundness error, where Q is the number of verification queries by the adversary.

Proposition 6.3. *Assuming the underlying aHMAC scheme has δ-correctness per Definition 6.12, and a bound Q on the number of verification query in the weak unforgeability experiment, then Construction 26 satisfy weak unforgeability with $\delta \cdot Q$ soundness error.*

Therefore, if assuming a polynomially bounded number of verification queries Q in the weak unforgeability experiment, then we can choose a sufficiently small $\delta \leq 1/(\mathrm{poly}(\lambda) \cdot Q)$ for the underlying aHMAC scheme and prove weak unforgeability with $1/\mathrm{poly}(\lambda)$ soundness error, for any $\mathrm{poly}(\lambda)$. This is how we obtain the HMAC instantiation using prime-order groups in Theorem 6.8 and 6.9.

Achieving Composability. In the literature of HMACs, a desirable feature is to allow further evaluations by $\mathsf{EvalTag}$ on previously evaluated tags. We refer to this as multi-hop verifiability formally defined as follows.

Definition 6.18 ((0-hop) δ-Verifiability)**.** *Let $\delta = \delta(\lambda)$ be an error bound. There exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every $\ell_z \in \mathbb{N}$, input $x \in \{0,1\}$ and $\mathsf{id} \in \{0,1\}^\lambda$, the following holds*

$$\Pr\left[\begin{array}{r} \mathsf{Verify}(\mathsf{sk}, \mathsf{ID}, \mathsf{id}, \\ x, \sigma'_x) = \top \end{array} \middle| \begin{array}{c} (\mathsf{evk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_z}) \\ \sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id}) \\ \sigma'_x \leftarrow \mathsf{EvalTag}(\mathsf{evk}, \mathsf{ID}, x, \sigma_x) \end{array}\right] \geq 1 - \delta(\lambda) - \mathrm{negl}(\lambda),$$

where $\mathsf{ID} : \{0,1\} \to \{0,1\}$ is the identity circuit.

Remark. 0-hop verifiability is trivially implied by 1-hop verifiability (Definition 6.16).

Definition 6.19 ((multi-hop) δ-Verifiability)**.** *Let $\delta = \delta(\lambda)$ be an error bound. For every polynomial $p(\lambda)$, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every composed labeled Boolean circuits $C := g(f_1, \dots, f_n)$, where where $g : \{0,1\}^n \to \{0,1\}^{\ell_z}$ and $|g| \le p(\lambda)$, ids $\mathsf{id}_i \in \{0,1\}^{\ell_{x i} \times \lambda}$ corresponding to f_i, and intermediate evaulation results $\mathbf{w} \in \{0,1\}^n$ and tags $\sigma_{\mathbf{w}} \in \mathbb{Z}^n$, the following implication holds with probability $\ge 1 - \delta(\lambda) - \mathrm{negl}(\lambda)$.*

Setup *Sample* $(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_z})$.

Premise *For $i \in [n]$,* $\mathsf{Verify}(\mathsf{sk}, f_i, \mathsf{id}_i, \mathbf{w}[i], \sigma_{\mathbf{w}}[i]) = \top$.

Conclusion

$$\mathsf{Verify}(\mathsf{sk}, C, \mathsf{id}, g(\mathbf{w}), \sigma_{\mathbf{z}}) = \top \;\middle|\; \begin{array}{l} \sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, g, \mathbf{w}, \sigma_{\mathbf{w}}), \\ \mathsf{id} := (\mathsf{id}_1, \dots, \mathsf{id}_n). \end{array}$$

The aHMAC definition in Definition 6.12 doesn't support multi-hop evaluation because the authenticated tags and evaluated tags have different forms. More specifically, we will need the evaluated tags (of x) to have an algebraic form $\mathbf{\Delta} x + \mathbf{k}_x$, where $\mathbf{\Delta}$ is a user-supplied integer vector. This is defined to make the evaluated tags compatible for further evaluation as a wire label by a garbling scheme (with "free-XOR" style labels) or as a memory share by a HSS scheme.

However, we note that our constructions of aHMAC (Construction 21, 22, or the one sketched in Section 6.4.3) can be straightforwardly modified to support multi-hop evaluations. Roughly, our evaluation algorithm $\mathsf{EvalTag}$ proceeds in two steps:

1. Evaluate the circuit C over input tags using evk. Every intermediate evaluated tag (of value x) has a consistent form $\bar{\mathbf{s}} x + \mathbf{k}_x$, where $\bar{\mathbf{s}}$ is a global internal secret vector.

2. In the end, apply a "key-switching" step to transform the output tags (of value x) to have the format $\mathbf{\Delta} x + \mathbf{k}'_x$, where $\mathbf{\Delta}$ is the user-supplied vector encoded in evk.

To support multi-hop evaluations, we can move the second step out of $\mathsf{EvalTag}$, and only perform it during verification Verify. This ensures all evaluated tags remain their internal format, and can be further evaluated freely.

Claim 1. *With the described modification to the underlying* aHMAC *scheme, (Construction 21, 22, or Section 6.4.3) and assuming it has δ-correctness, then Construction 26 satsify both 0-hop δ-verifiability and multi-hop δ-verifiability.*

Note that weak unforgeability is not affected by this modification, because (1) the view of an adversary, consisting of un-evaluated tags and evk, does not change and (2) a forgery of a tag in the internal format can be publicly transformed into a forgery in the original format by applying the "key-switching" step.

Amplifying Weak to Standard Unforgeability. Finally, we show that we can generically combine two instances of HMAC schemes with weak unforgeability to satisfy standard unforgeability. We describe the construction below.

Construction 27 (HMAC from Weak to Standard Unforgeability)**.** Ingredients:

- An HMAC scheme HMAC with weak unforgeability.

$(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^{\ell_z})$: Setup two instances of HMAC:

$$(\mathsf{sk}_0, \mathsf{evk}_0) \leftarrow \mathsf{HMAC.KeyGen}(1^\lambda, 1^{\ell_z}), \quad (\mathsf{sk}_1, \mathsf{evk}_1) \leftarrow \mathsf{HMAC.KeyGen}(1^\lambda, 1^{\ell_z}),$$

and output $\mathsf{sk} := (\mathsf{sk}_0, \mathsf{sk}_1)$ and $\mathsf{evk} := (\mathsf{evk}_0, \mathsf{evk}_1)$.

$\sigma_x \leftarrow \mathsf{Auth}(\mathsf{sk}, x, \mathsf{id})$: Parse $\mathsf{sk} = (\mathsf{sk}_0, \mathsf{sk}_1)$, and authenticate x using two instances of HMAC:

$$\sigma_0 \leftarrow \mathsf{HMAC.Auth}(\mathsf{sk}_0, x, \mathsf{id}), \quad \sigma_1 \leftarrow \mathsf{HMAC.Auth}(\mathsf{sk}_1, x, \mathsf{id}).$$

Output $\sigma_x := (\sigma_0, \sigma_1)$.

$\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \mathbf{x}, \sigma_{\mathbf{x}}):$ Parse $\mathsf{evk} = (\mathsf{evk}_0, \mathsf{evk}_1)$, and evaluate two different circuits on $\sigma_{\mathbf{x}}$ using two instances of HMAC:

$$\sigma_{\mathbf{z},0} \leftarrow \mathsf{HMAC.EvalTag}(\mathsf{evk}_0, C, \mathbf{x}, \sigma_{\mathbf{x}}),$$

$$\sigma_{y,1} \leftarrow \mathsf{HMAC.EvalTag}(\mathsf{evk}_1, \mathsf{XOR}, \mathbf{x}, \sigma_{\mathbf{x}}),$$

where $y = \mathsf{XOR}(\mathbf{x})$ computes the XOR of all bits in $\mathbf{x}$. Output $\sigma_{\mathbf{z}} = (\sigma_{\mathbf{z},0}, y, \sigma_{y,1})$

$\mathsf{Verify}(\mathsf{sk}, C, \textsf{\textbf{id}}, \mathbf{z}, \sigma_{\mathbf{z}}):$ Parse $\mathsf{sk} = (\mathsf{sk}_0, \mathsf{sk}_1)$, and $\sigma_{\mathbf{z}} = (\sigma_{\mathbf{z},0}, y, \sigma_{y,1})$. Then verify the evaluated tags $\sigma_{\mathbf{z},0}$ and $\sigma_{y,1}$ using two instances of HMAC:

$$b_0 \leftarrow \mathsf{HMAC.Verify}(\mathsf{sk}_0, C, \textsf{\textbf{id}}, \mathbf{z}, \sigma_{\mathbf{z},0}),$$

$$b_1 \leftarrow \mathsf{aHMAC.Verify}(\mathsf{sk}_1, \mathsf{XOR}, \textsf{\textbf{id}}, y, \sigma_{y,1}).$$

Output $\top$ only if $b_0 = b_1 = \top$. Otherwise, output $\bot$.

The verifiability correctness of Construction 27 follows directly from that of the underlying HMAC scheme. We now show that Construction 27 satsify unforgeability per Definition 6.17.

Proposition 6.4. *Assuming the underlying HMAC scheme satsify weak unforgeability per Definition 6.17, then Construction 27 satsify (standard) unforgeability per Definition 6.17.*

Proof. We now show a series of hybrids that transitions from $\mathsf{Hyb}_0 := \mathrm{Exp}^{\mathcal{A}}_{\mathrm{UF}}(\lambda)$ to Hyb_2, where the probability of the experiment outputs "Win" is negligible.

Hyb_0**:** This is the experiment $\mathrm{Exp}^{\mathcal{A}}_{\mathrm{UF}}(\lambda)$.

Hyb_1**:** Instead of answering each verification query $(C^{(i)}, \textsf{\textbf{id}}^{(i)}, \mathbf{z}^{(i)}, \sigma^{(i)}_{\mathbf{z}})$ using sk and Verify, proceed as follows.

- If there is an $\mathsf{id} \in \textsf{\textbf{id}}^{(i)}$ that's un-queried, directly answer $\bot$.
- Otherwise, run Verify to answer as in Hyb_0.

Note that Hyb_1 is the same as Hyb_0, unless there is a verification query with unqueried $\textsf{\textbf{id}}$. We claim that this bad event happens with negligible probability, due to the weak unforgeability of (the second instance of) HMAC.

Claim 6.2. *If the underlying* HMAC *scheme satisfy weak unforgeability, then in* Hyb_0 *all verification queries with an un-queried* id *evaluate to* $\bot$*, except with negligible probability.*

We have $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

Hyb_2 : Instead of checking the forgery $(C, \mathbf{id}, \mathbf{z}, \sigma_z)$ by running Verify, proceed as follows.

- If there is an $\mathsf{id} \in \mathbf{id}^{(i)}$ that's un-queried, directly answer "Loss".
- Otherwise, run Verify to answer as in Hyb_1.

By analogous arguments as the previous hybrid, we have $\mathsf{Hyb}_2 \approx_c \mathsf{Hyb}_1$ due to the weak unforgeability of (the second instance of) HMAC.

By a hybrid argument, we conclude that $\mathrm{Exp}_{\mathrm{UF}}^{\mathcal{A}}(\lambda) \equiv \mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_2$. We claim that the probability of Hyb_2 outputing "Win" is negligible due to the weak unforgeability of (the first instance of) HMAC, which concludes the proof.

Claim 6.3. *If the underlying* HMAC *scheme satisfy weak unforgeability, then* Hyb_2 *outputs "Loss" except with negligible probability.*

We now prove the first claim.

of the first claim. For contradiction, assume there exists an efficient adversary $\mathcal{A}$ that with non-negligible probability $p(\lambda)$ produces at least one verification query in the Hyb_0 experiment, $(C^{(i)}, \mathbf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma_{\mathbf{z}}^{(i)})$, where $\mathbf{id}^{(i)}$ contains an un-queried id, but Verify outputs $\top$. We call such queries "bad" ones. Then we define the following reduction $\mathcal{B}$ to break the weak unforgeability of the underlying HMAC scheme.

- Let $Q \leq \mathrm{poly}(\lambda)$ be a bound on the number of verification queries from $\mathcal{A}$, and $j \leftarrow [Q]$ be $\mathcal{B}$'s guess of the first "bad" query from $\mathcal{A}$.
- Recall that in Construction 26, authentication tags and verification queries are handled using two independent instances of HMAC schemes. $\mathcal{B}$ sets up the first instance on its own by running

$$(\mathsf{sk}_0, \mathsf{evk}_0) \leftarrow \mathsf{HMAC}(1^\lambda, 1^{\ell_z}).$$

It queries the challenger to handle the second instance of HMAC as we explain below.

- For every authentication query $(x^{(i)}, \mathsf{id}^{(i)})$, $\mathcal{B}$ computes $\sigma_0 \leftarrow \mathsf{HMAC.Auth}(\mathsf{sk}_0, x^{(i)}, \mathsf{id}^{(i)})$ on its own, and query the challenger $(x^{(i)}, \mathsf{id}^{(i)})$ to obtain σ_1. $\mathcal{B}$ answers $\sigma_x := (\sigma_0, \sigma_1)$ to $\mathcal{A}$.
- For every authentication query $(C^{(i)}, \mathbf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma_{\mathbf{z}}^{(i)} = (\sigma_{\mathbf{z},0}, y, \sigma_{y,0}))$, $\mathcal{B}$ checks if $\mathbf{id}^{(i)}$ contains an un-queried id.
  - If yes, and if $i < j$, then $\mathcal{B}$ aborts.
  - If yes, and if $i = j$, then $\mathcal{B}$ outputs $(\mathsf{XOR}, \mathbf{id}^{(i)}, y, \sigma_{y,0})$ as the forgery to the challenger.
  - If no, then $\mathcal{B}$ computes $b_0 \leftarrow \mathsf{HMAC.Verify}(\mathsf{sk}_0, C^{(i)}, \mathbf{id}, \mathbf{z}, \sigma_{\mathbf{z},0})$, and query the challenger with $(\mathsf{XOR}, \mathbf{id}^{(i)}, y, \sigma_{y,0})$ to obtain b_1. $\mathcal{B}$ answers $\top$ to $\mathcal{A}$ if $b_0 = b_1 = \top$, and $\perp$ otherwise.

Note that if $\mathcal{B}$ guess correctly the index of the first "bad" query from $\mathcal{A}$, then it successfully wins the weak unforgeability experiment. This happens with non-negligible probability $\geq p(\lambda)/Q$. Hence we are done. □

We next prove the second claim.

of the second claim. For contradiction, assume there exists an efficient adversary $\mathcal{A}$ that wins the Hyb_2 experiment with non-negligible probability $p(\lambda)$. Then we define the following reduction $\mathcal{B}$ to break the weak unforgeability of the underlying aHMAC scheme.

- Recall that in Construction 26, authentication tags and verification queries are handled using two independent instances of HMAC schemes. $\mathcal{B}$ sets up the second instance on its own by running

$$(\mathsf{sk}_1, \mathsf{evk}_1) \leftarrow \mathsf{HMAC}(1^\lambda, 1^{\ell_z}).$$

  It queries the challenger to handle the first instance of HMAC as we explain below.
- For every authentication query $(x^{(i)}, \mathsf{id}^{(i)})$, $\mathcal{B}$ computes $\sigma_1 \leftarrow \mathsf{HMAC.Auth}(\mathsf{sk}_1, x^{(i)}, \mathsf{id}^{(i)})$ on its own, and query the challenger $(x^{(i)}, \mathsf{id}^{(i)})$ to obtain σ_0. $\mathcal{B}$ answers $\sigma_x := (\sigma_0, \sigma_1)$ to $\mathcal{A}$.

- For every authentication query $(C^{(i)}, \mathsf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma_{\mathbf{z}}^{(i)} = (\sigma_{\mathbf{z},0}, y, \sigma_{y,0}))$, $\mathcal{B}$ checks if $\mathsf{id}^{(i)}$ contains an un-queried id.
  - If yes, answer $\perp$ to $\mathcal{A}$ as in Hyb_2.
  - If no, then $\mathcal{B}$ computes $b_1 \leftarrow \mathsf{HMAC.Verify}(\boldsymbol{\Delta}, \mathsf{sk}_0, \mathsf{XOR}, \mathsf{id}, y, \sigma_{y,1})$, and query the challenger with $(C^{(i)}, \mathsf{id}^{(i)}, \mathbf{z}^{(i)}, \sigma_{\mathbf{z},0})$ to obtain b_0. $\mathcal{B}$ answers $\top$ to $\mathcal{A}$ if $b_0 = b_1 = \top$, and $\perp$ otherwise.
- Given a forgery $(C, \mathsf{id}, \mathbf{z}, \sigma_{\mathbf{z}} = (\sigma_{\mathbf{z},0}, y, \sigma_{y,1}))$ from $\mathcal{A}$, $\mathcal{B}$ checks if id contains un-queried ids.
  - If yes, then abort.
  - If no, then output $(C, \mathsf{id}, \mathbf{z}, \sigma_{\mathbf{z},0})$ to the challenger as a forgery.

Note that if $\mathcal{A}$ wins in the emulated experiment, then $\mathcal{B}$ also wins in the weak unforgeability experiment. By assumption, this happens with non-negligible probability $p(\lambda)$. Hence we are done. □

6.5 Application: Succinct Partial Garbling

In this section, we show how to construct succinct partial garbling schemes for circuits from aHMACs. In Section 6.5.1, we show the simpler case using an aHMAC with negl correctness errors. In Section 6.5.2, we show the more general case using an aHMAC with 1/poly correctness errors, and a robust secret sharing scheme, e.g., Shamir's scheme.

Theorem 6.10 (Succinct Partial Garbling for Circuits)**.** *Let $\mathcal{C} = \{\mathcal{C}_\lambda\}$ be the class of two-input Boolean circuits, i.e.*

$$\mathcal{C}_\lambda := \{\textit{all Boolean circuits of form } C(\mathbf{x}, \mathbf{y}) = C_{\mathrm{priv}}(C_{\mathrm{pub}}(\mathbf{x}), \mathbf{y})\}.$$

There exists a succinct partial garbling scheme for $\mathcal{C}$ where the garbling size is $|\widehat{C}| = |C_{\mathrm{priv}}| \cdot \mathrm{poly}(\lambda)$ bits under any of the assumptions from Theorem 6.5:

1. *CP-DDH in either the NIDLS framework or prime-order groups;*

2. the KDM-DCR assumption.

When replacing the aHMAC with a leveled aHMAC in the above constructions, we obtain a partial garbling whose size scales linearly with both the private computation complexity $|C_{\text{priv}}|$ and the public computation depth D_{pub}. This scheme satisfies our succinctness definition (Definition 6.2) when restricted to the class of bounded-depth circuits. As the constructions remain unchanged otherwise, we omit writing them out again.

Theorem 6.11 (Succinct Partial Garbling for Bounded-Depth Circuits)**.** *Let $\mathcal{C}$ be the class of two-input Boolean circuits as in Theorem 6.10. There exists a partial garbling scheme for $\mathcal{C}$ with garbling size $|\widehat{C}| = (|C_{\text{priv}}| + D_{\text{pub}}) \cdot \text{poly}(\lambda)$ bits, where D_{pub} denotes the depth of C_{pub}, under any of the assumptions from Theorem 6.6:*

1. DDH in the NIDLS framework or prime-order groups.

2. The DCR assumption.

As a directly implication, for any polynomial $d(\lambda)$, let $\mathcal{C}^d := \{\mathcal{C}^d_\lambda\}$ be the class of bounded-depth two-input Boolean circuits, i.e.

$$\mathcal{C}^d_\lambda := \{\textit{all Boolean circuits of form } C(\mathbf{x}, \mathbf{y}) = C_{\text{priv}}(C_{\text{pub}}(\mathbf{x}), \mathbf{y}),\ \textit{and with depth } \leq d(\lambda)\}.$$

There exists a succinct *partial garbling scheme for $\mathcal{C}^d$ with garbling size $|\widehat{C}| = (|C_{\text{priv}}| + d(\lambda)) \cdot \text{poly}(\lambda)$ bits, under the above assumptions.*

6.5.1 Construction from negl*-Correct aHMACs*

Construction 28 (Succinct Partial Garbling)**.** Ingredients:

- An aHMAC scheme aHMAC with negl-correctness error.
- Any Boolean garbling scheme BG with λ-bit labels.
- A secret-key encryption scheme E with λ-bit keys encrypting λ-bit messages.

We construct a succinct partial garbling scheme for the class of Boolean circuits of unbounded size: $\mathcal{C} = \{\mathcal{C}_\lambda\}$, where every $\mathcal{C}_\lambda$ contains all Boolean circuits of the form $C(\mathbf{x}, \mathbf{y}) = C_{\text{priv}}(C_{\text{pub}}(\mathbf{x}), \mathbf{y})$. We refer to $C_{\text{pub}} : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_w}$ as the public sub-circuit, and $C_{\text{priv}} : \{0,1\}^{\ell_w} \times \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_z}$, the private sub-circuit.

$(\widehat{C}, \{K_x^{(i)}\}, \{K_y^{(i)}\}) \leftarrow \mathsf{Garb}(1^\lambda, C, \{K_z^{(i)}\})$:

1. Generate the garbling $\widehat{C_{\text{priv}}}$ of the private sub-circuit:

$$(\widehat{C_{\text{priv}}}, \{K_w^{(i)}\}, \{K_y^{(i)}\}) \leftarrow \mathsf{BG.Garb}(C_{\text{priv}}, \{K_z^{(i)}\}).$$

2. Sample ℓ_w pairs of secret keys $\{\Delta_j, \overline{\Delta}_j\}$ to encrypt output labels of C_{pub}:

$$\begin{aligned} \text{for } j \in [\ell_w], \quad & \mathsf{ct}_j \leftarrow \mathsf{E.Enc}(\Delta_j, K_w^{(j)}(1)) \\ & \overline{\mathsf{ct}}_j \leftarrow \mathsf{E.Enc}(\overline{\Delta}_j, K_w^{(j)}(0)). \end{aligned}$$

3. Generate aHMAC tags for public inputs as their labels (using deterministically derived distinct $\textbf{id}, \overline{\textbf{id}}$):

$$\begin{aligned} (\mathsf{sk}, \mathsf{evk}) &\leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, \boldsymbol{\Delta}), & \boldsymbol{\Delta} &:= (\ldots, \Delta_j, \ldots), \\ (\overline{\mathsf{sk}}, \overline{\mathsf{evk}}) &\leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, \overline{\boldsymbol{\Delta}}), & \overline{\boldsymbol{\Delta}} &:= (\ldots, \overline{\Delta}_j, \ldots), \\ \sigma_b^{(i)} &\leftarrow \mathsf{aHMAC.Auth}(\mathsf{sk}, b, \textbf{id}[i]), & &// \text{ for } i \in [\ell_x], b \in \{0,1\}, \\ \overline{\sigma}_b^{(i)} &\leftarrow \mathsf{aHMAC.Auth}(\overline{\mathsf{sk}}, b, \overline{\textbf{id}}[i]). \end{aligned}$$

   Define $K_x^{(i)}$ such that $K_x^{(i)}(b) = (\sigma_b^{(i)}, \overline{\sigma}_b^{(i)})$.

4. Evaluate aHMAC keys $\mathbf{k}_{\text{pub}}, \overline{\mathbf{k}}_{\text{pub}}$ which will be used as "decryption helpers":

$$\mathbf{k}_{\text{pub}} \leftarrow \mathsf{aHMAC.EvalKey}(\mathsf{sk}, C_{\text{pub}}, \textbf{id}), \quad \overline{\mathbf{k}}_{\text{pub}} \leftarrow \mathsf{aHMAC.EvalKey}(\overline{\mathsf{sk}}, \overline{C_{\text{pub}}}, \overline{\textbf{id}}),$$

   where $\overline{C_{\text{pub}}}$ computes the complement of C_{pub}.

Output $\widehat{C} := (\widehat{C_{\text{priv}}}, \{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}, \mathsf{evk}, \overline{\mathsf{evk}}, \mathbf{k}_{\text{pub}}, \overline{\mathbf{k}}_{\text{pub}})$, $\{K_x^{(i)}\}$ and $\{K_y^{(i)}\}$.

$\mathbf{z} \leftarrow \mathsf{Eval}(C, \widehat{C}, \{x^{(i)}, L_x^{(i)}\}, \{L_y^{(i)}\})$:

Parse $\widehat{C_{\text{priv}}}$, $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}$, and $\mathsf{evk}, \overline{\mathsf{evk}}, \mathbf{k}_{\text{pub}}, \overline{\mathbf{k}}_{\text{pub}}$ from $\widehat{C}$. Let $\mathbf{x} := (\ldots, x_i, \ldots)$. Parse $\{\sigma^{(i)}, \overline{\sigma}^{(i)}\}$ from $\{L_x^{(i)}\}$, and let

$$\sigma_{\mathbf{x}} := (\ldots, \sigma^{(i)}, \ldots)_{i \in [\ell_x]}, \quad \overline{\sigma}_{\mathbf{x}} := (\ldots, \overline{\sigma}^{(i)}, \ldots)_{i \in [\ell_x]}.$$

1. Evaluate aHMAC tags according to C_{pub}:

$$\sigma_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\mathsf{evk}, C_{\text{pub}}, \mathbf{x}, \sigma_{\mathbf{x}}), \ \overline{\sigma}_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\overline{\mathsf{evk}}, \overline{C_{\text{pub}}}, \mathbf{x}, \overline{\sigma}_{\mathbf{x}}).$$

Note that the aHMAC correctness should ensure $\sigma_{\mathbf{w}} = \mathbf{\Delta} \odot \mathbf{w} + \mathbf{k}_{\text{pub}}$, *and* $\overline{\sigma}_{\mathbf{w}} = \overline{\mathbf{\Delta}} \odot \overline{\mathbf{w}} + \overline{\mathbf{k}}_{\text{pub}}$ *over* $\mathbb{Z}$, *where* $\mathbf{w} = C_{\text{pub}}(\mathbf{x})$, *and* $\overline{\mathbf{w}} = \mathbf{1} - \mathbf{w} = \overline{C_{\text{pub}}}(\mathbf{x})$.

2. Recover one of the decryption keys Δ_j, $\overline{\Delta}_j$ for each bit of $\mathbf{w} = C_{\text{pub}}(\mathbf{x})$.

$$\begin{array}{ll} \text{If } \mathbf{w}[j] = 1 & \Delta_j \leftarrow \sigma_{\mathbf{w}}[j] - \mathbf{k}_{\text{pub}}[j] \quad (\text{over } \mathbb{Z}), \\ \text{o/w} & \overline{\Delta}_j \leftarrow \overline{\sigma}_{\mathbf{w}}[j] - \overline{\mathbf{k}}_{\text{pub}}[j] \quad (\text{over } \mathbb{Z}). \end{array}$$

3. Decrypt input labels $\{L_w^{(i)}\}$ to the private sub-circuit C_{priv}:

$$\begin{array}{ll} \text{If } \mathbf{w}[j] = 1 & L_w^{(j)} \leftarrow \mathsf{E.Dec}(\Delta_j, \mathsf{ct}_j), \\ \text{o/w} & L_w^{(j)} \leftarrow \mathsf{E.Dec}(\overline{\Delta}_j, \overline{\mathsf{ct}}_j). \end{array}$$

Then evaluate the garbling $\{L_z^{(i)}\} \leftarrow \mathsf{BG.Eval}(C_{\text{priv}}, \widehat{C_{\text{priv}}}, \{L_w^{(i)}\}, \{L_y^{(i)}\})$.

Correctness: As noted in the construction, correctness follows straightforwardly from that of the aHMAC scheme and Boolean garbling.

Efficiency: We summarize bit-lengths of the components, assuming the Boolean garbling scheme BG has label size $O(\lambda)$ bits and garbling size $|C_{\text{priv}}| \cdot \text{poly}(\lambda)$ bits, the encryption scheme E has $O(1)$-rate ciphertexts, and the aHMAC scheme aHMAC has evaluation keys of $\ell_z \cdot \text{poly}(\lambda)$ bits. They all exist under any of the assumptions from which we construct aHMACs in Theorem 6.5.

- $\{L_x^{(i)}\}$ each consists of two aHMAC tags, which has $\text{poly}(\lambda)$ bits.
- $\{L_y^{(i)}\}$ are standard Boolean garbling labels, and each has $O(\lambda)$ bits.
- $\widehat{C}$ consists of the following, and has $|C_{\text{priv}}| \cdot \text{poly}(\lambda)$ bits overall.
  - A Boolean garbling $\widehat{C_{\text{priv}}}$, which has $|C_{\text{priv}}| \cdot O(\lambda)$ bits;
  - $2\ell_z$ ciphertexts $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}$, each having $O(\lambda)$ bits;
  - 2 aHMAC evaluation keys $\mathsf{evk}, \overline{\mathsf{evk}}$, each having $\ell_z \cdot \text{poly}(\lambda)$;
  - 2 aHMAC evaluated keys $\mathbf{k}_{\text{pub}}, \mathbf{k}_{\overline{\text{pub}}}$, each having $\ell_z \cdot \text{poly}(\lambda)$.

Security: We will next show a more general construction from aHMACs with $1/\text{poly}$ correctness error. We omit proving security of the current simpler case, and refer readers to the more general proof (of Lemma 6.11).

Lemma 6.10. *Construction 28 is a secure partial garbling scheme.*

6.5.2 Construction from $1/\text{poly}$*-Correct aHMACs*

Definition 6.20 (t-out-of-n Robust Secret Sharing)**.** *A t-out-of-n robust secret sharing scheme with message space $\mathcal{M}$ consists of two efficient algorithms:*

- $\mathsf{Share}(s \in \mathcal{M})$ *takes a secret s, and outputs n shares $\{s_i\}_{[n]}$ where $s_i \in \mathcal{M}$.*
- $\mathsf{Recon}(\{s_i\}_{[n]})$ *takes n shares, and recovers a secret $s \in \mathcal{M}$.*

Robust Reconstruction. *For all messages $s \in \mathcal{M}$, all subsets $T \subset [n]$ with $|T| \leq t$, and any adversary $\mathcal{A}$, the following holds:*

$$\Pr\left[\mathsf{Recon}(\{s_i^*\}_{[n]}) = s \;\middle|\; \begin{array}{r} \{s_i\} \leftarrow \mathsf{Share}(s), \\ \{s_i^*\}_T \leftarrow \mathcal{A}(\{s_i\}_{[n]}), \\ s_i^* = s \textit{ for } i \notin T. \end{array}\right] = 1$$

Privacy. *For any two messages $s, s' \in \mathcal{M}$, all subsets $T \subset [n]$ with $|T| \leq t$, the following holds:*

$$\mathsf{Share}(s)_T \equiv \mathsf{Share}(s')_T$$

Remark. For $t < n/3$, the standard Shamir's secret sharing is also a robust secret sharing. This suffices for our application. For $n/3 \leq t < n/2$, there also exists robust secret sharing schemes with a negligible correctness error and larger share size, e.g. [RB89, CFOR12].

Construction 29 (Correctness and Privacy Amplification)**.** Ingredients:

- An aHMAC scheme aHMAC with $1/(2\lambda)$-correctness error.
- A $(\lambda-1)$-out-of-3λ RSS scheme RSS with message space $\mathcal{M} = \{0,1\}^\lambda$.
- Any Boolean garbling scheme BG with λ-bit labels.
- A secret-key encryption scheme E with λ-bit keys encrypting λ-bit messages.

Compared to Construction 28, the new construction for Garb differs in Step 3, 4, and for Eval differs in Step 1, 2. In the following, we focus on the differences.

$(\widehat{C}, \{K_x^{(i)}\}, \{K_y^{(i)}\}) \leftarrow \mathsf{Garb}(1^\lambda, C)$:

3'. Generate 3λ shares $\{\Delta_{j,t}, \overline{\Delta}_{j,t}\}_{t\in[3\lambda]}$ from each pairs of secret keys:

$$\text{for } j \in [\ell_w], \qquad \{\Delta_{j,t}\}_{t\in[3\lambda]} \leftarrow \mathsf{Share}(\Delta_j),$$
$$\{\overline{\Delta}_{j,t}\}_{t\in[3\lambda]} \leftarrow \mathsf{Share}(\overline{\Delta}_j).$$

Write $\mathsf{Keys}_t = \{\Delta_{j,t}, \overline{\Delta}_{j,t}\}_{j\in[\ell_w]}$ to denote the t-th share of the key pairs.

4'. Apply Step 3, 4 from Construction 28 independently on each share Keys_t to generate

$$\mathsf{evk}_t, \overline{\mathsf{evk}}_t, \mathbf{k}_{\text{pub},t}, \overline{\mathbf{k}}_{\text{pub},t}, \quad \text{and for } i \in [\ell_x],\ K_{x,t}^{(i)}.$$

Define $K_x^{(i)}$ such that $K_x^{(i)}(b) = (\ldots, K_{x,t}^{(i)}(b), \ldots)_{t\in[3\lambda]}$, and as shorthands write $\mathsf{Tables}_t = (\mathsf{evk}_t, \overline{\mathsf{evk}}_t, \mathbf{k}_{\text{pub},t}, \overline{\mathbf{k}}_{\text{pub},t})$.

Output $\widehat{C} := (\widehat{C_{\text{priv}}}, \{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}, \{\mathsf{Tables}_t\})$, $\{K_x^{(i)}\}$ and $\{K_y^{(i)}\}$).

$\mathbf{z} \leftarrow \mathsf{Eval}(C, \widehat{C}, \{x^{(i)}, L_x^{(i)}\}, \{L_y^{(i)}\})$:

Parse $\widehat{C_{\text{priv}}}$, $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}$, and $\{\mathsf{Tables}_t\}$ from $\widehat{C}$. Let $\mathbf{x} := (\ldots, x_i, \ldots)$. Parse $\{\sigma_t^{(i)}, \overline{\sigma}_t^{(i)}\}$ from $\{L_x^{(i)}\}$, and let

$$\sigma_{\mathbf{x},t} := (\ldots, \sigma_t^{(i)}, \ldots)_{i \in [\ell_x]}, \quad \overline{\sigma}_{\mathbf{x},t} := (\ldots, \overline{\sigma}_t^{(i)}, \ldots)_{i \in [\ell_x]}.$$

1'. For each $t \in [3\lambda]$, apply Step 1, 2 from Construction 28 independently on each Tables_t to recover (half of) the t-th share $\mathsf{Keys}_t = \{\Delta_{j,t}, \overline{\Delta}_{j,t}\}$ for each bit of $\mathbf{w} = C_{\text{pub}}(\mathbf{x})$.

$$\text{If } \mathbf{w}[j] = 1, \quad \{\Delta_{j,t}\}_{t \in [3\lambda]}, \quad \text{o/w}, \quad \{\overline{\Delta}_{j,t}\}_{t \in [3\lambda]}.$$

Note that due to the $1/(2\lambda)$-correctness error from aHMAC, some of the recovered shares (but less than λ of them) may be incorrect. But this suffices for (robust) reconstruction of the secret sharing scheme.

2'. Apply robust secret share reconstruction to recover the decryption keys:

$$\text{If } \mathbf{w}[j] = 1, \quad \Delta_j \leftarrow \mathsf{Recon}(\{\Delta_{j,t}\}), \quad \text{o/w}, \quad \overline{\Delta}_j \leftarrow \mathsf{Recon}(\{\overline{\Delta}_{j,t}\}).$$

Then continue as in Construction 28 using the recovered decryption keys.

Correctness: As noted, correctness follows from the robust reconstruction property of the secret sharing scheme, which is able to tolerate up to $\lambda - 1$ incorrect shares.

Efficiency: Compared to Construction 28, the label sizes remain $\text{poly}(\lambda)$ bits each, while the garbled circuit size is increased by at most 3λ times. We still have $|\widehat{C}| = |C_{\text{priv}}| \cdot \text{poly}(\lambda)$.

Security: We state and prove the following security lemma.

Lemma 6.11. *Construction 29 is a secure partial garbling scheme.*

of Lemma 6.11. The security of a partial garbling scheme (Definition 6.1) requires a simulator Sim, given a two-input circuit C, a public input $\mathbf{x}$, and output labels $\{L_z^{(i)}\}$, to simulate a garbled circuit $\widehat{C}$ and input labels $\{L_x^{(i)}\}$, $\{L_y^{(i)}\}$. It simulates them as follows:

- Run the (standard) Boolean garbling simulator $\mathsf{BG.Sim}$ to compute

$$\widetilde{C_{\text{priv}}}, \{\widetilde{L}_w^{(i)}\}, \{\widetilde{L}_y^{(i)}\} \leftarrow \mathsf{BG.Sim}(1^\lambda, C_{\text{priv}}, \{L_z^{(i)}\}).$$

- Sample random encryptions keys $\boldsymbol{\Delta} = (\ldots, \Delta_j, \ldots)_j$, $\overline{\boldsymbol{\Delta}} = (\ldots, \overline{\Delta}_j, \ldots)_{j\in[\ell_w]}$, and simulate ciphertexts $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}_{j\in[\ell_w]}$ of labels $L_w^{(j)}$ according to $\mathbf{w} = C_{\text{pub}}(\mathbf{x})$.

$$\mathsf{ct}_j \leftarrow \begin{cases} \mathsf{E.Enc}(\Delta_j, L_w^{(j)}) \\ \mathsf{E.Enc}(\Delta_j, 0) \end{cases} \qquad \overline{\mathsf{ct}}_j \leftarrow \begin{cases} \mathsf{E.Enc}(\overline{\Delta}_j, 0) & \text{if } \mathbf{w}[j] = 1 \\ \mathsf{E.Enc}(\overline{\Delta}_j, L_w^{(j)}) & \text{o/w} \end{cases}. \tag{6.17}$$

- Secret share the encryption keys according to $\mathbf{w}$:

$$\{\Delta_{j,t}\}_t \leftarrow \begin{cases} \mathsf{Share}(\Delta_j), \\ \mathsf{Share}(0), \end{cases} \qquad \{\overline{\Delta}_{j,t}\}_t \leftarrow \begin{cases} \mathsf{Share}(0) & \text{if } \mathbf{w}[j] = 1, \\ \mathsf{Share}(\overline{\Delta}_j) & \text{o/w.} \end{cases}. \tag{6.18}$$

- Proceed as in Construction 29 to compute $\mathsf{Tables}_t = (\mathsf{evk}_t, \overline{\mathsf{evk}}_t, \mathbf{k}_{\text{pub},t}, \overline{\mathbf{k}}_{\text{pub},t})$ and labels $\{L_{x,t}^{(i)}\}_i$ from the t-th shares denoted as $\mathsf{Keys}_t = \{\Delta_{j,t}, \overline{\Delta}_{j,t}\}_j$. As a shorthand, we write

$$(\mathsf{Tables}_t, \{L_{x,t}^{(i)}\}_i) \leftarrow \mathsf{Comps}_{\mathbf{x},C_{\text{pub}}}(\mathsf{Keys}_t)$$

to mean the computations in this step, summarized here for reference.

$$\begin{aligned}
&\mathsf{Comps}_{\mathbf{x},C_{\text{pub}}}(\mathsf{Keys} = \{\Delta_j, \overline{\Delta}_j\}_j): \\
&\text{Denote } \boldsymbol{\Delta} = (\ldots, \Delta_j, \ldots)_j,\ \overline{\boldsymbol{\Delta}} = (\ldots, \overline{\Delta}_j, \ldots)_j, \\
&\text{Compute} \\
&(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(1^\lambda, \boldsymbol{\Delta}), \qquad (\overline{\mathsf{sk}}, \overline{\mathsf{evk}}), \leftarrow \mathsf{KeyGen}(1^\lambda, \overline{\boldsymbol{\Delta}}), \\
&\quad \mathbf{k}_{\text{pub}} \leftarrow \mathsf{EvalKey}(\mathsf{sk}, C_{\text{pub}}, \mathbf{id}), \qquad \overline{\mathbf{k}}_{\text{pub}} \leftarrow \mathsf{EvalKey}(\overline{\mathsf{sk}}, \overline{C_{\text{pub}}}, \overline{\mathbf{id}}) \\
&\quad \sigma_x^{(i)} \leftarrow \mathsf{Auth}(\mathsf{sk}, \mathbf{x}[i], \mathbf{id}[i]), \qquad \overline{\sigma}_x^{(i)} \leftarrow \mathsf{Auth}(\overline{\mathsf{sk}}, \mathbf{x}[i], \overline{\mathbf{id}}[i]), \\
&\text{Output } \mathsf{Tables} = (\mathsf{evk}, \overline{\mathsf{evk}}, \mathbf{k}_{\text{pub}}, \overline{\mathbf{k}}_{\text{pub}}),\ \text{and } \{L_x^{(i)} = (\sigma_x^{(i)}, \overline{\sigma}_x^{(i)})\}_i.
\end{aligned} \tag{6.19}$$

We further use the notation $\mathsf{Keys}_t[\mathbf{w}]$ to mean the subset:

$$\mathsf{Keys}_t[\mathbf{w}] = \{\Delta_{j,t} : \mathbf{w}[j] = 1\} \cup \{\overline{\Delta}_{j,t} : \mathbf{w}[j] = 0\}.$$

- Output the simulated garbled circuit $\widetilde{C} = (\widetilde{C_{\text{priv}}}, \{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}, \{\mathsf{Tables}_t\})$, and input labels $\{\widetilde{L}_x^{(i)} = (\ldots, L_{x,t}^{(i)}, \ldots)_t\}$ and $\{\widetilde{L}_y^{(i)}\}$.

We first argue that the function $\mathsf{Comps}_{\mathbf{x}, C_{\text{pub}}}$ is a $1/(2\lambda)$-leaking computation with respect to the subset $\mathsf{Keys}_t[\overline{\mathbf{w}}]$.

Claim 6.4. *For every polynomial $p(\lambda)$, and every efficient distinguisher $\mathcal{A}$, there exists a negligiable function* negl *such that for every $\lambda \in \mathbb{N}$, sequence of Boolean circuits $\{f_\lambda\}$ with $|f_\lambda| \leq p(\lambda)$, inputs $\{\mathbf{x}_\lambda\}$, and inputs $\{\mathsf{Keys}_\lambda\}$, the following holds.*

$$\begin{aligned}&\left|\Pr[\mathcal{A}(\mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys})) = 1] - \Pr[\mathcal{A}(\mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys}'))) = 1]\right| \\ &< 1/\lambda + \text{negl}(\lambda),\end{aligned}$$

where $\mathsf{Keys}'[\mathbf{w}] = \mathsf{Keys}[\mathbf{w}]$, and $\mathsf{Keys}'[\overline{\mathbf{w}}] = 0$, for $\mathbf{w} = f(\mathbf{x})$.

of Claim. We show a series of hybrids that transition from $\mathsf{Hyb}'_0 = \mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys})$ to $\mathsf{Hyb}'_3 = \mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys}')$.

Hyb'_0 : The output of this hybrid is summarized in Equation 6.19.

Hyb'_1 : Instead of computing $\mathbf{k}_f$, $\overline{\mathbf{k}}_f$ as in Equation 6.19, Hyb'_1 simulates them as

$$\begin{aligned}
&\sigma_{\mathbf{x}} := (\ldots, \sigma_x^{(i)}, \ldots)_i, && \overline{\sigma}_{\mathbf{x}} := (\ldots, \overline{\sigma}_x^{(i)}, \ldots)_i, \\
&\sigma_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\mathsf{evk}, f, \mathbf{x}, \sigma_{\mathbf{x}}), && \overline{\sigma}_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\overline{\mathsf{evk}}, \overline{f}, \mathbf{x}, \overline{\sigma}_{\mathbf{x}}), \\
&\mathbf{k}_f[j] \leftarrow \begin{cases} \sigma_{\mathbf{w}}[j] - \Delta_j, \\ \sigma_{\mathbf{w}}[j], \end{cases} && \overline{\mathbf{k}}_f[j] \leftarrow \begin{cases} \overline{\sigma}_{\mathbf{w}}[j], & \text{if } \mathbf{w}[j] = 1, \\ \overline{\sigma}_{\mathbf{w}}[j] - \overline{\Delta}_j, & \text{o/w}, \end{cases}
\end{aligned} \tag{6.20}$$

where $\mathbf{w} = f(\mathbf{x})$. The $1/(2\lambda)$-correctness of aHMAC (Definition 6.12) ensures that the simulation is correct except with probability $\leq 1/(2\lambda)$. Hence

$$|\Pr[\mathcal{A}(\mathsf{Hyb}'_1) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}'_0) = 1| \leq 1/(2\lambda).$$

Hyb'_2 : Instead of computing evk, $\overline{\mathsf{evk}}$, $\sigma_{\mathbf{x}}$, and $\overline{\sigma}_{\mathbf{x}}$ as in Equation 6.19, Hyb'_2 simulates them using the simulator $\mathsf{aHMAC.Sim}$

$$(\mathsf{evk}, \sigma_{\mathbf{x}}) \leftarrow \mathsf{aHMAC.Sim}(1^\lambda), \quad (\overline{\mathsf{evk}}, \overline{\sigma}_{\mathbf{x}}) \leftarrow \mathsf{aHMAC.Sim}(1^\lambda).$$

The security of aHMAC ensures

$$|\Pr[\mathcal{A}(\mathsf{Hyb}'_2) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}'_1) = 1| \leq \text{negl}(\lambda).$$

Note that in Hyb'_2, the input keys $\{\Delta_j, \overline{\Delta}_j\}$ are only used for deriving $\mathbf{k}_f$, $\overline{\mathbf{k}}_f$ from the evaluated tags $\sigma_{\mathbf{w}}$, $\overline{\sigma}_{\mathbf{w}}$, as in Equation 6.20. In particular, Hyb'_2 is independent of $\mathsf{Keys}[\overline{\mathbf{w}}]$.

Hyb'_3 : The output of this hybrid is $\mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys}')$, where $\mathsf{Keys}'[\mathbf{w}] = \mathsf{Keys}[\mathbf{w}]$, and $\mathsf{Keys}'[\overline{\mathbf{w}}] = 0$.

The same arguments from Hyb'_1, Hyb'_2, in reverse order, shows

$$|\Pr[\mathcal{A}(\mathsf{Hyb}'_3) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}'_2) = 1| \leq 1/(2\lambda) + \text{negl}(\lambda).$$

By a hybrid argument, we conclude that $|\Pr[\mathcal{A}(\mathsf{Hyb}'_3) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}'_0) = 1| \leq 1/\lambda + \text{negl}(\lambda)$, which proves the claim. $\square$

We will then use the following lemma from [BGIK22], which is further based on a computational hardcore lemma from [MT10]. The lemma intuitively says that a δ-leaking distribution with respect to some secret m can be simulated by another distribution that completely leaks m with probability δ, and perfectly hides m with probability $1 - \delta$.

Lemma 6.12 (Simulating Leaky Functions [BGIK22])**.** *Let* Leaky *be an efficiently computable randomized function with domain* $\mathcal{M}(\lambda)$*, and* $\delta = \delta(\lambda)$ *be a bound such that for every sequence of inputs* $\{m_\lambda\}, \{m'_\lambda\}$*, every polynomial distinguisher* $\mathcal{A}$*, it holds for sufficiently large* λ *that*

$$\left|\Pr[\mathcal{A}(1^\lambda, \mathsf{Leaky}(m_\lambda)) = 1] - \Pr[\mathcal{A}(1^\lambda, \mathsf{Leaky}(m'_\lambda)) = 1]\right| < \delta(\lambda).$$

Then, there exists randomized functions:

- $\mathsf{Erase}_\delta : \mathcal{M} \to \mathcal{M} \cup \{\bot\}$ *such that for every* $m \in \mathcal{M}$,

$$\Pr[\mathsf{Erase}_\delta(m) = m] \le \delta, \quad \Pr[\mathsf{Erase}_\delta(m) = \bot] = 1 - \Pr[\mathsf{Erase}_\delta(m) = m].$$

- $\mathsf{SimLeaky}$ *such that for every sequence* $\{m_\lambda\}$,

$$\{\mathsf{Leaky}(m_\lambda)\} \approx_c \{\mathsf{SimLeaky}(\mathsf{Erase}_\delta(m_\lambda))\},$$

where Erase_δ *and* $\mathsf{SimLeaky}$ *depend on the same random coins.*

Let $\mathsf{Leaky}(\mathsf{Keys}[\overline{\mathbf{w}}]) := \mathsf{Comps}(\mathsf{Keys})$, it follows that there exists $\mathsf{Erase}_{1/\lambda}$, $\mathsf{SimLeaky}$ such that $\mathsf{Leaky}(\mathsf{Keys}[\overline{\mathbf{w}}]) \approx_c \mathsf{SimLeaky}(\mathsf{Erase}_{1/\lambda}(\mathsf{Keys}[\overline{\mathbf{w}}]))$.

We now show a series of hybrids that transitions from the real-world distribution in Definition 6.1 (Hyb_0) to the above simulated distribution (Hyb_6).

Hyb_0 : We summarize the real-world distribution of the garbled circuit $\widehat{C} = (\widehat{C_{\text{priv}}}, \{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}, \{\mathsf{Tables}_t\})$ and labels $\{L_x^{(i)} = (\ldots, L_{x,t}^{(i)}, \ldots)\}$, $\{L_y^{(i)}\}$.

$$\begin{array}{r|c} L_y^{(i)} = K_y^{(i)}(\mathbf{y}[i]), \quad \widehat{C_{\text{priv}}}, & \widehat{C_{\text{priv}}}, \{K_w^{(j)}\}, \{K_y^{(i)}\} \leftarrow \mathsf{BG.Garb}(h, \{K_z^i\}), \\ \mathsf{ct}_j \leftarrow \mathsf{E.Enc}(\Delta_j, K_w^{(j)}(1)), & \Delta_j, \overline{\Delta}_j \leftarrow \{0,1\}^\lambda \\ \overline{\mathsf{ct}}_j \leftarrow \mathsf{E.Enc}(\overline{\Delta}_j, K_w^{(j)}(0)), & \end{array} \tag{6.21}$$

$$\begin{array}{r|l} (\{L_{x,t}^{(i)}\}, \mathsf{Tables}_t) & \{\Delta_{j,t}\} \leftarrow \mathsf{Share}(\Delta_j),\ \{\overline{\Delta}_{j,t}\} \leftarrow \mathsf{Share}(\overline{\Delta}_j) \\ \leftarrow \mathsf{Comps}_{\mathbf{x},f}(\mathsf{Keys}_t). & \mathsf{Keys}_t = \{\Delta_{j,t}, \overline{\Delta}_{j,t}\}_j \end{array} \tag{6.22}$$

Hyb_1 : Instead of computing $\{L_{x,t}^{(i)}\}, \mathsf{Tables}_t$ as in Equation 6.19, Hyb_1 simulates them using the $\mathsf{Erase}_{1/\lambda}$ and $\mathsf{SimLeaky}$ algorithms from Lemma 6.12:

$$(\{\widetilde{L}_{x,t}^{(i)}\}, \widetilde{\mathsf{Tables}_t}) \leftarrow \mathsf{SimLeaky}(\mathsf{Erase}_{1/\lambda}(\mathsf{Keys}_t[\overline{\mathbf{w}}])).$$

Lemma 6.12 ensures that $\mathsf{Hyb}_1 \approx_c \mathsf{Hyb}_0$.

Hyb_2 : Proceeds as in Hyb_1, but aborts if there are $\ge \lambda$ instances (among 3λ) of $\mathsf{Erase}_{1/\lambda}$ that doesn't erase its input.

Since each instance of $\mathsf{Erase}_{1/\lambda}$ independently erases with probability $> 1 - 1/\lambda$, by Chernoff bound, abort happens with negligiable probability. Hence $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_1$.

Hyb_3 : Instead of computing the shares in as in Equation 6.19, Hyb_3 simulates them according to $\mathbf{w}$ as in Equation 6.18.

Note that the changed shares are exactly those in $\mathsf{Keys}_t[\overline{\mathbf{w}}]$, which are erased except for $\leq \lambda - 1$ indices t. The $(\lambda - 1)$-privacy of secret sharing (Definition 6.20) ensures $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_2$.

Hyb_4 : Change back to computing $(\{L_{x,t}^{(i)}\}, \mathsf{Tables}_t)$ from $\mathsf{Comps}_{\mathbf{x},C_{\text{pub}}}$ as in Equation 6.19, instead of using Erase_λ and $\mathsf{SimLeaky}$, as they are potentially inefficient. (The shares are still simulated as in Equation 6.18.)

Lemma 6.12 again ensures that $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_3$.

We draw attention to the keys Δ_j for $\mathbf{w}[j] = 0$, and $\overline{\Delta_j}$ for $\mathbf{w}[j] = 1$. Hyb_3 has changed from computing secret shares of those keys to secret shares of zeros. Hence they are not used in the current experiment except for encrypting the corresponding ciphertexts in $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}$.

Hyb_5 : Instead of computing the ciphertexts $\{\mathsf{ct}_j, \overline{\mathsf{ct}}_j\}$ as in Equation 6.21, simulate them as in Equation 6.17 (re-created here), where $L_w^{(j)} := K_w^{(j)}[\mathbf{w}[j]]$.

$$\mathsf{ct}_j \leftarrow \begin{cases} \mathsf{E.Enc}(\Delta_j, L_w^{(j)}) \\ \mathsf{E.Enc}(\Delta_j, 0) \end{cases} \qquad \overline{\mathsf{ct}}_j \leftarrow \begin{cases} \mathsf{E.Enc}(\overline{\Delta}_j, 0) & \text{if } \mathbf{w}[j] = 1 \\ \mathsf{E.Enc}(\overline{\Delta}_j, L_w^{(j)}) & \text{o/w} \end{cases},$$

The semantic security of the encryption scheme E ensures that $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_4$.

Hyb_6 : Instead of computing $\widehat{C_{\text{priv}}}$, $\{L_y^{(i)}\}$ and $\{L_w^j = K_w^{(j)}[\mathbf{w}[j]]\}$ as in Equation 6.21, simulate them using the simulator $\mathsf{BG.Sim}$ guaranteed by the security of Boolean garbling:

$$(\widetilde{C_{\text{priv}}}, \{\widetilde{L}_w^{(j)}\}, \{\widetilde{L}_y^{(i)}\}) \leftarrow \mathsf{BG.Sim}(1^\lambda, C_{\text{priv}}, \{L_z^{(i)}\}).$$

The security of Boolean garbling ensures $\mathsf{Hyb}_6 \approx \mathsf{Hyb}_5$.

By a hybrid argument, we conclude that $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_5$, which proves the lemma. □

6.5.3 Implications: Succinct Secret Sharing, Garbling, and PSM

We first point out an immediate implication to succinct secret sharing for partite functions (See [ABI+23] for a formal definition). Such a secret sharing scheme has n pairs of shareholders (i.e., $2n$ in total), and an access structure define by a function $f : \{0,1\}^n \to \{0,1\}$ in the following way:

1. For all $\mathbf{x} \in \{0,1\}^n$ such that $f(\mathbf{x}) = 1$, the subset of n share holders, one from each i-th pair "selected" by $\mathbf{x}[i]$, can recover the secret.

2. A subset that contains two shareholders from any pair can recover the secret.

3. Subsets other than the above learn nothing about the secret.

Such a secret sharing scheme can be implemented by a partial garbling of the circuit $C(\mathbf{x}, s) = f(\mathbf{x}) \cdot s$, where s is the secret. Each of the n pairs of shareholders is assigned the two possible partial garbling input labels for $\mathbf{x}[i]$, and the garbled circuit $\widehat{C}$ can be released as public information or sent to all shareholders. [8]

Corollary 6.4 (Succinct Secret Sharing for Partite Functions)**.** *There exists a succinct secret sharing for partite functions specified as Boolean circuits (of unbounded size), where the share sizes are* $\mathrm{poly}(\lambda)$ *bits, assuming any for the assumptions from Theorem 6.10:*

1. *CP-DDH in either the NIDLS framework or prime-order groups;*

2. *the KDM-DCR assumption.*

[8]Technically we have only ensured condition 1, 3 of the access structure. We can additionally send an additive share of the secret to each pair of shareholders to ensure they can always jointly recover the secret.

There also exists a scheme for partite functions specified as bounded-depth (and unbounded size) Boolean circuits, where the share sizes are $\mathrm{poly}(\lambda)$ *bits, assuming any for the assumptions from Theorem 6.11:*

1. *DDH in either the NIDLS framework or prime-order groups;*
2. *The DCR assumption.*

Next, we sketch how to "upgrade" a partial garbling to a (fully private) standard garbling using a homomorphic encryption (HE) scheme, following the blueprint of [GKP+13b]. To garble a program P, we define

$$P'(\mathbf{x}, \mathbf{y}) := \mathsf{HE.Dec}(\mathbf{y}, \mathsf{HE.Eval}(P, \mathbf{x})),$$

which is supposed to take HE ciphertexts as the public input $\mathbf{x}$, and a HE decryption key as the private input $\mathbf{y}$. A partial garbling of P' then implements a standard garbling of P. Intuitively, the partial garbling security ensures the HE decryption key (as the private input $\mathbf{y}$) is hidden, while the HE security ensures the actual input encrypted in HE ciphertexts (as the public input $\mathbf{x}$) are hidden.

Assuming a succinct partial garbling for circuits and a compact HE in the above construction, the resulting garbling scheme is also succinct (Definition 6.3). We further note that if the HE evaluation algorithm $\mathsf{HE.Eval}$ has bounded computation depth, then a succinct partial garbling for bounded-depth circuits suffices.

Definition 6.21 (Homomorphic Encryption Schemes (HE))**.** *A (secret-key) homomorphic encryption scheme for the class of programs* $\mathcal{P} = \{\mathcal{P}_\lambda\}$ *with Boolean inputs consists of four efficient algorithms.*

- $\mathsf{KeyGen}(1^\lambda)$ *outputs public parameters* pp *and a secret key* sk*.*
- $\mathsf{Enc}(\mathsf{pp}, \mathsf{sk}, x \in \{0,1\})$ *takes a secret key* sk *and a message* x*. It outputs a ciphertext* ct*.*
- $\mathsf{Eval}(\mathsf{pp}, P \in \mathcal{P}_\lambda, \{\mathsf{ct}_i\})$ *takes a program* $P : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_y}$ *and* ℓ_x *ciphertexts. It outputs a evaluated ciphertext* ct^**.*

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}^*)$ *takes a (evaluated) ciphertext and outputs messages* $\mathbf{y} \in \{0,1\}^{\ell_y}$.

Correctness. *For every* $\lambda \in \mathbb{N}$, *program* $P \in \mathcal{P}_\lambda$ *with* ℓ_x *inputs, and input* $\mathbf{x} \in \{0,1\}^{\ell_x}$, *the following holds:*

$$\Pr\left[P(\mathbf{x}) = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}^*) \;\middle|\; \begin{array}{c} (\mathsf{sk}, \mathsf{pp}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, \mathbf{x}[i]) \\ \mathsf{ct}^* \leftarrow \mathsf{Eval}(\mathsf{pp}, P, \{\mathsf{ct}_i\}) \end{array} \right] = 1.$$

Security. *The standard semantic security for secret-key encryption schemes should hold. (We omit writing it out here.)*

Compactness. *An evaluated ciphertext* ct^* *by* Eval *has bit-length* $\mathrm{poly}(\lambda, \ell_y)$ *independent of the program size, except the output length* ℓ_y.

Lemma 6.13 (HE Schemes)**.** *There exist the following constructions:*

1. *[IP07, DGI+19] Assuming the DCR assumption or DDH in prime-order groups, for any polynomial* $\ell(\lambda)$, *there exists a compact homomorphic encryption scheme for the class of branching programs with bounded length by* $\ell(\lambda)$ *and unbounded size, i.e.,* $\mathcal{P}^\ell = \{\mathcal{P}^\ell_\lambda\}$ *where* $\mathcal{P}^\ell_\lambda$ *consits of branching programs with length below* $\ell(\lambda)$ *and size below* $2^{\mathrm{poly}(\lambda)}$.

2. *[BGN05] Assuming the subgroup decision problem in bilinear groups of composite order, there exists a compact somewhat homomorphic encryption scheme for the class of quadratic polynomials (mod 2) of unbounded size, i.e.,* $\mathcal{Q} = \{\mathcal{Q}_\lambda\}$ *where* $\mathcal{Q}_\lambda$ *consits of quadratic polynomials with below* $2^{\mathrm{poly}(\lambda)}$ *number of monomials.*

Furthermore, the computation depth of the Eval *algorithms in the above schemes are bounded by fixed polynomials* $\mathrm{poly}(\lambda)$, *independent of program sizes.*

Theorem 6.12 (Succinct Garbling for Bounded-Length BPs)**.** *There exists a succinct garbling for bounded-length (and unbounded size) branching programs assuming (1) a succinct partial garbling scheme for bounded-depth circuits and (2) a compact homomorphic encryption scheme for bounded-length branching programs.*

Theorem 6.13 (Succinct Garbling for Quadratic Poly)**.** *There exists a succinct garbling for quadratic polynomials (mod 2, and unbounded size) assuming (1) a succinct partial garbling scheme for bounded-depth circuits and (2) a compact homomorphic encryption scheme for quadratic polynomials.*

We point out truth tables as special cases of bounded-length branching programs: a truth table for ℓ_x inputs can be represented by a decision tree of depth ℓ_x, which is also a branching program of length ℓ_x. We therefore obtain succinct garbling for truth tables where the garbling size only depends polynomially on the input length.

As outlined in [FKN94a], a garbling scheme implements a multi-party private simultaneous messages (PSM) protocol. we obtain the following corollary.

Corollary 6.5 (k-party Computational PSM for Truth Tables)**.** *For any constant k,* [9] *any k-party function $f : [N]^k \to [N]$ has a computationally secure PSM protocol with* $\mathrm{poly}(\lambda, \log N)$ *communication and* $\mathrm{poly}(\lambda, N)$ *computation.*

Note that we have imposed composability of garbled circuits at the syntax level (Definition 6.1). So we can use the succinct garbling schemes for a program class $\mathcal{P} = \{\mathcal{P}_\lambda\}$ from Theorem 6.12 and 6.13 in an outer Boolean garbling scheme to handle general P-gates for any $P \in \mathcal{P}_\lambda$.

Corollary 6.6. *There exists a garbling scheme for circuits composed of general gates P that each implements any bounded-length (and unbounded size) branching program or any quadratic polynomial (mod* 2*, of unbounded size), where the garbling size is #wires* $\cdot\, \mathrm{poly}(\lambda)$*, under the union of following assumptions:*

- *Any of the assumptions from Theorem 6.11;*
- *The DCR assumption, or DDH in prime-order groups;*
- *The subgroup decision problem in bilinear groups of composite order.*

[9] For super-constant k, the protocol will have communication $\mathrm{poly}(\lambda, \log(N^k))$, and computation $\mathrm{poly}(\lambda, N^k)$.

6.6 Application: 1-Key Selective CPRF for Circuits

The notion of constrained PRFs (CPRF) is first proposed (concurrently) in [BW13, KPTZ13, BGI14], where besides the usual pair of algorithms KeyGen and Eval for PRFs, there exists another pair $\mathsf{Constrain}$ and CEval. $\mathsf{Constrain}$ produces a constrained key sk_C with respect to a Boolean circuit C. CEval can use sk_C to evaluate the PRF on any point $\mathbf{x}$ such that $C(\mathbf{x}) = 0$.

The original definitions consider the setting where an adversary may adaptively obtain multiple constrained keys with respect to different circuits. This is referred to as multi-key CPRF. However, for circuit constraints (even low depth ones), multi-key CPRF is only known from heavy tools like indistinguishability obfuscation (iO) and functional encryption. Known non-iO based constructions [BV15, CC17, BTVW17, AMN+18, CVW18, PS18, CMPR23] only achieve the weaker definition, where the adversary obtains only a single selectively chosen constrained key. This is referred to as 1-key selective CPRF, and is also what we achieve in this work.

Definition 6.22 (Constrained Pseudorandom Functions). *Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ be classes of Boolean circuits where circuits in C_λ have domain $\mathcal{X}_\lambda$ and range $\{0,1\}$. A constrained pseudorandom function (CPRF) with domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_\lambda$, key space $\mathcal{K} = \{\mathcal{K}_\lambda\}_\lambda$, and range $\mathcal{Z} = \{\mathcal{Z}_\lambda\}_\lambda$ supporting circuit constraints $\mathcal{C}$ consists of four efficient algorithms:*

- $\mathsf{KeyGen}(1^\lambda)$ *outputs public parameters* pp *and a master secret key* msk.
- $\mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x \in \mathcal{X})$ *takes as inputs the master secret key* msk *and an input x. It outputs an evaluation result $z_0 \in \mathcal{Z}$.*
- $\mathsf{Constrain}(\mathsf{pp}, \mathsf{msk}, C \in \mathcal{C})$ *takes as inputs the master secret key* msk *and a constraint circuit C. It outputs a constrained key* sk_C.
- $\mathsf{CEval}(\mathsf{pp}, \mathsf{sk}_C, x)$ *takes as inputs the constrained key* sk_C *and an input x. It outputs an evaluation result $z_1 \in \mathcal{Z}$.*

Correctness: *There exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$, any circuit*

$C \in \mathcal{C}_\lambda$, and input $x \in \mathcal{X}_\lambda$ such that $C(x) = 0$, the following holds:

$$\Pr\left[\begin{array}{c|r} \mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x) & (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ = \mathsf{CEval}(\mathsf{pp}, \mathsf{sk}_C, x) & \mathsf{sk}_C \leftarrow \mathsf{Constrain}(\mathsf{pp}, \mathsf{msk}, C) \end{array}\right] \geq 1 - \text{negl}(\lambda).$$

1-Key Selective Security: *For any efficient adversary $\mathcal{A}$, there exists a negligible function* $\text{negl}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$:*

$$\left|\Pr[\text{Exp}^{\mathcal{A},0}_{\text{CPRF}}(\lambda) = 1] - \Pr[\text{Exp}^{\mathcal{A},1}_{\text{CPRF}}(\lambda) = 1]\right| \leq \text{negl}(\lambda),$$

where the experiment $\text{Exp}^{\mathcal{A},b}_{\text{CPRF}}$ is as follows:

1. *Launch $\mathcal{A}(1^\lambda)$ and receives a selective challenge constraint $C \in \mathcal{C}_\lambda$.*

2. *Run $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\mathsf{sk}_C \leftarrow \mathsf{Constrain}(\mathsf{msk}, C)$, and send $\mathsf{pp}, \mathsf{sk}_C$ to the adversary $\mathcal{A}$.*

3. *Answer queries x from $\mathcal{A}$ with evaluations $z \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x)$.*

4. *Receive from $\mathcal{A}$ a challenge x^* that's never queried before and such that $C(x^*) \neq 0$. Sends z^* to $\mathcal{A}$ computed as follows:*

$$z^* \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, x^*) \quad \text{if } b = 0$$
$$z^* \leftarrow \mathcal{Z}_\lambda \quad \text{if } b = 1.$$

5. *Answer queries $x \neq x^*$ from $\mathcal{A}$ as in step 3.*

6. *In the end, $\mathcal{A}$ outputs a bit b' as the experiment result.*

As explained in the technical overview, our construction relies on an extended syntax of homomorphic secret sharing schemes (HSS), formalized in Definition 6.23. We recap the observation of [CMPR23] that common HSS schemes for restricted multiplication straight-line programs (RMS) satisfy this syntax.

In an RMS program P, all inputs $\mathbf{x}$ are first converted into memory (i.e. intermediate) wires. Additions are allowed between two memory wires, but multiplications are restricted to between a memory wire and an input. Note that the initial conversion can be implemented by multiplying input wires with a constant memory wire of 1.

The observation is that if one change the initial conversion to using a constant memory wire of some value w instead of 1, while keeping the rest of the evaluation unchanged, then the final result becomes $w \cdot P(\mathbf{x})$.

The observation becomes useful in the context of evaluating RMS programs using HSS, because in common schemes the share format of memory wires are much simpler than the share format of inputs. In particular, any subtractive share (over $\mathbb{Z}$) of $\mathbf{\Delta} \cdot w$ is a valid memory share of w, where $\mathbf{\Delta}$ is a secret vector in the HSS scheme.

Definition 6.23 (Extended Homomorphic Secret Sharing)**.** *An extended homomorphic secret sharing scheme for a class of programs $\mathcal{P}$ (defined over a ring $\mathcal{R}$) with input space $\mathcal{I} \subseteq \mathcal{R}$ consists of three efficient algorithms:*

- $\mathsf{Setup}(1^\lambda)$ *outputs a public key* pk, *a pair of evaluation keys* $\mathsf{evk}_0, \mathsf{evk}_1$, *and an "extension secret" as an integer vector* $\mathbf{\Delta} \in [2^\lambda]^{\ell_d}$.
- $\mathsf{Input}(\mathsf{pk}, x \in \mathcal{I})$ *takes as inputs the public key* pk *and an input* x. *It outputs a pair of input shares* I_0, I_1.
- $\mathsf{ExtEval}(\beta \in \{0,1\}, \mathsf{evk}_\beta, \mathbf{I}_\beta, \mathbf{\Delta}_\beta, P \in \mathcal{P})$ *takes as inputs a party identity* β, *its evaluation key* evk_β *and input shares* $\mathbf{I}_\beta$, *its share of the extension secret* $\mathbf{\Delta}_\beta$, *and a program* P. *It outputs its share of the evaluation result* $z_\beta \in \mathcal{R}$.

Extended Evaluation Correctness: *For all polynomial* $p(\lambda)$, *there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for all* $\lambda \in \mathbb{N}$, *any program* $P \in \mathcal{P}$ *with* n *inputs and* 1 *output, and with size* $|P| \leq p(\lambda)$, *any inputs* $\mathbf{x} \in \mathcal{I}^n$, *any extension bit* $w \in \{0,1\}$, *and any share vector*

$\mathbf{\Delta}_0 \in \mathbb{Z}^{\ell_d}$, *the following holds:*

$$\Pr\left[\begin{array}{l|c} & (\mathsf{pk}, \mathsf{evk}_0, \mathsf{evk}_1, \mathbf{\Delta}) \leftarrow \mathsf{Setup}(1^\lambda) \\ & (I_0^{(i)}, I_1^{(i)}) \leftarrow \mathsf{Input}(\mathsf{pk}, \mathbf{x}[i]) \\ z_1 - z_0 = w \cdot P(\mathbf{x}) & \mathbf{I}_\beta := (I_\beta^{(0)}, \dots, I_\beta^{(\ell_x - 1)}) \\ (\text{over } \mathcal{R}) & \mathbf{\Delta}_1 := \mathbf{\Delta}_0 + \mathbf{\Delta} \cdot w \quad (\text{over } \mathbb{Z}) \\ & z_\beta \leftarrow \mathsf{ExtEval}(b, \mathsf{evk}_\beta, \mathbf{I}_\beta, \mathbf{\Delta}_\beta, P) \end{array}\right] \geq 1 - \text{negl}(\lambda).$$

Security: *For any efficient adversary $\mathcal{A}$, there exists a negligible function* $\text{negl}(\lambda)$ *such that for all $\lambda \in \mathbb{N}$ and for all $\beta \in \{0,1\}$:*

$$\left|\Pr[\text{Exp}_{\text{HSS}}^{\mathcal{A},\beta,0}(\lambda) = 1] - \Pr[\text{Exp}_{\text{HSS}}^{\mathcal{A},\beta,1}(\lambda) = 1]\right| \leq \text{negl}(\lambda),$$

where the experiment $\text{Exp}_{\text{HSS}}^{\mathcal{A},\beta}$ *is as follows:*

1. *Launch $\mathcal{A}(1^\lambda)$ and receive challenge inputs $x_0, x_1 \in \mathcal{I}$.*

2. *Run* $(\mathsf{pk}, \mathsf{evk}_0, \mathsf{evk}_1, \mathbf{\Delta}) \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $(I_0, I_1) \leftarrow \mathsf{Input}(\mathsf{pk}, x_b)$. *Then send* $(\mathsf{pk}, \mathsf{evk}_\beta, I_\beta)$ *to $\mathcal{A}$.*

3. *In the end, $\mathcal{A}$ outputs a bit b' as the experiment result.*

Remark. From an extended HSS we can obtain a "normal" HSS by viewing each party's evaluation key evk_β together with a subtractive share of $\mathbf{\Delta}$ as its overall evaluation key: $\mathsf{evk}_\beta^* := (\mathsf{evk}_\beta, \mathbf{\Delta}_\beta)$. This corresponds to setting the extension bit $w = 1$. Hence the evaluated shares satisfy $z_1 - z_0 = w \cdot P(\mathbf{x}) = P(\mathbf{x})$.

Lemma 6.14 (Extended HSS)**.** *There exists an extended HSS scheme for NC1 assuming either of the following:*

1. *[OSY21] The DCR assumption.*

2. *[ADOS22] DDH and the small exponent assumption (Definition 6.9) in the NIDLS framework.*

In this section, we construct a CPRF by composing a leveled aHMAC and an HSS with extended evaluations.

Theorem 6.14 (CPRF for Circuits)**.** *For any polynomial $\ell(\lambda)$, let $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ be the class of Boolean circuits with sizes bounded by $\ell(\lambda)$:*

$$\mathcal{C}_\lambda := \left\{C : \{0,1\}^\lambda \to \{0,1\} : |C| < \ell(\lambda)\right\}.$$

There exists a 1-key selective CPRF for $\mathcal{C}$ assuming either of the following:

1. *DDH and the small exponent assumption in the NIDLS framework.*

2. *The DCR assumption.*

Construction 30 (1-Key Selective CPRF for Circuits)**.** Ingredients:

- A leveled aHMAC scheme aHMAC with negl-correctness error.
- An extended HSS scheme HSS for a class $\mathcal{P}$ (defined over a ring $\mathcal{R}$) with input space $\mathcal{I} \subseteq \mathcal{R}$.
- A PRF $\mathsf{F} : \mathcal{K} \times \{0,1\}^\lambda \to \mathcal{R}$ with evaluation in $\mathcal{P}$, and with a compatible key space $\mathcal{K} = \mathcal{I}^{\ell_k}$.

We construct a CPRF for the class of polynomial-sized circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ where

$$\mathcal{C}_\lambda := \left\{C : \{0,1\}^\lambda \to \{0,1\} : |C| < \mathrm{poly}(\lambda)\right\}.$$

As shorthands, in the following we write $U_{\mathbf{x}}(\cdot) = U(\cdot, \mathbf{x})$, and $\mathsf{F}_{\mathbf{x}}(\cdot) = \mathsf{F}(\cdot, \mathbf{x})$, where U is a universal circuit such that $U_{\mathbf{x}}(C) = C(\mathbf{x})$ for all circuits $C \in \mathcal{C}_\lambda$.

$(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$:

Sample a PRF key $\mathbf{s} \leftarrow \mathcal{I}^{\ell_k}$, and generate HSS input shares $\mathbf{I}_0 = (\ldots, I_0^{(i)}, \ldots)$, and $\mathbf{I}_1 = (\ldots, I_1^{(i)}, \ldots)$ of this key:

$$(\mathsf{pk}, \mathsf{HSS.evk}_0, \mathsf{HSS.evk}_1, \mathbf{\Delta}) \leftarrow \mathsf{HSS.Setup}(1^\lambda),$$
$$\text{for } i \in [\ell_k] : \quad (I_0^{(i)}, I_1^{(i)}) \leftarrow \mathsf{HSS.Input}(\mathsf{pk}, \mathbf{s}[i]).$$

Next generate aHMAC keys $\mathsf{sk}, \mathsf{aHMAC.evk}$ w.r.t. the extension secret $\mathbf{\Delta}$:

$$(\mathsf{sk}, \mathsf{aHMAC.evk}) \leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, 1^{\mathrm{Depth}(U)}, \mathbf{\Delta}).$$

Output $\mathsf{pp} = \mathsf{pk}$ and $\mathsf{msk} = (\{I_\beta, \mathsf{HSS.evk}_\beta\}, \mathsf{sk}, \mathsf{aHMAC.evk})$.

$z_0 \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{msk}, \mathbf{x})$:

Parse I_0, $\mathsf{HSS.evk}_0$, and $\{\mathsf{sk}_j\}$ from msk.

Deterministically derive distinct **id** associated with inputs to $U_{\mathbf{x}}$, (i.e. the constraint circuit C,) and derive the zero-share $\mathbf{\Delta}_0$ of the extension secret:

$$\mathbf{\Delta}_0 := \mathbf{k}_U \leftarrow \mathsf{aHMAC.EvalKey}(\mathsf{sk}, U_{\mathbf{x}}, \textbf{id}), \text{[10]}$$

Next run extended HSS evaluation on the zero-shares of the input I_0 and the extension secret $\mathbf{\Delta}_0$:

$$z_0 \leftarrow \mathsf{HSS.ExtEval}(0, \mathsf{evk}_0, \mathbf{I}_0, \mathbf{\Delta}_0, \mathsf{F}_{\mathbf{x}}).$$

$\mathsf{sk}_C \leftarrow \mathsf{Constrain}(\mathsf{pp}, \mathsf{msk}, C)$:

Parse I_1, $\mathsf{HSS.evk}_1$, and $\mathsf{sk}, \mathsf{aHMAC.evk}$ from msk.

Deterministically derive distinct **id** associated to the inputs to $U_{\mathbf{x}}$, and derive tags $\sigma_C := (\ldots, \sigma_C^{(i)}, \ldots)$ for C (viewed as a string):

$$\text{for } i \in \mathsf{bitLen}(C) \quad \sigma^{(i)} \leftarrow \mathsf{aHMAC.Auth}(\mathsf{sk}, C[i], \textbf{id}[i]),$$

Output $\mathsf{sk}_C = (C, I_1, \mathsf{HSS.evk}_1, \sigma_C, \mathsf{aHMAC.evk})$.

---

[10] The syntax of aHMAC requires an ℓ_d output circuit. We implicitly duplicate the one-bit output of $U_{\mathbf{x}}$ to ℓ_d bits here, and also in the construction of CEval.

$z_1 \leftarrow \mathsf{CEval}(\mathsf{pp}, \mathsf{sk}_C, \mathbf{x})$:

Parse C, I_1, $\mathsf{HSS.evk}_1$, σ_C, and $\mathsf{aHMAC.evk}$ from sk_C.

Derive the one-share $\mathbf{\Delta}_1$ of the extension secret:

$$\mathbf{\Delta}_1 := \sigma_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\mathsf{evk}, U_{\mathbf{x}}, C, \sigma_C).$$

Note that aHMAC correctness ensures $\mathbf{\Delta}_1 = \mathbf{\Delta} \cdot C(\mathbf{x}) + \mathbf{\Delta}_0$ *over* $\mathbb{Z}$*.*

Next run extended HSS evaluation on the one-shares of the input I_1 and the extension secret $\mathbf{\Delta}_1$:

$$z_1 \leftarrow \mathsf{HSS.ExtEval}(1, \mathsf{evk}_1, \mathbf{I}_1, \mathbf{\Delta}_1, \mathsf{F}_{\mathbf{x}}).$$

Note that extended HSS correctness ensures $z_1 = z_0 + C(\mathbf{x}) \cdot \mathsf{F}(\mathbf{s}, \mathbf{x})$*.*

Correctness: As noted in the construction, the evaluation results z_0, z_1 from Eval and CEval satisfy $z_1 = z_0 + C(\mathbf{x}) \cdot \mathsf{F}(\mathbf{s}, \mathbf{x})$, where C is the constraint circuit. When $C(\mathbf{x}) = 0$, we have $z_1 = z_0$ as desired.

Security: We state and prove the following security lemma.

Lemma 6.15. *Construction 30 is a 1-key selectively secure CPRF scheme.*

Proof. We show a series of hybrid experiments that transitions from the experiment $\mathsf{Hyb}_0 = \mathrm{Exp}^{\mathcal{A},0}_{\mathrm{CPRF}}$ to $\mathsf{Hyb}_5 = \mathrm{Exp}^{\mathcal{A},1}_{\mathrm{CPRF}}$ as defined in Definition 6.22.

Hyb_0 : For reference, we summarize the adversary $\mathcal{A}$'s view, w.r.t a challenge circuit C in this experiment:

- In the beginning, $\mathcal{A}$ receives public parameters $\mathsf{pp} = \mathsf{HSS.pk}$ and a constrained key sk_C the boxed terms in the following:

$$\begin{aligned}&(\boxed{\mathsf{HSS.pk}}, \mathsf{HSS.evk}_0, \boxed{\mathsf{HSS.evk}_1}, \mathbf{\Delta}) \leftarrow \mathsf{HSS.Setup}(1^\lambda),\\ &\mathbf{s} \leftarrow \mathcal{I}^{\ell_k}, \quad (I_0^{(i)}, \boxed{I_1^{(i)}}) \leftarrow \mathsf{HSS.Input}(\mathsf{HSS.pk}, \mathbf{s}[i]),\end{aligned} \tag{6.23}$$

$$\begin{aligned}&(\mathsf{aHMAC.sk}, \boxed{\mathsf{aHMAC.evk}}) \leftarrow \mathsf{aHMAC.KeyGen}(1^\lambda, \mathbf{\Delta}),\\ &\boxed{\sigma^{(i)}} \leftarrow \mathsf{aHMAC.Auth}(\mathsf{aHMAC.sk}, C[i], \mathbf{id}[i]),\end{aligned} \tag{6.24}$$

- For any number of (adaptively chosen) queries $\mathbf{x} \in \{0,1\}^\lambda$, $\mathcal{A}$ receives evaluations

$$\begin{aligned} \mathbf{I}_0 &= (\ldots, I_0^{(i)}, \ldots), \\ \boldsymbol{\Delta}_0 &= \mathbf{k}_U \leftarrow \mathsf{aHMAC.EvalKey}(\mathsf{aHMAC.sk}, U_{\mathbf{x}}, \textbf{id}), \\ \boxed{z_0} &\leftarrow \mathsf{HSS.ExtEval}(0, \mathsf{HSS.evk}_0, \mathbf{I}_0, \boldsymbol{\Delta}_0, \mathsf{F}_{\mathbf{x}}). \end{aligned} \tag{6.25}$$

  One of the query, $\mathbf{x}^*$ (with evaluation z_0^*) satisfying $C(\mathbf{x}) \neq 0$ is called the challenge query.

Hyb_1 : Instead of computing the evaluations to queries $\mathbf{x}$, including the challenge query, as in Equation 6.25, Hyb_1 simulate them as

$$\begin{aligned} \mathbf{I}_1 &:= (\ldots, I_1^{(i)}, \ldots), \quad \sigma_C := (\ldots, \sigma^{(i)}, \ldots), \\ \boldsymbol{\Delta}_1 &= \sigma_{\mathbf{w}} \leftarrow \mathsf{aHMAC.EvalTag}(\mathsf{aHMAC.evk}, U_{\mathbf{x}}, C, \sigma_C), \\ z_1 &\leftarrow \mathsf{HSS.ExtEval}(1, \mathsf{HSS.evk}_1, \mathbf{I}_1, \boldsymbol{\Delta}_1, \mathsf{F}_{\mathbf{x}}), \\ \boxed{z_0} &\leftarrow z_1 - C(\mathbf{x}) \cdot \mathsf{F}(\mathbf{s}, \mathbf{x}). \end{aligned}$$

The correctness of our Construction ensures that the above is an equivalent way of computing z_0, except with a negligible error probability. Hence $|\Pr[\mathcal{A}(\mathsf{Hyb}_1) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_0) = 1| \leq \text{negl}(\lambda)$.

Note that in Hyb_1, the evaluations are derived from the aHMAC tags σ_C and $\mathsf{aHMAC.evk}$, without depending on $\mathsf{aHMAC.sk}$ anymore.

Hyb_2 : Instead of computing σ_C and $\mathsf{aHMAC.evk}$ as in Equation 6.24, Hyb_2 simulates them using the simulator $\mathsf{aHMAC.Sim}$

$$(\boxed{\mathsf{evk}, \sigma_C}) \leftarrow \mathsf{aHMAC.Sim}(1^\lambda, 1^{\text{Depth}(U)}).$$

aHMAC security ensures $|\Pr[\mathcal{A}(\mathsf{Hyb}_2) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_1) = 1| \leq \text{negl}(\lambda)$.

Note that in Hyb_2, the share of extension secret $\boldsymbol{\Delta}_1$ is derived from the simulated aHMAC tags σ_C and evk, without depending on the actual secret $\boldsymbol{\Delta}$ anymore.

Hyb_3 : Instead of computing the HSS shares $I_1^{(i)}$ according to the PRF secret key $\mathbf{s}$ as in Equation 6.23, Hyb_3 simulate them as

$$(I_0^{(i)}, \boxed{I_1^{(i)}}) \leftarrow \mathsf{HSS.Input}(\mathsf{HSS.pk}, 0).$$

The security of HSS ensures $|\Pr[\mathcal{A}(\mathsf{Hyb}_3) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_2) = 1| \leq \text{negl}(\lambda)$.

Note that in Hyb_3, the PRF secret key $\mathbf{s}$ is only used for evaluating $F(\mathbf{s}, \mathbf{x})$, and nowhere else.

Hyb_4 : Instead of answering the challenge query $\mathbf{x}^*$, satisfying $C(\mathbf{x}) = 1$, as $z_0^* \leftarrow z_1 - 1 \cdot \mathsf{F}(\mathbf{s}, \mathbf{x})$, Hyb_4 simulates it as

$$z_0^* \leftarrow z_1 - \mathsf{Uniform}(\mathcal{Z}_\lambda) \equiv \mathsf{Uniform}(\mathcal{Z}_\lambda),$$

where $\mathcal{Z}_\lambda$ is the output space of the CPRF and F.

PRF security (of F) ensures $|\Pr[\mathcal{A}(\mathsf{Hyb}_4) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_3) = 1| \leq \text{negl}(\lambda)$.

Hyb_5 : This is the experiment $\text{Exp}_{\text{CPRF}}^{1,\lambda}$. The same arguments from Hyb_1 to Hyb_4, in reverse order, shows $|\Pr[\mathcal{A}(\mathsf{Hyb}_5) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_4) = 1| \leq \text{negl}(\lambda)$.

By a hybrid argument, we conclude that $|\Pr[\mathcal{A}(\mathsf{Hyb}_5) = 1] - \Pr[\mathcal{A}(\mathsf{Hyb}_0) = 1| \leq \text{negl}(\lambda)|$ which proves the lemma. $\square$

	Class	$\|\sigma\|$	Assumption	BBox	Comp.
HSig [GU24, GVW15b]	Cir	poly(λ)	subexp iO + FHE + NIZK or SNARK-NP	✗	✓
HMAC [GW13]	Cir	poly(λ)	FHE	✓	✓
HSig [GVW15b]	Bnd-Dep-Cir	poly(Dep, λ)	SIS	✓	✓
HSig [ACG24]	Bnd-Dep-Cir	poly(Dep, λ)	subexp k-Lin (Pairing) or subexp LWE or subexp DDH	✗	✓
HMAC [CF13]	poly deg polynomials	Deg · poly(λ) / poly(λ)$^\bigstar$	OWF / DDH	✓	✓
aHMAC [This Work]	Cir	poly(λ)	CP-DDH or KDM-DCR	✓	✓
aHMAC [This Work]	Bnd-Dep Cir	Dep · poly(λ)	DDH	✓	✓
Schemes with Limited Composability					
HSig [CFT22]	NC^1	poly(λ)	DHE	✓	✗
HSig [KLVW23]	Cir	poly(λ)	k-Lin (Pairing) or LWE or subexp DDH	✗	✗
HSig [WW24]	bounded size Cir	poly(λ)	bilateral k-Lin (Pairing)	✓	✗

The top part lists schemes that are composable, while the bottom part lists weaker schemes that are not. λ denotes the security parameter, Dep the depth of a circuit, and Deg the degree of a polynomial. DHE stands for the Diffie-Hellman exponent assumption. $\bigstar$ indicates that the computational costs of the DDH-based HMAC scheme of [CF13] scales with the degree of the polynomial evaluated.

Table 6.1: Comparison between homomorphic MAC and signature schemes, in terms of *class of functions supported*, size $|\sigma|$ of the tags/signatures, *assumptions*, *black-box* usage of cryptography, and *composability*.

Chapter 7

A UNIFIED FRAMEWORK FOR SUCCINCT GARBLING FROM HSS

This chapter is adapted from preprint work [ILL25].

7.1 Introduction

Introduced by Yao in the 1980s, a garbling scheme [Yao82, BHR12] allows a "garbler" to transform a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^m$ into a *garbled circuit* $\hat{C}$ along with a pair of short keys $\mathbf{k}_{i,0}, \mathbf{k}_{i,1}$ for each input bit x_i. Given the circuit C, the garbled circuit $\hat{C}$, and an *encoded input* consisting of the input labels $\mathbf{k}_x = \{\mathbf{k}_{i,x_i}\}_{i\in[n]}$ for an unknown input x, an efficient "evaluator" can compute $C(x)$ while learning nothing else about x. An important feature of garbled circuits is that the input keys are short, growing only with the security parameter λ and not with the size of the circuit C.

Garbling schemes serve as fundamental building blocks in cryptography and have found a variety of applications, including constant-round secure computation [Yao82, BMR90, FKN94b, GS18b, BL18, HIKR23], low-complexity cryptography [AIK05], proof systems [GGP10], single-key functional encryption [SS10], offline-online secure computation [CEMY09, AIKW13], and many more. See [App17] for a survey.

A central research direction is to minimize the size of garbled circuits. This is motivated by the fact that in typical applications, the garbler needs to communicate the garbled circuit to the evaluator. Minimizing size has a compounded impact when the same garbled circuit is distributed to multiple parties, translating into proportional bandwidth and storage savings across all recipients. For example, a garbled circuit implementing a complex algorithm can be generated and broadcasted to many receivers during an offline phase, or even stored on a

blockchain, allowing fast online computation once the inputs are known.

Succinct Garbling. In this work, we focus on the task of obtaining *succinct* garbled circuits. Our default notion of succinctness requires that the bit-length of the garbled circuit be smaller than the description size of the original circuit. Concretely, we say that a garbling scheme (for Boolean circuits) is succinct, if for sufficiently large C we have $|\hat{C}| < |C| \log |C|$, where $|C|$ denotes the size of a Boolean circuit C (number of gates, with fan-in 2[1]) and $|\hat{C}|$ denotes the bit length of the string $\hat{C}$;[2] note that describing a general circuit C requires at least $|C| \log |C|$ bits. This is a natural threshold, since it implies that a succinct garbled circuit is less expensive to communicate than the original circuit. Once this minimal threshold of succinctness is crossed, we can aim for a full spectrum of better levels of succinctness. Concrete succinctness goals we consider in this work include garbling with 1-bit-per-gate, or even $O(1/\log \log \lambda)$-bits-per-gate, where λ is the security parameter. (These are all amortized costs, assuming $|C| \gg n, m, \lambda$.) The ultimate goal is achieving *full succinctness*, where the garbled circuit size is independent of the size of the original circuit.

Over the past four decades, tremendous progress has been made on reducing the garbled circuit size, even reaching the ultimate goal of full succinctness. However, significant gaps remain, especially when requiring the constructions to be efficient enough to be implemented.

- *Fast Non-Succinct Garbling Schemes.* Yao's original garbled circuit construction [Yao82] and its optimizations [BMR90, NPS99, KS08a, PSSW09, KMR14, GLNP15, ZRE15, RR21] remain the most practical general-purpose garbling schemes, as they only rely on fast symmetric-key cryptography. The current state-of-the-art, due to Rosulek and Roy [RR21], garbles an AND gate using $1.5\lambda + 5$ bits, while XOR and NOT gates are free, using a random oracle. Note that this does not meet our succinctness criterion,

[1] By default, the number of gates counts all fan-in 2 gates, including XOR, AND, OR. But in fact, our results apply also to circuits with a much richer set of gates that include all possible fan-in-$O(\log \lambda)$ gates; see below.

[2] Note that C and $\hat{C}$ are syntactically different objects. C is a circuit, while $\hat{C}$ is a binary string. Adopting standard notation, $|C|$ denotes the number of *gates* in the original circuit C, while $|\hat{C}|$ denotes the bit-length of the garbled circuit.

since security against poly($|C|$)-time adversaries implies that $\lambda > \log |C|$. Indeed, high communication cost is typically the main practical bottleneck in applications that rely on Yao-style garbling.

- *Fully Succinct Garbling Schemes.* On the other extreme, fully succinct garbled circuits have been shown feasible under standard assumptions. Though theoretically optimal, the concrete garbling size is astronomically large, making these constructions practically infeasible. In addition, existing schemes rely either on indistinguishability obfuscation (iO) [KLW15, BCG+18] or on a non-black-box combination of fully homomorphic encryption (FHE) and attribute-based encryption (ABE) [GKP+13b, BGG+14, HLL23]. These do not only incur a high computational overhead but also rely on a limited class of assumptions, such as circular-secure LWE (e.g., [Gen09, BV11, GSW13]) or combinations of multiple assumptions (e.g., LPN over large fields, local PRG, and DLin over bilinear groups).

As is often the case in cryptography, diversifying assumptions may also lead to efficiency benefits. This motivates the following question:

Can succinct garbling schemes be based on a broader set of assumptions,
with improved efficiency?

Two recent works have made major progress on succinct garbling without the heavy machinery of iO, FHE, or ABE. The work of [LWYY24] presented garbled circuits with 1-bit-per-gate based on variants of RLWE or NTRU, while [ILL24b] achieved *fully succinct* garbling for weak classes of programs, including truth-tables and DFAs, from a variety of group-based assumptions. The latter results build on a fully succinct *partial* garbling scheme for general circuits, applying a computation on a secret input on top of a computation on a public input. Here full succinctness requires the garbled circuit size to be independent of the complexity of the public part of the computation.

7.1.1 Our Results in a Nutshell

Following the recent momentum, we present a unified framework for constructing succinct garbled circuits with 1-bit-per-gate using techniques for *homomorphic secret sharing* (HSS) [BGI16, BGI+18, BKS19, OSY21, RS21, MORS24]. Our unified framework can be instantiated using prime order groups or Paillier groups or lattices, relying on circular-security variants of the power-DDH assumption or a circular power-RLWE assumption. (See discussion of these assumptions below.) We further show how to avoid circular security altogether in a "leveled" variant of our unified framework, where the garbled circuits contain additional components of size $D \cdot \mathrm{poly}(\lambda)$ that depend the circuit depth D but not on the circuit size. Note that the leveled version already gives 1-bit-per-gate garbling for low-depth circuits, including NC^1 or depth-λ circuits, that arise in many applications. We summarize our results on succinct Boolean garbling in the following theorem, and compare it with prior schemes in Table 7.1.

Theorem (Succinct Boolean Garbling, Informal)**.** *Assuming either: (1) circular Power-DDH in Paillier groups (Definition 7.3), or (2) a variant of circular Power-DDH in prime-order groups (Definition 7.7), or (3) circular Power-RLWE (Definition 7.5), there is a garbling scheme for Boolean circuits C with garbled circuit size $|\widehat{C}| = |C| + \mathrm{poly}(\lambda)$.*

Assuming Power-DDH (Definition 7.2) in Paillier or prime-order groups, or Power-RLWE (Definition 7.4), there is a leveled variant with garbled circuit size $|\widehat{C}| = |C| + D \cdot \mathrm{poly}(\lambda)$ for circuits of depth D.

See Theorem 7.2 and Corollary 7.1 for the formal statements on instantiations from Paillier groups and lattices. The instantiation using prime order groups is slightly more complex and proceeds in two steps. In the first step, we obtain a 1-bit-per-gate Boolean garbling with inverse polynomial correctness and privacy errors assuming the circular Power-DDH assumption, as formally stated in Theorem 7.2 and Corollary 7.1. Then in the second step, we make the errors negligible, using correctness and privacy amplification, assuming a variant of the circular Power-DDH assumption (Definition 7.7). Importantly, it turns out that the

amplification does not increase the (amortized) per-gate garbling size, keeping 1-bit-per-gate. See Section 7.5.5.

Extension 1: Beating 1-bit-per-gate. Our framework can be further extended in two interesting ways. First, for *layered* circuits, we can improve the garbling size to $O(1/\log\log\lambda)$-bits-per-gate. This gives the first garbling scheme for a natural and general class of circuits that goes below the bar of 1-bit-per-gate, without relying on iO or FHE plus ABE. Previously, this was only achieved for simple programs such as DFAs [ILL24b]. In fact, this follows as a corollary of a more general construction of a garbling scheme for circuits built from "supergates", including all gates with $O(\log\lambda)$-fan-in, and the garbling size is 1 bit per "supergate."

Extension 2: Succinct arithmetic garbling. We extend our unified framework to garble *arithmetic* circuits, whose gates perform additions and multiplications modulo p or over the integers. Under the above assumptions, we garble arithmetic circuits modulo p with $O(\log p)$-bits-per-gate for general moduli p (for small modulus $p = \text{poly}(\lambda)$, the constant behind the big-O is 1). Note that garbling schemes for mod-p computations automatically imply garbling schemes for bounded integer computations where the wire values are guaranteed to be smaller than p.

This represents significant progress on the front of succinct arithmetic garbling, without relying on iO or FHE plus ABE. As summarized in Table 7.2, the state-of-the-art arithmetic garbling schemes require $\tilde{\Omega}(\log p \cdot \lambda)$-bits-per-gate for $\mathbb{Z}_p$ computation [BLLL23, LL24, Hea24]. We eliminate the λ-multiplicative overhead.

For the simpler task of bounded integer garbling, prior works [BLLL23, MORS24] have shown how to trade the $\Omega(\lambda)$-multiplicative overhead for an additive $\text{poly}(\lambda)$-overhead, achieving $(\log p + \text{poly}(\lambda))$-bits-per-gate, based on DCR. We improve the state-of-the-art by diversifying the assumptions, adding prime-order groups and lattices, and removing the large additive $\text{poly}(\lambda)$ overhead.

Concrete Succinctness. Our 1-bit-per-gate Boolean garbling scheme improves the concrete

	Class	$\|\hat{C}\| + \|\hat{x}\|$	Tool	Assumption	
Yao [Yao82]	general	$O(\lambda) \cdot \|C\|$	Symmetric		OWF
Fully Succinct e.g.,[BGG+14]	general	$\mathrm{poly}(\lambda) \cdot (n+m)$	iO FHE+ABE	Lattice	cir-LWE
Fully Succinct [ILL24b]	weak classes e.g. DFA	$\mathrm{poly}(\lambda) \cdot (n+m)$	HSS	Group	cir-P-DDH
[LWYY24]	general	$\|C\| + \mathrm{poly}(\lambda) \cdot n$	SHE	Lattice	cir-RLWE cir-NTRU
This Work	general	$\|C\| + \mathrm{poly}(\lambda) \cdot n$	HSS	Group Lattice	cir-P-DDH cir-P-RLWE
	layered	$\frac{\|C\|}{\log\log\lambda} + \mathrm{poly}(\lambda) \cdot n$			
	general	$\|C\| + \mathrm{poly}(\lambda) \cdot (n+D)$			P-DDH P-RLWE
	layered	$\frac{\|C\|}{\log\log\lambda} + \mathrm{poly}(\lambda) \cdot (n+D)$			

For assumptions, we list both the mathematical structure and the concrete assumption. λ denotes the security parameter, $|C|$ the number of gates in C, n the input length, m the output length, and D the depth. OWF stands for one-way function and SHE for somewhat homomorphic encryption.

Table 7.1: Comparison between Boolean garbling schemes, in terms of *the class of Boolean circuits handled*, *garbled circuit size*, *the cryptographic tool used*, and *assumptions*.

	Ring	$\lvert\hat{C}\rvert + \lvert\hat{x}\rvert$	Tool	Assumption	
[BLLL23]	$\mathbb{Z}$	$\lvert C\rvert \cdot (O(\ell) + \text{poly}(\lambda))$	LHE	Paillier	DCR
[MORS24]	$\mathbb{Z}$	$\lvert C\rvert \cdot (\ell + \text{poly}(\lambda))$	HSS	Paillier	cir-DCR
[BLLL23]	$\mathbb{Z}_p$	$\lvert C\rvert \cdot \lambda \cdot (O(\log p) + \text{poly}(\lambda))$	LHE	Paillier	strong DCR
[LL24, Hea24]	$\mathbb{Z}_p$	$\lvert C\rvert \cdot \lambda \cdot \widetilde{O}(\log p)$	Symm.		CRH
This Work	$\mathbb{Z}_p$	$\lvert C\rvert \cdot O(\log p) + \text{poly}(\lambda) \cdot n$ ★	HSS	Paillier Prime Group Lattice	cir-P-DDH cir-P-RLWE
		$\lvert C\rvert \cdot O(\log p) + \text{poly}(\lambda) \cdot (n + D)$ ★			P-DDH P-RLWE

For integer computations, the wire values must be smaller than an a priori upper bound 2^ℓ. For assumptions, we list both the mathematical object it relies on and the concrete assumption. λ denotes the security parameter, $\lvert C\rvert$ the number of gates in C, n the input length, and D the depth. CRH stands for Correlation Robust Hash, LHE for linearly homomorphic encryption. Strong DCR refers to DCR where the secret exponent of the hard subgroup is chosen to be a random λ-bit number, instead of $O(\log N)$-bit number. ★ indicates that when the prime is $O(\log(\lambda))$-bit long, the size of garbling is $\lvert C\rvert \cdot \log p + \text{poly}(\lambda) \cdot n$ and $\lvert C\rvert \cdot \log p + \text{poly}(\lambda) \cdot (n + D)$ respectively, eliminating the hidden constant factor multiplied with $\log p$.

Table 7.2: Comparison between *arithmetic* garbling schemes, in terms of *ring supported*, *garbled circuit size*, *the cryptographic tool used*, and *assumptions*.

garbling size even with just a moderately large number of gates. Recall that asymptotically the garbling size is $|\hat{C}| = |C| + \text{poly}(\lambda)$. Here the $\text{poly}(\lambda)$ additive term represents the size of some global public data pd. The concrete size of this global data determines when using our schemes yields smaller garbled circuits compared with Yao-style garbled circuits. The break-even point depends on the instantiation, as well as the choice of a PRG seed length. For our estimation below and in Section 7.7, we optimistically assume an "HSS-friendly" PRG with 128-bit seed and output length $\approx |C|$. Designing MPC/FHE/HSS-friendly PRGs is an active research direction; see, e.g., [ARS+15, GRR+16, BCG+17, CCKK21, ABG+24, FLLL24, CCH+24] and references therein. For such PRGs, there is typically a tradeoff between computational cost and seed length; the size of the global data pd in our constructions scales linearly with the seed length. On the other hand, the number of restricted multiplications (between an intermediate value and an input bit) needed for evaluating each output bit of the PRG, which is upper bounded by the branching program size, directly influences the computational cost. While research on HSS-friendly PRGs is still in its infancy, there is a large space of possible designs to explore. We hope that the goal of practical succinct garbling will further motivate research on the concrete efficiency of HSS-friendly PRGs.

Table 7.3 in Section 7.7 summarizes the concrete sizes of the global data. The Paillier instantiation has just 0.38MB global data, the simple instantiation using Prime-order groups *with inverse polynomial errors*[3] has 5.1MB global data which can be optimized down to just 0.13MB, and our lattice instantiation has 71MB global data. Comparing with using optimized Yao's garbled circuits with estimated size[4] of $\lambda|C|$, our 1-bit-per-gate garbling is smaller when the original circuit satisfies $|C| > |\mathsf{pd}|/(\lambda - 1)$. Concretely, the break-even point is $|C| = 24K$ for Paillier instantiation, $8K$ for optimized prime order groups, and $4.5M$

[3] As mentioned above, these errors can be made negligible via correctness and security amplification. The amplification step increases the size of the global data and computational efficiency by a factor of $\omega(1)$ factor asymptotically, but does not increase the per-gate communication cost. For concrete efficiency, we consider the simpler instantiation without amplification.

[4] The state-of-the-art optimization over Yao's garbled circuit is by [RR21], which contains 1.5λ bits per AND gate, and garbling XOR is free. We use $\lambda|C|$ as a rough estimation of the garbled circuit size.

for lattices. For moderately large circuits, e.g., of size 10^7 gates, our optimized prime order group instantiation is smaller than Yao-style garbling by a factor of 116 (from 160MB to 1.38MB), while our Paillier group instantiation is smaller by a factor of 98 with size 1.63MB. For reference, the plain circuit description size is at least 29MB.

Compared with the recent work of Liu et al. [LWYY24] that constructed 1-bit-per-gate garbled circuits based on circular RLWE, the public data in the latter has a much larger size of $10GB$. This results from the use of the Gentry-Sahai-Water fully homomorphic encryption scheme [GSW13], which has much larger ciphertexts than the HSS encodings used in this work.

Finally, we compare with fully succinct garbling schemes. The iO-based constructions are not currently implementable, while the FHE+ABE-based constructions [GKP+13b, BGG+14, HLL23] have astronomically large input labels and/or computational costs, and hence are also impractical. The label size for each input bit of the RLWE instantiation of the succinct garbling scheme of [GKP+13b, BGG+14] is $\Omega(n^2 \log q^4)$, where n and q are RLWE degree and modulus satisfying $n^{1-\epsilon} > \log q > D$ for some $\epsilon \in (0,1)$ and D is the circuit depth. This means each input label has size $\omega(D^6)$, which is prohibitive even for small depth such as 100. The recent work [HLL23] removes the constraint of $\log q > D$, allowing for smaller modulus and degree. However, this requires performing "boostrapping" inside ABE, which is very computationally expensive.

In summary, for circuits of moderate size around 10^5 to 10^6 gates, the garbled circuits of our schemes are concretely smaller than all prior constructions.

Towards Practical Succinct Garbling. Concretely, our garbling schemes require evaluating a PRG using HSS, in addition to a few other HSS operations per gate. Assuming each output bit of the PRG can be evaluated using a restricted multiplication straightline (RMS) program of size S, or alternatively a branching program of size S, then garbling and evaluating a general Boolean circuit require $|C| \cdot (4S + O(1))$ homomorphic RMS operations. In particular, since the HSS restricted multiplication operation is much more expensive than the HSS addition operation, if the PRG requires $S_\times$ restricted multiplications, the per-gate cost of garbling is

dominated by $4S_\times + O(1)$ homomorphic restricted multiplication. As discussed above, we optimistically conjecture an HSS-friendly PRG with a large stretch (as our garbling size scales linearly with the seed length), and where each output bit can be evaluated using a reasonably small number of restricted multiplication operations. Then in the lattice instantiation, the per-gate computation boils down to computing a small number of multiplication/addition of $\mathcal{R}_q$ elements and rounding.

Our Assumptions. The leveled version of our unified framework can be based on natural flavors of the Power Decisional Diffie Hellman (P-DDH) assumption (Definition 7.2), introduced in [GJM03, CNs07, AHI11] and further used in [GHKW17, KY18, AMN⁺18, BMZ19, ILL24b], in Paillier or prime-order groups. P-DDH postulates that for appropriately sampled group element g and exponents s and a, b sampled randomly from a range $[\ell]$, the triple (g, g^s, g^{s^2}) is indistinguishable from (g, g^a, g^b).

To remove the $D \cdot \mathrm{poly}(\lambda)$ additive term in the size of "leveled" garbled circuits, we need the following circular-security variant of this assumptions. The Circular Power Decisional Diffie Hellman (CP-DDH) assumption (Definition 7.3) asserts that a circular encryption of bits of the secret key s using powers of the secret key is pseudorandom. More precisely, for appropriately sampled group elements g, f and random exponents s and $\{a_i, b_i, c_i\}_i$, the following computational indistinguishability holds:

$$\textbf{CP-DDH: } g, g^s, g^{s^2}, (g^{a_i}, g^{sa_i}, g^{s^2 a_i} \cdot f^{s[i]})_{i \in [\log s]} \quad \approx_c \quad g, g^s, g^d, (g^{a_i}, g^{b_i}, g^{c_i})_{i \in [\log s]}.$$ [5]

The P-DDH and CP-DDH assumptions can be postulated over Paillier or prime order groups. For the Paillier group, the assumption can be further simplified (still sufficient for succinct garbling) to $(g^r, g^{rs}, g^{rs^2}(1+N)^s)$ being pseudorandom, where g is a generator of the hard subgroup and the exponents r, s are randomly sampled. This optimization is introduced for concrete efficiency; see Section 7.7. For prime-order groups, Power-DDH and Circular Power-DDH hold in the standard generic group model (GGM) [Sho97], as shown in [ILL24b].

[5]Compared to [ILL24b], our formulation here includes g^{s^2} in the indistinguishability, which we believe is more natural and easier to use. See also the remark under Definition 7.3.

In particular, our succinct garbling scheme can be instantiated in the (prime-order) GGM, under the mild assumption of a PRF in NC^1. Furthermore, under the CP-DDH assumption in prime order groups, we only obtain succinct garbling with inverse polynomial errors. As mentioned above, we can amplify correctness and privacy to make the errors negligible without hurting the amortized per-gate garbling size. This requires a variant of the CP-DDH assumption, which instead of hiding the bits of the secret s, hides bits of the secret shifted by a public constant s', $t = s + s'$.

$$\textbf{CP-DDH*:}\ g, g^s, g^{s^2}, s', (g^{a_i}, g^{sa_i}, g^{s^2 a_i} \cdot f^{t[i]})_{i\in[\log s]} \quad \approx_c \quad g, g^s, g^d, (g^{a_i}, g^{b_i}, g^{c_i})_{i\in[\log s]}\ , \text{ where } t = s + s'$$

Alternatively, our garbling schemes can be based on the Power-RLWE assumption (Definition 7.4) for the leveled version and circular Power-RLWE assumption (Definition 7.5) for the full-fledged version. Introduced in [ARS24], Power-RLWE postulates that RLWE samples with *small* secrets s and s^2, and the same public vector $\mathbf{a}$ in a polynomial ring $\mathcal{R}_q$, $(\mathbf{a}, s\mathbf{a} + \mathbf{e}_1, s^2\mathbf{a} + \mathbf{e}_2)$ is pseudorandom. The circular variant further uses the last sample to hide the secret s, assuming the pseudorandomness of $(\mathbf{a}, s\mathbf{a} + \mathbf{e}_1, s^2\mathbf{a} + \mathbf{e}_2 + s\Delta)$, where Δ is a constant.

7.1.2 Related Works

In this section we provide a detailed comparison between our results and prior or concurrent related works.

Comparison with [ILL24b]. The work of [ILL24b] constructed fully succinct garbling schemes for weak classes of programs, including truth tables, DFA, and decision trees, based on different group-based assumptions. This builds on a fully succinct *partial garbling schemes* (equivalently, conditional disclosure of secrets), where most of the input is public. cIn comparison, our garbling schemes achieve a weaker level of succinctness, but apply to all circuits while fully hiding the input. Our work provides a lattice-based instantiation of

the succinct partial garbling scheme from [ILL24b] and the underling homomorphic MAC primitive.

Comparison with [LWYY24]. Another recent work [LWYY24] constructed 1-bit-per-gate garbled circuits using special somewhat homomorphic encryption schemes, namely the GSW scheme instantiated using circular variants of RLWE or NTRU. In comparison, our unified framework presents a more general design principle using HSS. It yields instantiations based on more diverse assumptions that include different group-based assumptions. Our garbled circuits have smaller concrete sizes as discussed in the introduction (also see Table 7.3), owing to the fact that HSS encoding is smaller than GSW ciphertexts. In addition, we show how to go below 1-bit-per-gate for layered circuits as well as an extension to arithmetic garbling.

Comparison with [MORS25]. The concurrent and independent work of [MORS25] achieves a similar set of results to this work based on similar techniques. We note the following differences. For Boolean garbling, both [MORS25] and this work achieve (amortized) 1 bit per gate for general circuits, and $O(1/\log\log\lambda)$ bits per gate for layered circuits under circular assumptions. Both works have leveled variants that avoid circular assumptions at the price of an additive $\mathrm{Depth}(C)\cdot\mathrm{poly}(\lambda)$ size overhead. The differences are in the underlying assumptions: the work of [MORS25] focuses on constructions in Paillier groups based on a circular DCR assumption (resp., standard DCR for the leveled variant), while our work presents a unified framework with instantiations in Paillier groups, prime-order groups, or lattices, based on the CP-DDH or CP-RLWE assumptions (resp. P-DDH or P-RLWE for the leveled variants).

The difference in assumptions stems from the fact that [MORS25] use a more sophisticated variant of the basic technique to base their construction (in the leveled case) on the standard DCR assumption. While DCR is more widely used than P-DDH in Paillier groups used in our work, these assumptions seem technically incomparable. We believe that adapting the technique from [MORS25] to our constructions will give leveled variants under Paillier groups, prime-order groups, or lattices based on the standard DDH or RLWE with small

secret assumptions. However, this seems to come at the price of a higher concrete overhead. The current work initiates a study of the concrete efficiency of group-based and lattice-based garbling, including an effort to optimize the additive terms.

For arithmetic garbling, the work of [MORS25] constructs a scheme over *bounded integers* by 2^ℓ, with (amortized) $(\ell + \lambda)$ bits per gate for general circuits, and $O((\ell + \lambda)/\log\log\lambda)$ bits per gate for layered circuits. In contrast, our work constructs schemes for computation over $\mathbb{Z}_R$ computation for *any modulus* R of ℓ bits, with (amortized) $O(\ell)$ bits per gate for general circuits. We believe that we can also obtain additional savings in cost for layered arithmetic circuits. Besides the distinction between supporting bounded integers vs. $\mathbb{Z}_R$ computation, the above differences in assumptions also hold for the arithmetic garbling results.

Comparison with [CHHK25]. The concurrent and independent work of [CHHK25] constructed Boolean garbling schemes with amortized per-gate garbling size below λ. Their first scheme is proven in the Generic Group model (GGM), achieving $\lambda/\sqrt{\log\lambda}$-bit-per-gate garbling size. Their second scheme is proven in the plain model under the Power-DDH assumption together with the existence of a tweakable correlation robust hash, attaining a garbled circuit size of $\lambda \cdot |C|/\sqrt{\log\lambda} + \text{poly}(\lambda) \cdot D$, where D is the depth of the circuit for *layered* circuits. In comparison, our Boolean garbling schemes achieve 1-bit-per-gate for general circuits, and $O(1/\log\log\lambda)$-bit-per-gate for layered circuits, again, removing the $\tilde{\Omega}(\lambda)$ multiplicative overhead. On the other hand, our schemes make a non-black-box use of a PRF (or high-stretch PRG), whereas their constructions can be cast unconditionally in the GGM.

Other Use of HSS in Garbling by [GN25]. The recent work of [GN25] constructed a garbling scheme that supports mixed circuits with both Boolean and bounded integer arithmetics using HSS techniques. The core innovation is using HSS techniques to implement an efficient garbling gadget for bit-decomposition that is compatible with the state-of-art arithmetic garbling scheme of [MORS24]. Overall, their scheme has a garbling size of (amortized) $O(\lambda)$ bits per Boolean gate, $(\ell + \lambda_{\mathsf{DCR}})$ bits per arithmetic gate (over integers bounded by 2^ℓ), and $O(\ell \cdot \lambda_{\mathsf{DCR}}/\log(\lambda))$ bits per bit-decomposition gate. In comparison,

we apply HSS techniques to construct significantly more succinct Boolean and arithmetic garbling with (amortized) 1 bit per Boolean gate, and $O(\ell)$ bits per arithmetic gate (over an ℓ-bit modulus).

Updated Version of [ILL24b]. As explained in the technical overview below, our results use the aHMAC constructions from [ILL24b] as one of the main building blocks. The updated version [ILL24a] presents new constructions of leveled aHMAC that improve the assumptions from P-DDH (in Paillier and prime-order groups) or P-RLWE respectively to standard DDH or RLWE. Applying the new constructions of leveled aHMAC, we can obtain all of our leveled garbled circuit results from DDH (in Paillier and prime-order groups) or RLWE.

Inspiration from Arithmetic Garbling. Our work builds upon a recent line of research [BLLL23, LL24, Hea24, MORS24] for improving the garbling size of *arithmetic* circuits. These circuits consist of addition and multiplication gates, evaluated over a ring, typically $\mathbb{Z}_p$ or $\mathbb{Z}$, and an input x consists of ring elements. Because there is a simple baseline solution that uses a Boolean garbling scheme to garble a Boolean circuit implementing the arithmetic circuit of interest, research naturally focuses on what can be done differently. The first work on arithmetic garbling by Applebaum et al. [AIK11] proposed an arithmetic generalization of input keys and labels – the keys of an input wire describes an affine functions $\mathbf{K}_i$ and the label for x_i is the output $\mathbf{K}_i(x_i)$. They then constructed an garbling scheme for bounded integer computation with such arithmetic input labels, based on LWE, which sends $\ell \cdot \text{poly}(\lambda)$ bits per gate when the wire values are bounded by 2^ℓ. Building upon [AIK11] and a subsequent work by Ball et al. [BMR16], recent works [BLLL23, LL24, Hea24, MORS24] have renewed research on arithmetic garbling on several different fronts: 1) diversifying assumptions, 2) supporting more models of computing, such as, $\mathbb{Z}_p$ computation, and mixed circuits with both arithmetic and Boolean gates, and 3) optimizing succinctness.

We focus on the succinctness aspect. The baseline solution using Yao's garbled circuits requires $\Omega(\lambda\ell\log\ell)$-bits-per-gate. Interestingly, the work of Ball et. al. [BLLL23] showed that bounded integer computations can be garbled with $O(\ell + \text{poly}(\lambda))$-bits-per-gates, trading the

$O(\lambda \log \ell)$ multiplicative factor for an additive poly(λ) term, assuming the DCR assumption over Paillier/Damgård-Jurik groups. Their technique relies on simple additive homomorphism supported by DCR, rather than iO or FHE plus ABE underlying fully succinct garbling. The work of [MORS24] further improved size to exactly $\ell +$ poly(λ)-bits-per-gate, by applying HSS techniques, assuming the circular security of Damgård-Jurik encryption.

These works shed new light on how to avoid the $O(\lambda)$-multiplicative factor overhead associated with Yao's garbled circuits, using lightweight tools. But their techniques are limited in two ways. First, the additive poly(λ) is large, proportional to $\log N$ where N is the Paillier modulus, and dominates when wire values are relatively small $\ell = o(\log N)$. In particular, when used to garble Boolean computation $\ell = 1$, the size is $O(\log N)$-bits-per-gate, worse than Yao's garbled circuits. Second, their methods do not extend to garbling $\mathbb{Z}_p$-arithmetic circuits. Despite past efforts [AIK11, BMR16, BLLL23, LL24, Hea24], the most succinct garbled $\mathbb{Z}_p$-circuits have size $\Omega(\lambda \log p)$-bit-per-gate, carrying the $\Omega(\lambda)$-multiplicative factor overhead.

As discussed before, the current work overcomes the above two limitations. Our unified framework also gives a $\mathbb{Z}_p$-garbling scheme with $O(\log p)$-bits-per-gate for general p, based on various group and lattice assumptions. Our technique is inspired by techniques developed in the context of arithmetic garbling, particularly the HSS-based technique from [MORS24].

7.2 Technical Overview

Starting Point: Succinct Garbling of [ILL24b]. Our starting point is the recent new approach to succinct garbling from [ILL24b], which combines a new primitive called fully succinct *partial* garbling and fully homomorphic encryption (FHE) to obtain fully succinct standard garbling. This follows the FHE+ABE blueprint of [GKP+13b, BGG+14], replacing succinct ABE by succinct partial garbling.

In more detail, a partial garbling scheme generalizes standard garbling to consider computations with public and private parts, $C(\mathbf{x}, \mathbf{y}) = C_{\text{Priv}}(\mathbf{y}, C_{\text{Pub}}(\mathbf{x}))$. A partial garbling of C computes a garbling $\widehat{C}$ and a pair of short keys $\mathbf{k}_{x,i,0}, \mathbf{k}_{x,i,0}$ for every bit in $\mathbf{x}$, as well as

$\mathbf{k}_{y,i,0}, \mathbf{k}_{y,i,1}$ for every bit in $\mathbf{y}$. The garbling $\widehat{C}$, the keys $\{\mathbf{k}_{x,i}\}$, $\{\mathbf{k}_{y,i}\}$ selected corresponding to inputs $\mathbf{x}, \mathbf{y}$, together with $\mathbf{x}$ in the clear reveals the computation result $\mathbf{z} = C(\mathbf{x}, \mathbf{y})$, and nothing else about the private input $\mathbf{y}$. The scheme of [ILL24b] achieves a fully succinct garbling size $|\widehat{C}| \leq |C_{\text{Priv}}| \cdot \text{poly}(\lambda)$, independent of the complexity of C_{Pub}.

The observation from [ILL24b] then is to apply partial garbling to the computation

$$C(\mathsf{ct}_{\mathbf{x}}, \mathsf{sk}) = \mathsf{Dec}(\mathsf{sk}, \mathsf{HEval}^f(\mathsf{ct}_{\mathbf{x}})),$$

i.e. with a public part $C_{\text{Pub}}(\mathsf{ct}_{\mathbf{x}}) = \mathsf{HEval}^f(\mathsf{ct}_{\mathbf{x}}) = \mathsf{ct}^*_{\mathbf{z}}$ computing homomorphic evaluation of some function f over FHE ciphertexts $\mathsf{ct}_{\mathbf{x}}$, and a private part $C_{\text{Priv}}(\mathsf{sk}, \mathsf{ct}^*_{\mathbf{z}})$ decrypting evaluated ciphertexts using the secret key sk as the private input. A partial garbling of C reveals the evaluation result $\mathbf{z} = f(\mathbf{x})$, and guarantees privacy of the secret key sk, which further guarantees privacy of $\mathbf{x}$ by FHE security. Therefore, a partial garbling of C can be viewed as a standard garbling of the function f. Furthermore, the size of $\widehat{C}$ only depends on the complexity of private computation, i.e. FHE decryption, $|\widehat{C}| \leq |\mathbf{z}| \cdot |\mathsf{Dec}| \cdot \text{poly}(\lambda) = |\mathbf{z}| \cdot \text{poly}(\lambda)$, and does not depend on the complexity of f. Hence the partial garbling of C is a fully succinct standard garbling of f.

While conceptually simple, this solution is far from practically useful due to the heavy computation complexity of FHE. A natural attempt is to use a less powerful, but much lighter-weight, homomorphic encryption (HE) scheme to obtain fully succinct garbling for many low-depth computations $\{f^i\}$, and composing them into a high-depth one: $f := f^T \circ f^{T-1} \circ \ldots \circ f_1$. However, some calculation shows a difficulty. Suppose each f^i is a Boolean circuit of depth D with width W. The succinct garbling of f^i costs $|\widehat{f^i}| = W \cdot \text{poly}(\lambda)$, while our target size is $\leq WD \cdot \log(WD)$ to achieve succinctness. Without new ideas, we would need a powerful HE supporting $D = \text{poly}(\lambda)$ depth computation to achieve succinctness.

Indeed, our new ideas require looking into the construction of [ILL24b], and finding new ways to garble the private computation C_{Priv} more efficiently, at the cost of supporting only a restricted form of computation.

The Construction of [ILL24b] in More Detail. The partial garbling construction

of [ILL24b] relies on a new primitive, algebraic homomorphic MAC (aHMAC), and the standard Yao's Boolean garbling to handle the public and private computations respectively. We give a simplified review here, assuming the the free-XOR [KS08a] key format in Yao's garbling.

The aHMAC Scheme. An aHMAC scheme is run between an authenticator and an evaluator. They both hold an evaluation key evk. The authenticator additionally holds a PRF key k, and a global secret $s \in \mathbb{Z}$.

- The authenticator when given a bounded integer $x_i \in [B]$ as input, and an associated id, computes its tag as $\sigma_{x_i} := s \cdot x_i + k_x^{(i)}$ over $\mathbb{Z}$, where $k_x^{(i)} = \mathsf{PRF}(\mathsf{k}, \mathsf{id})$ is derived from the id.
- The evaluator when given inputs $\mathbf{x}$ and tags $\sigma_{\mathbf{x}} = \{\sigma_{x_i}\}$ can evaluate any arithmetic circuit C (with bounded intermediate values by B) using the evaluation key: $\sigma_{\mathbf{z}} \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \sigma_{\mathbf{x}}, \mathbf{x})$.
- The authenticator when given only the ids, hence the derived keys $\mathbf{k}_x = \{k_x^{(i)}\}$, can evaluate the same circuit: $\mathbf{k}_z \leftarrow \mathsf{EvalKey}(\mathsf{evk}, C, \mathbf{k}_x)$.

The scheme guarantees the evaluated tags and keys are consistent: $\sigma_{\mathbf{z}} = s \cdot \mathbf{z} + \mathbf{k}_z$ over $\mathbb{Z}$, and also that the evaluation key evk and tags $\sigma_{\mathbf{x}}$ don't leak anything about the global secret s.

In this work, we view a pair of tag and key σ_{x_i}, $k_x^{(i)}$ as an additive share of sx_i over $\mathbb{Z}$, written as $\langle sx_i \rangle_0 = k_x^{(i)}$, and $\langle sx_i \rangle_1 = \sigma_{x_i}$. We view the algorithms $\mathsf{EvalKey}, \mathsf{EvalTag}$ as homomorphically evaluating additive shares of $s\mathbf{x}$ between a garbler P_G and an evaluator P_E, who both hold an evaluation key evk with respect to the global secret s. Note that the $\mathsf{EvalTag}$ algorithm by the evaluator also needs $\mathbf{x}$ in the clear.

$$\begin{array}{ll} \underline{P_G(\mathsf{evk})} & \underline{P_E(\mathsf{evk}, \mathbf{x})} \\ \langle s\mathbf{z} \rangle_0 \leftarrow \mathsf{EvalKey}(\mathsf{evk}, C, \langle s\mathbf{x} \rangle_0), & \langle s\mathbf{z} \rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C, \langle s\mathbf{x} \rangle_1, \mathbf{x}). \end{array}$$

The construction of [ILL24b] guarantees that given any additive shares of $s\mathbf{x}$, as long as all intermediate values of $C(\mathbf{x})$ are bounded by B, the results of $\mathsf{EvalKey}, \mathsf{EvalTag}$ also form additive shares of $s\mathbf{z}$, where $\mathbf{z} = C(\mathbf{x})$.

Yao's Garbling. In Yao's garbling of a Boolean circuit C (assuming the free-XOR [KS08a] key format), the garbler P_G samples a random key k_j for every wire j in C, and a global secret s. We view the keys $\{k_j\}$ and the global secret s as $O(\lambda)$-bit integers in this overview.

P_G provides a garbled table for each gate to the evaluator P_E, such that if P_E obtains a set of labels $\{l_i = s \cdot x_i + k_i\}$ according to an input $\mathbf{x} = \{x_i\}$, then she can use the garbled tables to recover a label $l_j = s \cdot v_j + k_j$ for every wire j corresponding to the correct wire value v_j. In order for P_E to recover the values z_o on the output wires o in C, a usual trick is to assume the least significant bit (LSB) of s is 1, so that $\mathrm{LSB}(l_o) = z_o \oplus \mathrm{LSB}(k_o)$. It suffices for P_G to send P_E $\mathrm{LSB}(k_o)$ for every output wire o.

We take an alternative view of Yao's garbling not as a static scheme, but as a protocol between a garbler P_G and an evaluator P_E.

- Initially P_G and P_E jointly hold additive shares of $s\mathbf{x}$ for some input $\mathbf{x} = \{x_i\}$: the garbler holds $\langle sx_i \rangle_0 = k_i$, and the evaluator holds $\langle sx_i \rangle_1 = l_i$.
- Then P_G sends garbled tables to P_E so that they jointly hold additive shares of sv_j for every wire value v_j in C.
- In the end, P_G and P_E jointly hold additive shares of $s\mathbf{z}$ for the output $\mathbf{z} = \{z_o\}$: the garbler holds $\langle sz_o \rangle_0 = k_o$, and the evaluator holds $\langle sz_o \rangle_1 = l_o$. P_G then sends $\{\mathrm{LSB}(k_o)\}$ to P_E to reveal $\mathbf{z}$.

The security of Yao's garbling guarantees that if the global secret s is not leaked by the initial additive shares $\langle s\mathbf{x} \rangle_1$ to P_E, then all communication from P_G to P_E can be simulated by P_E, given only the output $\mathbf{z}$. To summarize the protocol between P_G and P_E, we write

$$(P_G : \langle s\mathbf{z} \rangle_0), (P_E : \langle s\mathbf{z} \rangle_1, \mathbf{z}) \leftarrow \mathsf{Yao}^C\big((P_G : \langle s\mathbf{x} \rangle_0), (P_E : \langle s\mathbf{x} \rangle_1)\big).$$

Succinct Partial Garbling from aHMAC and Yao. We again describe the partial garbling scheme for evaluating $C(\mathbf{x}, \mathbf{y}) = C_{\mathrm{Priv}}(\mathbf{y}, C_{\mathrm{Pub}}(\mathbf{x}))$ as a protocol between the garbler P_G and the evaluator P_E, which we believe is more intuitive. (See Section 7.5 for viewing garbling as a 2PC protocol.) It represents a valid garbling scheme as long as P_G's communication is independent of the inputs $\mathbf{x}, \mathbf{y}$ except in an initialization phase.

1. In the initialization phase, P_G sets up the aHMAC scheme with a global secret s, and evaluation key evk. He then samples random additive shares $\langle s\mathbf{x}\rangle_0$, $\langle s\mathbf{x}\rangle_1$, for the public input $\mathbf{x}$, and $\langle s\mathbf{y}\rangle_0$, $\langle s\mathbf{y}\rangle_1$ for the private input $\mathbf{y}$. In the end, P_G sends evk, $\mathbf{x}$, $\langle s\mathbf{x}\rangle_1$ and $\langle s\mathbf{y}\rangle_1$ to P_E.

2. To evaluate the public computation C_{Pub}, [6] P_G and P_E locally run $\mathsf{EvalKey}$ and $\mathsf{EvalTag}$ respectively on their shares $\langle s\mathbf{x}\rangle_0$ and $\langle s\mathbf{x}\rangle_1$.

$$\begin{aligned} P_G: &\quad \langle s\mathbf{w}\rangle_0 \leftarrow \mathsf{EvalKey}(\mathsf{evk}, C_{\text{Pub}}, \langle s\mathbf{x}\rangle_0), \\ P_E: &\quad \langle s\mathbf{w}\rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{evk}, C_{\text{Pub}}, \langle s\mathbf{x}\rangle_1, \mathbf{x}). \end{aligned} \tag{7.1}$$

3. To evaluate the private computation C_{Priv}, P_G and P_E jointly run Yao's garbling.

$$\begin{aligned} &(P_G : \langle s\mathbf{z}\rangle_0), (P_E : \langle s\mathbf{z}\rangle_1, \mathbf{z}) \\ &\quad \leftarrow \mathsf{Yao}^{C_{\text{Priv}}}\big((P_G : \langle s\mathbf{y}\rangle_0, \langle s\mathbf{w}\rangle_0), (P_E : \langle s\mathbf{y}\rangle_1, \langle s\mathbf{w}\rangle_1)\big). \end{aligned} \tag{7.2}$$

   The evaluator P_E outputs $\mathbf{z}$ in the end.

In the above protocol, communication from P_G to P_E after the initialization phase corresponds to the garbling material $\widehat{C}$ in the garbling scheme. We note since the public computation C_{Pub} is evaluated by local procedures, with no communication, we indeed obtain a fully succinct partial garbling scheme.

Recall that in this work, we intend to perform homomorphic evaluation of some low-depth computation f^i over HE ciphertexts $\mathsf{ct}_{\mathbf{x}^i}$ using the public computation, and then HE decryption using the private computation. Furthermore, in order to compose multiple such evaluations, $\ldots f^{i+1} \circ f^i \circ \ldots$, we need to also implement HE re-encryption using the private computation. We illustrate the modified steps 2 and 3 below.

2' To evaluate the public computation $C_{\text{Pub}} := \mathsf{HEval}^{f^i}$, P_G and P_E locally run $\mathsf{EvalKey}$

[6] A Boolean circuit C_{Pub} can be implemented by an arithmetic circuit over integers bounded by 2.

and $\mathsf{EvalTag}$ respectively on their shares $\langle s \cdot \mathsf{ct}_{\mathbf{x}^i} \rangle_0$, $\langle s \cdot \mathsf{ct}_{\mathbf{x}^i} \rangle_1$. [7]

$$P_G: \quad \langle s \cdot \mathsf{ct}^*_{\mathbf{x}^{i+1}} \rangle_0 \leftarrow \mathsf{EvalKey}(\mathsf{evk}, \mathsf{HEval}^{f^i}, \langle s \cdot \mathsf{ct}_{\mathbf{x}^i} \rangle_0),$$
$$P_E: \quad \langle s \cdot \mathsf{ct}^*_{\mathbf{x}^{i+1}} \rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{evk}, \mathsf{HEval}^{f^i}, \langle s \cdot \mathsf{ct}_{\mathbf{x}^i} \rangle_1, \mathsf{ct}_{\mathbf{x}^i}),$$

where ct^* denotes homomorphically evaluated ciphertexts.

3' To evaluate the private computation $C_{\text{Priv}} := \mathsf{Enc} \circ \mathsf{Dec}$, P_G and P_E jointly run Yao's garbling.

$$\begin{aligned}&(P_G : \langle s \cdot \mathsf{ct}_{\mathbf{x}^{i+1}} \rangle_0), (P_E : \langle s \cdot \mathsf{ct}_{\mathbf{x}^{i+1}} \rangle_1, \mathsf{ct}_{\mathbf{x}^{i+1}}) \\ &\quad \leftarrow \mathsf{Yao}^{\mathsf{Enc}\circ\mathsf{Dec}}\big((P_G : \langle s \cdot \mathsf{sk} \rangle_0, \langle s \cdot \mathsf{ct}^*_{\mathbf{x}^{i+1}} \rangle_0), (P_E : \langle s \cdot \mathsf{sk} \rangle_1, \langle s \cdot \mathsf{ct}^*_{\mathbf{x}^{i+1}} \rangle_1)\big).\end{aligned}$$

Note that the results are shares of fresh HE ciphertexts $\mathsf{ct}_{\mathbf{x}^{i+1}}$, so the parties can then repeat Step 2' and 3' for the next evaluation of f^{i+1}.

As explained, the communication by Yao's garbling to implement $\mathsf{Enc} \circ \mathsf{Dec}$ is too much for our purpose. Instead, our idea is to use homomorphic secret sharing (HSS) to replace Yao's garbling in the above protocol.

It may first seem a bit odd to consider HSS as a replacement for garbling. Indeed, in the setting of HSS, both parties *depend* on the input, while in the setting of garbling, P_G needs to be independent of the input. Our observation is that in the private computation implemented by Yao, $\mathsf{Enc} \circ \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\mathbf{x}})$, the most complicated computations, e.g. evaluating a PRG, involve only the secret key sk, which is indeed independent of the actual input $\mathbf{x}$! One can therefore hope to rely on HSS for the complicated computations involving only sk, and in the end incorporate $\mathsf{ct}_{\mathbf{x}}$ in the remaining simpler steps.

Replacing Yao with HSS. An HSS scheme runs between two parties P_0, P_1. In common constructions, such as [ADOS22, BGI16, BKS19], they both hold encryptions of an input $\mathbf{y}$, denoted $I_{\mathbf{y}}$, and jointly an additive share of a global secret s over $\mathbb{Z}$ consistent with

[7] Technically, we mean shares of $s \cdot \mathsf{Bits}(\mathsf{ct}_{\mathbf{x}^i})$ here, and shares of $s \cdot \mathsf{Bits}(\mathsf{sk})$ in step 3'. But we choose to abuse notations to avoid cluttering.

the encryptions. Each party P_b can locally evaluate any NC1 Boolean circuits C over the encrypted inputs via $\mathsf{HSS.Eval}_b$ such that the two outputs form additive shares of $s \cdot \mathbf{z}$ and $\mathbf{z}$, where $\mathbf{z} = C(\mathbf{y})$ is the evaluation result.

$$\begin{array}{ll} \underline{P_0(I_\mathbf{y}, \langle s \rangle_0)} & \underline{P_1(I_\mathbf{y}, \langle s \rangle_1)} \\ \langle s\mathbf{z} \rangle_0, \langle \mathbf{z} \rangle_0 \leftarrow \mathsf{HSS.Eval}_0(I_\mathbf{y}, C, \langle s \rangle_0), & \langle s\mathbf{z} \rangle_1, \langle \mathbf{z} \rangle_1 \leftarrow \mathsf{HSS.Eval}_1(I_\mathbf{y}, C, \langle s \rangle_1). \end{array}$$

In an additional step, the party P_0 may send its share $\langle \mathbf{z} \rangle_0 \pmod 2$ to P_1 to reveal the Boolean evaluation result $\mathbf{z}$.

It was observed in [CMPR23] that the above HSS schemes allow for an extended evaluation procedure, where if replacing the additive share of s with shares of $s \cdot w$ and w for some integer w, then the extended evaluation results form additive shares of $s \cdot w \cdot \mathbf{z}$ and $w \cdot \mathbf{z}$. In other words, the HSS evaluation results over encrypted inputs $\mathbf{y}$ can be additionally multiplied with an integer w, when the two parties hold additive shares of sw and w.

$$\begin{array}{ll} \underline{P_0(I_\mathbf{y}, \langle s \rangle_0)} & \underline{P_1(I_\mathbf{y}, \langle s \rangle_1)} \\ \langle sw\mathbf{z} \rangle_0, \langle w\mathbf{z} \rangle_0 & \langle sw\mathbf{z} \rangle_1, \langle w\mathbf{z} \rangle_1 \\ \quad \leftarrow \mathsf{ExtEval}(I_\mathbf{y}, C, \langle sw \rangle_0, \langle w \rangle_0), & \quad \leftarrow \mathsf{ExtEval}(I_\mathbf{y}, C, \langle sw \rangle_1, \langle w \rangle_1). \end{array}$$

Taking the extended evaluation one step further, we can consider a matrix $\mathbf{W}$ as the additional input, and compute $\mathbf{z}' = \mathbf{W} \cdot C(\mathbf{y})$ (over $\mathbb{Z}$) as the final output. Including the additional step where P_0 sends its share of $\mathbf{z}' \pmod 2$ to P_1 to reveal $\mathbf{z}'$, we obtain an HSS evaluation "protocol" for NC1 Boolean circuits C, denoted HSS^C:

$$\begin{aligned} &(P_G : \langle s\mathbf{z}' \rangle_0), (P_E : \langle s\mathbf{z}' \rangle_1, \mathbf{z}') \\ &\qquad \leftarrow \mathsf{HSS}^C\big((P_G : I_\mathbf{y}, \langle s\mathbf{W} \rangle_0, \langle \mathbf{W} \rangle_0), (P_E : I_\mathbf{y}, \langle s\mathbf{W} \rangle_1, \langle \mathbf{W} \rangle_1)\big), \end{aligned}$$

which we replace Yao's garbling with in step 3 (Equation 7.2) and 3' from the previous paragraph. [8] The communication cost from HSS is only $|\mathbf{z}'|$ bits, much smaller than that of Yao.

[8] Readers may notice a mismatch, where from step 2 (Equation 7.1), the parties hold shares of $s\mathbf{w}$, but not of $\mathbf{w}$. This is not an issue, as P_E can compute $\mathbf{w}$ in the clear from the public input $\mathbf{x}$. The parties now hold a trivial share: $\langle \mathbf{w} \rangle_0 = 0$, $\langle \mathbf{w} \rangle_1 = \mathbf{w}$.

One detail to note is that in Yao's garbling we are free to use the global secret s from aHMAC also as the secret in Yao, but now with HSS, we need compatible instantiations with aHMAC, (see Section 7.4) so that they can share a common secret s. This usage of aHMAC and HSS requires us to prove security of the overall garbling scheme in a non-black-box way.

As anticipated, the computation implemented by this protocol is restricted: $\mathbf{z}' = \mathbf{W} \cdot C(\mathbf{y})$ over $\mathbb{Z}$, for an NC1 circuit C. In order to use $\mathsf{HSS}^{\mathsf{Enc}\circ\mathsf{Dec}}$ in place of $\mathsf{Yao}^{\mathsf{Enc}\circ\mathsf{Dec}}$ in Step 3', we need to find a suitable HE scheme where the $\mathsf{Enc} \circ \mathsf{Dec}$ circuit can be implemented in this restricted way:

$$\underbrace{\mathsf{ct}_{\mathbf{x}}}_{\mathbf{z}'} = \mathsf{Enc} \circ \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}^*_{\mathbf{x}}) = \underbrace{\mathsf{Bits}(\mathsf{ct}^*_{\mathbf{x}})}_{\mathbf{W}} \cdot \underbrace{C(\mathsf{sk})}_{C(\mathbf{y})} \text{ over } \mathbb{Z}.$$

While such an HE scheme may seem hard to find, our observation is that the size of evaluated ciphertexts $|\mathsf{ct}^*_{\mathbf{x}}|$ don't matter in our scheme, as the communication cost from HSS is exactly the size of a fresh ciphertext $|\mathsf{ct}_{\mathbf{x}}|$. In fact, viewing one-time-pad as a trivial HE scheme suffices! We illustrate a simple case of homomorphically multiplying one-time-pad ciphertexts, and the $\mathsf{Enc} \circ \mathsf{Dec}$ computation.

$$\begin{aligned}
&\mathsf{ct}_x := x \oplus r_x, \quad \mathsf{ct}_y := y \oplus r_y, \quad \text{ where } r_x = \mathsf{PRF}(\mathsf{sk}, 1), r_y = \mathsf{PRF}(\mathsf{sk}, 2).\\
&\mathsf{ct}^*_z = \mathsf{HMult}(\mathsf{ct}_x, \mathsf{ct}_y) = (\mathsf{ct}_x, \mathsf{ct}_y, \mathsf{ct}_x \cdot \mathsf{ct}_y).\\
&\mathsf{Enc} \circ \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}^*_z) = (\mathsf{ct}_x \oplus r_x) \cdot (\mathsf{ct}_y \oplus p_2) \oplus r_z, \text{ where } r_z = \mathsf{PRF}(\mathsf{sk}, 3)\\
&\qquad = C_1(\mathsf{sk})\mathsf{ct}_x + C_2(\mathsf{sk})\mathsf{ct}_y + C_3(\mathsf{sk})\mathsf{ct}_x \cdot \mathsf{ct}_y + C_4(\mathsf{sk}) \text{ over } \mathbb{Z}.\\
&\qquad = \mathsf{Bits}(\mathsf{ct}^*_z) \cdot C(\mathsf{sk}) \text{ over } \mathbb{Z} \text{ where } C := (C_1, C_2, C_3, C_4).
\end{aligned}$$

The final equality, writing Boolean operations as a polynomial over $\mathbb{Z}$, uses the fact that $x \oplus y = x + y - 2xy$ over $\mathbb{Z}$ for $x, y \in \{0, 1\}$. In the following, we directly write $\overline{\mathbf{x}}$ to denote one-time-padded $\mathbf{x}$, instead of $\mathsf{ct}_{\mathbf{x}}$.

In summary, our final Boolean garbling scheme for a circuit C starts with two parties P_G, P_E holding additive shares $\langle s\overline{\mathbf{x}} \rangle$ and P_E holding $\overline{\mathbf{x}}$ in the clear, where $\overline{\mathbf{x}}$ represents one-time-padded inputs. For every gate in C, in a topological order, both parties applie aHMAC evaluations to "homomorphically" add or multiply two one-time-padded inputs, and then

run HSS to decrypt and re-encrypt the resulting bit. The communication cost per gate is excatly 1-bit from the HSS protocol.

Generalization: Evaluating $O(\log \lambda)$-ary Gates. Observe that the technique of combining aHMAC and HSS from the previous paragraph can be viewed as a more general protocol for computation over some public masked input $\overline{\mathbf{x}}$ and a private secret key sk for deriving the masks. Using aHMAC we can evaluate any arithmetic circuit C_{Pub} (with bounded intermediate values) on $\overline{\mathbf{x}}$, and with HSS we can evaluate any NC1 Boolean circuit C_{Priv} on sk. The two results are then multiplied as an inner product over $\mathbb{Z}$. We summarize it as a protocol $\mathsf{aHMAC\text{-}HSS}^{C_{\text{Pub}},C_{\text{Priv}}}$:

$$\begin{aligned}&(P_G : \langle sz' \rangle_0), (P_E : \langle sz' \rangle_1, z')\\ &\qquad \leftarrow \mathsf{aHMAC\text{-}HSS}^{C_{\text{Pub}},C_{\text{Priv}}}\big((P_G : I_{\mathsf{sk}}, \langle s\overline{\mathbf{x}} \rangle_0), (P_E : I_{\mathsf{sk}}, \langle s\overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}})\big),\\ &// \ z' = \langle C_{\text{Pub}}(\overline{\mathbf{x}}), C_{\text{Priv}}(\mathsf{sk}) \rangle.\end{aligned}$$

The communication cost of this protocol is 1 bit. Note that the result z' is revealed to the evaluator P_E, hence should always be masked by a pseudo-random pad derived from sk.

Given this more general view, we can in fact use $\mathsf{aHMAC\text{-}HSS}^{C_{\text{Pub}},C_{\text{Priv}}}$ to compute any function g over $O(\log \lambda)$ masked input bits, and re-mask the resulting value. In particular, we choose $C_{\text{Pub}}(\overline{\mathbf{x}})$ to compute a one-hot vector $(0, \ldots, 0, 1, 0, \ldots, 0)$, where all but the $\overline{\mathbf{x}}$-th component are 0. (See Fact 1.) And we choose $C_{\text{Priv}}(\mathsf{sk}) = (\ldots, C_{\text{Priv},\mathbf{v}}(\mathsf{sk}), \ldots)_{\mathbf{v}}$ to compute a vector listing evaluated values $g(\mathbf{x})$ (and then masked) for all possible values of $\overline{\mathbf{x}}$.

$$\begin{aligned}\forall \mathbf{v} \in \{0,1\}^{|\overline{\mathbf{x}}|}, \quad & C_{\text{Pub},\mathbf{v}}(\overline{\mathbf{x}}) = 1 \text{ iff } \overline{\mathbf{x}} = \mathbf{v}\\ & C_{\text{Priv},\mathbf{v}}(\mathsf{sk}) = g(\mathbf{v} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{id})) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{id}').\\ // \ z' = \langle C_{\text{Pub}}(\overline{\mathbf{x}}), & C_{\text{Priv}}(\mathsf{sk}) \rangle = g(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{id}').\end{aligned}$$

The $\mathsf{id}, \mathsf{id}'$ from the above means some distinct ids assigned to every wire of the overall circuit consisting of these $O(\log \lambda)$-ary gates.

In summary, our generalized technique can garble Boolean circuits consisting of arbitrary $O(\log \lambda)$-ary gates, costing 1 bit per such gate. As applications, we show how to obtain a

scheme for *layered* circuits C^{Layer} with garbling size $|\widehat{C^{\mathsf{Layer}}}| \leq O(|C^{\mathsf{Layer}}|/\log\log\lambda) + \mathrm{poly}(\lambda)$ in Section 7.5, and a scheme for arithmetic circuits C over $\mathbb{Z}_R$ with garbling size $|\widehat{C}| \leq O(|C|\log R) + \mathrm{poly}(\lambda)$ in Section 7.6.

Other Extensions. Our techniques rely on two primitives:

- aHMAC which has been instantiated under the circular power-DDH (CP-DDH) assumptions in Paillier groups or prime-order groups in [ILL24b];
- HSS which has been instantiated under the DDH assumption in Paillier groups [ADOS22], prime-order groups [BGI16], and the RLWE assumption [BKS19].

We introduce a new lattice assumption, CP-RLWE, analogous to the CP-DDH assumption in groups, and show three instantiations of our technique of combining aHMAC and HSS under either CP-DDH in Paillier groups, in prime-order groups, or CP-RLWE. As noted earlier, since we require using a common secret in both aHMAC and HSS, we have to prove the security of our garbling schemes in a non-black-box way.

The work of [ILL24b] also constructed leveled variants of aHMAC that avoids the circular assumptions at the cost of a larger evaluation key evk with size linear in the supported evaluation depth. We also construct leveled garbling schemes using leveled aHMAC and (normal) HSS at the cost of increasing the garbling size by $\mathrm{Depth}(C) \cdot \mathrm{poly}(\lambda)$ bits. They can be instantiated under P-DDH plus DDH in Paillier groups, P-DDH in prime-order groups, or P-RLWE. (See Section 7.5.2 for details.)

Finally, we note that existing aHMAC and HSS instantiations under prime-order groups suffer a $1/\mathrm{poly}(\lambda)$ correctness error. This causes a $1/\mathrm{poly}(\lambda)$ error for both correctness and *privacy* in our garbling scheme under prime-order groups. We show in Section 7.5.5 how to adapt existing HSS amplification techniques [BGI17] to our setting to remove the $1/\mathrm{poly}(\lambda)$ error at the price of increased computation cost and, in the non-leveled variant, assuming a variant of CP-DDH (Definition 7.7).

7.3 Preliminaries

In this chapter, we focus on constructing a Boolean garbling scheme (Definition 2.1) that achieves the following notion of succinctness. Roughly, it requires the garbling size to be smaller than the circuit description size.

Definition 7.1 (Succinct Boolean Garbling Schemes)**.** *We say a Boolean garbling scheme is* succinct *if there exists a polynomial $p(\lambda)$ such that for every supported ring $\mathcal{R}$ and every $\lambda \in \mathbb{N}$, sufficiently large circuits C (over $\mathcal{R}$) with $|C| > p(\lambda)$ have garbling sizes $|\widehat{C}| \leq |C| \cdot \log |C|$.*

7.3.1 Basic Notations.

For an integer value within some range $x \in [B]$, we write $\mathsf{Bits}(x)$ to denote its bit-representation as a Boolean vector of dimension $\lceil \log B \rceil$, and $\mathsf{BitComp}(\mathbf{x} \in \{0,1\}^{\lceil \log B \rceil})$ to denote the linear function that recovers x from its bit-representation.

We write $\langle x \rangle_0, \langle x \rangle_1$ to denote a pair of additive shares (over a ring $\mathcal{R}$) of the value x, i.e. the notation represents two arbitrary values $v_0, v_1 \in \mathcal{R}$ such that $v_1 = v_0 + x$ over $\mathcal{R}$. In this work we will consider additive shares over the integers $\mathcal{R} = \mathbb{Z}$ and over the polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where n is a power-of-two.

When describing invocations of (sub-)protocols between two parties P_G, P_E, we write

$$(P_G : O_G), (P_E : O_E) \leftarrow \mathsf{Protocol}\left((P_G : I_G), (P_E : I_E)\right)$$

to mean the parties respectively hold inputs I_G, I_E when entering the protocol, and obtain outputs O_G, O_E after the protocol.

We assume all gates and wires in a circuit are labeled by distinct ids in $\{0,1\}^\lambda$. We write $\mathsf{InWires}(C)$ to denote the ids of all input wires to C, and $\mathsf{InWires}(\mathsf{g}), \mathsf{OutWire}(\mathsf{g})$ to respectively denote the ids of input wires to, and output wire from a gate $\mathsf{g} \in C$.

When writing invocations of a function $f : \mathcal{X} \to \mathcal{Y}$, we use the short-hand $f(\mathbf{x} \in \mathcal{X}^\ell) \in \mathcal{Y}^\ell$ to mean parallel invocations of f on every component of the vector $\mathbf{x}$. For example, given a

$\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$, we write

$$\overline{\mathbf{x}} = \mathbf{x} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(\mathsf{g}))$$

to mean computing masked inputs $\overline{\mathbf{x}}$ to some gate g using parallel invocations of a PRF (w.r.t. different wire ids) under a secret key sk.

7.3.2 Hardness Assumptions in Groups

We consider two types of groups, Paillier groups (Definition 2.7) (of composite orders) and prime-order groups (Definition 2.6) in this work. See Definition 6.7 for the DDH assumption formulated in these groups, and Lemma 2.1 for a quick review of some facts about Paillier groups.

We next introduce two variants of the standard DDH assumption in those groups. Our first variant, power-DDH, was first introduced by [CNs07, AHI11] in prime-order groups, and formulated in Paillier groups (as an instance of the NIDLS framework) by [ARS24]. Roughly, the assumption states that a group element g raised to the powers of a random secret exponent $s, s^2, s^3, \ldots$ still "look random". In this work we only need the weaker version that consider the first and second powers s, s^2.

Definition 7.2 (Power-DDH Assumption [CNs07, AHI11, ARS24]). *We say the power-DDH assumption (P-DDH) holds in Paillier groups if the following holds for every polynomial $\zeta(\lambda)$:*

$$\left\{ \mathsf{pp}, g, g^s, g^{s^2} \;\middle|\; \begin{array}{c} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ s \leftarrow [N]. \end{array} \right\}_\lambda$$
$$\approx_c \left\{ \mathsf{pp}, g, g^a, g^b \;\middle|\; \begin{array}{c} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b \leftarrow [N^{\zeta+2}]. \end{array} \right\}_\lambda .$$

We say P-DDH holds in prime-order groups if the following holds:

$$\left\{ \mathsf{pp}, g, g^s, g^{s^2} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda$$

$$\approx_c \left\{ \mathsf{pp}, g, g^a, g^b \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ a, b \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda .$$

Remark. As remarked in [ILL24b], in prime-order groups, power-DDH implies DDH: the reduction given a power-DDH tuple (g, g^s, g^{s^2}) samples $a, b \leftarrow \mathbb{Z}_p$ to re-randomize the tuple as $(g, g^{s \cdot a}, g^{s \cdot b}, g^{s^2 \cdot ab})$, which becomes a valid DDH tuple. If the reduction is given a random tuple (g, g^s, g^r), the re-randomized is also random. We show below that power-DDH also implies DDH in Paillier groups, via a slightly different reduction suggested by Lawrence Roy.

Lemma 7.1. *Power-DDH implies DDH in prime-order or Paillier groups.*

Proof. The implication in prime-order groups is sketched in the remark above. We focus on proving the implication in Paillier groups via a series of hybrid distributions that transitions from the left-hand side to the right-hand side in the DDH assumption (Definition 6.7).

Hyb_0**:** This is the left-hand side in the DDH assumption:

$$\left\{ \mathsf{pp}, g, g^a, g^b, g^{ab} \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ \quad g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b \leftarrow [N]. \end{array} \right\}_\lambda .$$

Hyb_1**:** In this hybrid, we sample the random exponents a, b from a much larger range $[N^{\zeta+2}]$.

$$\left\{ \mathsf{pp}, g, g^a, g^b, g^{ab} \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ \quad g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b \leftarrow [N^{\zeta+2}]. \end{array} \right\}_\lambda .$$

By P-DDH in Paillier groups (looking only at the first three terms), we have $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$.

Hyb_2: In this hybrid, we shift the random components a, b by a common random factor $r \leftarrow [N]$.

$$\left\{ \mathsf{pp}, g, g^{(a+r)}, g^{(b+r)}, g^{(a+r)(b+r)} \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b \leftarrow [N^{\zeta+2}], r \leftarrow [N]. \end{array} \right\}_\lambda .$$

Since a, b are sampled at random from a much larger range than r, they statistically "smudges" the term r. We have $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$.

Hyb_3: In this hybrid, we replace the square term r^2 in the exponent $(a + r)(b + r)$.

$$\left\{ \mathsf{pp}, g, g^{(a+r)}, g^{(b+r)}, g^{ab+(a+b)r+c} \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b, c \leftarrow [N^{\zeta+2}], r \leftarrow [N]. \end{array} \right\}_\lambda .$$

By P-DDH in Paillier groups, we have $\mathsf{Hyb}_2 \approx_c \mathsf{Hyb}_3$.

Hyb_4: In this hybrid, we combine two steps: first artificially add a square term r^2 to the exponent $ab + (a + b)r + c$, and then replace the terms $(a + r), (b + r)$ with a, b.

$$\left\{ \mathsf{pp}, g, g^{a}, g^{b}, g^{ab+c} \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b, c \leftarrow [N^{\zeta+2}]. \end{array} \right\}_\lambda .$$

The first step is statistically indistinguishable because c is sampled at random from a much larger range than r^2, hence smudges the term r^2. The second step is also statistically indistinguishable because a, b are sampled from much larger ranges than r, hence smudges the term r. We have $\mathsf{Hyb}_3 \approx \mathsf{Hyb}_4$.

Hyb_5: In this hybrid, we sample the random exponents a, b from a much smaller range $[N]$.

$$\left\{ \mathsf{pp}, g, g^{a}, g^{b}, g^{ab+c} \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b \leftarrow [N], c \leftarrow [N^{\zeta+2}]. \end{array} \right\}_\lambda .$$

By P-DDH in Paillier groups, we have $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_5$.

hyb$_6$**:** In this hybrid, we remove the term ab from the exponent $ab + c$.

$$\left\{ \mathsf{pp}, g, g^a, g^b, g^c \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ \quad g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b \leftarrow [N], c \leftarrow [N^{\zeta+2}]. \end{array} \right\}_\lambda .$$

Since c is sampled at random from a much larger range than a, b now, it statistically smudges the term ab. We have $\mathsf{Hyb}_5 \approx \mathsf{Hyb}_6$.

Hyb_7**:** This is the right-hand side in the DDH assumption:

$$\left\{ \mathsf{pp}, g, g^a, g^b, g^c \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ \quad g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ a, b, c \leftarrow [N^{\zeta+2}]. \end{array} \right\}_\lambda .$$

By P-DDH, we have $\mathsf{Hyb}_7 \approx \mathsf{Hyb}_8$. □

The next circular variant was first introduced by [ILL24b] both over Paillier groups and prime-order groups. It further assumes that the DDH sample using s^2 as the secret exponent can securely hide (bits of) the secret s itself, after proper re-randomization.

Definition 7.3 (Circular-Power-DDH [ILL24b])**.** *We say the circular-power-DDH assumption (CP-DDH) holds in Paillier groups if the following holds for every polynomial* $\zeta(\lambda)$*:*

$$\left\{ \begin{array}{l} \mathsf{pp}, g, g^s, g^{s^2}, g^{a_i}, g^{sa_i}, g^{s^2 a_i}(1+N)^{s[i]} \\ \quad (\text{for } i \in \lceil \log N \rceil) \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ \quad g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ s, \{a_i\} \leftarrow [N]. \end{array} \right\}_\lambda$$

$$\approx_c \left\{ \begin{array}{l} \mathsf{pp}, g, g^s, g^d, g^{a_i}, g^{b_i}, g^{c_i} \\ \quad (\text{for } i \in \lceil \log N \rceil) \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta), \\ \quad g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ s, d, \{a_i, b_i, c_i\} \leftarrow [N^{\zeta+2}]. \end{array} \right\}_\lambda .$$

We say CP-DDH holds in prime-order groups if the following holds:

$$\left\{ \begin{array}{l} \mathsf{pp}, g, g^s, g^{s^2}, g^{a_i}, g^{sa_i}, g^{s^2 a_i + s[i]} \\ \quad (\text{for } i \in \lceil \log p \rceil) \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s, \{a_i\} \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda$$

$$\approx_c \left\{ \begin{array}{l} \mathsf{pp}, g, g^s, g^d, g^{a_i}, g^{b_i}, g^{c_i} \\ \quad (\text{for } i \in \lceil \log p \rceil) \end{array} \;\middle|\; \begin{array}{l} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s, d, \{a_i, b_i, c_i\} \leftarrow \mathbb{Z}_p. \end{array} \right\}_\lambda .$$

Remark. We modify the formulation from [ILL24b] to include the g^{s^2} term in the indistinguishability, so that it implies both DDH and P-DDH and looks more natural. The proof (Theorem 4 in [ILL24b]) that CP-DDH in prime-order groups holds in the generic group model (GGM) still goes through for our variant.

7.3.3 Lattice Hardness Assumptions

In this work, we consider two variants to the standard RingLWE assumption (Definition 2.11) over polynomial rings of the form $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$, where $n(\lambda)$ is a power-of-2. Let $q(\lambda) > 2$ be a modulus, $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda) \subseteq \mathcal{R}$ be error and secret distributions. The standard RingLWE assumption (w.r.t. $\mathcal{R}, q, \mathcal{D}_{\mathsf{sk}}, \mathcal{D}_{\mathsf{err}}$) states that for every polynomial $m(\lambda)$, the following computational indistinguishability holds:

$$\left\{ \begin{array}{l|l} \mathbf{a},\ s \cdot \mathbf{a} + \mathbf{e} & s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e} \leftarrow \mathcal{D}_{\mathsf{err}}^m, \\ \text{(over } \mathcal{R}_q) & \mathbf{a} \leftarrow \mathcal{R}_q^m \end{array} \right\}_\lambda \approx_c \left\{ \mathbf{a}, \mathbf{b} \leftarrow \mathcal{R}_q^m \right\}_\lambda,$$

where $\mathcal{R}_q = \mathcal{R}/(q\mathcal{R})$. The first variant considers the case where two vectors of RingLWE samples are computed using the same public vector $\mathbf{a}$, correlated secrets s, s^2, and fresh errors $\mathbf{e}_1, \mathbf{e}_2$. This is a weaker version of the power RingLWE assumption first introduced in [ARS24], which considers multiple powers of s instead of just 2.

Definition 7.4 (Power RingLWE[ARS24])**.** *We say the power RingLWE (P-RLWE) assumption holds with respect to the ring $\mathcal{R}(\lambda)$, a modulus $q(\lambda)$, error and secret distributions $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda)$ if the following holds for every polynomial $m(\lambda)$:*

$$\left\{ \begin{array}{l|l} \mathbf{a},\ s \cdot \mathbf{a} + \mathbf{e}_1,\ s^2 \cdot \mathbf{a} + \mathbf{e}_2 & s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e}_1, \mathbf{e}_2 \leftarrow \mathcal{D}_{\mathsf{err}}^m, \\ \text{(over } \mathcal{R}_\mathrm{q}) & \mathbf{a} \leftarrow \mathcal{R}_q^m \end{array} \right\}_\lambda \approx_c \left\{ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow \mathcal{R}_q^m \right\}_\lambda$$

Remark. P-RLWE implies the standard RLWE, which just requires indistinguishability of the first 2 terms in the above.

Our next circular variant further assumes that the RingLWE sample using s^2 as the secret can securely hide the secret s itself.

Definition 7.5 (Circular Power RingLWE)**.** *We say the circular power RingLWE (CP-RLWE) assumption holds with respect to the ring $\mathcal{R}(\lambda)$, two modulus $p(\lambda), q(\lambda)$ such that $q = p \cdot \Delta$, error and secret distributions $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda)$ if the following holds for every polynomial $m(\lambda)$:*

$$\left\{ \begin{array}{l|l} \mathbf{a},\ s \cdot \mathbf{a} + \mathbf{e}_1,\ s^2 \cdot \mathbf{a} + \mathbf{e}_2 + s \cdot \Delta & s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e}_1, \mathbf{e}_2 \leftarrow \mathcal{D}_{\mathsf{err}}^m, \\ \text{(over } \mathcal{R}_{\mathrm{q}}) & \mathbf{a} \leftarrow \mathcal{R}_q^m \end{array} \right\}_\lambda \approx_c \left\{ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow \mathcal{R}_q^m \right\}_\lambda$$

7.4 aHMAC and HSS as Evaluation Procedures

In this work, we make use of two tools from prior works, an algebraic homomorphic MAC scheme (aHMAC) [ILL24b], and a homomorphic secret sharing scheme (HSS) with extended evaluations [CMPR23, ARS24]. At a highlevel, both schemes are run between a pair of parties, which we call the garbler and the evaluator, who jointly hold (not neccessarily additive) secret shares with respect to some input values $\mathbf{x}$.

- An aHMAC scheme allows the parties to locally evaluate arithmetic circuits (over bounded integers) on their input shares, if the evaluator additionally knows the inputs $\mathbf{x}$ in the clear.
- An HSS scheme allows the parties to locally evaluate a weaker program class (including NC1 Boolean circuits), but without requiring the evaluator to learn $\mathbf{x}$.

When garbling and evaluating a circuit, our techniques require interleaving aHMAC and HSS evaluations on secret shares of intermediate wire values. In particular, they require conversions between the two schemes' share formats.

For this reason, (except in the leveled variants of our grabling constructions,) we need to setup the two schemes using correlated secret randomness, and hence cannot directly invoke their standard security definitions. In the following lemmas we focus only on their correctness properties, and expose the underlying construction detail of their "setup" algorithm (for generating public data pd).

We stress that our garbling schemes will use the evaluation procedures of HSS and aHMAC

as subroutines, and we will directly prove the security of our garbling schemes without relying on the security aHMAC and HSS in a black-box way.

7.4.1 aHMAC and HSS under Paillier Groups

The following lemmas summarize the aHMAC constructions under Paillier groups from [ILL24b], including both the non-leveled and leveled variants. We refer readers to [ILL24b] for more details. We note that [ILL24b] presents the constructions in the language of NIDLS framework [ADOS22], which covers Paillier groups, class groups, and a variant of Joye-Libert encryption as known instantiations. In this work, we chose to focus on Paillier groups for clarity. Our results can be generalized to fit NIDLS framework, and enjoy other instantiations covered by it.

Lemma 7.2 (aHMAC Gate Evaluation under Paillier Groups)**.** *Let $B < 2^{\mathrm{poly}(\lambda)}$ be a bound on input values, and $\zeta = \lceil \log B/(2\lambda) \rceil + 1$. There exist two pairs of efficient algorithms:*

- $\mathsf{MultKey}(\mathsf{pd}, w_0^x, w_0^y)$ *takes as inputs public data* pd *and two integer values* $w_0^x, w_0^y \in \mathbb{Z}$*. It outputs an integer* $w_0^z \in \mathbb{Z}$*.*
- $\mathsf{MultTag}(\mathsf{pd}, w_1^x, w_1^y, x, y)$ *takes as input public data* pd *and four integer values* $w_1^x, w_1^y, x, y \in \mathbb{Z}$*. It outputs an integer* $w_1^z \in \mathbb{Z}$*.*
- $\mathsf{AddKey}, \mathsf{AddTag}$ *have the same syntax as* $\mathsf{MultKey}, \mathsf{MultTag}$*, respectively.*

For every $\lambda \in \mathbb{N}$, $\mathsf{pp} = (N, \zeta)$ in the support of $\mathsf{Pai.Gen}(1^\lambda, 1^\zeta)$, secret exponents, $s, s' \in [N]$, inputs $x, y \in [B]$ such that $xy < B$, and additive shares (over $\mathbb{Z}$) $\langle sx \rangle_0, \langle sx \rangle_1, \langle sy \rangle_0, \langle sy \rangle_1$, the following holds:

$$\Pr\left[\begin{array}{l|l} w_1^z = w_0^z + s' \cdot xy & w_0^z = \mathsf{MultKey}(\mathsf{pd}, \langle sx \rangle_0, \langle sy \rangle_0) \\ \text{(over } \mathbb{Z}) & w_1^z = \mathsf{MultTag}(\mathsf{pd}, \langle sx \rangle_1, \langle sy \rangle_1, x, y) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

$$\Pr\left[\begin{array}{l|l} w_1^z = w_0^z + s \cdot (x + y) & w_0^z = \mathsf{AddKey}(\mathsf{pd}, \langle sx \rangle_0, \langle sy \rangle_0) \\ \text{(over } \mathbb{Z}) & w_1^z = \mathsf{AddTag}(\mathsf{pd}, \langle sx \rangle_1, \langle sy \rangle_1, x, y) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of pd*, which is computed as follows:*

$$g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ \mathbf{r} \leftarrow [N]^{\lceil \log N \rceil},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda$$

$$\mathsf{pd} := (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{r}}, g^{\mathbf{r}s}, \{g^{\mathbf{r}[i]s^2} \cdot (1+N)^{\mathsf{Bits}(s')[i]}\}).$$

We write $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, s')$ *to denote public data* pd *computed as above with freshly sampled* g*,* $\mathbf{r}$*, and* seed*.*

In the above, if we choose the secret exponents $s' = s$, then we can compose MultKey, MultTag, MultKey, MultTag to obtain algorithms EvalKey, EvalTag that respectively evaluates an arithmetic C over additive shares. Note that each invocation of those algorithms imposes a bound B on the underlying wire values. We therefore only consider bounded integer evaluations.

Definition 7.6 (Admissible Input w.r.t. B)**.** *Let C be an arithmetic circuit (with ℓ_x inputs) over $\mathbb{Z}$. We say an input $\mathbf{x} \in \mathbb{Z}^{\ell_x}$ is admissible w.r.t. some positive integer B if all intermediate wire values of $C(\mathbf{x})$ are bounded by B.*

Lemma 7.3 (aHMAC Circuit Evaluation under Paillier Groups)**.** *Under the same setting as Lemma 7.2, and assume the existence of a PRG, there exists a pair of efficient algorithms:*

- $\mathsf{EvalKey}(\mathsf{pd}, C, \mathbf{w}_0^x)$ *takes public data* pd*, an arithmetic circuit* $C : \mathbb{Z}^{\ell_x} \to \mathbb{Z}^{\ell_z}$*, and a vector* $\mathbf{w}_0^x \in \mathbb{Z}^{\ell_x}$*. It outputs a vector* $\mathbf{w}_0^z \in \mathbb{Z}^{\ell_z}$*.*
- $\mathsf{EvalTag}(\mathsf{pd}, C, \mathbf{w}_1^x, \mathbf{x})$ *takes public data* pd*, an arithmetic circuit* C*, two vectors* $\mathbf{w}_0^x, \mathbf{x} \in \mathbb{Z}^{\ell_x}$*. It outputs a vector* $\mathbf{w}_1^z \in \mathbb{Z}^{\ell_z}$*.*

For every polynomial $p(\lambda)$, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$,* $\mathsf{pp} = (N, \zeta)$ *in the support of* $\mathsf{Pai.Gen}(1^\lambda, 1^\zeta)$*, secret exponents, $s \in [N]$, arithmetic circuit C with $|C| \le p(\lambda)$, admissible inputs $\mathbf{x}$ w.r.t B, and additive shares (over $\mathbb{Z}$) $\langle s\mathbf{x} \rangle_0, \langle s\mathbf{x} \rangle_1$, the following holds:*

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_1^z = \mathbf{w}_0^z + s \cdot C(\mathbf{x}) & \mathbf{w}_0^z = \mathsf{EvalKey}(\mathsf{pd}, C, \langle s\mathbf{x} \rangle_0) \\ (\text{over } \mathbb{Z}) & \mathbf{w}_1^z = \mathsf{EvalTag}(\mathsf{pd}, C, \langle s\mathbf{x} \rangle_1, \mathbf{x}) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of pd*, which is computed as* $\mathsf{pd} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, s)$*.*

Alternatively, by using a vector of different secret exponents $\mathbf{s} \in \mathbb{Z}^{d+1}$ we can compose $\mathsf{MultKey}$, $\mathsf{MultTag}$, $\mathsf{MultKey}$, $\mathsf{MultTag}$ to obtain *leveled* variants of algorithms $\mathsf{EvalKey}^d$, $\mathsf{EvalTag}^d$ that respectively evaluates an arithmetic C of depth bounded by d. [9]

Lemma 7.4 (aHMAC Leveled Circuit Evaluation under Paillier Groups)**.** *Under the same setting as Lemma 7.2, assuming the existence of a PRG, for every polynomial depth bound $d(\lambda)$, there exists a pair of efficient deterministic algorithms:*

- $\mathsf{EvalKey}^d(\mathsf{pd}, C, \mathbf{w}_0^x)$ *takes public data* pd*, an arithmetic circuit* $C : \mathbb{Z}^{\ell_x} \to \mathbb{Z}^{\ell_z}$ *of depth at most d, and a vector $\mathbf{w}_0^x \in \mathbb{Z}^{\ell_x}$. It outputs a vector $\mathbf{w}_0^z \in \mathbb{Z}^{\ell_z}$.*
- $\mathsf{EvalTag}^d(\mathsf{pd}, C, \mathbf{w}_1^x, \mathbf{x})$ *takes public data* pd*, an arithmetic circuit C of depth at most d, two vectors $\mathbf{w}_0^x, \mathbf{x} \in \mathbb{Z}^{\ell_x}$. It outputs a vector $\mathbf{w}_1^z \in \mathbb{Z}^{\ell_z}$.*

For every polynomial $p(\lambda)$, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$,* $\mathsf{pp} = (N, \zeta)$ *in the support of* $\mathsf{Pai.Gen}(1^\lambda, 1^\zeta)$*, secret exponents, $\mathbf{s} \in [N]^{d+1}$, arithmetic circuit C with $|C| \le p(\lambda)$ and* $\mathrm{Depth}(C) < d(\lambda)$*, admissible inputs $\mathbf{x}$ w.r.t B, and additive shares (over $\mathbb{Z}$) $\langle \mathbf{s}[0] \cdot \mathbf{x} \rangle_0$, $\langle \mathbf{s}[0] \cdot \mathbf{x} \rangle_1$, the following holds:*

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s}[d] \cdot C(\mathbf{x}) & \mathbf{w}_0^z = \mathsf{EvalKey}^d(\mathsf{pd}, C, \langle \mathbf{s}[0] \cdot \mathbf{x} \rangle_0) \\ \text{(over } \mathbb{Z}) & \mathbf{w}_1^z = \mathsf{EvalTag}^d(\mathsf{pd}, C, \langle \mathbf{s}[0] \cdot \mathbf{x} \rangle_1, \mathbf{x}) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of $\mathsf{pd} = \{\mathsf{pd}^{(j)}\}_{[d]}$*, computed as*

$$\mathsf{pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[j], \mathbf{s}[j+1]).$$

The following lemma summarizes the HSS construction under Paillier groups from [ADOS22]. Again, the results in [ADOS22] are presented in the language of NIDLS framework, which covers Paillier groups as a particular instantiation.

[9]In the leveled variant, shares of two intermediate wires to a gate can have different secret "levels". We can artificially increase the lower wire by multiplying with a constant wire of value 1. In this work, we assume the inputs to a computation contains a constant wire 1 (with secret level 0). Multiplying 1 with itself then provides constant wires with any secret level as needed. This assumption does not affect the asymptotic size of our garbling schemes.

Lemma 7.5 (HSS Extended Evaluation under DCR Groups [ADOS22]). *Under the same setting as Lemma 7.2, and assume the existence of a PRG, there exists a pair of efficient algorithms,* $\mathsf{ExtEval}_0$, $\mathsf{ExtEval}_1$, *where*

- $\mathsf{ExtEval}_b(\mathsf{pd}_{\mathbf{y}}, C, \mathbf{w}_b^x, \mathbf{v}_b^x)$*: takes public data* $\mathsf{pd}_{\mathbf{y}}$ *(with respect to some vector* $\mathbf{y} \in \{0,1\}^{\ell_y}$*), an NC1 Boolean circuit* $C : \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_x}$*, and two vectors* $\mathbf{w}_b^x, \mathbf{v}_b^x \in \mathbb{Z}^{\ell_x}$*. It outputs a pair of integers* $w_b^z, v_b^z \in \mathbb{Z}$*.*

For every logarithmic function $d(\lambda) \leq O(\log \lambda)$*, and every polynomial* $p(\lambda)$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$, $\mathsf{pp} = (N, \zeta)$ *in the support of* $\mathsf{Pai.Gen}(1^\lambda, 1^\zeta)$*, Boolean circuit* $C : \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_x}$ *with* $|C| < p(\lambda)$ *and* $\mathrm{Depth}(C) < d(\lambda)$*, secret exponents* $s \in [N]$*, inputs* $\mathbf{x} \in [B]^{\ell_x}$ *and* $\mathbf{y} \in \{0,1\}^{\ell_y}$*, and additive shares (over* $\mathbb{Z}$*)* $\langle s\mathbf{x} \rangle_0, \langle s\mathbf{x} \rangle_1, \langle \mathbf{x} \rangle_0, \langle \mathbf{x} \rangle_1$*, the following holds:*

$$\Pr\left[\begin{array}{l} w_1^z = w_0^z + sz \\ v_1^z = v_0^z + z \\ (\text{over } \mathbb{Z}) \end{array} \middle| \begin{array}{r} w_b^z, v_b^z = \mathsf{ExtEval}_b(\mathsf{pd}_{\mathbf{y}}, C, \langle s\mathbf{x} \rangle_b, \langle \mathbf{x} \rangle_b) \\ z := \mathsf{InnerPord}(\mathbf{x}, C(\mathbf{y})) \ (\text{over } \mathbb{Z}). \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of $\mathsf{pd}_{\mathbf{y}}$*, which is computed as follows.*

$$\begin{aligned} & g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ \mathbf{r}, \mathbf{r}' \leftarrow [N]^{\ell_y},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda \\ \mathsf{pd}_{\mathbf{y}}^{(i)} &:= \begin{pmatrix} g^{\mathbf{r}[i]}, & g^{\mathbf{r}[i]s} \cdot (1+N)^{\mathbf{y}[i]}, \\ g^{\mathbf{r}'[i]s}, & g^{\mathbf{r}'[i]} \cdot (1+N)^{\mathbf{y}[i]} \end{pmatrix} \quad \forall i \in [\ell_y], \\ \mathsf{pd}_{\mathbf{y}} &:= (\mathsf{pp}, \mathsf{seed}, \{\mathsf{pd}_{\mathbf{y}}^{(i)}\}). \end{aligned}$$

We write $\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, \mathbf{y})$ *to denote public data* $\mathsf{pd}_{\mathbf{y}}$ *computed as above with freshly sampled* g*,* $\mathbf{r}$*,* $\mathbf{r}'$*, and* seed*.*

7.4.2 aHMAC and HSS under Prime-Order Groups

The following lemmas summarize the aHMAC constructions under prime-order groups from [ILL24b], including both the non-leveled and leveled variants. The main difference

between these constructions and those under Paillier groups is that these only achieve $\delta = 1/\text{poly}(\lambda)$ correctness, and have computation costs scaling with $\sqrt{1/\delta}$. We refer readers to [ILL24b], for more details.

Lemma 7.6 (aHMAC Gate Evaluation under Prime-Order Groups)**.** *Let $B < \text{poly}(\lambda)$ be a bound on input values, $\delta = 1/\text{poly}(\lambda)$ be an error bound, and $\mathsf{Pri.Gen}$ be an instance generation algorithm for prime-order groups. There exists two pairs of efficient deterministic algorithms: $\mathsf{MultKey}, \mathsf{MultTag}, \mathsf{AddKey}, \mathsf{AddTag}$ with analogous syntax to Lemma 7.2.*

For every $\lambda \in \mathbb{N}$, $\mathsf{pp} = (G, p, g)$ in the support of $\mathsf{Pri.Gen}(1^\lambda)$, secret exponents, $\mathbf{s}, \mathbf{s}' \in \{0,1\}^{\lceil \log p \rceil}$, inputs $x, y \in [B]$ such that $xy < B$, and additive shares (over $\mathbb{Z}$) $\langle \mathbf{s}x \rangle_0, \langle \mathbf{s}x \rangle_1, \langle \mathbf{s}y \rangle_0, \langle \mathbf{s}y \rangle_1$, the following holds:

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s}' \cdot xy & w_0^z = \mathsf{MultKey}(\mathsf{pd}, \langle \mathbf{s}x \rangle_0, \langle \mathbf{s}y \rangle_0) \\ \text{(over } \mathbb{Z}) & w_1^z = \mathsf{MultTag}(\mathsf{pd}, \langle \mathbf{s}x \rangle_1, \langle \mathbf{s}y \rangle_1, x, y) \end{array}\right] > 1 - \delta(\lambda) - \text{negl}(\lambda),$$

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s} \cdot (x+y) & w_0^z = \mathsf{AddKey}(\mathsf{pd}, \langle \mathbf{s}x \rangle_0, \langle \mathbf{s}y \rangle_0) \\ \text{(over } \mathbb{Z}) & w_1^z = \mathsf{AddTag}(\mathsf{pd}, \langle \mathbf{s}x \rangle_1, \langle \mathbf{s}y \rangle_1, x, y) \end{array}\right] > 1 - \delta(\lambda) - \text{negl}(\lambda),$$

over the randomness of pd, which is computed as follows:

$$\mathbf{r} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil}, \mathsf{seed} \leftarrow \{0,1\}^\lambda, s := \mathsf{BitComp}(\mathbf{s}),$$
$$\mathsf{pd} := (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{r}}, g^{\mathbf{r}s}, g^{\mathbf{r}s^2 + \mathbf{s}'}).$$

We write $\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathbf{s}')$ to denote public data pd computed as above with freshly sampled $\mathbf{r}$, and seed.

Remark. The lemma also holds when the secret exponent $\mathbf{s}'$ has a different dimention $\ell \geq \lceil \log p \rceil$ than $\mathbf{s}$. In this case, the public data are computed with $\mathbf{r} \leftarrow \mathbb{Z}_p^\ell$ to match the dimension of $\mathbf{s}'$. We need to support this edge case when using aHMAC together with the HSS scheme based on BHHO encryption (Lemma 7.10), whose secret exponents has a dimension of $\lceil 3 \log p \rceil$.

Lemma 7.7 (aHMAC Circuit Evaluation under Prime-Order Groups)**.** *Under the same setting as Lemma 7.6, assuming the existence of a PRG, there exists a pair of efficient deterministic algorithms:* $\mathsf{EvalKey}, \mathsf{EvalTag}$ *with analogous syntax to Lemma 7.3.*

For every polynomial $p(\lambda)$, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*,* $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$*, secret exponents,* $\mathbf{s} \in \{0,1\}^{\lceil 3 \log p \rceil}$*, arithmetic circuit C with $|C| \leq p(\lambda)$, admissible inputs* $\mathbf{x}$ *w.r.t B, and additive shares (over $\mathbb{Z}$)* $\langle \mathbf{s} \otimes \mathbf{x} \rangle_0, \langle \mathbf{s} \otimes \mathbf{x} \rangle_1$*, the following holds:*

$$\Pr\left[\begin{matrix}\mathbf{W}_1^z = \mathbf{W}_0^z + \mathbf{s} \otimes C(\mathbf{x}) \\ \text{(over } \mathbb{Z}) \end{matrix}\ \middle|\ \begin{matrix}\mathbf{W}_0^z = \mathsf{EvalKey}(\mathsf{pd}, C, \langle \mathbf{s} \otimes \mathbf{x} \rangle_0) \\ \mathbf{W}_1^z = \mathsf{EvalTag}(\mathsf{pd}, C, \langle \mathbf{s} \otimes \mathbf{x} \rangle_1, \mathbf{x})\end{matrix}\right] > 1 - \delta(\lambda) - \mathrm{negl}(\lambda),$$

over the randomness of pd*, which is computed as* $\mathsf{pd} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathbf{s})$.

Lemma 7.8 (aHMAC Leveled Circuit Evaluation under Prime-Order Groups)**.** *Under the same setting as Lemma 7.6, assuming the existence of a PRG, for every polynomial depth bound $d(\lambda)$, there exists a pair of efficient deterministic algorithms:* $\mathsf{EvalKey}^d, \mathsf{EvalTag}^d$ *with analogous syntax to Lemma 7.4.*

For every polynomial $p(\lambda)$, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*,* $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$*, secret exponents* $\mathbf{S} \in \{0,1\}^{(d+1) \times \lceil 3 \log p \rceil}$*, arithmetic circuit C with $|C| \leq p(\lambda)$ and* $\mathrm{Depth}(C) < d(\lambda)$*, admissible inputs* $\mathbf{x}$ *w.r.t B, and additive shares (over $\mathbb{Z}$)* $\langle \mathbf{S}[0] \otimes \mathbf{x} \rangle_0$, $\langle \mathbf{S}[0] \otimes \mathbf{x} \rangle_1$*, the following holds:*

$$\Pr\left[\begin{matrix}\mathbf{W}_1^z = \mathbf{W}_0^z + \mathbf{S}[d] \otimes C(\mathbf{x}) \\ \text{(over } \mathbb{Z}) \end{matrix}\ \middle|\ \begin{matrix}\mathbf{W}_0^z = \mathsf{EvalKey}^d(\mathsf{pd}, C, \langle \mathbf{S}[0] \otimes \mathbf{x} \rangle_0) \\ \mathbf{W}_1^z = \mathsf{EvalTag}^d(\mathsf{pd}, C, \langle \mathbf{S}[0] \otimes \mathbf{x} \rangle_1, \mathbf{x})\end{matrix}\right]$$
$$> 1 - \delta(\lambda) - \mathrm{negl}(\lambda),$$

over the randomness of $\mathsf{pd} = \{\mathsf{pd}^{(j)}\}_{[d]}$*, computed as*

$$\mathsf{pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{S}[j], \mathbf{S}[j+1]).$$

The following two lemmas summarize the HSS constructions under prime-order groups from [BGI16]. The first variant was proven secure assuming circular security of ElGamal

encryption. We modify it slightly:

$$\text{ElGamal CT of } \mathbf{y} \text{ under s}: \quad g^{\mathbf{r}}, g^{\mathbf{r}\cdot s+\mathbf{y}}$$
$$\text{modified}: \quad g^{\mathbf{r}\cdot s}, g^{\mathbf{r}\cdot s^2+\mathbf{y}}.$$

The modified ciphertext can be decrypted in the same way using s, but we can now use CP-DDH to replace circular security assumption of ElGamal when proving security of the joint usage of aHMAC and this variant is secure.

Lemma 7.9 (HSS Extended Evaluation Based on ElGamal [BGI16])**.** *Under the same setting as Lemma 7.6, assuming the existence of a PRG, there exists a pair of efficient deterministic algorithms,* $\mathsf{ExtEval}_0$, $\mathsf{ExtEval}_1$, *with analogous syntax to Lemma 7.5.*

For every logarithmic function $d(\lambda) \leq O(\log \lambda)$, *polynomial* $p(\lambda)$, *there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$, $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$, *circuit* $C : \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_x}$ *with* $|C| < p(\lambda)$, $\mathrm{Depth}(C) < d(\lambda)$, *secret exponents* $\mathbf{s} \in \lceil \log p \rceil$, *inputs* $\mathbf{x} \in [B]^{\ell_x}$, $\mathbf{y} \in \{0,1\}^{\ell_y}$, *and additive shares (over* $\mathbb{Z}$*)* $\langle \mathbf{x} \otimes \mathbf{s} \rangle_0, \langle \mathbf{x} \otimes \mathbf{s} \rangle_1, \langle \mathbf{x} \rangle_0, \langle \mathbf{x} \rangle_1$, *the following holds:*

$$\Pr\left[\begin{array}{l|c} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s} \cdot z & w_b^z, v_b^z = \mathsf{ExtEval}_b(\mathsf{pd}_{\mathbf{y}}, C, \langle \mathbf{x} \otimes \mathbf{s} \rangle_b, \langle \mathbf{x} \rangle_b) \\ v_1^z = v_0^z + z & z := \mathsf{InnerPord}(\mathbf{x}, C(\mathbf{y})) \;\; (\text{over } \mathbb{Z}). \\ (\text{over } \mathbb{Z}) & \end{array}\right] > 1 - \delta(\lambda) - \mathrm{negl}(\lambda),$$

over the randomness of $\mathsf{pd}_{\mathbf{y}}$, *which is computed as follows.*

$$\mathbf{r} \leftarrow \mathbb{Z}_p^{\ell_y},\ \mathbf{R} \leftarrow \mathbb{Z}_p^{\ell_y \times \lceil \log p \rceil},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda,\ s := \mathsf{BitComp}(\mathbf{s})$$
$$\mathsf{pd}_{\mathbf{y}} := (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{r}\cdot s},\ g^{\mathbf{r}\cdot s^2+\mathbf{y}},\ g^{\mathbf{R}\cdot s},\ g^{\mathbf{R}\cdot s^2+\mathbf{y}\otimes\mathbf{s}}).$$

We write $\mathsf{HSS}^{\mathsf{EG}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathbf{y})$ *to denote public data* $\mathsf{pd}_{\mathbf{y}}$ *computed as above with freshly sampled* $\mathbf{r}$, $\mathbf{R}$, *and* seed.

The second variant was proven secure assuming only DDH, without assuming circular security, by using BHHO [BHHO08] encryption instead of ElGamal. We use this variant in our leveled garbling scheme, together with leveled aHMAC, so that we can use P-DDH instead of CP-DDH when proving security of garbling scheme.

Lemma 7.10 (HSS Extended Evaluation under BHHO [BGI16])**.** *Under the same setting as Lemma 7.6, assuming the existence of a PRG, there exists a pair of efficient deterministic algorithms,* $\mathsf{ExtEval}_0$*,* $\mathsf{ExtEval}_1$*, with analogous syntax to Lemma 7.5.*

For every logarithmic function $d(\lambda) \leq O(\log \lambda)$*, polynomial* $p(\lambda)$*, there exists a negligible function* $\text{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*,* $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$*, Boolean circuit* $C : \{0,1\}^{\ell_y} \to \{0,1\}^{\ell_x}$ *with* $|C| < p(\lambda)$*,* $\text{Depth}(C) < d(\lambda)$*, secret exponents* $\mathbf{s} \in \{0,1\}^{\lceil 3 \log p \rceil}$*, inputs* $\mathbf{x} \in [B]^{\ell_x}$*,* $\mathbf{y} \in \{0,1\}^{\ell_y}$*, and additive shares (over* $\mathbb{Z}$*)* $\langle \mathbf{x} \otimes \mathbf{s} \rangle_0$*,* $\langle \mathbf{x} \otimes \mathbf{s} \rangle_1$*,* $\langle \mathbf{x} \rangle_0$*,* $\langle \mathbf{x} \rangle_1$*, the following holds:*

$$\Pr\left[\begin{array}{l|c} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s} \cdot z & w_b^z, v_b^z = \mathsf{ExtEval}_b(\mathsf{pd}_{\mathbf{y}}, C, \langle \mathbf{x} \otimes \mathbf{s} \rangle_b, \langle \mathbf{x} \rangle_b) \\ v_1^z = v_0^z + z & z := \mathsf{InnerPord}(\mathbf{x}, C(\mathbf{y}))\ (\text{over } \mathbb{Z}). \\ (\text{over } \mathbb{Z}) & \end{array}\right] > 1 - \delta(\lambda) - \text{negl}(\lambda),$$

over the randomness of $\mathsf{pd}_{\mathbf{y}}$*, which is computed as follows.*

$$\mathbf{c} \leftarrow \mathbb{Z}_p^{\lceil 3 \log p \rceil},\ \mathbf{r} \leftarrow \mathbb{Z}_p^{\ell_y},\ \mathbf{R} \leftarrow \mathbb{Z}_p^{\ell_y \times \lceil 3 \log p \rceil},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda,$$
$$\mathsf{pd}_{\mathbf{y}} := (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{c} \otimes \mathbf{r}}, g^{\mathbf{c} \otimes \mathbf{R}}, g^{\mathsf{InnerPord}(\mathbf{c},\mathbf{s}) \cdot \mathbf{r} + \mathbf{y}}, g^{\mathsf{InnerPord}(\mathbf{c},\mathbf{s}) \cdot \mathbf{R} + \mathbf{y} \otimes \mathbf{s}}).$$

We write $\mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathbf{y})$ *to denote public data* $\mathsf{pd}_{\mathbf{y}}$ *computed as above with freshly sampled* $\mathbf{c}$*,* $\mathbf{r}$*,* $\mathbf{R}$*, and* seed*.*

7.4.3 aHMAC and HSS under Lattices

The following lemmas summarize analgous aHMAC constructions to Lemma 7.2, 7.3, and 7.4 under lattices. We present details of these constructions (hence prove the lemmas) in Section 7.4.4.

Lemma 7.11 (aHMAC Gate Evaluation under Lattices)**.** *Let* $B < 2^{\text{poly}(\lambda)}$ *be a bound on input values,* $\mathcal{R}$ *be the polynomial ring* $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ *where* $n(\lambda)$ *is a power-of-two,* $p \geq B \cdot \lambda^{\omega(1)}$*,* $q = p \cdot \Delta$ *be two moduli, where* $\Delta = B \cdot p \cdot \lambda^{\omega(1)}$ *is a scaling factor, and* $\mathcal{D}_{\mathsf{sk}}(\lambda), \mathcal{D}_{\mathsf{err}}(\lambda)$ *be error and secret distributions with coefficients bounded by* $\text{poly}(\lambda)$*. There*

exists two pairs of efficient deterministic algorithms, $\mathsf{MultKey}, \mathsf{MultTag}, \mathsf{AddKey}, \mathsf{AddTag}$*, with analogous syntax to Lemma 7.2.*

For every $\lambda \in \mathbb{N}$*, secret elements* $s, s' \in \mathcal{D}_{\mathsf{sk}}$*, inputs* $x, y \in [B]$ *such that* $xy < B$*, and additive shares (over* $\mathcal{R}$*)* $\langle sx \rangle_0, \langle sx \rangle_1, \langle sy \rangle_0, \langle sy \rangle_1$*, the following holds:*

$$\Pr\left[\begin{array}{l|l} w_1^z = w_0^z + s' \cdot xy & w_0^z = \mathsf{MultKey}(\mathsf{pd}, \langle sx \rangle_0, \langle sy \rangle_0) \\ (\text{over } \mathcal{R}) & w_1^z = \mathsf{MultTag}(\mathsf{pd}, \langle sx \rangle_1, \langle sy \rangle_1, x, y) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

$$\Pr\left[\begin{array}{l|l} w_1^z = w_0^z + s \cdot (x + y) & w_0^z = \mathsf{AddKey}(\mathsf{pd}, \langle sx \rangle_0, \langle sy \rangle_0) \\ (\text{over } \mathcal{R}) & w_1^z = \mathsf{AddTag}(\mathsf{pd}, \langle sx \rangle_1, \langle sy \rangle_1, x, y) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of pd*, which is computed (over* $\mathcal{R}_q$*) as follows:*

$$\begin{aligned} \mathsf{pp} &:= (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}}) \\ a &\leftarrow \mathcal{R}_q,\ e_1, e_2 \leftarrow \mathcal{D}_{\mathsf{err}},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda, \\ \mathsf{pd} &:= (\mathsf{pp}, \mathsf{seed}, a,\ s \cdot a + e_1,\ s^2 \cdot a + e_2 - s' \cdot \Delta). \end{aligned}$$

We write $\mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}, s, s')$ *to denote public data* pd *computed as above with freshly sampled* a*,* e_1, e_2*, and* seed*.*

Lemma 7.12 (aHMAC Circuit Evaluation under Lattices)**.** *Under the same setting as Lemma 7.11, assuming the existence of a PRG, there exists a pair of efficient deterministic algorithms:* $\mathsf{EvalKey}, \mathsf{EvalTag}$ *with analogous syntax as Lemma 7.3.*

For every polynomial $p(\lambda)$*, there exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, secret elements* $s \in \mathcal{D}_{\mathsf{sk}}$*, arithmetic circuit* C *with* $|C| \leq p(\lambda)$*, admissible inputs* $\mathbf{x}$ *w.r.t* B*, and additive shares (over* $\mathcal{R}$*)* $\langle s\mathbf{x} \rangle_0, \langle s\mathbf{x} \rangle_1$*, the following holds:*

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_1^z = \mathbf{w}_0^z + s \cdot C(\mathbf{x}) & \mathbf{w}_0^z = \mathsf{EvalKey}(\mathsf{pd}, C, \langle s\mathbf{x} \rangle_0) \\ (\text{over } \mathcal{R}) & \mathbf{w}_1^z = \mathsf{EvalTag}(\mathsf{pd}, C, \langle s\mathbf{x} \rangle_1, \mathbf{x}) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of pd*, which is computed as* $\mathsf{pd} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}, s, s)$*.*

Lemma 7.13 (aHMAC Leveled Circuit Evaluation under Lattices)**.** *Under the same setting as Lemma 7.11, assuming the existence of a PRG, for every polynomial depth bound* $d(\lambda)$*,*

there exists a pair of efficient deterministic algorithms: $\mathsf{EvalKey}^d, \mathsf{EvalTag}^d$ *with analogous syntax as Lemma 7.4.*

For every polynomial $p(\lambda)$, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$, secret elements $\mathbf{s} \in \mathcal{D}_{\mathsf{sk}}^{d+1}$, arithmetic circuit C with $|C| \leq p(\lambda)$ and* $\mathrm{Depth}(C) < d(\lambda)$*, admissible inputs $\mathbf{x}$ w.r.t B, and additive shares (over $\mathcal{R}$) $\langle \mathbf{s}[0] \cdot \mathbf{x} \rangle_0$, $\langle \mathbf{s}[0] \cdot \mathbf{x} \rangle_1$, the following holds:*

$$\Pr\left[\begin{array}{l|l} \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s}[d] \cdot C(\mathbf{x}) & \mathbf{w}_0^z = \mathsf{EvalKey}^d(\mathsf{pd}, C, \langle \mathbf{s}[0] \cdot \mathbf{x} \rangle_0) \\ (\text{over } \mathcal{R}) & \mathbf{w}_1^z = \mathsf{EvalTag}^d(\mathsf{pd}, C, \langle \mathbf{s}[0] \cdot \mathbf{x} \rangle_1, \mathbf{x}) \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of $\mathsf{pd} = \{\mathsf{pd}^{(j)}\}_{[d]}$*, computed as*

$$\mathsf{pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[j], \mathbf{s}[j+1]).$$

The following lemma summarizes the HSS construction under lattices from [BKS19]. We refer readers to [BKS19] for more details.

Lemma 7.14 (HSS Extended Evaluation under Lattices [BKS19])**.** *Under the same setting as Lemma 7.11, assuming the existence of a PRG, there exists a pair of efficient deterministic algorithms,* $\mathsf{ExtEval}_0$*,* $\mathsf{ExtEval}_1$*, with analogous syntax to Lemma 7.5.*

For every logarithmic function $d(\lambda)$, every polynomial $p(\lambda)$, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$, Boolean circuit $C : \{0,1\}^{\ell_y} \rightarrow \{0,1\}^{\ell_x}$ with* $|C| < p(\lambda)$, $\mathrm{Depth}(C) < d(\lambda)$*, secret elements $s \in \mathcal{D}_{\mathsf{sk}}$, inputs $\mathbf{x} \in [B]^{\ell_x}$ and $\mathbf{y} \in \{0,1\}^{\ell_y}$, and additive shares (over $\mathcal{R}$) $\langle s\mathbf{x} \rangle_0, \langle s\mathbf{x} \rangle_1, \langle \mathbf{x} \rangle_0, \langle \mathbf{x} \rangle_1$, the following holds:*

$$\Pr\left[\begin{array}{l|r} w_1^z = w_0^z + sz & w_b^z, v_b^z = \mathsf{ExtEval}_b(\mathsf{pd}_{\mathbf{y}}, C, \langle s\mathbf{x} \rangle_b, \langle \mathbf{x} \rangle_b) \\ v_1^z = v_0^z + z & \\ (\text{over } \mathcal{R}) & z := \mathsf{InnerPord}(\mathbf{x}, C(\mathbf{y}))\ (\text{over } \mathbb{Z}). \end{array}\right] > 1 - \mathrm{negl}(\lambda),$$

over the randomness of $\mathsf{pd}_{\mathbf{y}}$*, which is computed (over* $\mathcal{R}_q$*) as follows.*

$$\mathbf{a} \leftarrow \mathcal{R}_q^{\ell_y},\ r_1, r_2 \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathbf{e}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_1', \mathbf{e}_2' \leftarrow \mathcal{D}_{\mathsf{err}}^{\ell_y}$$

$$\mathbf{b} := s \cdot \mathbf{a} + \mathbf{e}, \quad \mathbf{c}_1 := r_1 \cdot \mathbf{a} + \mathbf{e}_1 + \mathbf{y} \cdot \Delta, \qquad \mathbf{c}_1' := r_1 \cdot \mathbf{b} + \mathbf{e}_1',$$

$$\mathbf{c}_2 := r_2 \cdot \mathbf{a} + \mathbf{e}_2, \qquad \mathbf{c}_2' := r_2 \cdot \mathbf{b} + \mathbf{e}_2' + \mathbf{y} \cdot \Delta.$$

$$\mathsf{pd}_{\mathbf{y}} := (\mathsf{pp}, \mathsf{seed}, \mathbf{c}_1, \mathbf{c}_1', \mathbf{c}_2, \mathbf{c}_2').$$

We write $\mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}, s, \mathbf{y})$ *to denote public data* $\mathsf{pd}_{\mathbf{y}}$ *computed as above with freshly sampled* $\mathbf{a}$*,* r_1, r_2*, the errors, and* seed*.*

7.4.4 aHMAC Constructions under Lattices

We show a construction of MultKey, MultTag, AddKey and AddTag, which proves Lemma 7.2.

Construction 31 (aHMAC Gate Evaluation under Lattices)**.** The construction is with respect to the following public parameters $\mathsf{pp} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$:

- a polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ where n is a power-of-two;
- two modulus $p > B \cdot \lambda^{\omega(1)}$, and $q = p \cdot \Delta$, where $\Delta > B^2 \lambda^{\omega(1)}$, for some input bound B.
- error and secret distributions $\mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}} \subseteq \mathcal{R}$ with coefficients bounded by $\mathrm{poly}(\lambda)$.

As described in Lemma 7.2, the public data with respect to two secrets $s, s' \in \mathcal{D}_{\mathsf{sk}}$ are sampled as follows

$$a \leftarrow \mathcal{R}_q,\ e_1, e_2 \leftarrow \mathcal{D}_{\mathsf{err}},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda,$$

$$\mathsf{pd} := (\mathsf{pp}, \mathsf{seed}, a, \underbrace{s \cdot a + e_1}_{b}, \underbrace{s^2 \cdot a + e_2 - s' \cdot \Delta}_{c}).$$

$w_0^z \leftarrow \mathsf{MultKey}(\mathsf{pd}, w_0^x \in \mathcal{R}, w_0^y \in \mathcal{R})$: Read seed from pd, and expand from it pseudo-random "shifting factors" $r^x, r^y, r^z \in \mathcal{R}_p$.

1. Shift the coefficients of w_0^x, w_0^y by the random factors $r^x, r^y \in \mathcal{R}_p$, and then reduce them mod p.

$$w_0^x \leftarrow (w_0^x + r^x \bmod p), \quad w_0^y \leftarrow (w_0^y + r^y \bmod p).$$

2. Read a from pd, and compute the output w_0^z as follows.

$$w_0^z \leftarrow (\lfloor a w_0^x w_0^y / \Delta \rfloor + r^z) \bmod p.$$

$w_1^z \leftarrow \mathsf{MultTag}(\mathsf{pd}, w_1^x \in \mathcal{R}, w_1^y \in \mathcal{R}, x \in \mathbb{Z}_B, y \in \mathbb{Z}_B):$ Read seed from pd, and expand from it pseudo-random "shifting factors" $r^x, r^y, r^z \in \mathcal{R}_p$.

1. Shift the coefficients of w_1^x, w_1^y by the random factors $r^x, r^y \in \mathcal{R}_p$, and then reduce them mod p. As a result, we have $\|w_1^x\|_\infty, \|w_1^y\|_\infty < p$.

$$w_1^x \leftarrow (w_1^x + r^x \bmod p), \quad w_1^y \leftarrow (w_1^y + r^y \bmod p).$$

Note that if the input satsify $w_1^x = sx + w_0^x$ over $\mathcal{R}$, where $x \in [B]$, then it also holds, except with negligible probability, that $(w_1^x + r^x \bmod p) = sx + (w_0^x + r^x \bmod p)$ over $\mathcal{R}$, as long as $\|sx\|_\infty \ll p$. (See Lemma 2 in [BKS19].) The same holds for w_1^y.

2. Read $a, b, c \in \mathcal{R}_q$ from pd, and compute the following over $\mathcal{R}_q$:

$$d = -a \cdot w_1^x \cdot w_1^y + b \cdot (x \cdot w_1^y + y \cdot w_1^x) - c \cdot x \cdot y.$$

Assuming $w_1^x = sx + w_0^x$, and $w_1^y = sy + w_0^y$, where $x, y \in [B]$ and $xy < B$, then the above computation equals

$$\begin{aligned} d =& - a \cdot (s^2xy + sxw_0^y + syw_0^x + w_0^x w_0^y) \\ &+ (sa + e_1) \cdot (2sxy + xw_0^y + yw_0^x) - (s^2 a + e_2 - s'\Delta) \cdot xy \\ =& s'xy\Delta - aw_0^x w_0^y + \underbrace{e_1(xw_1^y + yw_1^x) + e_2 xy}_{\|error\|_\infty \le B \cdot p \cdot \mathrm{poly}(\lambda) \ll \Delta} \end{aligned}$$

3. Round the coefficients of d by Δ, and shift resulting coefficients again by the random factor $r^z \in \mathcal{R}_p$.

$$w_1^z \leftarrow (\lfloor d/\Delta \rfloor + r^z \bmod p).$$

We have shown that the error term from d is much smaller than Δ. Hence the rounding step removes it, except with negligible probability. (See Lemma 1 in [BKS19].)

$$w_1^z = \lfloor d/\Delta \rceil + r^z = s'xy + \underbrace{\lfloor a w_0^x w_0^y/\Delta \rceil + r^z}_{w_0^z} \mod p.$$

Shifting by the random factor r^z ensures that $w_1^z = s'xy + w_0^z$ over $\mathcal{R}$ holds except with negligible probability, as long as $\|s'xy\|_\infty \ll p$.

$w_0^z \leftarrow \mathsf{AddKey}(\mathsf{pd}, w_0^x, w_0^y):$ output $w_0^z = w_0^x + w_0^y$ over $\mathcal{R}$.

$w_1^z \leftarrow \mathsf{AddTag}(\mathsf{pd}, w_0^x, w_0^y, x, y):$ output $w_1^z = w_1^x + w_1^y$ over $\mathcal{R}$.

Note that assuming $w_1^x = sx + w_0^x$, and $w_1^y = sy + w_0^y$, then we have

$$w_1^z = s(x+y) + \underbrace{w_0^x + w_0^y}_{w_0^z}.$$

We have directly analyzed the correctness in the construction, and have proven Lemma 7.2. By composing the algorithms $\mathsf{MultKey}$, $\mathsf{MultTag}$, AddKey and AddTag in an analogous way to the instantiations under Paillier groups (see Section 7.4.1), we derive Lemma 7.11 and 7.12 as corollaries.

Additionally, we note that our new lattice construction implies an aHMAC scheme (and a leveled variant) as originally defined in [ILL24b]. We refer readers to [ILL24b] for the definition of an aHMAC scheme.

Theorem 7.1 (aHMAC Under Lattices)**.** *Assuming CP-RLWE (Definition 7.5) with respect to the public parameters $\mathsf{pp}^{\mathsf{Lat}}$ specified in Section 7.5.3, there exists an aHMAC scheme achieving* negl-*correctness, with an* evk *of size $\ell_z \cdot \mathrm{poly}(\lambda)$ bits.*

Alternatively, assuming P-RLWE (Definition 7.4, with respect to $\mathsf{pp}^{\mathsf{Lat}}$), there exists a leveled aHMAC scheme achieving negl-*correctness, with an* evk *of size $(\ell_z + D) \cdot \mathrm{poly}(\lambda)$ bits.*

7.5 Succinct Boolean Garbling Schemes

For a more intuitive presentation, we first show a 2PC protocol $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 7.1, 7.2, under Paillier groups) for evaluating Boolean circuits between a garbler P_G and an evaluator P_E:

- In an Init phase, the garbler P_G sends public data and input shares w.r.t. a Boolean vector $\mathbf{x} \in \{0,1\}^{\ell_x}$ to the evaluator P_E;
- In an Eval phase, the two parties jointly evaluate gates of a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$ in topological order;
- In a Final phase, the garbler P_G sends some decryption data to reveal the final output $\mathbf{z} \in \{0,1\}^{\ell_z}$ to the evaluator P_E.

We note that all messages in this protocol are from the garbler P_G to the evaluator P_E. We further divide them into two parts, each satisfying a special property.[10]

1. *Input shares w.r.t. to the vector* $\mathbf{x}$. We ensure they are decomposable, i.e., each bit in this communication depends only on a single bit of $\mathbf{x}$.

2. *Garbling materials.* These include the public data during Init, the decryption data during Final, and all communication during Eval. We ensure they are independent of the input $\mathbf{x}$.

Therefore, we can directly "compile" the above 2PC protocol into a garbling scheme as follows.

- The Garb algorithm outputs (1) key functions $\{K_x^{(i)}\}$ such that labels $\{L_x^{(i)} := K_x^{(i)}(\mathbf{x}[i])\}$ exactly equal to the input shares w.r.t. $\mathbf{x}$, and (2) a garbling $\widehat{C}$ that contains the garbling materials.
- The Eval algorithm performs all steps of P_E in the 2PC protocol to recover the final output.

The core of our construction is a sub-protocol $\mathsf{BoolGateEval}$ (Figure 7.3, and Section 7.5.1) for evaluating an arbitrary Boolean gate with up to $O(\log \lambda)$ input wires, and a single output, costing 1 bit per gate.

The sub-protocol itself stays unchanged when we instantiate the underlying primitives aHMAC and HSS under Paillier groups, prime-order groups, or lattices. The only changes under different instantiations lie in the Init phases of the main protocol, during which P_G computes public data pd differently. We describe the Init phase under Paillier groups in Figure 7.1, under lattices in Section 7.5.3, and under prime-order groups in Section 7.5.4. In summary, we obtain the following theorem.

Theorem 7.2 (Garbling $O(\log \lambda)$-ary Gates)**.** *Let $\mathcal{C}^{\mathsf{Arb}} = \{\mathcal{C}^{\mathsf{Arb}}_\lambda\}$ be the class of circuits (of unbounded size) consisting of arbitrary gates with $O(\log \lambda)$ input wires and 1 output wires.*

Assuming CP-DDH in Paillier groups or CP-RLWE with respect to the public parameters $\mathsf{pp}^{\mathsf{Lat}}$ specified in Section 7.5.3, there exists a garbling scheme for $\mathcal{C}^{\mathsf{Arb}}$ over $\mathbb{Z}_2$, where the garbling size $\widehat{C}$ for a circuit $C \in \mathcal{C}^{\mathsf{Arb}}_\lambda$ is $|\widehat{C}| \leq |C| + \mathrm{poly}(\lambda)$.

Assuming CP-DDH in prime-order groups, there exists a garbling scheme for $\mathcal{C}^{\mathsf{Arb}}$ over $\mathbb{Z}_2$ achieving the same garbling size as above, but with 1/poly correctness and privacy errors*. The errors can be made negligible assuming a variant of CP-DDH (Definition 7.7).*

The proof of Theorem 7.2 follows from Proposition 7.1, 7.3, and 7.5, which are proven in Section 7.5.1, 7.5.3, and 7.5.4 respectively. We show amplification techniques in Section 7.5.5 for removing correctness and privacy errors from prime-order group instantiations. Applying Theorem 7.2 to garbling standard Boolean circuits with binary gates gives a scheme costing 1 bit per gate.

Corollary 7.1 (Boolean Garbling)**.** *Assuming any of the assumptions in Theorem 7.2, there exists a garbling scheme for all Boolean circuits C (with binary gates) with garbling size $|\widehat{C}| \leq |C| + \mathrm{poly}(\lambda)$.*

---

[10]Without the special properties, a trivial protocol is letting the garbler P_G directly send $\mathbf{z} := C(\mathbf{x})$ to the evaluator P_E.

The scheme assuming CP-DDH in prime-order groups has 1/poly correctness and privacy errors*, which can be made negligible assuming a variant of CP-DDH (Definition 7.7).*

In the special case of layered circuits C^{Layer}, we can re-write C^{Layer} into another circuit C' in terms of general gates for $\log\log\lambda$-depth computations, with the guarantee that $|C'| < O(|C^{\mathsf{Layer}}|/\log\log\lambda)$. (See Lemma 4.12 in [BGI16].) Since each general gate depends on at most $\log\lambda$ input values, we can apply Theorem 7.2 to garble C' which yields a scheme costing $O(1/\log\log\lambda)$ bits per gate.

Corollary 7.2 (Boolean Garbling for Layered Circuits)**.** *Assuming any of the assumptions in Theorem 7.2, there exists a garbling scheme for all layered Boolean circuits C^{Layer} (with binary gates) with garbling size $|\widehat{C}^{\mathsf{Layer}}| \leq O(|C^{\mathsf{Layer}}|/\log\log\lambda) + \mathrm{poly}(\lambda)$.*

The scheme assuming CP-DDH in prime-order groups has 1/poly correctness and privacy errors*, which can be made negligible assuming a variant of CP-DDH (Definition 7.7).*

In Section 7.5.2, we describe a leveled variant of the 2PC protocol $\mathsf{LBoolCircEval}$ (Figure 7.4 , 7.5, under Paillier groups), based on a leveled variant of the core sub-protocol $\mathsf{LBoolGateEval}$ (Figure 7.6, 7.7). We describe the Init phases of this variant under lattices and prime-order groups in Section 7.5.3 and 7.5.4 respectively. In summary, we obtain the following theorem.

Theorem 7.3 (Leveled Garbling of $O(\log\lambda)$-ary Gates)**.** *Let $\mathcal{C}^{\mathsf{Arb}} = \{\mathcal{C}^{\mathsf{Arb}}_\lambda\}$ be the class of circuits (of unbounded size) consisting of arbitrary gates with $O(\log\lambda)$ input wires and* 1 *output wires.*

Assuming P-DDH and DDH in Paillier groups, or P-RLWE with respect to the public parameters $\mathsf{pp}^{\mathsf{Lat}}$ specified in Section 7.5.3, there exists a garbling scheme for $\mathcal{C}^{\mathsf{Arb}}$ over $\mathbb{Z}_2$, where the garbling size $\widehat{C}$ for a circuit $C \in \mathcal{C}^{\mathsf{Arb}}_\lambda$ is $|\widehat{C}| \leq |C| + \mathrm{Depth}(C) \cdot \mathrm{poly}(\lambda)$.

The proof of Theorem 7.3 follows from Proposition 7.2, 7.4, and 7.6, which are proven in Section 7.5.1, 7.5.3, and 7.5.4 respectively. Security amplification for the leveled variant under prime-order groups is analogous to the non-leveled variant as described in Section 7.5.5. We obtain two corollaries analogous to the non-leveled case.

Corollary 7.3 (Leveled Boolean Garbling)**.** *Assuming any of the assumptions in Theorem 7.3, there exists a garbling scheme for all Boolean circuits C (with binary gates) with garbling size* $|\widehat{C}| \leq |C| + \mathrm{Depth}(C) \cdot \mathrm{poly}(\lambda)$.

Corollary 7.4 (Leveled Boolean Garbling for Layered Circuits)**.** *Assuming any of the assumptions in Theorem 7.2, there exists a garbling scheme for all layered Boolean circuits C^{Layer} (with binary gates) with garbling size* $|\widehat{C}^{\mathsf{Layer}}| \leq O(|C^{\mathsf{Layer}}|/\log\log\lambda) + \mathrm{Depth}(C) \cdot \mathrm{poly}(\lambda)$.

7.5.1 Sub-protocol for Garbling $O(\log\lambda)$-ary Boolean Gates

In the sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$, both parties P_G, P_E hold public data $\mathsf{pd} = (\mathsf{aHMAC.pd}, \mathsf{HSS.pd}_{\mathsf{sk}})$ prepared in the Init phase of the main protocol (Figure 7.1), and jointly hold additive shares $\langle s\overline{\mathbf{x}}\rangle$, where s is a global secret exponent sampled during Init, and $\overline{\mathbf{x}}$ represent masked input values to the gate $\mathsf{g} \in C$ (with masks derived from sk). Additionally, P_E holds $\overline{\mathbf{x}}$ in the clear.

$$\text{Inputs: } (P_G : \mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_0), \quad (P_E : \mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}).$$

Their goal is to jointly obtain shares of $\langle s \cdot \overline{z}\rangle$, where $\overline{z}$ is the masked output of g. Additionally, P_E should hold $\overline{z}$ in the clear.

$$\text{Outputs: } (P_G : \langle s\overline{z}\rangle_0), \quad (P_E : \langle s\overline{z}\rangle_1, \overline{z}).$$

Their first steps are local aHMAC and HSS evaluations by both parties over the input shares $\langle s\overline{\mathbf{x}}\rangle$. For now, assume there are arithmetic circuits $C_{\mathbf{v}}$ (Fact 1) and Boolean circuits $C_{\mathsf{g},\mathbf{v}}$ (Equation 7.6) which satsify

$$\overline{z} = \sum_{\mathbf{v}\in\{0,1\}^{\ell_x}} C_{\mathbf{v}}(\overline{\mathbf{x}}) \cdot C_{\mathsf{g},\mathbf{v}}(\mathsf{sk}) \text{ over } \mathbb{Z}. \tag{7.5}$$

The parties apply aHMAC to locally evaluate $C_{\mathbf{v}}$ over $\langle s\overline{\mathbf{x}}\rangle$ to obtain shares $\langle s \cdot C(\overline{\mathbf{x}})\rangle$. They also locally hold additive shares of $\langle C(\overline{\mathbf{x}})\rangle$ as P_E can compute $C(\overline{\mathbf{x}})$ on its own.

$$P_G : \langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0 \leftarrow \mathsf{EvalKey}(\mathsf{aHMAC.pd}, C_{\mathbf{v}}, \langle s\overline{\mathbf{x}}\rangle_0),$$
$$P_E : \langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{aHMAC.pd}, C_{\mathbf{v}}, \langle s\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}).$$

The parties next apply HSS to locally evaluate $C_{\mathsf{g},\mathbf{v}}$ over the public data $\mathsf{HSS.pd}_{\mathsf{sk}}$ and additionally "multiply" the results with $C(\overline{\mathbf{x}})$ to obtain shares $\langle s\overline{z}\rangle$ and $\langle \overline{z}\rangle$.

$$P_G : (\langle s\overline{z}\rangle_0, \langle \overline{z}\rangle_0) \leftarrow \mathsf{ExtEval}_0(\mathsf{HSS.pd}_{\mathsf{sk}}, (\ldots, C_{\mathsf{g},\mathbf{v}}, \ldots), \left\{\langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0, \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0\right\}_{\mathbf{v}})$$
$$P_E : (\langle s\overline{z}\rangle_1, \langle \overline{z}\rangle_1) \leftarrow \mathsf{ExtEval}_1(\mathsf{HSS.pd}_{\mathsf{sk}}, (\ldots, C_{\mathsf{g},\mathbf{v}}, \ldots), \left\{\langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1, \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1\right\}_{\mathbf{v}}).$$

In summary, the parties now hold shares $\langle s\overline{z}\rangle$, $\langle \overline{z}\rangle$ through local computations. In the last step, P_G sends its share $\langle \overline{z}\rangle \bmod 2$ to P_E, who then recovers $\overline{z}$.

It remains to specify the arithmetic circuits $C_{\mathbf{v}}$ and Boolean circuits $C_{\mathsf{g},\mathbf{v}}$ satisfying Equation 7.5. For this, we let $C_{\mathbf{v}}$ implement the indicator polynomial $p_{\mathbf{v}}$ specified as follows.

Fact 1 (Indicator Polynomial)**.** For every positive integer $\ell_x \in \mathbb{N}$, every vector $\mathbf{v} \in \{0,1\}^{\ell_x}$, there exists a polynomial $p_{\mathbf{v}}$ (over $\mathbb{Z}$) such that

$$p_{\mathbf{v}}(\mathbf{x}) = \begin{cases} 1 & \text{for } \mathbf{x} = \mathbf{v} \\ 0 & \text{for } \mathbf{x} \in \{0,1\}^{\ell_x}, \mathbf{x} \neq \mathbf{v}. \end{cases}$$

Furthermore, $p_{\mathbf{v}}$ can be implemented by an arithmetic circuit $C_{\mathbf{v}} : \mathbb{Z}^{\ell_x} \to \mathbb{Z}$ of size $|C_{\mathbf{v}}| \leq O(\ell_x)$, $\mathrm{Depth}(C_{\mathbf{v}}) \leq O(\log \ell_x)$ and such that all Boolean inputs $\mathbf{x} \in \{0,1\}^{\ell_x}$ are admissible w.r.t. the bound $B = 2$.

We define $C_{\mathsf{g},\mathbf{v}}(\mathsf{sk})$ to compute a masked output $\overline{z}$ pretending $\overline{\mathbf{x}} = \mathbf{v}$.

$$C_{\mathsf{g},\mathbf{v}}(\mathsf{sk}) = \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \oplus \mathsf{g}\big(\mathbf{v} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(\mathsf{g}))\big). \tag{7.6}$$

Effectively, Equation 7.5 computes all possible evaluation results of $\overline{z}$ via $C_{\mathsf{g},\mathbf{v}}(\mathsf{sk})$, and selects the correct one via $C_{\mathbf{v}}(\overline{\mathbf{x}})$, which only equals 1 when $\mathbf{v} = \overline{\mathbf{x}}$. Assuming PRF is in NC1 and $\ell_x = O(\log \lambda)$, $C_{\mathsf{g},\mathbf{v}}$ can indeed be evaluated using HSS.

We summarize the sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ in Figure 7.3. Note that in each invocation, the only communication is *one bit* $b := \langle \overline{z} \rangle_0$ sent from the garbler P_G to the evaluator P_E. We summarize its correctness and security in the following lemmas.

Lemma 7.15 (Correctness of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under Paillier Groups)**.** *Let* $\ell(\lambda) \leq O(\log \lambda)$ *be a bound on input length. There exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$, *every Boolean circuit* C *with a gate* g *of* $\ell_x \leq \ell(\lambda)$ *inputs, every masked input* $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, $\mathsf{pp} = (N, \zeta)$ *in the support of* $\mathsf{Pai.Gen}(1^\lambda, 1^2)$, *secret exponent* $s \in [N]$, *additive shares (over* $\mathbb{Z}$*)* $\langle s\overline{\mathbf{x}} \rangle_0, \langle s\overline{\mathbf{x}} \rangle_1$, *and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$, *the following holds:*

$$\Pr\left[\begin{array}{c|l} & \mathsf{pd} \textit{ sampled per Equation 7.3}, \\ w_1^z = w_0^z + s\overline{z}, & (P_G : w_0^z), (P_E : w_1^z, \overline{z}) \\ z = \mathsf{g}(\mathbf{x}) & \quad \leftarrow \mathsf{BoolGateEval}^{C,\mathsf{g}}\big((P_G : \mathsf{pd}, \langle s\overline{\mathbf{x}} \rangle_0), (P_E : \mathsf{pd}, \langle s\overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}})\big) \\ & z := \overline{z} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)),\ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)) \end{array}\right]$$
$$\geq 1 - \mathrm{negl}(\lambda).$$

Proof. The correctness of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ follows from that of $\mathsf{EvalKey}$, $\mathsf{EvalTag}$ and that of $\mathsf{ExtEval}_b$. □

Lemma 7.16 (Security of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under Paillier Groups)**.** *Under the same setting as Lemma 7.15, there exists an efficient simulator* Sim *that, given the masked output* $\overline{z}$, statistically *simulates* P_G*'s message in the sub-protocol* $\mathsf{BoolGateEval}^{C,\mathsf{g}}$.

More precisely, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$, *every Boolean circuit* C *with a gate* g *of* $\ell_x \leq \ell(\lambda)$ *inputs, every masked input* $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, $\mathsf{pp} = (N, \zeta)$ *in the support of* $\mathsf{Pai.Gen}(1^\lambda, 1^2)$, *secret exponent* $s \in [N]$, *additive shares (over* $\mathbb{Z}$*)* $\langle s\overline{\mathbf{x}} \rangle_0, \langle s\overline{\mathbf{x}} \rangle_1$, *and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$, *the following holds.*

$$\begin{array}{r|l} & \mathsf{pd} \textit{ sampled per Equation 7.3}, \\ \mathsf{SD}\big(\mathsf{msg}_G(\mathsf{pd}, \langle s\overline{\mathbf{x}} \rangle_0), & \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)), \\ \mathsf{Sim}(\mathsf{pd}, \langle s\overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}}, \overline{z})\big) \leq \mathrm{negl}(\lambda), & \overline{z} := \mathsf{g}(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \end{array}$$

where $\mathsf{msg}_G(\mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_0)$ *denotes* P_G*'s message to* P_E *in* $\mathsf{BoolGateEval}^{C,\mathsf{g}}$.

Proof. The simulator computes $\langle \overline{z}\rangle_1$ following exactly P_E's steps, and then simulates $\langle \overline{z}\rangle_0$ mod 2 (which is the message from P_G) as $\overline{z} - \langle \overline{z}\rangle_1$ mod 2. □

Using the correctness and security of the core sub-protocol, BoolGateEval under Paillier groups, we can now prove those of our garbling scheme under Paillier groups (compiled from the 2PC protocol BoolCircEval).

Proposition 7.1 (Garbling of $O(\log\lambda)$-ary Gates under Paillier Groups)**.** *Assuming CP-DDH in Paillier groups, the garbling scheme compiled from the protocol* $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ *(Figure 7.1, 7.2) is correct and secure.*

Proof of Proposition 7.1. The correctness of the protocol follows from that of BoolGateEval (Lemma 7.15). Hence the correctness of the compiled garbling scheme follows. We focus now on proving security.

First, we recap the compiled garbling scheme. Given a circuit C, the garbler proceeds as follows.

- Sample Paillier public parameters $\mathsf{pp} = (N, \zeta)$, a secret exponent $s \leftarrow [N]$, a PRF key $\mathsf{sk} \leftarrow \{0,1\}^\lambda$, and compute public data pd per Equation 7.3:

$$\begin{aligned} &\mathsf{pp} = (N, 2) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^2)\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ \mathsf{seed} \leftarrow \{0,1\}^\lambda, \\ &\quad s \leftarrow [N],\ \mathbf{r} \leftarrow [N]^{\lceil \log N \rceil}, \quad \mathbf{r}', \mathbf{r}'' \leftarrow [N]^\lambda, \\ &\mathsf{pd} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{r}}, g^{\mathbf{r}s}, g^{\mathbf{r}s^2} \odot (1+N)^{\mathsf{Bits}(s)}, \\ &\qquad g^{\mathbf{r}'}, g^{\mathbf{r}'s} \odot (1+N)^{\mathsf{Bits}(\mathsf{sk})}, g^{\mathbf{r}''s}, g^{\mathbf{r}''} \odot (1+N)^{\mathsf{Bits}(\mathsf{sk})}). \end{aligned} \tag{7.8}$$

  where $\odot$ denotes component-wise multiplication between two vectors.
- For every input wire i, sample a pad $k^{(i)} \leftarrow [N \cdot \lambda^{\omega(1)}]$, and define the key function $K^{(i)}$ as follows:

$$K^{(i)}(b) := (\overline{b}, s \cdot \overline{b} + k^{(i)}), \qquad \text{where } \overline{b} := b \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(C)[i]).$$

- For every gate $\mathsf{g} \in C$, in the topological order, where a pad $k^{(i)}$ is defined for all $i \in \mathsf{InWires}(g)$, follow the subprotocol $\mathsf{BoolGateEval}$ as P_G with $(\mathsf{pd}, \{k^{(i)}\}_{\mathsf{InWires}(g)})$ as inputs, and compute its message b_{g}. The output of the subprotocol defines a pad $k^{(j)}$ for the output wire $j = \mathsf{OutWire}(\mathsf{g})$.
- Compute the masks on output wires $\mathbf{o} = \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWires}(C))$.
- Output the garbling $\widehat{C} = (\mathsf{pd}, \{b_{\mathsf{g}}\}_C, \mathbf{o})$ consisting of the public data, the bit b_{g} for every $\mathsf{g} \in C$, and the masks on the output wires. Output the key functions $\{K^{(i)}\}$ as defined above.

Next, we describe the simulator Sim required by Definition 2.1. It takes the circuit C and the evaluation results $\mathbf{z}$ as input, and simulates the garbling $\widehat{C}$ and input labels $\{L^{(i)}\}$ as follows.

- Sample Paillier public parameter $\mathsf{pp} = (N, \zeta)$ honestly, and sample random elements as public data $\widetilde{\mathsf{pd}}$:

$$\begin{aligned} \mathsf{pp} = (N, 2) &\leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^2)\ g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ \mathsf{seed} \leftarrow \{0,1\}^\lambda, \\ s \leftarrow [N],\ \mathbf{a}, \mathbf{b}, \mathbf{c} &\leftarrow [N^3]^{\lceil \log N \rceil},\ \mathbf{a}', \mathbf{b}', \mathbf{a}'', \mathbf{b}'' \leftarrow [N^3]^\lambda, \\ \widetilde{\mathsf{pd}} &= (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}}, g^{\mathbf{a}'}, g^{\mathbf{b}'}, g^{\mathbf{a}''}, g^{\mathbf{b}''}). \end{aligned} \tag{7.9}$$

- For every input wire i, sample a label $\tilde{l}^{(i)} \leftarrow [N \cdot \lambda^\omega]$, and a masked bit $\tilde{x}^{(i)} \leftarrow \{0,1\}$. The simulated input labels are $\widetilde{L}^{(i)} = (\tilde{x}^{(i)}, \tilde{l}^{(i)})$. Further sample a masked bit $\tilde{x}^{(j)} \leftarrow \{0,1\}$ for every wires j in C, including the output wires.
- For every gate $g \in C$, in the topological order, where a label $\tilde{l}^{(i)}$ and a masked bit $\tilde{x}^{(i)}$ are defined for all $i \in \mathsf{InWires}(\mathsf{g})$, follow the the subprotocol $\mathsf{BoolGateEval}$ as P_E except the last step (See Figure 7.3) with $(\widetilde{\mathsf{pd}}, \{\tilde{l}^{(i)}, \tilde{x}^{(i)}\})$ as inputs. The computation results (corresponding to $\langle s\bar{z}\rangle_1$ in Figure 7.3) define a label $\tilde{l}^{(j)}$ for the output wire $j = \mathsf{OutWire}(\mathsf{g})$. Then run the simulator guaranteed by the security of the subprotocol (Lemma 7.16):

$$\tilde{b}_{\mathsf{g}} \leftarrow \mathsf{Sim}'(\widetilde{\mathsf{pd}}, \{\tilde{l}^{(i)}, \tilde{x}^{(i)}\}_{\mathsf{InWires}(\mathsf{g})}, \tilde{z}^{(j)}),$$

where $\tilde{z}^{(i)}$ is the masked bit assigned to wire $j = \mathsf{OutWire}(\mathsf{g})$.

- Let $\widetilde{\mathbf{z}}$ be the masked bits assigned to output wires $\mathsf{OutWires}(C)$, simulate the masks on output wires as $\tilde{\mathbf{o}} = \widetilde{\mathbf{z}} \oplus \mathbf{z}$.
- Output the simulated garbling $\widetilde{C} = (\widetilde{\mathsf{pd}}, \{\tilde{b}_{\mathsf{g}}\}, \tilde{\mathbf{o}})$, and the simulated input labels $\{\widetilde{L}^{(i)}\}$.

We now show a series of hybrid experiments, where Hyb_0 describe the honest distribution of $\widehat{C}$ and $\{L^{(i)} = K^{(i)}(\mathbf{x}[i])\}$ for some input $\mathbf{x}$, and Hyb_5 describe the simulated distribution $\widetilde{C}$ and $\{\widetilde{L}^{(i)}\}$.

Hyb_0 The real distribution of $\widehat{C}$ and $\{L^{(i)} = K^{(i)}(\mathbf{x}[i])\}$ computed according to the garbling scheme. (See the recap earlier.)

Hyb_1 In this hybrid, instead of computing the bits $\{b_{\mathsf{g}}\}$ as the garbler's message following the subprotocol $\mathsf{BoolGateEval}$, simulate them using the simulator Sim' guaranteed by the security of the subprotocol:

- First compute the correct wire value $x^{(j)}$ on each wire j in C, and then the masked bit $\overline{x}^{(j)} = x^{(j)} \oplus \mathsf{PRF}(\mathsf{sk}, j)$.
- Let $l^{(i)} = k^{(i)} + s\overline{x}^{(i)}$ be the labels on input wires. For every gate $\mathsf{g} \in C$, in topological order, follow the subprotocol as P_E (except the last step) with $(\mathsf{pd}, \{l^{(i)}, \overline{x}^{(i)}\}_{\mathsf{InWires(g)}})$ as inputs to compute a label $l^{(j)}$ for the output wire $j = \mathsf{OutWire}(\mathsf{g})$. Then run the simulator

$$\tilde{b}_{\mathsf{g}} \leftarrow \mathsf{Sim}'(\mathsf{pd}, \{\tilde{l}^{(i)}, \tilde{x}^{(i)}\}, \overline{z}^{(j)}),$$

where $\overline{z}^{(j)}$ is the masked bit on wire j.

By the correctness and security of $\mathsf{BoolGateEval}$ (Lemma 7.15 and 7.16), the simulated bits b_{g} are statistically close to the correctly computed ones in Hyb_0. Hence we have $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$. Note that in Hyb_1, the pads $k^{(i)}$ sampled for the input wires are not used for computing the bits b_{g} anymore.

Hyb_2 In this hybrid, instead of computing the input labels as $l^{(i)} = \overline{x}^{(i)}s + k^{(i)}$, directly sample $\tilde{l}^{(i)} \leftarrow [N \cdot \lambda^{\omega(1)}]$.

The two ways of sample $l^{(i)}$ and $\tilde{l}^{(i)}$ are statistically close, hence we have $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_1$. Note that in Hyb_2, the secret exponent s within pd is not used for computing the input labels or anywhere else.

Hyb_3 In this hybrid, instead of computing the masks on the output wires directly using the PRF, $\mathbf{o} = \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWires}(C))$, compute it as $\widetilde{\mathbf{o}} = \overline{\mathbf{z}} \oplus \mathbf{z}$, where $\overline{\mathbf{z}}$ are the masked wire values on the output wires. As the two ways of computing $\mathbf{o}$ and $\widetilde{\mathbf{o}}$ are equivalent, we have $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_2$.

Hyb_4 In this hybrid, instead of computing the public data pd as in Equation 7.8, simulate it with random elements as in Equation 7.9.

We claim the two ways of sampling pd are computationally indistinguishable (Claim 2). Hence we have $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_3$. Note that in Hyb_4, the PRF key sk are only used for computing masked wire values $\overline{x}^{(i)} = x^{(i)} \oplus \mathsf{PRF}(\mathsf{sk}, i)$, and in particular not the public data anymore.

Hyb_5 In this hybrid, instead of computing the masked wire values $\overline{x}^{(i)}$ using a PRF, directly sample them at random $\tilde{x}^{(i)} \leftarrow \{0, 1\}$.

By the security of PRF, we have $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_4$. Note that Hyb_5 computes exactly the simulated distribution of $\widetilde{C}$ and $\{\widetilde{L}^{(i)}\}$.

By a hybrid argument, we conclude $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_5$. It remains to prove the following claim.

Claim 2. *For all* $\mathsf{sk} \in \{0, 1\}^\lambda$, *the distribution of* pd *defined by Equation 7.8 and* $\widetilde{\mathsf{pd}}$ *by Equation 7.9 are computationally indistinguishable.*

Proof. We show a series of hybrid that transitions from the distribution of Equation 7.8 to Equation 7.9.

Hyb'_0 This is the distribution of Equation 7.8.

Hyb'_1 In this hybrid, instead of computing the aHMAC public data as

$$g^{\mathbf{r}}, g^{\mathbf{r}s}, g^{\mathbf{r}s^2} \odot (1+N)^{\mathsf{Bits}(s)}$$

where $\mathbf{r}, s$ are random exponents from $[N]$, simulate them as random elements

$$g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}},$$

where $\mathbf{a}, \mathbf{b}, \mathbf{c}$ are random exponents from $[N^3]$. By CP-DDH in Paillier groups (Definition 7.3), we have $\mathsf{Hyb}'_1 \approx_c \mathsf{Hyb}'_0$.

Hyb'_2 In this hybrid, instead of computing the HSS public data as

$$g^{\mathbf{r}'}, g^{\mathbf{r}'s} \odot (1+N)^{\mathsf{Bits}(\mathsf{sk})}, g^{\mathbf{r}''s}, g^{\mathbf{r}''} \odot (1+N)^{\mathsf{Bits}(\mathsf{sk})},$$

where $\mathbf{r}', \mathbf{r}'', s$ are random exponents from $[N]$, simulate them as

$$g^{\mathbf{a}'}, g^{\mathbf{b}'} \odot (1+N)^{\mathsf{Bits}(\mathsf{sk})}, g^{\mathbf{a}''}, g^{\mathbf{b}''} \odot (1+N)^{\mathsf{Bits}(\mathsf{sk})},$$

where $\mathbf{a}', \mathbf{b}', \mathbf{a}'', \mathbf{b}''$ are random exponents from $[N^3]$. By DDH (Definition 6.7, which is implied by CP-DDH) in Paillier groups, we have $\mathsf{Hyb}'_2 \approx_c \mathsf{Hyb}'_1$.

Hyb'_3 In this hybrid, instead of multiplying the term $(1+N)^{\mathsf{Bits}(sk)}$ to random elements $g^{\mathbf{b}'}$ and $g^{\mathbf{b}''}$ as above, directly compute HSS public data at random

$$g^{\mathbf{a}'}, g^{\mathbf{b}'}, g^{\mathbf{a}''}, g^{\mathbf{b}''}.$$

By Lemma 2.1, the element g sampled by $\mathsf{Pai.Samp}(\mathsf{pp})$ has the guarantee that $\langle g \rangle$ contains the subgroup generated by $(1+N)$. Therefore, $g^{\mathbf{b}'}$ with a random exponent $\mathbf{b}'$ from $[N^3]$ perfectly hides the multiplicative factor $(1+N)^{\mathsf{Bits}(\mathsf{sk})}$. We have $\mathsf{Hyb}'_3 \equiv \mathsf{Hyb}'_2$.

Note that Hyb'_3 computes exactly the distribution of Equation 7.9.

By a hybrid argument, we conclude that $\mathsf{Hyb}'_0 \approx_c \mathsf{Hyb}'_3$, which proves the claim. $\square$

7.5.2 A Leveled Variant under Paillier Groups

Compared to the non-leveled protocol, the main changes in the leveled variant are (1) in the core sub-protocol $\mathsf{LBoolGateEval}$ both parties now run *leveled* aHMAC local evaluations, and (2) the leveled aHMAC and (normal) HSS instances no longer rely on common secret exponents.

The two differences together allow us to avoid circular security arguments in this variant. On the other hand, they require much larger public data, of size linear in the circuit depth. We now explain the differences in more detail.

- During Init, the garbler prepares appropriate public data (Equation 7.10) for Depth(C) instances of leveled aHMAC and HSS to support the following two types of local evaluations. Assume P_G, P_E jointly holds additive shares $\langle s^{(t)} \cdot \overline{\mathbf{x}} \rangle$, and P_E additionally holds $\overline{\mathbf{x}}$.

$$\text{Inputs: } (P_G : \mathsf{pd}, \langle s^{(t)} \cdot \overline{\mathbf{x}} \rangle_0), \quad (P_E : \mathsf{pd}, \langle s^{(t)} \cdot \overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}}).$$

  1. In the first type, they locally run leveled aHMAC evaluations on the input shares $\langle s^{(t)} \cdot \overline{\mathbf{x}} \rangle$ over arithmetic circuits $C_{\mathbf{v}}$ (of depth d_{Ind}; Fact 1) to obtain additive shares $\langle \mathbf{k}[t] \cdot C_{\mathbf{v}}(\overline{\mathbf{x}}) \rangle$, where $\mathbf{k}[t]$ is an independent secret exponent in the t-th HSS instance.

$$\text{Via aHMAC Eval: } (P_G : \langle \mathbf{k}[t] \cdot C_{\mathbf{v}}(\overline{\mathbf{x}}) \rangle_0), \quad (P_E : \langle \mathbf{k}[t] \cdot C_{\mathbf{v}}(\overline{\mathbf{x}}) \rangle_1).$$

  They then locally run HSS to evaluate $C^{(i)}_{\mathsf{g},\mathbf{v}}$ over public data $\mathsf{HSS.pd}_{\mathsf{sk},s^{(t+1)}}$ where $s^{(t+1)}$ is an independent secret exponent in the $(t+1)$-th leveled aHMAC instance, and $C^{(i)}_{\mathsf{g},\mathbf{v}}(\mathsf{sk}, s^{(t+1)})$ is defined to compute $C_{\mathsf{g},\mathbf{v}}(\mathsf{sk}) \cdot \mathsf{Bits}(s^{(t+1)})[i]$ (see Equation 7.6). The result can be additionally "multiplied" by $C_{\mathbf{v}}(\overline{\mathbf{x}})$ via HSS extended evaluation. In the end, they jointly hold additive shares of $\langle \mathsf{Bits}(s^{(t+1)})[i] \cdot \overline{z} \rangle$ and $\langle \overline{z} \rangle$.

$$\begin{aligned} \text{Via HSS Eval: } & (P_G : \langle \mathsf{Bits}(s^{(t+1)})[i] \cdot \overline{z} \rangle_0, \langle \overline{z} \rangle_0), \\ & (P_E : \langle \mathsf{Bits}(s^{(t+1)})[i] \cdot \overline{z} \rangle_1, \langle \overline{z} \rangle_1). \end{aligned}$$

Finally, they locally linearly combine the shares $\langle \mathsf{Bits}(s^{(t+1)})[i] \cdot \overline{z} \rangle$ into shares $\langle s^{(t+1)} \cdot \overline{z} \rangle$.

$$\text{Via Linear Comb.: } (P_G : \langle s^{(t+1)} \cdot \overline{z} \rangle_0), \quad (P_E : \langle s^{(t+1)} \cdot \overline{z} \rangle_1).$$

2. In the second type, they locally run leveled aHMAC evaluations on the input shares $\langle s^{(t)} \cdot \overline{\mathbf{x}} \rangle$ over the identity arithmetic circuit C_{id} (of appropriate depth) to obtain additive shares $\langle s^{(t')} \cdot \overline{\mathbf{x}} \rangle$, where $t' > t$, and $s^{(t')}$ is an independent secret exponent in the t'-th leveled aHMAC instance.

$$\text{Via aHMAC Eval: } (P_G : \langle s^{(t')} \cdot \overline{\mathbf{x}} \rangle_0), \quad (P_E : \langle s^{(t')} \cdot \overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}}).$$

- During Eval, for every gate $\mathsf{g} \in C$ of depth t, assume P_G, P_E jointly holds additive shares $\langle s^{(t)} \cdot \overline{\mathbf{x}} \rangle$, and P_E additionally holds $\overline{\mathbf{x}}$.

  First, they apply type-1 local evaluations to obtain additive shares of $\langle s^{(t+1)} \overline{z} \rangle, \langle \overline{z} \rangle$. Next, P_G sends his share $\langle \overline{z} \rangle$ mod 2 to P_E who then recovers $\overline{z}$ mod 2. Finally, for every gate $\mathsf{g}' \in C$ of depth $t' > t$ that uses $\overline{z}$ as an input, they apply type-2 local evaluations to obtain additive shares of $\langle s^{(t')} \overline{z} \rangle$.

Implementing the leveled variant requires careful book-keeping of the public data. We give full details of the leveled variant of our 2PC protocol under Paillier groups in Figure 7.4 , 7.5, and the leveled variant of the core sub-protocol in Figure 7.6, 7.7.

Note that the total communication from P_G to P_E consists of *one bit* per invocation of the sub-protocol $\mathsf{LBoolGateEval}$, plus public data of size $\text{Depth}(C) \cdot \text{poly}(\lambda)$, assuming all gates in C has fan-in $O(\log(\lambda))$. We summarize the correctness and security of the sub-protocol $\mathsf{LBoolGateEval}$ in the following lemmas.

Lemma 7.17 (Correctness of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under Paillier Groups)**.** *Let $\ell(\lambda) \leq O(\log \lambda)$ be a bound on input length, and $d_{\text{Ind}} = O(\log \log \lambda)$ be the depth of the indicator arithmetic circuit over ℓ inputs (Fact 1).*

There exists a negligible function $\text{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C (of depth d_C) with a gate g of $\ell_x \leq \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$,* $\mathsf{pp} = (N, \zeta)$

in the support of $\mathsf{Pai.Gen}(1^\lambda, 1^2)$*, secret exponent* $\mathbf{s} \in [N]^{d_C \cdot d_{\mathrm{Ind}}+1}$*, additive shares (over* $\mathbb{Z}$*)* $\langle s^{(t)}\overline{\mathbf{x}}\rangle_0, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1$*, and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$*, the following holds: (where we use the shorthand* $s^{(t)} = \mathbf{s}[t \cdot d_{\mathrm{Ind}}]$*)*

$$\Pr\left[\begin{array}{c|l} & \mathsf{pd} \textit{ sampled per Equation 7.10}, \\ w_1^z = w_0^z + s^{(t+1)}\overline{z}, & (P_G : w_0^z), (P_E : w_1^z, \overline{z}) \leftarrow \mathsf{LBoolGateEval}^{C,\mathsf{g}} \\ z = \mathsf{g}(\mathbf{x}) & \quad \big((P_G : \mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_0), (P_E : \mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}})\big) \\ & z := \overline{z} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)), \\ & \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)) \end{array}\right]$$
$$\geq 1 - \mathrm{negl}(\lambda).$$

Proof. The correctness of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ follows from that of the leveled variants of $\mathsf{EvalKey}$, $\mathsf{EvalTag}$ (Lemma 7.4) and that of $\mathsf{ExtEval}_b$ (Lemma 7.5). □

Lemma 7.18 (Security of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under Pailler Groups)**.** *Under the same setting as Lemma 7.17, there exists an efficient simulator* Sim *that, given the masked output* $\overline{z}$*,* statistically *simulates* P_G*'s message in the sub-protocol* $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$.

More precisely, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, every Boolean circuit* C *with a gate* g *of* $\ell_x \leq \ell(\lambda)$ *inputs, every masked input* $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$*,* $\mathsf{pp} = (N, \zeta)$ *in the support of* $\mathsf{Pai.Gen}(1^\lambda, 1^2)$*, secret exponents* $\mathbf{s} \in [N]^{d_C \cdot d_{\mathrm{Ind}}+1}$*, additive shares (over* $\mathbb{Z}$*)* $\langle s^{(t)}\overline{\mathbf{x}}\rangle_0, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1$*, and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$*, the following holds.*

$$\left.\begin{array}{r} \mathsf{SD}\big(\mathsf{msg}_G(\mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_0), \\ \mathsf{Sim}(\mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}, \overline{z})\big) \leq \mathrm{negl}(\lambda), \end{array}\right| \begin{array}{l} \mathsf{pd} \textit{ sampled per Equation 7.10}, \\ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)), \\ \overline{z} := \mathsf{g}(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \end{array}$$

where $\mathsf{msg}_G(\mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_0)$ *denotes* P_G*'s message to* P_E *in* $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$.

Proof. Analogous to the proof of Lemma 7.16. □

Using the correctness and security of $\mathsf{LBoolGateEval}$ under Paillier groups, we can now prove those of our leveled garbling scheme under Paillier groups (compiled from the 2PC protocol $\mathsf{LBoolCircEval}$).

Proposition 7.2 (Leveled Garbling of $O(\log \lambda)$-ary Gates under Paillier Groups)**.** *Assuming P-DDH and DDH in Paillier groups, the garbling scheme compiled from the protocol* $\mathsf{LBoolCircEval}^{C,\mathsf{Pai}}$ *(Figure 7.4, 7.5) is correct and secure.*

Proof of Proposition 7.2. The correctness of the protocol follows from that of $\mathsf{LBoolGateEval}$ (Lemma 7.17). Hence the correctness of the compiled garbling scheme follows.

The security proof follows the same arguments as those for Proposition 7.1, except the public data pd are computed and simulated differently. In the honest protocol, they are computed as follows according to Equation 7.10, with respect to a PRF key $\mathsf{sk} \in \{0,1\}^\lambda$:

$$
\begin{aligned}
&\mathsf{pp} = (N,2) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^2),\ \mathbf{s} \leftarrow [N]^{d+1},\ \mathbf{k} \leftarrow [N]^{d_C}, \\
&\quad /\!/ \text{ For short, write } s^{(t)} = \mathbf{s}[t \cdot d_{\mathrm{Ind}}],\ s^{(t)}_{\mathsf{end}} = \mathbf{s}[(t+1)\cdot d_{\mathrm{Ind}} - 1]. \\
&\quad \forall j \in [d], \quad \mathsf{aHMAC.pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[j], \mathbf{s}[j+1]), \\
&\quad \forall t \in [d_C], \quad \mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s^{(t)}_{\mathsf{end}}, \mathbf{k}[t]), \\
&\quad \forall t \in [d_C], \quad \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}} \leftarrow \mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{k}[t], \mathsf{sk} \| \mathsf{Bits}(s^{(t+1)})), \\
&\mathsf{pd} := \big(\{\mathsf{aHMAC.pd}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}}, \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}}\}_{t\in[d_C]}\big).
\end{aligned}
\tag{7.11}
$$

In the simulation, the are computed as follows:

$$
\begin{aligned}
&\mathsf{pp} = (N,2) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^2), \\
&\quad \forall j \in [d], \quad \mathsf{aHMAC}.\widetilde{\mathsf{pd}}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil}), \\
&\quad \forall t \in [d_C], \quad \mathsf{aHMAC}.\widetilde{\mathsf{pd}'}^{(t)}, \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil}), \\
&\quad \forall t \in [d_C], \quad \mathsf{HSS}.\widetilde{\mathsf{pd}}^{(t)} \leftarrow \mathsf{HSS}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil + \lambda}), \\
&\widetilde{\mathsf{pd}} := \big(\{\mathsf{aHMAC}.\widetilde{\mathsf{pd}}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC}.\widetilde{\mathsf{pd}'}^{(t)}, \mathsf{HSS}.\widetilde{\mathsf{pd}}^{(t)}\}_{t\in[d_C]}\big),
\end{aligned}
\tag{7.12}
$$

where $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\ell})$ is as follows

$$g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow [N^3]^{\ell},\ \mathsf{seed} \leftarrow \{0,1\}^{\lambda}$$
$$\mathsf{aHMAC}^{\mathsf{Pai}}.\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}}), \tag{7.13}$$

and $\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\ell})$ as as follows

$$g \leftarrow \mathsf{Pai.Samp}(\mathsf{pp}),\ \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \leftarrow [N^3]^{\ell},\ \mathsf{seed} \leftarrow \{0,1\}^{\lambda}$$
$$\mathsf{HSS}^{\mathsf{Pai}}.\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}}, g^{\mathbf{d}}). \tag{7.14}$$

We show an analogous claim (to Claim 2) which completes the proof.

Claim 3. *For all* $\mathsf{sk} \in \{0,1\}^{\lambda}$*, the distribution of* pd *defined by Equation 7.11 and* $\widetilde{\mathsf{pd}}$ *by Equation 7.12 are computationally indistinguishable.*

Proof. We show a series of hybrid that transitions from the distribution of Equation 7.11 to Equation 7.12.

Hyb'_0 This is the distribution of Equation 7.11.

$\mathsf{Hyb}'_{0,1}$ Instead of computing the first instance of aHMAC public data $\mathsf{aHMAC.pd}^{(0)}$ as $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathsf{s}[0], \mathsf{s}[1])$, simulate it as $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil})$. We claim (Claim 4) the two ways of generating $\mathsf{aHMAC.pd}^{(0)}$ are computationally indistinguishable. Hence we have $\mathsf{Hyb}'_{0,1} \approx_c \mathsf{Hyb}'_0$.

$\mathsf{Hyb}'_{0,j}$ for $1 < j < d_{\mathrm{Ind}} - 1$, instead of computing the j-th instance of aHMAC public data from

$$\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathsf{s}[j], \mathsf{s}[j+1]),$$

simulate is as

$$\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil}).$$

By Claim 4 again, we have $\mathsf{Hyb}'_{0,j} \approx_c \mathsf{Hyb}'_{0,j-1}$.

$\mathsf{Hyb}'_{0,j}$ for $j = d_{\text{Ind}} - 1$, instead of computing the aHMAC public data $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}^{(j)}$, $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}'^{(0)}$ from (where $s^{(0)}_{\mathsf{end}} := \mathbf{s}[d_{\text{Ind}} - 1]$)

$$\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[s^{(0)}_{\mathsf{end}}, \mathbf{s}[d_{\text{Ind}}]), \quad \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[s^{(0)}_{\mathsf{end}}], \mathbf{k}[0]),$$

simulate them as

$$\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil}), \quad \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil}).$$

By Claim 4 again, we have $\mathsf{Hyb}'_{0,j} \approx_c \mathsf{Hyb}'_{0,j-1}$.

$\mathsf{Hyb}'_{0,j}$ for $j = d_{\text{Ind}}$, instead of computing the HSS public data $\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}^{(0)}$ from

$$\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{k}[0], \mathsf{sk} \| \mathsf{Bits}(s^{(1)})),$$

simulate it as

$$\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log N \rceil + \lambda}).$$

We claim (Claim 5) the two ways of generating $\mathsf{HSS}.\mathsf{pd}^{(0)}$ are computationally indistinguishable. Hence we have $\mathsf{Hyb}'_{0,j} \approx_c \mathsf{Hyb}'_{0,j-1}$.

$\mathsf{Hyb}'_{t,j}$ for $1 < t < d_C$, $1 \le j \le d_{\text{Ind}}$ is analogous to the case of $\mathsf{Hyb}'_{0,j}$, except replacing the 0 in its description with t.

We have $\mathsf{Hyb}'_{t,1} \approx_c \mathsf{Hyb}'_{t-1,d_{\text{Ind}}}$, and $\mathsf{Hyb}'_{t,j} \approx_c \mathsf{Hyb}'_{t,j-1}$. Note that $\mathsf{Hyb}'_{d_C-1,d_{\text{Ind}}}$ computes exactly the distribution of Equation 7.12.

By a hybrid argument, we conclude that $\mathsf{Hyb}'_0 \approx_c \mathsf{Hyb}'_{d_C-1,d_{\text{Ind}}}$, which proves the claim. It remains to prove the following sub-claims.

Claim 4. *For all $s' \in \mathbb{Z}$, with bit-length $\ell \le \text{poly}(\lambda)$ the following computational indistinguishability holds*

$$\left\{\mathsf{pp}, \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, s') \,\middle|\, s \leftarrow [N]\right\}_\lambda$$
$$\approx_c \left\{\mathsf{pp}, \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^\ell)\right\}_\lambda$$

where the public parameter pp is sampled as $\mathsf{pp} = (N, 2) \leftarrow \mathsf{Pai}.\mathsf{Gen}(1^\lambda, 1^2)$ in both sides.

Proof. This follows directly from P-DDH and DDH (Definition 7.3 and 7.2) in Paillier groups. □

Claim 5. *For all $s' \in \mathbb{Z}$, with bit-length $\ell \leq \mathrm{poly}(\lambda)$, and $\mathsf{sk} \in \{0,1\}^\lambda$ the following computational indistinguishability holds*

$$\left\{\mathsf{pp}, \mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, \mathsf{sk}\|\mathsf{Bits}(s')) \,\middle|\, s \leftarrow [N]\right\}_\lambda$$
$$\approx_c \left\{\mathsf{pp}, \mathsf{HSS}^{\mathsf{Pai}}.\mathsf{Sim}(\mathsf{pp}, 1^\ell)\right\}_\lambda$$

where the public parameter pp *is sampled as* $\mathsf{pp} = (N, 2) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^2)$ *in both sides.*

Proof. This follows directly from DDH (Definition 6.7) in Paillier groups. □

7.5.3 Instantiations under Lattices

In this section, we instantiate the non-leveled and leveled variants of our 2PC protocols under lattices, $\mathsf{BoolCircEval}^{C,\mathsf{Lat}}$, $\mathsf{LBoolCircEval}^{C,\mathsf{Lat}}$. As explained in the begining of Section 7.5, the protocols stay mostly unchanged, except for the Init phases, during which P_G computes public data pd differently. We show them in Figure 7.8 and 7.9 respectively.

Parameter Settings. Our instantiations under lattices uses the following public parameter settings: A polynomial ring $\mathcal{R}(\lambda) = \mathbb{Z}[X]/(X^{n(\lambda)} + 1)$, two moduli $p(\lambda), q(\lambda)$, and error and secret distributions $\mathcal{D}_{\mathsf{err}}(\lambda), \mathcal{D}_{\mathsf{sk}}(\lambda)$ where

- $n \leq \mathrm{poly}(\lambda)$ is a power-of-two,
- $p \geq \lambda^{\omega(1)}$, $q = p \cdot \Delta$, and $\Delta \geq p \cdot \lambda^{\omega(1)}$;
- $\mathcal{D}_{\mathsf{err}}(\lambda)$, $\mathcal{D}_{\mathsf{sk}}(\lambda)$ have coefficients bounded by $\mathrm{poly}(\lambda)$.

We write $\mathsf{pp}^{\mathsf{Lat}} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$.

The Non-leveled Variant. The non-leveled 2PC protocol is shown in Figure 7.8. It uses the same core sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ (Figure 7.3) which stays unchanged. We summarize the correctness and security of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under lattices in the following lemmas. Their proofs are completely analogous to those of Lemma 7.15 and 7.16, hence are omitted.

Lemma 7.19 (Correctness of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under Lattices)**.** *Let $\ell(\lambda) \leq O(\log \lambda)$ be a bound on input length, and $\mathsf{pp}^{\mathsf{Lat}}$ be the public parameters specified in Section 7.5.3. There exists a negligible function* $\operatorname{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C with a gate g of $\ell_x \leq \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, secret exponent $s \in \mathcal{D}_{\mathsf{sk}}$, additive shares (over $\mathcal{R}$) $\langle s\overline{\mathbf{x}}\rangle_0, \langle s\overline{\mathbf{x}}\rangle_1$, and PRF key $\mathsf{sk} \in \{0,1\}^\lambda$, the following holds:*

$$\Pr\left[\begin{array}{c|l} & \mathsf{pd} \text{ sampled per Equation 7.15}, \\ w_1^z = w_0^z + s\overline{z}, & (P_G : w_0^z), (P_E : w_1^z, \overline{z}) \\ z = \mathsf{g}(\mathbf{x}) & \quad \leftarrow \mathsf{BoolGateEval}^{C,\mathsf{g}}\big((P_G : \mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_0), (P_E : \mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}})\big) \\ & z := \overline{z} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)), \quad \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)) \end{array}\right]$$
$$\geq 1 - \operatorname{negl}(\lambda).$$

Lemma 7.20 (Security of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under Lattices)**.** *Under the same setting as Lemma 7.19, there exists an efficient simulator Sim that, given the masked output $\overline{z}$,* statistically *simulates P_G's message in the sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$.*

More precisely, there exists a negligible function $\operatorname{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C with a gate g of $\ell_x \leq \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, secret exponent $s \in \mathcal{D}_{\mathsf{sk}}$, additive shares (over $\mathcal{R}$) $\langle s\overline{\mathbf{x}}\rangle_0, \langle s\overline{\mathbf{x}}\rangle_1$, and PRF key $\mathsf{sk} \in \{0,1\}^\lambda$, the following holds:*

$$\begin{array}{r|l} \mathsf{SD}\big(\mathsf{msg}_G(\mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_0), & \mathsf{pd} \text{ sampled per Equation 7.15}, \\ & \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)), \\ \mathsf{Sim}(\mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}, \overline{z})\big) \leq \operatorname{negl}(\lambda), & \overline{z} := \mathsf{g}(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \end{array}$$

where $\mathsf{msg}_G(\mathsf{pd}, \langle s\overline{\mathbf{x}}\rangle_0)$ denotes P_G's message to P_E in $\mathsf{BoolGateEval}^{C,\mathsf{g}}$.

Using the correctness and security of the core sub-protocol, $\mathsf{BoolGateEval}$ under lattices, we can now prove those of our garbling scheme under lattices (compiled from the 2PC protocol $\mathsf{BoolCircEval}$).

Proposition 7.3 (Garbling of $O(\log \lambda)$-ary Gates under Lattices)**.** *Assuming CP-RLWE with respect to the public parameters* $\mathsf{pp}^{\mathsf{Lat}}$ *specified in Section 7.5.3, the garbling scheme compiled from the protocol* $\mathsf{BoolCircEval}^{C,\mathsf{Lat}}$ *(Figure 7.8) is correct and secure.*

Proof of Proposition 7.3. The correctness of the protocol follows from that of $\mathsf{BoolGateEval}$ (Lemma 7.19). Hence the correctness of the compiled garbling scheme follows.

The security proof follows the same arguments as those for Proposition 7.1, except the public data pd are computed and simulated differently. In the honest protocol, they are computed as follows according to Equation 7.15, with respect to a PRF key $\mathsf{sk} \in \{0,1\}^\lambda$, and the public parameters $\mathsf{pp}^{\mathsf{Lat}} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ described in Section 7.5.3.

$$\begin{aligned}
&\mathsf{seed} \leftarrow \{0,1\}^\lambda,\ s, r_1, r_2 \leftarrow \mathcal{D}_{\mathsf{sk}},\ a \leftarrow \mathcal{R}_q, \mathbf{a}' \leftarrow \mathcal{R}_q^\lambda,\\
&e_1, e_2 \leftarrow \mathcal{D}_{\mathsf{err}},\ \mathbf{e}', \mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}''_1, \mathbf{e}''_2 \leftarrow \mathcal{D}_{\mathsf{err}}^\lambda,\ \mathbf{b} := s\mathbf{a}' + \mathbf{e}'\\
&\mathsf{pd} = (\mathsf{pp}^{\mathsf{Lat}}, \mathsf{seed}, a, sa + e_1, s^2 a + e_2 - s\Delta\\
&\qquad r_1\mathbf{a}' + \mathbf{e}'_1 + \mathsf{Bits}(\mathsf{sk})\Delta,\ r_1\mathbf{b} + \mathbf{e}''_1,\\
&\qquad r_2\mathbf{a}' + \mathbf{e}'_2,\ r_2\mathbf{b} + \mathbf{e}''_2 + \mathsf{Bits}(\mathsf{sk})\Delta).
\end{aligned} \tag{7.16}$$

In the simulation, they are computed as random elements:

$$\begin{aligned}
&\mathsf{seed} \leftarrow \{0,1\}^\lambda,\ a, b, c \leftarrow \mathcal{R}_q,\ \mathbf{b}', \mathbf{c}', \mathbf{b}'', \mathbf{c}'' \leftarrow \mathcal{R}_q^\lambda,\\
&\widetilde{\mathsf{pd}} = (\mathsf{pp}^{\mathsf{Lat}}, \mathsf{seed}, a,\ b,\ c,\ \mathbf{b}'\,\mathbf{c}', \mathbf{b}'',\ \mathbf{c}'').
\end{aligned} \tag{7.17}$$

We show the analogous claim (to Claim 2) which completes the arguments for this proof.

Claim 6. *For all* $\mathsf{sk} \in \{0,1\}^\lambda$*, the distribution of* pd *defined by Equation 7.16 and* $\widetilde{\mathsf{pd}}$ *by Equation 7.17 are computationally indistinguishable.*

Proof. We show a series of hybrid that transitions from the distribution of Equation 7.16 to Equation 7.17.

Hyb'_0 This is the distribution of Equation 7.16.

Hyb'_1 In this hybrid, instead of computing the aHMAC public data, together with the intermediate value $\mathbf{b}$ as

$$a, sa + e_1, s^2a + e_2 - s\Delta,\ \mathbf{b} := s\mathbf{a}' + \mathbf{e}'$$

where $a, \mathbf{a}'$ are random elements in $\mathcal{R}_q$, s is a secret sampled from $\mathcal{D}_{\mathsf{sk}}$, and $e_1, e_2, \mathbf{e}'$ are errors from $\mathcal{D}_{\mathsf{err}}$, simulate them as random elements $a, b, c, \mathbf{b}$ from $\mathcal{R}_q$. By CP-RLWE (Definition 7.5), we have $\mathsf{Hyb}'_1 \approx_c \mathsf{Hyb}'_0$.

Hyb'_2 In this hybrid, instead of computing the HSS public data as

$$r_1\mathbf{a}' + \mathbf{e}'_1 + \mathsf{Bits}(\mathsf{sk})\Delta,\ r_1\mathbf{b} + \mathbf{e}''_1,\ r_2\mathbf{a}' + \mathbf{e}'_2,\ r_2\mathbf{b} + \mathbf{e}''_2 + \mathsf{Bits}(\mathsf{sk})\Delta,$$

where $\mathbf{a}, \mathbf{a}', \mathbf{b}$ are random elements from $\mathcal{R}_q$, r_1, r_2 secrets sampled from $\mathcal{D}_{\mathsf{sk}}$, and $\mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}''_1, \mathbf{e}''_2$ are errors from $\mathcal{D}_{\mathsf{err}}$, simulate them as

$$\mathbf{b}' + \mathsf{Bits}(\mathsf{sk})\Delta,\ \mathbf{c}',\ \mathbf{b}'',\ \mathbf{c}'' + \mathsf{Bits}(\mathsf{sk})\Delta,$$

where $\mathbf{b}', \mathbf{c}', \mathbf{b}'', \mathbf{c}''$ are random elements from $\mathcal{R}_q$. By RLWE (which is implied by CP-RLWE) we have $\mathsf{Hyb}'_2 \approx_c \mathsf{Hyb}'_1$.

Hyb'_3 In this hybrid, instead of adding the term $\mathsf{Bits}(sk)\Delta$ to random elements $\mathbf{b}'$ and $\mathbf{c}''$ as above, directly compute HSS public data as random elements $\mathbf{b}', \mathbf{c}', \mathbf{b}'', \mathbf{c}''$ from $\mathcal{R}_q$.

Since $\mathbf{b}', \mathbf{c}''$ are random, they perfectly hide the additive factor $\mathsf{Bits}(sk)\Delta$. We have $\mathsf{Hyb}'_3 \equiv \mathsf{Hyb}'_2$. Note that Hyb'_3 computes exactly the distribution of Equation 7.17.

By a hybrid argument, we conclude that $\mathsf{Hyb}'_0 \approx_c \mathsf{Hyb}'_3$, which proves the claim. □

The Leveled Variant. The leveled 2PC protocol is shown in Figure 7.9. It uses the same core sub-protocol $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ (Figure 7.6, 7.7) which stays unchanged. We summarize the correctness and security of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under lattices in the following lemmas. Their proofs are completely analogous to those of Lemma 7.17 and 7.18, hence are omitted.

Lemma 7.21 (Correctness of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under Lattices)**.** *Let $\ell(\lambda) \leq O(\log \lambda)$ be a bound on input length, $\mathsf{pp}^{\mathsf{Lat}}$ be the public parameters specified in Section 7.5.3, and $d_{\mathrm{Ind}} = O(\log\log\lambda)$ be the depth of the indicator arithmetic circuit over ℓ inputs (Fact 1).*

There exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C (of depth d_C) with a gate g of $\ell_x \leq \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, secret exponents $\mathbf{s} \in \mathcal{D}_{\mathsf{sk}}^{d_C \cdot d_{\mathrm{Ind}}+1}$, additive shares (over $\mathcal{R}$) $\langle s^{(t)}\overline{\mathbf{x}}\rangle_0, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1$, and PRF key $\mathsf{sk} \in \{0,1\}^\lambda$, the following holds: (where we use the shorthand $s^{(t)} = \mathbf{s}[t \cdot d_{\mathrm{Ind}}]$)*

$$\Pr\left[\begin{array}{l|l} & \mathsf{pd} \textit{ sampled per Equation 7.18}, \\ w_1^z = w_0^z + s^{(t+1)}\overline{z}, & (P_G : w_0^z), (P_E : w_1^z, \overline{z}) \leftarrow \mathsf{LBoolGateEval}^{C,\mathsf{g}} \\ \quad z = \mathsf{g}(\mathbf{x}) & \quad \big((P_G : \mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_0), (P_E : \mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}})\big) \\ & z := \overline{z} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)), \\ & \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)) \end{array}\right]$$
$$\geq 1 - \mathrm{negl}(\lambda).$$

Lemma 7.22 (Security of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under Lattices)**.** *Under the same setting as Lemma 7.21, there exists an efficient simulator* Sim *that, given the masked output $\overline{z}$,* statistically *simulates P_G's message in the sub-protocol* $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$.

More precisely, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C with a gate g of $\ell_x \leq \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, secret exponents $\mathbf{s} \in \mathcal{D}_{\mathsf{sk}}^{d_C \cdot d_{\mathrm{Ind}}+1}$, additive shares (over $\mathbb{Z}$) $\langle s^{(t)}\overline{\mathbf{x}}\rangle_0, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1$, and PRF key $\mathsf{sk} \in \{0,1\}^\lambda$, the following holds:*

$$\left.\begin{array}{r} \mathsf{SD}\big(\mathsf{msg}_G(\mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_0), \qquad\qquad \\ \mathsf{Sim}(\mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}, \overline{z})\big) \leq \mathrm{negl}(\lambda), \end{array}\right| \begin{array}{l} \mathsf{pd} \textit{ sampled per Equation 7.18}, \\ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)), \\ \overline{z} := \mathsf{g}(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \end{array}$$

where $\mathsf{msg}_G(\mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_0)$ denotes P_G's message to P_E in $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$.

Using the correctness and security of $\mathsf{LBoolGateEval}$ under lattices, we can now prove those of our leveled garbling scheme under lattices (compiled from the 2PC protocol $\mathsf{LBoolCircEval}$).

Proposition 7.4 (Leveled Garbling of $O(\log\lambda)$-ary Gates under Lattices)**.** *Assuming P-RLWE with respect to the public parameters* $\mathsf{pp}^{\mathsf{Lat}}$ *specified in Section 7.5.3, the garbling scheme compiled from the protocol* $\mathsf{LBoolCircEval}^{C,\mathsf{Lat}}$ *(Figure 7.9) is correct and secure.*

Proof of Proposition 7.4. The correctness of the protocol follows from that of $\mathsf{LBoolGateEval}$ (Lemma 7.21). Hence the correctness of the compiled garbling scheme follows.

The security proof follows the same arguments as those for Proposition 7.1, except the public data pd are computed and simulated differently. In the honest protocol, they are computed as follows according to Equation 7.18, with respect to a PRF key $\mathsf{sk} \in \{0,1\}^\lambda$ and the public parameters $\mathsf{pp}^{\mathsf{Lat}} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ described in Section 7.5.3:

$$
\begin{aligned}
&\mathbf{s} \leftarrow \mathcal{D}_{\mathsf{sk}}^{d+1},\ \mathbf{k} \leftarrow \mathcal{D}_{\mathsf{sk}}^{d_C},\\
&\quad // \text{ For short, write } s^{(t)} = \mathbf{s}[t \cdot d_{\mathrm{Ind}}],\ s^{(t)}_{\mathsf{end}} = \mathbf{s}[(t+1)\cdot d_{\mathrm{Ind}} - 1].\\
&\quad \forall j \in [d],\quad \mathsf{aHMAC.pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, \mathbf{s}[j], \mathbf{s}[j+1]),\\
&\quad \forall t \in [d_C],\quad \mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, s^{(t)}_{\mathsf{end}}, \mathbf{k}[t]),\\
&\quad \forall t \in [d_C],\quad \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}} \leftarrow \mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, \mathbf{k}[t], \mathsf{sk}\|\mathsf{Bits}(s^{(t+1)})),\\
&\mathsf{pd} := \big(\{\mathsf{aHMAC.pd}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}}, \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}}\}_{t\in[d_C]}\big).
\end{aligned}
\tag{7.19}
$$

In the simulation, the are computed as follows:

$$
\begin{aligned}
&\forall j \in [d],\quad \mathsf{aHMAC}.\widetilde{\mathsf{pd}}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp}^{\mathsf{Lat}}),\\
&\forall t \in [d_C],\quad \mathsf{aHMAC}.\widetilde{\mathsf{pd}'}^{(t)}, \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp}^{\mathsf{Lat}}),\\
&\forall t \in [d_C],\quad \mathsf{HSS}.\widetilde{\mathsf{pd}}^{(t)} \leftarrow \mathsf{HSS}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp}^{\mathsf{Lat}}),\\
&\widetilde{\mathsf{pd}} := \big(\{\mathsf{aHMAC}.\widetilde{\mathsf{pd}}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC}.\widetilde{\mathsf{pd}'}^{(t)}, \mathsf{HSS}.\widetilde{\mathsf{pd}}^{(t)}\}_{t\in[d_C]}\big),
\end{aligned}
\tag{7.20}
$$

where $\mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp})$ is as follows

$$
\begin{aligned}
&a, b, c \leftarrow \mathcal{R}_q,\ \mathsf{seed} \leftarrow \{0,1\}^\lambda\\
&\mathsf{aHMAC}^{\mathsf{Pai}}.\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, a, b, c),
\end{aligned}
\tag{7.21}
$$

and $\mathsf{HSS}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp})$ as as follows

$$\begin{aligned} &\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \leftarrow \mathcal{R}_q^{n \log q + \lambda}, \ \mathsf{seed} \leftarrow \{0,1\}^\lambda \\ &\mathsf{HSS}^{\mathsf{Pai}}.\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}). \end{aligned} \tag{7.22}$$

We show an analogous claim (to Claim 3) which completes the proof.

Claim 7. *For all* $\mathsf{sk} \in \{0,1\}^\lambda$*, the distribution of* pd *defined by Equation 7.19 and* $\widetilde{\mathsf{pd}}$ *by Equation 7.20 are computationally indistinguishable.*

Proof. The proof is again analogous to that of Claim 3, based on the following two sub-claims.

Claim 8. *For all* $s' \in \mathcal{D}_{\mathsf{sk}}$*, the following computational indistinguishability holds*

$$\begin{aligned} &\left\{\mathsf{pp}^{\mathsf{Lat}}, \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, s, s') \,\middle|\, s \leftarrow \mathcal{D}_{\mathsf{sk}}\right\}_\lambda \\ &\approx_c \left\{\mathsf{pp}^{\mathsf{Lat}}, \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp}^{\mathsf{Lat}})\right\}_\lambda \end{aligned}$$

Proof. This follows directly from P-RLWE (Definition 7.4). □

Claim 9. *For all* $s' \in [N]$*, and* $\mathsf{sk} \in \{0,1\}^\lambda$ *the following computational indistinguishability holds*

$$\begin{aligned} &\left\{\mathsf{pp}^{\mathsf{Lat}}, \mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, s, \mathsf{sk} \| \mathsf{Bits}(s')) \,\middle|\, s \leftarrow \mathcal{D}_{\mathsf{sk}}\right\}_\lambda \\ &\approx_c \left\{\mathsf{pp}^{\mathsf{Lat}}, \mathsf{HSS}^{\mathsf{Lat}}.\mathsf{Sim}(\mathsf{pp}^{\mathsf{Lat}})\right\}_\lambda . \end{aligned}$$

Proof. This follows from RLWE, which is implied by P-RLWE. □

7.5.4 Instantiations under Prime-Order Groups

In this section, we instantiate the non-leveled and leveled variants of our 2PC protocols under prime-order groups, $\mathsf{BoolCircEval}^{C,\mathsf{Pri}}$, $\mathsf{LBoolCircEval}^{C,\mathsf{Pri}}$. As explained in the begining of Section 7.5, the protocols stay mostly unchanged, except for the Init phases, during which P_G computes public data pd differently. We show them in Figure 7.10 and 7.11, 7.12 respectively.

The Non-leveled Variant. The non-leveled 2PC protocol is shown in Figure 7.10. It uses the same core sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ (Figure 7.3) which stays unchanged. We

summarize the correctness and security of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under prime-order groups in the following lemmas. Their proofs are completely analogous to those of Lemma 7.15 and 7.16, hence are omitted.

Lemma 7.23 (Correctness of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under Prime-Order Groups)**.** *Let* $\ell(\lambda) \leq O(\log \lambda)$ *be a bound on input length, and* $\delta = 1/(\mathrm{poly}(\lambda) \cdot |C|)$ *be the error bound specified in Figure 7.10. There exists a negligible function* $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, every Boolean circuit* C *with a gate* g *of* $\ell_x \leq \ell(\lambda)$ *inputs, every masked input* $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$*,* $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$*, secret exponents* $\mathbf{s} \in \{0,1\}^{\lceil \log p \rceil}$*, additive shares (over* $\mathbb{Z}$*)* $\langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_0, \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_1$*, and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$*, the following holds:*

$$\Pr\left[\begin{array}{c|l} & \mathsf{pd} \textit{ sampled per Equation 7.23}, \\ w_1^z = w_0^z + \mathbf{s}\overline{z}, & (P_G : w_0^z), (P_E : w_1^z, \overline{z}) \\ z = \mathsf{g}(\mathbf{x}) & \quad \leftarrow \mathsf{BoolGateEval}^{C,\mathsf{g}}\big((P_G : \mathsf{pd}, \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_0), (P_E : \mathsf{pd}, \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}})\big) \\ & z := \overline{z} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)), \quad \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)) \end{array}\right]$$
$$\geq 1 - \delta(\lambda) - \mathrm{negl}(\lambda).$$

Lemma 7.24 (Security of $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ under Prime-Order Groups)**.** *Under the same setting as Lemma 7.23, there exists an efficient simulator* Sim *that, given the masked output* $\overline{z}$, statistically *simulates* P_G*'s message in the sub-protocol* $\mathsf{BoolGateEval}^{C,\mathsf{g}}$.

More precisely, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, every Boolean circuit* C *with a gate* g *of* $\ell_x \leq \ell(\lambda)$ *inputs, every masked input* $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$*,* $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$*, secret exponents* $\mathbf{s} \in \{0,1\}^{\lceil \log p \rceil}$*, additive shares (over* $\mathbb{Z}$*)* $\langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_0, \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_1$*, and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$*, the following holds:*

$$\begin{array}{l|l} & \mathsf{pd} \textit{ sampled per Equation 7.23}, \\ \mathsf{SD}\big(\mathsf{msg}_G(\mathsf{pd}, \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_0), & \quad \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)), \\ \quad \mathsf{Sim}(\mathsf{pd}, \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}}, \overline{z})\big) \leq \mathrm{negl}(\lambda) + \delta(\lambda), & \quad \overline{z} := \mathsf{g}(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \end{array}$$

where $\mathsf{msg}_G(\mathsf{pd}, \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_0)$ *denotes* P_G*'s message to* P_E *in* $\mathsf{BoolGateEval}^{C,\mathsf{g}}$.

Using the correctness and security of the core sub-protocol, $\mathsf{BoolGateEval}$ under prime-order groups, we can now prove those of our garbling scheme under prime-order groups (compiled from the 2PC protocol $\mathsf{BoolCircEval}$).

Proposition 7.5 (Garbling $O(\log \lambda)$-ary Gates under Prime-Order Groups)**.** *Assuming CP-DDH in prime-order groups, the garbling scheme compiled from the protocol* $\mathsf{BoolCircEval}^{C,\mathsf{Pri}}$ *(Figure 7.10) achieves* $1/\mathrm{poly}$ *correctness and privacy error.*

Proof of Proposition 7.5. The correctness of the protocol, with an error $\delta \cdot |C| = 1/\mathrm{poly}(\lambda)$ follows from that of $\mathsf{BoolGateEval}$ (Lemma 7.23), and a union bound on the error probability over all gates in C. Hence the correctness of the compiled garbling scheme follows.

The security proof follows the same arguments as those for Proposition 7.1, except for two differences.

- In Hyb_1, which changes from following the subprotocol $\mathsf{BoolGateEval}$ as P_G with $(\mathsf{pd}, \{k^{(i)}\})$ as inputs, into following the subprotocol as P_E with $(\mathsf{pd}, \{l^{(i)}, \overline{x}^{(i)}\})$ as inputs, there is an error probability for every gate in C. Therefore, the statistical distance between Hyb_1 and Hyb_0 is bounded by $\mathrm{negl}(\lambda) + \delta(\lambda) \cdot |C| \leq 1/\mathrm{poly}(\lambda)$.
- The public data pd are computed and simulated differently as explained below. We need to argue the honestly computed pd and the simulated are computationally indistinguishable, which completes the argument for this proof.

In the honest protocol, the public data pd are computed as follows according to Equation 7.23, with respect to a PRF key $\mathsf{sk} \in \{0,1\}^\lambda$.

$$\begin{aligned}
&\mathsf{seed} \leftarrow \{0,1\}^\lambda,\ \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda),\\
&\quad s \leftarrow \mathbb{Z}_p, \mathbf{r} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil},\ \mathbf{r}' \leftarrow \mathbb{Z}_p^\lambda,\ \mathbf{R} \leftarrow \mathbb{Z}_p^{\lambda \times \lceil \log p \rceil},\\
&\mathsf{pd} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{r}}, g^{\mathbf{r}s}, g^{\mathbf{r}s^2 + \mathsf{Bits}(s)},\\
&\qquad\quad g^{\mathbf{r}'s}, g^{\mathbf{r}'s^2 + \mathsf{Bits}(\mathsf{sk})}, g^{\mathbf{R}s}, g^{\mathbf{R}s^2 + \mathsf{Bits}(\mathsf{sk}) \otimes \mathsf{Bits}(s)}).
\end{aligned} \tag{7.24}$$

In the simulation, they are computed as random elements:

$$\begin{aligned}\mathsf{seed} \leftarrow \{0,1\}^\lambda,\ \mathsf{pp} = (G,p,g) \leftarrow \mathsf{Pri.Gen}(1^\lambda),\\ \mathbf{a},\mathbf{b},\mathbf{c} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil},\ \mathbf{a}',\mathbf{b}' \leftarrow \mathbb{Z}_p^\lambda,\ \mathbf{A},\mathbf{B} \leftarrow \mathbb{Z}_p^{\lambda\times\lceil \log p\rceil},\\ \mathsf{pd} = (\mathsf{pp},\mathsf{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}}, g^{\mathbf{a}'}, g^{\mathbf{b}'}, g^{\mathbf{A}}, g^{\mathbf{B}}).\end{aligned} \tag{7.25}$$

We show the analogous claim (to Claim 2) which completes the arguments for this proof.

Claim 10. *For all* $\mathsf{sk} \in \{0,1\}^\lambda$*, the distribution of* pd *defined by Equation 7.24 and* $\widetilde{\mathsf{pd}}$ *by Equation 7.25 are computationally indistinguishable.*

Proof. We show a series of hybrid that transitions from the distribution of Equation 7.24 to Equation 7.25.

Hyb'_0 This is the distribution of Equation 7.16.

Hyb'_1 In this hybrid, instead of computing the last two terms of HSS public data as

$$\mathbf{H}_1 = g^{\mathbf{R}s}, \mathbf{H}_2 = g^{\mathbf{R}s^2 + \mathsf{Bits}(\mathsf{sk})\otimes \mathsf{Bits}(s)},$$

where $\mathbf{R}$ are random exponents, simulate them based on the aHMAC public data $g^{\mathbf{r}s}, g^{\mathbf{r}s^2+\mathsf{Bits}(s)}$ as follows.

$$\widetilde{\mathbf{H}}_1 = g^{\mathsf{Bits}(\mathsf{sk})\otimes \mathbf{r}s + \mathbf{R}s}, \quad \widetilde{\mathbf{H}}_2 = g^{\mathsf{Bits}(\mathsf{sk})\otimes(\mathbf{r}s^2+\mathsf{Bits}(s)) + \mathbf{R}s^2}.$$

By the randomness of $\mathbf{R}$, we have $\mathsf{Hyb}'_1 \equiv \mathsf{Hyb}'_0$.

Hyb'_2 In this hybrid, instead of computing the aHMAC public data as

$$g^{\mathbf{r}}, g^{\mathbf{r}s}, g^{\mathbf{r}s^2+\mathsf{Bits}(s)},$$

where $\mathbf{r}, s$ are random exponents, simulate them as random elements

$$g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}},$$

for random exponents $\mathbf{a},\mathbf{b},\mathbf{c}$. By CP-DDH in prime-order groups, (Definition 7.3), we have $\mathsf{Hyb}'_2 \approx_c \mathsf{Hyb}'_1$.

Hyb'_3 In this hybrid, instead of computing the HSS public data as

$$g^{\mathbf{r}'s}, g^{\mathbf{r}'s^2+\mathsf{Bits}(\mathsf{sk})}, \widetilde{\mathbf{H}}_1 = g^{\mathsf{Bits}(\mathsf{sk})\otimes\mathbf{b}+\mathbf{R}s}, \quad \widetilde{\mathbf{H}}_2 = g^{\mathsf{Bits}(\mathsf{sk})\otimes\mathbf{c}+\mathbf{R}s^2}.$$

where $\mathbf{r}', \mathbf{R}, \mathbf{b}, \mathbf{c}$ are random exponents, simulate them as

$$g^{\mathbf{a}'}, g^{\mathbf{b}'+\mathsf{Bits}(\mathsf{sk})}\widetilde{\mathbf{H}}_1, = g^{\mathsf{Bits}(\mathsf{sk})\otimes\mathbf{b}+\mathbf{A}}, \quad \widetilde{\mathbf{H}}_2 = g^{\mathsf{Bits}(\mathsf{sk})\otimes\mathbf{c}+\mathbf{B}}.$$

where $\mathbf{a}, \mathbf{a}', \mathbf{b}, \mathbf{b}', \mathbf{A}, \mathbf{B}$ are exponents. By P-DDH (Definition 7.2, which is implied by CP-DDH) in prime-order groups, we have $\mathsf{Hyb}'_3 \approx_c \mathsf{Hyb}'_2$.

Hyb'_4 In this hybrid, remove the additive terms involving $\mathsf{Bits}(\mathsf{sk})$ from the exponents.

Due to the randomness of $\mathbf{b}', \mathbf{A}, \mathbf{B}$, We have $\mathsf{Hyb}'_4 \equiv \mathsf{Hyb}'_3$. Note that Hyb'_4 computes exactly the distribution of Equation 7.25.

By a hybrid argument, we conclude that $\mathsf{Hyb}'_0 \approx_c \mathsf{Hyb}'_4$, which proves the claim. □

The Leveled Variant. The leveled 2PC protocol is shown in Figure 7.11, 7.12. It uses the same core sub-protocol $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ (Figure 7.6, 7.7) which stays unchanged. We summarize the correctness and security of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under prime-order groups in the following lemmas. Their proofs are completely analogous to those of Lemma 7.17 and 7.18, hence are omitted.

Lemma 7.25 (Correctness of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under Prime-Order Groups)**.** *Let $\ell(\lambda) \leq O(\log\lambda)$ be a bound on input length, $\delta = 1/(\mathrm{poly}(\lambda)\cdot|C|)$ be the error bound specified in Figure 7.11, 7.12 and $d_{\mathrm{Ind}} = O(\log\log\lambda)$ be the depth of the indicator arithmetic circuit over ℓ inputs (Fact 1).*

There exists a negligible function $\mathrm{negl}(\lambda)$ such that for every $\lambda \in \mathbb{N}$, every Boolean circuit C (of depth d_C) with a gate g of $\ell_x \leq \ell(\lambda)$ inputs, every masked input $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$, $\mathsf{pp} = (G, p, g)$ in the support of $\mathsf{Pri.Gen}(1^\lambda)$, secret exponents $\mathbf{S} \in \{0,1\}^{(d+1)\times\lceil\log p\rceil}$, additive

shares (over $\mathbb{Z}$*)* $\langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_0, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_1$*, and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$*, the following holds:*

$$\Pr\left[\begin{array}{l|l} & \mathsf{pd} \textit{ sampled per Equation 7.26}, \\ \mathbf{w}_1^z = \mathbf{w}_0^z + \mathbf{s}^{(t+1)}\overline{z}, & (P_G : w_0^z), (P_E : w_1^z, \overline{z}) \leftarrow \mathsf{LBoolGateEval}^{C,\mathsf{g}} \\ \quad z = \mathsf{g}(\mathbf{x}) & \quad \big((P_G : \mathsf{pd}, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_0), (P_E : \mathsf{pd}, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}})\big) \\ & z := \overline{z} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)), \\ & \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)) \end{array}\right]$$
$$\geq 1 - \delta(\lambda) - \mathrm{negl}(\lambda).$$

Lemma 7.26 (Security of $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$ under Prime-Order Groups)**.** *Under the same setting as Lemma 7.25, there exists an efficient simulator* Sim *that, given the masked output* $\overline{z}$*,* statistically *simulates* P_G*'s message in the sub-protocol* $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$*.*

More precisely, there exists a negligible function $\mathrm{negl}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, every Boolean circuit* C *with a gate* g *of* $\ell_x \leq \ell(\lambda)$ *inputs, every masked input* $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$*,* $\mathsf{pp} = (G, p, g)$ *in the support of* $\mathsf{Pri.Gen}(1^\lambda)$*, secret exponents* $\mathbf{S} \in \{0,1\}^{(d+1)\times\lceil \log p \rceil}$*, additive shares (over* $\mathbb{Z}$*)* $\langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_0, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_1$*, and PRF key* $\mathsf{sk} \in \{0,1\}^\lambda$*, the following holds:*

$$\begin{array}{l|l} & \mathsf{pd} \textit{ sampled per Equation 7.26}, \\ \mathsf{SD}\big(\mathsf{msg}_G(\mathsf{pd}, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_0), & \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)), \\ \quad \mathsf{Sim}(\mathsf{pd}, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_1, \overline{\mathbf{x}}, \overline{z})\big) \leq \mathrm{negl}(\lambda) + \delta(\lambda), & \overline{z} := \mathsf{g}(\mathbf{x}) \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \end{array}$$

where $\mathsf{msg}_G(\mathsf{pd}, \langle \mathbf{s}^{(t)} \otimes \overline{\mathbf{x}} \rangle_0)$ *denotes* P_G*'s message to* P_E *in* $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$*.*

Using the correctness and security of LBoolGateEval under prime-order groups, we can now prove those of our leveled garbling scheme under prime-order groups (compiled from the 2PC protocol LBoolCircEval).

Proposition 7.6 (Leveled Garbling of $O(\log\lambda)$-ary Gates under Prime-Order Groups)**.** *Assuming P-DDH in prime-order groups, the garbling scheme compiled from the protocol* $\mathsf{LBoolCircEval}^{C,\mathsf{Pri}}$ *(Figure 7.11, 7.12) achieves* $1/\mathrm{poly}$ *correctness and privacy error.*

Proof of Proposition 7.6. The correctness of the protocol follows from that of $\mathsf{LBoolGateEval}$ (Lemma 7.25). Hence the correctness of the compiled garbling scheme follows.

The security proof follows the same arguments as those for Proposition 7.1, except the public data pd are computed and simulated differently. In the honest protocol, they are computed as follows according to Equation 7.26, with respect to a PRF key $\mathsf{sk} \in \{0,1\}^\lambda$:

$$\begin{aligned}
&\mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\
&\mathbf{S} \in \{0,1\}^{(d+1)\times\lceil \log p \rceil} \text{ where } s_j \leftarrow \mathbb{Z}_p,\ \mathbf{S}[j] := \mathsf{Bits}(s_j), \\
&\mathbf{K} \leftarrow \{0,1\}^{d_C \times \lceil 3\log p \rceil},\ \mathsf{sk} \leftarrow \{0,1\}^\lambda \\
&\quad // \text{ For short, write } \mathbf{s}^{(t)} = \mathbf{S}[t \cdot d_{\mathrm{Ind}}],\ \mathbf{s}^{(t)}_{\mathsf{end}} = \mathbf{S}[(t+1)\cdot d_{\mathrm{Ind}} - 1]. \\
&\quad \forall j \in [d], \quad \mathsf{aHMAC.pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Pri}}, \mathbf{S}[j], \mathbf{S}[j+1]), \\
&\quad \forall t \in [d_C], \quad \mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Pri}}, \mathbf{s}^{(t)}_{\mathsf{end}}, \mathbf{K}[t]), \\
&\quad \forall t \in [d_C], \qquad \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}} \leftarrow \mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, \mathbf{K}[t], \mathsf{sk}\|\mathsf{Bits}(\mathbf{s}^{(t+1)})), \\
&\mathsf{pd} := \big(\{\mathsf{aHMAC.pd}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}}, \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}}\}_{t\in[d_C]}\big).
\end{aligned} \tag{7.27}$$

In the simulation, the are computed as follows:

$$\begin{aligned}
&\mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\
&\quad \forall j \in [d], \quad \mathsf{aHMAC}.\widetilde{\mathsf{pd}}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil \log p \rceil}), \\
&\quad \forall t \in [d_C], \quad \mathsf{aHMAC}.\widetilde{\mathsf{pd}'}^{(t)}, \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{Sim}(\mathsf{pp}, 1^{\lceil 3\log p \rceil}), \\
&\quad \forall t \in [d_C], \qquad \mathsf{HSS}.\widetilde{\mathsf{pd}}^{(t)} \leftarrow \mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{Sim}(\mathsf{pp}^{\mathsf{Lat}}), \\
&\widetilde{\mathsf{pd}} := \big(\{\mathsf{aHMAC}.\widetilde{\mathsf{pd}}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC}.\widetilde{\mathsf{pd}'}^{(t)}, \mathsf{HSS}.\widetilde{\mathsf{pd}}^{(t)}\}_{t\in[d_C]}\big),
\end{aligned} \tag{7.28}$$

where $\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{Sim}(\mathsf{pp}, 1^\ell)$ is as follows

$$\begin{aligned}
&\mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow \mathbb{Z}_p^\ell,\ \mathsf{seed} \leftarrow \{0,1\}^\lambda \\
&\mathsf{aHMAC}^{\mathsf{Pai}}.\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}}),
\end{aligned} \tag{7.29}$$

and $\mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{Sim}(\mathsf{pp})$ as as follows

$$\begin{aligned}
&\mathbf{a}, \mathbf{b}, \leftarrow \mathbb{Z}_q^{\lceil \log q \rceil + \lambda},\ \mathbf{C}, \mathbf{D} \leftarrow \mathbb{Z}_q^{(\lceil \log q \rceil + \lambda)\times\lceil 3\log p \rceil},\ \mathsf{seed} \leftarrow \{0,1\}^\lambda \\
&\mathsf{HSS}^{\mathsf{BHHO}}.\widetilde{\mathsf{pd}} = (\mathsf{pp}, \mathsf{seed}, g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{C}}, g^{\mathbf{D}}).
\end{aligned} \tag{7.30}$$

We show an analogous claim (to Claim 3) which completes the proof.

Claim 11. *For all* $\mathsf{sk} \in \{0,1\}^\lambda$*, the distribution of* pd *defined by Equation 7.27 and* $\widetilde{\mathsf{pd}}$ *by Equation 7.28 are computationally indistinguishable.*

Proof. The proof is again analogous to that of Claim 3, based on the following two sub-claims.

Claim 12. *For all* $\mathbf{s}' \in \{0,1\}^\ell$*, with* $\ell \leq \mathrm{poly}(\lambda)$ *the following computational indistinguishability holds*

$$\left\{\mathsf{pp}, \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathbf{s}') \,\middle|\, s \leftarrow \mathbb{Z}_p, \mathbf{s} := \mathsf{Bits}(s)\right\}_\lambda$$
$$\approx_c \left\{\mathsf{pp}, \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{Sim}(\mathsf{pp}, 1^\ell)\right\}_\lambda$$

where the public parameter pp *is sampled as* $\mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda)$ *in both sides.*

Proof. This follows from P-DDH (Definition 7.2) and DDH (Definition 6.7, which is implied by P-DDH) in prime-order groups. □

Claim 13. *For all* $\mathbf{s}' \in \{0,1\}^\ell$*, with* $\ell \leq \mathrm{poly}(\lambda)$ *and* $\mathsf{sk} \in \{0,1\}^\lambda$ *the following computational indistinguishability holds*

$$\left\{\mathsf{pp}, \mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{pd}(\mathsf{pp}, s, \mathsf{sk} \| \mathsf{Bits}(s')) \,\middle|\, s \leftarrow \mathcal{D}_{\mathsf{sk}}\right\}_\lambda$$
$$\approx_c \left\{\mathsf{pp}, \mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{Sim}(\mathsf{pp}, 1^\ell)\right\}_\lambda.$$

where the public parameter pp *is sampled as* $\mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda)$ *in both sides.*

Proof. This follows from the security of the BHHO [BHHO08] encryption scheme, which is based on DDH (Definition 6.7) in prime-order groups. □

7.5.5 Security Amplification for Prime-Order Group Instantiations.

In this section, we show how to adapt the amplification techniques from [BGI17] to remove the 1/poly privacy and correctness errors of our garbling scheme under prime-order groups. For simplicity, we focus on the non-leveled variant in this section. The leveled variant can be amplified in the analogous way.

The Reason For the Errors. The the 1/poly privacy and correctness errors of our garbling scheme both come from the 1/poly correctness errors in the aHMAC and HSS constructions under prime-order groups. While it's clear correctness of our garbling scheme depends on those of aHMAC and HSS, it's less obvious how privacy depends on those. We briefly review our proof strategy (see the proof of Proposition 7.5) to illustrate the cause of this error.

The relevant step in our proof consists the following hybrid experiments for computing the garbled circuits $\widehat{C}$, and labels $\{L^{(i)}\}$.

Hyb_0 : This is the real world distribution.

- First sample a global secret $\mathbf{s}$ and a PRF key sk. Compute public data pd w.r.t. $\mathsf{seed}, \mathsf{s}, \mathsf{sk}$ following Equation 7.23.
- Next sample a random pad $\mathbf{k}^{(i)}$ for every input wire i in C, and compute the labels $L^{(i)}$ as $L^{(i)} = \mathbf{s} \cdot (\mathbf{x}[i] \oplus \mathsf{PRF}(\mathsf{sk}, i)) + \mathbf{k}^{(i)}$, where $\mathbf{x}$ is the input.
- For every gate g in C, in topological order, run aHMAC and HSS (as P_G described in Figure 7.3) evaluations over $\{\mathbf{k}^{(i)}\}$, for input wires i to g. The results are a pad $\mathbf{k}^{(j)}$ and an integer $r^{(j)}$. Set $b^{(j)} = r^{(j)} \bmod 2$.
- In the end, compute $\mathbf{o} = \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWires}(C))$ and set $\widehat{C} = (\mathsf{pd}, \{b^{(j)}\}, \mathbf{o})$.

Hyb_1 : In this hybrid, compute the bits $\{b^{(j)}\}$ differently.

- For every output wire j of some gate $\mathsf{g} \in C$, compute the correct wire value $x^{(j)}$ according to the input $\mathbf{x}$. Then set $\overline{x}^{(j)} = x^{(j)} \oplus \mathsf{PRF}(\mathsf{sk}, j)$.
- For every gate g in C, in topological order, run aHMAC and HSS (as P_E described in Figure 7.3) evaluations over $\{L^{(i)}, x^{(i)}\}$, for the input wires i to g. The results are a label $L^{(j)}$ and an integer $u^{(j)}$. Set the bit $b^{(j)} = \overline{x}^{(j)} + u^{(j)} \bmod 2$.

If there are no error in the aHMAC and HSS evaluations, then we have $L^{(j)} = \mathbf{s} \cdot \overline{x}^{(j)} + \mathbf{k}^{(j)}$, and $u^{(j)} = \overline{x}^{(j)} + r^{(j)}$ for every output wire j of some gate $\mathsf{g} \in C$. Hence conditioned on no error occurs, $\mathsf{Hyb}_0, \mathsf{Hyb}_1$ compute the same distribution.

In our construction (Figure 7.10) we set the error chance of each aHMAC and HSS evaluation to be $\leq 1/(\text{poly}(\lambda)|C|)$. Hence by a union bound, no error occurs except with 1/poly chance, creating a 1/poly statistical distance between Hyb_0 and Hyb_1.

Removing the Correctness Error. The correctness errors from both aHMAC and HSS evaluations stem from the following distributed discrete logarithm (DDLog) technique, which underlies their constructions (Lemma 7.7 and 7.9). In particular, the following DDLog evaluation is invoked for every intermediate multiplication within aHMAC and HSS.

Lemma 7.27 (Distributed Discrete Log with Error [BGI16, DKK18])**.** *For any cyclic group G with order p and a generator g, there exists an algorithm $\mathsf{DDLog}_{G,g}$:*

- $\mathsf{DDLog}_{G,g}(\delta \in (0,1], B \in [p], \phi : G \to \{0,1\}^{\lceil \log(2B/\delta) \rceil}, a \in G)$ *takes an error bound δ, a message bound B, a function ϕ mapping group elements to bit strings, and an element a. It outputs a value $\alpha \in \mathbb{Z}_p$.*

The algorithm requires $O(\sqrt{B/\delta})$ group operations, and has the guarantee that for all $0 < \delta \leq 1$, $B < p$, $a \in G$, and $m \leq B$:

$$\Pr\left[\begin{array}{l|l} \mathsf{DDLog}_{G,g}(\delta, B, \phi, a \cdot g^m) & \\ =\mathsf{DDLog}_{G,g}(\delta, B, \phi, a) + m \bmod p & \phi \leftarrow \$ \end{array}\right] \geq 1 - \delta,$$

where $\phi \leftarrow \$$ means sampling at random from all possible mappings.

The DDLog algorithm is setup with a sufficiently small error bound δ such that the overall error probability (through a union bound) of all aHMAC and HSS multiplications is bounded by 1/poly. A (pseudo-)random mapping function ϕ used for DDLog is specified by the public PRG seed included in the public data pd of aHMAC and HSS.

To remove the correctness error, we follow the observation from [BGI17] that when two parties locally run DDLog on two elements $a, a \cdot g^m$, one of the party, which we call the left party, can actually detect potential errors as long as there is a bound B on the value m:

- The left party aborts with probability $\leq \delta$ over the randomness of ϕ;

- When the left party doesn't abort, both parties output the correct results except with negligible probability.

Armed with this detection technique, we can remove the correctness error from our grabling scheme: when the garbler – who acts as the left party in DDLog – aborts, it restarts with fresh randomness.

By setting the error probability δ in each DDLog invocation to be sufficiently small, $\leq 1/(\text{poly}(\lambda) \cdot |C|)$, the garbler only restarts with $1/\text{poly}(\lambda)$ probability. In expectation, it takes a constant number of restarts before the garbler produces a garbled circuit that's guaranteed to be correct.

Removing the Privacy Error. While restarting removes the 1/poly correctness error, there is still a 1/poly privacy error in the resulting scheme.

Looking again at the two hybrids $\mathsf{Hyb}_0, \mathsf{Hyb}_1$ in our proof strategy, it may seem with restarting we have ensured no error occurs during all aHMAC and HSS evaluations, and have removed the 1/poly statistical difference between Hyb_0 and Hyb_1. However, the subtle issue is that in Hyb_0, the experiment runs DDLog as the left party to detect errors and restarts, while in Hyb_1 the experiment runs DDLog as the right party, who does not have the same restarting pattern.

To simulate the restarting pattern of Hyb_0, our first step is to use another observation from [BGI17]: the right party running DDLog can actually predict potential abort from the left party with possibly false positives:

- The right party can additionally output a bit pred, which equals 1 with probability $\leq 2\delta$ over the randomness of ϕ;
- When $\mathsf{pred} = 0$, the left party does not abort.

Armed with this prediction technique, in Hyb_1, the experiment can proceed as the right party running DDLog as long as $\mathsf{pred} = 0$.

However, when the experiment sees $\mathsf{pred} = 1$ during some DDLog invocation, it then needs to re-run this particular DDLog as the left party to check if there needs to be a restart (as

$\mathsf{pred} = 1$ may be a false positive). As we explain next, re-running the DDLog as the left party relies on some leakages on the global secret s and the PRF key sk. We then show how to deal with those leakages by adapting techniques from [BGI17].

Leakages in aHMAC and HSS. In order to explain the leakage, we now expose a bit more detail on how the left and right parties, which correspond to P_G and P_E respectively in Figure 7.3, run DDLog within aHMAC and HSS evaluations.

- aHMAC evaluations are over input shares $\langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle$, where $\mathbf{s}$ is the global secret. The results are output shares $\langle \mathbf{s} \cdot C_{\mathbf{v}}(\overline{\mathbf{x}}) \rangle$, where $C_{\mathbf{v}}$ is an arithmetic circuit with intermediate values bounded by $B = 2$. The right party additionally holds $\overline{\mathbf{x}}$ in the clear.

  In each invocation of DDLog, the left and right parties respectively hold inputs of the form $a, a \cdot g^{\mathbf{s}[i] \cdot v}$:

  $$\begin{aligned} \text{Left}: &\quad \mathsf{DDLog}(\delta, \phi, B, a), \\ \text{Right}: &\quad \mathsf{DDLog}(\delta, \phi, B, a \cdot g^{\mathbf{s}[i] \cdot v}), \end{aligned}$$

  where v is an intermediate wire value of $C_{\mathbf{v}}(\overline{\mathbf{x}})$.

  When the right party predicts a potential abort and needs to re-run DDLog as the left party, it needs to know both $\mathbf{s}[i]$ and v. As all intermediate wire values v are known to the right party in the clear, the only leakage required is a certain bit $\mathbf{s}[i]$ from the global secret.

- HSS evaluations are over input shares $\langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle$, and $\langle \overline{\mathbf{x}} \rangle$. The results are output shares $\{\langle \mathbf{s} \cdot \mathsf{InnerPord}(\overline{\mathbf{x}}, C_{\mathsf{g}}(\mathsf{sk})) \rangle\}$, where C_{g} is a Boolean circuit (implementable by an arithmetic circuit with $B = 2$), and sk is the global PRF key. The right party additionally holds $\overline{\mathbf{x}}$ in the clear.

  In each invocation of DDLog, the left and right parties respectively hold inputs of the form $a, a \cdot g^{\mathbf{s}[i] \cdot \overline{\mathbf{x}}[j] \cdot v}$:

  $$\begin{aligned} \text{Left}: &\quad \mathsf{DDLog}(\delta, \phi, B = 2, a), \\ \text{Right}: &\quad \mathsf{DDLog}(\delta, \phi, B = 2, a \cdot g^{\mathbf{s}[i] \cdot \overline{\mathbf{x}}[j] \cdot v}), \end{aligned}$$

where v is an intermediate wire value of $C_{\mathsf{g}}(\mathsf{sk})$.

When the right party predicts a potential abort and needs to re-run DDLog as the left party, it needs to know $\mathbf{s}[i]$, $\overline{\mathbf{x}}[j]$, and v. As $\overline{\mathbf{x}}$ is known to the right party in the clear, the leakage required is a certain bit $\mathbf{s}[i]$ from the global secret, and an intermediate wire value v from $C_{\mathsf{g}}(\mathsf{sk})$.

In summary, the Hyb_1 experiment proceeds as the righ party running DDLog in aHMAC and HSS evaluations, as long as $\mathsf{pred} = 0$. In the case $\mathsf{pred} = 1$, it relies on the following leakage to re-run the DDLog as the left party.

- If the DDLog is within an aHMAC evaluation, the leakage is a bit $\mathbf{s}[i]$ from the global secret $\mathbf{s}$.
- If the DDLog is within an HSS evaluation, the leakage is a bit $\mathbf{s}[i]$ and an intermediate wire value v in $C_{\mathsf{g}}(\mathsf{sk})$.

If the re-run as the left party indeed aborts, then Hyb_1 restarts with fresh randomness, and all the leakages have no effect. However if the re-run does not abort, (i.e. $\mathsf{pred} = 1$ is a false positive), then Hyb_1 continues as the right party. This restarting pattern exactly simulates that of Hyb_0, so we have $\mathsf{Hyb}_0 \equiv \mathsf{Hyb}_1$.

Note that as $\mathsf{pred} = 1$ happens independently in each DDLog invocation with $\leq 2\delta \leq 1/(\text{poly}(\lambda) \cdot |C|)$ probability, in an eventual accepting Hyb_1 experiment with no aborts, there are at most some $\omega(1) \leq \lambda$ instances of leakages except with negligible probability.

In conclusion, the overall leakage in Hyb_1 are (1) $\leq \lambda$ bits from the global secret $\mathbf{s}$ and (2) $\leq \lambda$ intermediate values in the circuit $C_{\mathsf{g}}(\mathsf{sk})$.

Removing the Leakages. We explain the solutions to each leakage type in more detail. They are adapted from the techniques introduced in [BGI17] for dealing with similar types leakages.

1. The global secret $\mathbf{s} \in \{0,1\}^{\lceil \log p \rceil}$ in our garbling scheme is sampled as follows, per

Equation 7.23:

$$s \leftarrow \mathbb{Z}_p, \quad \mathbf{s} := \mathsf{Bits}(s).$$

Our security proof (of Proposition 7.5) relies on the s being a random exponent and the CP-DDH assumption (Definition 7.3) to argue that the public data pd leaks nothing about the PRF key sk. With $\leq \lambda$ bits of leakage from $\mathbf{s}$, the secret exponent s is no longer random.

Our solution is to create $\lambda + 1$ additive shares of s, and define $\mathbf{s}$ to be the bits of all $\lambda + 1$ shares. Any $\leq \lambda$ bits leaked from $\mathbf{s}$ are now statistically independent of the secret exponent s, which remains random. In more detail, we modify Equation 7.23 and correspondingly $\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}$, $\mathsf{HSS}^{\mathsf{EG}}.\mathsf{pd}$ as follows:

$$\begin{aligned}
&\text{Modified Equation 7.23}: \\
&\quad \forall i \in [\lambda + 1],\ s_i \leftarrow \mathbb{Z}_p, \quad s := \sum_i s_i \bmod p, \quad \mathbf{s} := (\ldots \| \mathsf{Bits}(s_i) \| \ldots). \\
&\text{Modified } \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}, \mathsf{HSS}^{\mathsf{EG}}.\mathsf{pd}: \\
&\quad \text{parse } \mathbf{s} = (\ldots \| \mathbf{s}_i \| \ldots), \quad s := \sum_i \mathsf{BitComp}(\mathbf{s}_i) \bmod p.
\end{aligned}$$

Now our proof argument of pd computed using the modified Equation 7.23 goes through, under a slight variant of the CP-DDH assumption that incorporates the extra secret sharing steps. (In the leveled variants, the P-DDH assumption unmodified suffices.)

Definition 7.7 (CP-DDH* Over Prime-Order Groups)**.** *We say CP-DDH* holds in*

prime-order groups if the following holds:

$$\left\{ \begin{matrix} \mathsf{pp}, s_1, \ldots, s_\lambda, g, g^s, g^{s^2}, \\ g^{\mathbf{a}}, g^{s \cdot \mathbf{a}}, g^{s^2 \cdot \mathbf{a} + (\ldots \| \mathsf{Bits}(s_i) \| \ldots)} \end{matrix} \middle| \begin{matrix} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s_0, s_1, \ldots, s_\lambda \leftarrow \mathbb{Z}_p,\ s := \sum_i s_i \bmod p \\ \mathbf{a} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil \cdot (\lambda+1)} \end{matrix} \right\}_\lambda$$

$$\approx_c \left\{ \begin{matrix} \mathsf{pp}, s_1, \ldots, s_\lambda, g, g^s, g^d, \\ g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}} \end{matrix} \middle| \begin{matrix} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s, s_1, \ldots, s_\lambda, d \leftarrow \mathbb{Z}_p, \\ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil}. \end{matrix} \right\}_\lambda .$$

Remark. This formulation can be further simplified to

$$\begin{aligned} &\left\{ \begin{matrix} \mathsf{pp}, s', g, g^s, g^{s^2}, \\ g^{\mathbf{a}}, g^{s \cdot \mathbf{a}}, g^{s^2 \cdot \mathbf{a} + \mathsf{Bits}(s + s' \bmod p)} \end{matrix} \middle| \begin{matrix} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s, s' \leftarrow \mathbb{Z}_p,\ \mathbf{a} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil} \end{matrix} \right\}_\lambda \\ \approx_c &\left\{ \begin{matrix} \mathsf{pp}, s', g, g^s, g^d, \\ g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{c}} \end{matrix} \middle| \begin{matrix} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s, s', d \leftarrow \mathbb{Z}_p,\ \mathbf{a}, \mathbf{b}, \mathbf{c} \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil}. \end{matrix} \right\}_\lambda . \end{aligned} \tag{7.31}$$

We sketch this through the following hybrid arguments:

Hyb'_0 This is the left-hand-side distribution from the CP-DDH* assumption, in a slightly more convenient form:

$$\begin{matrix} \mathsf{pp}, s_1, \ldots, s_\lambda, g, g^s, g^{s^2}, \\ \{g^{\mathbf{a}_i}, g^{s \cdot \mathbf{a}_i}, g^{s^2 \cdot \mathbf{a}_i + \mathsf{Bits}(s_i)}\}_{i=0,1,\ldots,\lambda}. \end{matrix} \middle| \begin{matrix} \mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ s_0, s_1, \ldots, s_\lambda \leftarrow \mathbb{Z}_p,\ s := \sum_i s_i \bmod p \\ \mathbf{a}_i \leftarrow \mathbb{Z}_p^{\lceil \log p \rceil} \end{matrix}$$

Hyb'_1 Equivalently sample $s \leftarrow \mathbb{Z}_p$ and set $s' := \sum_{i>0} s_i \bmod p$, and $s_0 := s - s' \bmod p$. The distribution is:

$$\begin{matrix} \mathsf{pp}, s_1, \ldots, s_\lambda, g, g^s, g^{s^2}, \\ g^{\mathbf{a}_0}, g^{s \cdot \mathbf{a}_0}, g^{s^2 \cdot \mathbf{a}_0 + \mathsf{Bits}(s - s' \bmod p)},\ \{g^{\mathbf{a}_i}, g^{s \cdot \mathbf{a}_i}, g^{s^2 \cdot \mathbf{a}_i + \mathsf{Bits}(s_i)}\}_{i=1,\ldots,\lambda}. \end{matrix}$$

We have $\mathsf{Hyb}'_0 \equiv \mathsf{Hyb}'_1$.

Hyb_2' Replace the terms $g^{s^2}, g^{s\cdot\mathbf{a}_0}, g^{s^2\cdot\mathbf{a}_0+\mathsf{Bits}(s-s' \bmod p)}$, with $g^d, g^{\mathbf{b}_0}, g^{\mathbf{c}_0}$ for random exponents $d, \mathbf{b}_0, \mathbf{c}_0$:

$$\mathsf{pp}, s_1, \ldots, s_\lambda, g, g^s, g^d,$$
$$g^{\mathbf{a}_0}, g^{\mathbf{b}_0}, g^{\mathbf{c}_0}, \{g^{\mathbf{a}_i}, g^{s\cdot\mathbf{a}_i}, g^{d\cdot\mathbf{a}_i+\mathsf{Bits}(s_i)}\}_{i=1,\ldots,\lambda}.$$

By the simplied assumption in Equation 7.31, we have $\mathsf{Hyb}_2' \approx_c \mathsf{Hyb}_1'$.

Hyb_3' Replace the terms $g^{s\cdot\mathbf{a}_i}$ and $g^{d\cdot\mathbf{a}_i}$ with $g^{\mathbf{b}_i}, g^{\mathbf{c}_i}$ for random exponents $\mathbf{b}_i, \mathbf{c}_i$:

$$\mathsf{pp}, s_1, \ldots, s_\lambda, g, g^s, g^d,$$
$$g^{\mathbf{a}_0}, g^{\mathbf{b}_0}, g^{\mathbf{c}_0}, \{g^{\mathbf{a}_i}, g^{\mathbf{b}_i}, g^{\mathbf{c}_i+\mathsf{Bits}(s_i)}\}_{i=1,\ldots,\lambda}.$$

By DDH (which is implied by Equation 7.31), we have $\mathsf{Hyb}_3' \approx_c \mathsf{Hyb}_2'$.

Hyb_4' Remove the additive terms $\mathsf{Bits}(s_i)$ from the exponents. By the randomness of exponents $\mathbf{c}_i$, we have $\mathsf{Hyb}_4' \equiv \mathsf{Hyb}_3'$. Note that the resulting is exactly the right-hand-side distribution from the CP-DDH* assumption.

2. To deal with the second leakage type, we need to ensure the leaked intermediate values from $C_{\mathsf{g}}(\mathsf{sk})$, for all gates $\mathsf{g} \in C$, are independent of the PRF key sk.

   The solution is to replace C_{g} with a leakage resilient circuit $\overline{C}_{\mathsf{g}}$ such that any set of $\leq \lambda$ intermediate values can be computationally simulated from the evaluation result only. We cite the result from [BGI17] that there exists a compiler from any NC1 Boolean circuit C_{g} to a leakage resilient one $\overline{C}_{\mathsf{g}}$ also in NC1, together with a compiler for the inputs sk to $\overline{\mathsf{sk}}$ such that $C_{\mathsf{g}}(\mathsf{sk}) = \overline{C}_{\mathsf{g}}(\overline{\mathsf{sk}})$.

   Lemma 7.28 (Leakage Resilient Circuits for NC1 [BGI17])**.** *Assuming there is a PRF in NC1. There exists a pair of compilers* $\mathsf{LR}^{\mathsf{Circ}}$ $\mathsf{LR}^{\mathsf{Input}}$ *satisfy the following.*

   Correctness: *For every logarithmic depth bound* $d \leq O(\log\lambda)$*, there exists another logarithmic bound* $d' \leq O(\log\lambda)$*, and a polynomial* $p \leq \mathrm{poly}(\lambda)$ *such that for every* $\lambda \in \mathbb{N}$*, Boolean circuit* $C : \{0,1\}^{\ell_x} \to \{0,1\}$ *of depth* $\leq d(\lambda)$*, and inputs* $\mathbf{x} \in \{0,1\}^{\ell_x}$*:*

- $\overline{C} \leftarrow \mathsf{LR}^{\mathsf{Circ}}(C)$ *has depth* $\leq d'(\lambda)$*;*
- $\overline{\mathbf{x}} \leftarrow \mathsf{LR}^{\mathsf{Input}}(\mathbf{x})$ *has bit-length* $\leq \ell_x \cdot p(\lambda)$*;*
- $C(\mathbf{x}) = \overline{C}(\overline{\mathbf{x}})$*.*

Leakage Resilience: *There exists a simulator* Sim *such that for every logarithmic depth bound* $d \leq O(\log \lambda)$*, Boolean circuits* $\{C_\lambda\}$ *of depth* $\leq d(\lambda)$*, inputs* $\{\mathbf{x}_\lambda\}$*, and sets of leakage wires* $\{S_\lambda\}$ *of size* $\leq \lambda$*:*

$$\left\{ \begin{matrix} \textit{Wire values} \\ \textit{in } S \textit{ of } \overline{C}(\overline{\mathbf{x}}) \end{matrix} \;\middle|\; \begin{matrix} \overline{C} \leftarrow \mathsf{LR}^{\mathsf{Circ}}(C), \\ \overline{\mathbf{x}} \leftarrow \mathsf{LR}^{\mathsf{Input}}(\mathbf{x}), \end{matrix} \right\}_\lambda \approx_c \{\mathsf{Sim}(C(\mathbf{x}))\}_\lambda$$

Now we just need to modify Equation 7.23 to compile the sampled PRF key sk into leakage resilient inputs $\overline{\mathsf{sk}}$, and correspondingly modify Figure 7.3 to use leakage relient circuits in HSS evaluations:

$$\begin{aligned} &\text{Modified Equation 7.23}: && \mathsf{sk} \leftarrow \{0,1\}^\lambda, \quad \overline{\mathsf{sk}} \leftarrow \mathsf{LR}^{\mathsf{Input}}(\mathsf{sk}) \\ &\text{Modified Figure 7.3 Step 2}: && \text{run } \mathsf{ExtEval}_b \text{ with } \overline{C}_{\mathsf{g}} \leftarrow \mathsf{LR}^{\mathsf{Circ}}(C). \end{aligned}$$

After applying the two solutions, we can now conclude that in Hyb_1 the leakages (required to simulating restarting patterns of Hyb_0) do not affect the remaining proof arguments.

The overhead of our solutions for removing leakages are (1) larger public data pd caused by a larger global secret vector $\mathbf{s}$ and a larger leakage resilient version of the PRF key $\overline{\mathsf{sk}}$ and (2) heavier computation in HSS evaluations caused by the leakage resilient circuits $\overline{C}_{\mathsf{g}}$.

7.6 Efficient Arithmetic Garbling Schemes

Our observation is that the Boolean garbling schemes from Theorem 7.2 (and their leveled variants), supporting evaluations of arbitrary $O(\log \lambda)$-ary gates, can implement arithmetics over small modulus $R(\lambda) \leq \mathrm{poly}(\lambda)$ very efficeintly. This is because multiplications and additions between two $\mathbb{Z}_R$ values can be implemented by $(2 \log R)$-ary Boolean gates, costing $\log R$ bits per multiplication or addition.

A small issue prevents directly using Theorem 7.2 to obtain arithmetic garbling schemes for small modulus. An arithmetic garbling scheme requires arithmetic labels for its inputs $\mathbf{x} \in \mathbb{Z}_R^{\ell_x}$, while the schemes from Theorem 7.2 require Boolean labels for the bit representations $\mathsf{Bits}(\mathbf{x})$.

Therefore, to obtain arithmetic garbling over polynomial modulus $R(\lambda) < \text{poly}(\lambda)$, we need a special garbling scheme in which the evaluation algorithm Eval takes in arithmetic labels for some input $\mathbf{x} \in \mathbb{Z}_R^{\ell_x}$, and outputs Boolean labels for their bit-representations $\mathsf{Bits}(\mathbf{x})$, as required by the scheme from Theorem 7.2. Fortunately, such schemes exist based on Chinese Remainder Theorem and the minimal assumption of one-way functions. (See Section 8 in [AIK11], and also [LL24, Hea24] for more efficient constructions based on stronger assumptions.)

Lemma 7.29 (Bit-Decomposition Garbling Scheme [AIK11])**.** *Assuming one-way functions exist, there exists a garbling scheme for the class of functions* $C : \mathbb{Z}_R \to \{0,1\}^{\lceil \log R \rceil \times \ell}$ [11] *specified by any modulus* R*, and any set of Boolean key functions* $K^{(i)} : \{0,1\} \to \{0,1\}^{\ell}$ *for* $i \in [\lceil \log R \rceil]$*:*

$$C(x) = \{K^{(i)}(\mathsf{Bits}(x)[i])\}.$$

The garbling size is $\text{poly}(\log R, \ell, \lambda)$*.*

Composing a bit-decomposition garbling scheme with Theorem 7.2 results in an arithmetic garbling scheme satisfying Definition 2.3, and creates only an additive cost of $|\mathbf{x}| \cdot \text{poly}(\lambda)$ to the garbling size. We therefore obtain the following corollary.

Corollary 7.5 (Arithmetic Garbling for Small Modulus)**.** *Let* $R(\lambda) \leq \text{poly}(\lambda)$ *be a modulus. Assuming any of the assumptions in Theorem 7.2, there exists a garbling scheme for all arithmetic circuits* C *(with binary gates,* ℓ_x *inputs) over* $\mathbb{Z}_R$ *with garbling size*

$$|\widehat{C}| \leq |C| \cdot \log R + \ell_x \cdot \text{poly}(\lambda).$$

---

[11] Technically we are generalizing Definition 2.3 to allow functions with different input and output rings: $\mathbb{Z}_R$ and $\mathbb{Z}_2$.

The scheme assuming CP-DDH in prime-order groups has 1/poly correctness and privacy errors*, which can be made negligible assuming a variant of CP-DDH (Definition 7.7).*

Alternatively, assuming any of the assumptions in Theorem 7.3, there exists a garbling scheme for all arithmetic circuits C (with binary gates, ℓ_x inputs) over $\mathbb{Z}_R$ with garbling size

$$|\widehat{C}| \leq |C| \cdot \log R + (\ell_x + \mathrm{Depth}(C)) \cdot \mathrm{poly}(\lambda).$$

Using Chinese Remainder Theorem, we can further compose multiple schemes, supporting co-prime polynomial moduli $\{R_i\}$, into one supporting a large modulus $R^* = \prod_i R_i$. We additionally show how to emulate an arbitrary modulus R using a sufficiently large one $R^* = O(R^2)$ in Section 7.6.1 (protocol $\mathsf{ArithCircEval}^C$, Figure 7.2). We show its instantiation under Paillier groups to illustrate our techniques. Other instantiations under prime-order groups and lattices differ only by how the public data are generated during the Init phase, analogous to the Boolean case. In summary, we obtain the following result.

Theorem 7.4 (Arithmetic Garbling for Large Modulus)**.** *Assuming any of the assumptions in Theorem 7.2, there exists a garbling scheme for all arithmetic circuits C (with binary gates, ℓ_x inputs) over an arbitrary modulus $\mathbb{Z}_R$ with garbling size*

$$|\widehat{C}| \leq O(|C| \cdot \log R) + \ell_x \cdot \mathrm{poly}(\lambda, \log R).$$

The scheme assuming CP-DDH in prime-order groups has 1/poly correctness and privacy errors*, which can be made negligible assuming a variant of CP-DDH (Definition 7.7).*

We can also obtain leveled variants analogous to the Boolean case to avoid circular assumptions at the cost of an additive $\mathrm{Depth}(C) \cdot \mathrm{poly}(\lambda, \log R)$ term in the size of the garbling.

Theorem 7.5 (Leveled Arithmetic Garbling for Large Modulus)**.** *Assuming any of the assumptions in Theorem 7.3, there exists a garbling scheme for all arithmetic circuits C (with binary gates, ℓ_x inputs) over an arbitrary modulus $\mathbb{Z}_R$ with garbling size*

$$|\widehat{C}| \leq O(|C| \cdot \log R) + (\ell_x + \mathrm{Depth}(C)) \cdot \mathrm{poly}(\lambda, \log R).$$

Note that we cannot use the schemes from Theorem 7.2 (and their leveled variants) to support general arithmetic gates with $\ell_x = \omega(1)$ inputs over polynomial modulus $R(\lambda) < \mathrm{poly}(\lambda)$, as their computation cost would become super-polynomial $R^{\ell_x} = \lambda^{\omega(1)}$. Therefore, we *do not* directly obtain analogous (to Theorem 7.2 and 7.4) arithmetic garbling schemes for layered circuits.

7.6.1 Handling Large R using Chinese Remainder Theorem

The overall protocol $\mathsf{ArithCircEval}^{C,\mathsf{Pai}}$ (under Paillier groups) is shown in Figure 7.2. It's mostly the same as the Boolean protocol $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ except how wire values are masked.

- In the Boolean protocol, each wire value (on wire i) is masked by a single bit derived by $\mathsf{PRF}(\mathsf{sk}, i)$.
- In the arithmetic protocol, each wire value (on wire i) is masked by an integer mod R derived by $\mathsf{PRF}(\mathsf{sk}, i)$.

The wire values are always represented as bits. In particular, the input shares during the Init phases are defined as additive shares of bit representations of the masked inputs. We can still compile such a protocol to an arithmetic garbling scheme with arithmetic input labels, relying on existing techniques [AIK11, LL24, Hea24] as explained in Section 7.6.

As in the Boolean case, we rely on a core sub-protocol $\mathsf{ArithGateEval}^{C,\mathsf{g}}$ (Figure 7.14, 7.15) to evaluate arithmetic gates $(+, \times)$. It proceeds in the following steps.

- First find enough number of $O(\log \lambda)$-bit primes $\{p_i\}$ such that their product $Q = \prod_i p_i$ is sufficiently large, $Q > R^2 \cdot \lambda^{\omega(1)}$. Let ℓ be the number of primes needed. Also define CRT representations with respect to Q as

$$\mathsf{CRT}(x) = \{x_i\} \text{ where } x_i := x \bmod p_i, \quad \mathsf{CRT}^{-1}(\{x_i\}) = x, \tag{7.33}$$

  and Boolean circuits computing those conversions.

$$\begin{aligned} C^{\mathsf{CRT}}(\mathsf{Bits}(x)) &:= \{\mathsf{Bits}(x \bmod p_i)\}_i = \mathsf{Bits}(\mathsf{CRT}(x)), \\ C^{\mathsf{InvCRT}}(\mathsf{Bits}(\mathsf{CRT}(x))) &:= \mathsf{Bits}(x). \end{aligned} \tag{7.34}$$

- P_G, P_E apply the aHMAC evaluations locally on the shares $\langle s\mathsf{Bits}(\overline{x})\rangle$, $\langle s\mathsf{Bits}(\overline{y})\rangle$ to obtain shares of their CRT representations:

$$\begin{array}{ll} \text{Input} & \langle s\mathsf{Bits}(\overline{x})\rangle, \langle s\mathsf{Bits}(\overline{y})\rangle \\ \text{via aHMAC} & \{\langle s\mathsf{Bits}(\overline{x}_i)\rangle\}, \{\langle s\mathsf{Bits}(\overline{y}_i)\rangle\}. \end{array}$$

- After decomposing the large values $\overline{x}, \overline{y}$ into small CRT representations, P_G, P_E jointly call $\mathsf{BoolGateEval}'^{C,\mathsf{g}}$ to evaluate the gate function g on each CRT components $\overline{x}_i$, $\overline{y}_i$:

$$\text{via } \mathsf{BoolGateEval}' \quad \{\langle s\mathsf{Bits}(\overline{z}_i)\rangle\},$$

  where $\mathsf{BoolGateEval}'^{C,\mathsf{g}}$ is the same as $\mathsf{BoolGateEval}^{C,\mathsf{g}}$ (Figure 7.3) except using a different Boolean circuit $C'_{g,\mathbf{v}}(\mathsf{sk})$ defined as follows.

  1. Parse $\mathbf{v}$ as two values $\overline{x}_i, \overline{y}_i \in \mathbb{Z}_{p_i}$.
  2. Compute x'_i, y'_i as

$$x'_i = \overline{x}_i - (r^x \bmod R), \quad y'_i = \overline{y}_i - (r^y \bmod R) \text{ where}$$
$$r^x \leftarrow \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)[0]), \quad r^y \leftarrow \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)[1]).$$

  3. Outputs $\mathsf{Bits}(\overline{z}_i)$ where $\overline{z}_i$ is computed as

$$\overline{z}_i = \mathsf{g}(x'_i, y'_i) + r^{(z)} \bmod p_i, \text{ where } r^z \leftarrow \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)).$$

  We note two facts of the values $\overline{z}_i$ computed by $\mathsf{BoolGateEval}'^{C,\mathsf{g}}$.

$$\begin{aligned} \mathsf{CRT}^{-1}(\{\overline{z}_i\}) &\equiv g(x, y) + \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)) \bmod R, \\ |\mathsf{CRT}^{-1}(\{\overline{z}_i\})| &\leq R^2 \cdot \lambda^{\omega(1)}. \end{aligned} \tag{7.35}$$

  where x, y are the actual wire values to the gate g.

- Finally, convert the shares of small CRT components $\langle s\overline{z}_i\rangle$ back into a share of an integer, and then compute the mod R circuit on it.

$$\begin{array}{ll} \text{via aHMAC} & \{\langle s\mathsf{Bits}(\overline{z}')\rangle\}, \text{ where } \overline{z}' = \mathsf{CRT}^{-1}(\{\overline{z}_i\}) \\ \text{via aHMAC} & \{\langle s\mathsf{Bits}(\overline{z})\rangle\}, \text{ where } \overline{z} = \overline{z}' \bmod R. \end{array}$$

The security of the subprotocol $\mathsf{ArithGateEval}^{C,\mathsf{g}}$ and of the overall protocol $\mathsf{ArithCircEval}^{C,\mathsf{Pai}}$ can be proved analogously to the Boolean case. Hence we omit them here.

	Concrete Size	Asymptotic
Ours (Paillier)	0.38 MB	$8\lambda\lceil\log N\rceil$
Ours (Prime-Order)	5.1 MB	$(4/\beta)\lambda^2\lceil\log p\rceil$
size opt. ver.	0.13 MB	
Ours (Lattice)	71 MB	$4\lambda n\lceil\log q\rceil$
[LWYY24] (Lattice)	10 GB	$4\lambda n\lceil\log q\rceil^2$

Our scheme under prime-order groups has a 1/poly correctness and privacy error. Here N is the Paillier modulus, p is the prime-order group size, β denotes digit-decomposition by 2^β (explained below), and n, q are the degree and the modulus of the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$. The estimation for [LWYY24] is optimistic.

Table 7.3: Concrete sizes for public data pd in our non-leveled schemes, and [LWYY24].

7.7 *Concrete Efficiency Analysis*

In this section, we analyze the concrete garbling sizes of our non-leveled schemes, which corresponds to the communication sizes in the protocols $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 7.1, 7.2), $\mathsf{BoolCircEval}^{C,\mathsf{Pri}}$ (Figure 7.10), and $\mathsf{BoolCircEval}^{C,\mathsf{Lat}}$ (Figure 7.8). They consist of two parts: (1) 1-bit per gate in the circuit (during the Eval phase), and (2) public data pd (during the Init phase). We analyze the size of the public data pd in different instantiations below, and summarize them in Table 7.3.

In the following, we use $\lambda = 128$ as the computational security parameter, and $\kappa = 40$ as the statistical security parameter. We also recall the following parameters:

- Under Paillier groups, N and ζ specify the group $\mathbb{Z}^*_{N^{\zeta+1}}$.
- Under prime-order groups, p denotes the group size.
- Under lattices, n and q specify the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$, where the degree n is a power-of-two.

Paillier Groups Instantiation. The public data for aHMAC evaluations contains a λ-bit seed, and $3\lceil \log N \rceil$ elements in $\mathbb{Z}^*_{N^{\zeta+1}}$. (See Lemma 7.2.) The public data for HSS evaluations can share the seed from aHMAC, and additionally contains 4λ elements in $\mathbb{Z}_{N^{\zeta+1}}$. In total:

$$|\mathsf{pd}^{\mathsf{Pai}}| = \lambda + (3\lceil \log N \rceil + 4\lambda) \cdot (\zeta + 1) \cdot \lceil \log N \rceil.$$

An optimization to reduce this size is to compress the public data of aHMAC. As long as the order of the sub-group generated by $(1 + N)$ is sufficiently larger than the secret exponent s, i.e., $N^\zeta \gg |s|$, we can compress the public data from consisting $3\lceil \log N \rceil$ elements to 3 elements:

$$g^{r^*},\ g^{r^* s},\ g^{r^* s^2} \cdot (1 + N)^s, \text{ where } r^* = \sum_i \mathbf{r}[i], \text{ and } \mathbf{r} \leftarrow [N]^{\lceil \log N \rceil}.$$

Note that the compressed version of pd can be derived from the original, hence is still secure. Furthermore, if aggressively assuming the secret exponent in CP-DDH only needs to have 2λ [12] instead of $\lceil \log N \rceil$ bits, it suffices to set $\zeta = 1$ to guarantee $N^\zeta = N \gg 2\lambda$. In total

$$|\mathsf{pd}^{\mathsf{Pai}}| = \lambda + (3 + 4\lambda) \cdot 2 \cdot \lceil \log N \rceil.$$

// w/ compressed aHMAC pd and small exponents.

Concretely, we set the Paillier modulus N to have 3072 bits (which is believed to provides 128 bits of security), which gives $|\mathsf{pd}^{\mathsf{Pai}}| = 0.38\text{MB}$.

Prime-Order Groups Instantiation. The public data for aHMAC evaluations contains a λ-bit seed, and $3\lceil \log p \rceil$ elements in $\mathbb{Z}_p$. (See Lemma 7.6.) The public data for HSS evaluations can share the seed from aHMAC, and additionally contains $2\lambda + 2\lambda\lceil \log p \rceil$ elements in $\mathbb{Z}_p$. (See Lemma 7.9.) In total:

$$|\mathsf{pd}^{\mathsf{Pri}}| = \lambda + (3\lceil \log p \rceil + 2\lambda + 2\lambda\lceil \log p \rceil) \cdot \lceil \log p \rceil.$$

We can reduce this size by similarly assuming the secret exponent only needs 2λ instead of $\lceil \log p \rceil$ bits. Furthermore, we can considering digit-decomposition instead of bit-decomposition

---

[12] We estimate a 2λ-bit exponent to have λ-bit security following the estimation for small-exponent ElGamal in [BGI17].

of the secret s. When using a base 2^β, the public data for aHMAC now only needs to contain $6\lambda/\beta$ elements, and the public data for HSS only needs to contain $2\lambda + (4/\beta)\lambda^2$ elements. [13] A final optimization is to use a random oracle (RO) to obtain the first elements in all ElGamal ciphertexts for free, as suggested in [BGI16], which reduces the public data for HSS by a factor of 2. In total:

$$|\mathsf{pd}^{\mathsf{Pri}}| = \lambda + \big((6/\beta + 1)\lambda + (2/\beta)\lambda^2\big) \cdot \lceil \log p \rceil.$$

// w/ small exponents, digit decomposition, and RO.

Concretely, we consider two settings. First, optimizing for computation time, we follow the optimized implementation from [BGI17] to use "conversion friendly" primes for p. As noted there, compared to a general prime, such conversion friendly primes needs to have a 50% larger bit-length to provide a similar level of security. Therefore, we estimate $\lceil \log p \rceil = 5000$ to provide 128 bits of security. We also follow [BGI17] to set $\beta = 4$, which gives $|\mathsf{pd}^{\mathsf{Pri}}| = 5.07$MB.

Second, optimizing for garbling size, we choose elliptic curves of 256-bit as the prime-order group, and more aggressively set $\beta = 8$, which gives $|\mathsf{pd}^{\mathsf{Pri}}| = 0.13$MB.

Lattice Instantiation. The public data for aHMAC evaluations contains a λ-bit seed, and 3 elements in $\mathcal{R}_q$. (See Lemma 7.11.) The public data for HSS evaluations can share the seed from aHMAC, and additionally contains 4λ elements in $\mathcal{R}_q$. (See Lemma 7.14.) In total:

$$|\mathsf{pd}^{\mathsf{Lat}}| = \lambda + (3 + 4\lambda) \cdot n \cdot \lceil \log q \rceil.$$

Concretely, we follow [BKS19] to use uniform ternary secrets with coefficients from $\{0, -1, 1\}$, and rounded Gaussian error distributions with parameter $\sigma = 8/\sqrt{2\pi}$. We choose a modulus with $\lceil \log q \rceil = 142$ bits, and the polynomial ring with degree $n = 2^{13} = 8192$. These settings are estimated [14] to achieve 128 bits of security, and a correctness error 2^{-40}. We get $|\mathsf{pd}^{\mathsf{Lat}}| = 71.42$MB.

[13] As a consequence of using digit decomposition, the computation cost of our scheme will increase (by a at least a factor of 2^β).

[14] Using the LWE security estimator: https://github.com/malb/lattice-estimator

Comparing with the Scheme of [LWYY24] Based on FHE. The scheme of [LWYY24] is based on the GSW [GSW13] fully homomorphic encryption (FHE) scheme (and assuming its KDM-security). Under the RLWE version of GSW, the garbling material contains 1 bit per gate, a λ-bit seed, public parameters pp, and λ FHE ciphertexts. We refer to the seed, pp and the ciphertexts as the public data $\mathsf{pd}^{\mathsf{GSW}}$ in this scheme. In more detail, pp consists of $2 + 8\lceil \log q \rceil$ elements in $\mathcal{R}_q$, and each ciphertext consists of $4\lceil \log q \rceil$ elements in $\mathcal{R}_q$. In total:

$$|\mathsf{pd}^{\mathsf{GSW}}| = \lambda + (2 + 8\lceil \log q \rceil + 4\lceil \log q \rceil \lambda) \cdot n \cdot \lceil \log q \rceil.$$

Compared to our lattice instantiation, the public data in [LWYY24] is asymptotically greater by a factor of $\log q$, assuming the polynomial ring degree n and the modulus q being equal.

In the GSW FHE scheme, the modulus q not only needs to satisfy a similar set of constraints to our lattice instantiation, but also needs to support homomorphic evaluations of a low-depth PRG. Therefore, we expect concretely the scheme of [LWYY24] needs a much larger q than ours, and consequently also a larger n, to achieve 128 bits of estimated security. Optimistically, under the same settings of $\lceil \log q \rceil = 142$ and $n = 2^{13} = 8192$, we get $|\mathsf{pd}^{\mathsf{GSW}}| = 10.00\text{GB}$.

Protocol $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$. It uses the following ingradients:

- aHMAC evaluation procedures EvalKey, EvalTag over bounded integers by $B = 2$, and public data generation procedure $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}$ under Paillier groups; (See Lemma 7.3;)
- HSS evaluation procedures $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$ and public data generation procedure $\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}$ under paillier groups; (See Lemma 7.5;)
- a PRF : $\{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ in NC1. [a]

Inputs: P_G holds a vector $\mathbf{x} \in \{0,1\}^{\ell_x}$, while P_E holds notinog.

Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \{0,1\}^{\ell_z}$.

- Init :

  1. P_G sends public data pd to the evaluator P_E.

$$\zeta := 2,\ \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta),$$
$$s \leftarrow [N],\ \mathsf{sk} \leftarrow \{0,1\}^\lambda$$
$$\mathsf{pd} := \big(\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, s), \mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s, \mathsf{sk})\big). \tag{7.3}$$

  2. P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle s\overline{\mathbf{x}}\rangle_1$ to P_E. [b]

$$\overline{\mathbf{x}} = \mathbf{x} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(C)),$$
$$\langle s\overline{\mathbf{x}}\rangle_1 := s\overline{\mathbf{x}} + \langle s\overline{\mathbf{x}}\rangle_0 \text{ (over } \mathbb{Z}\text{), where } \langle s\overline{\mathbf{x}}\rangle_0 \leftarrow [N \cdot \lambda^{\omega(1)}]^{\ell_x}.$$

---

[a] With a bound on $|C|$, the PRF can be replaced with a PRG where each bit can be evaluated in NC1.

[b] $(\overline{\mathbf{x}}, \langle s\overline{\mathbf{x}}\rangle_1)$ jointly is what we call input shares in the overview text.

Figure 7.1: 2PC for Boolean circuits under Paillier groups, the Init phase.

Protocol $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ **Continued**

- Eval : P_G, P_E evaluate gates $\mathsf{g} \in C$ in the topological order while maintaining the following invariant:

  1. P_G, P_E jointly hold additive shares $\langle s\overline{\mathbf{x}}_{\mathsf{g}} \rangle$, where $\overline{\mathbf{x}}_{\mathsf{g}}$ are masked input wire values to the gate g

  $$\overline{\mathbf{x}}_{\mathsf{g}} = \mathbf{x}_{\mathsf{g}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(\mathsf{g})). \tag{7.4}$$

  2. P_E holds the masked wire values $\overline{\mathbf{x}}_{\mathsf{g}}$.

  To evaluate the gate g, P_G, P_E jointly call the sub-protocol $\mathsf{BoolGateEval}$.

  $$\begin{aligned}&(P_G : \langle s\overline{z}_{\mathsf{g}} \rangle_0), (P_E : \langle s\overline{z}_{\mathsf{g}} \rangle_1, \overline{z}_{\mathsf{g}}) \\ &\qquad \leftarrow \mathsf{BoolGateEval}^{C,\mathsf{g}} \big((P_G : \mathsf{pd}, \langle s\overline{\mathbf{x}}_{\mathsf{g}} \rangle_0), (P_E : \mathsf{pd}, \langle s\overline{\mathbf{x}}_{\mathsf{g}} \rangle_1, \overline{\mathbf{x}}_{\mathsf{g}})\big)\end{aligned}$$

- Final : P_G sends masks $\mathsf{PRF}(\mathsf{sk}, \mathsf{OutWires}(C))$ mod 2 on all output wires to P_E, who then recovers the output $\mathbf{z}$ by removing the masks mod2. [a]

[a] The final message from P_G to P_E can be avoided via an optimization: let $\mathsf{BoolGateEval}$ compute z, instead of masked $\overline{z}$, for values on the output wires.

Figure 7.2: 2PC for Boolean circuits, the $\mathsf{Eval}, \mathsf{Final}$ phases.

Sub-protocol $\mathsf{BoolGateEval}^{C,\mathsf{g}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean gate $\mathsf{g} \in C$.

Inputs: P_G, P_E both hold public data $\mathsf{pd} = (\mathsf{aHMAC.pd}, \mathsf{HSS.pd}_{\mathsf{sk}})$ (as defined in Equation 7.3), and jointly hold additive shares $\langle s\overline{\mathbf{x}}\rangle$, where $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$ is a masked input vector. P_E additionally holds the vector $\overline{\mathbf{x}}$.

Outputs: P_G, P_E jointly output additive shares $\langle s\overline{z}\rangle$, where $\overline{z} \in \{0,1\}$ is the masked output. P_E additionally holds the bit $\overline{z}$.

- P_G, P_E obtain additive shares $\langle s\overline{z}\rangle$ and $\langle \overline{z}\rangle$ through local computations, where

$$\overline{z} := z \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(g)),\ z := \mathsf{g}(\mathbf{x}),\ \mathbf{x} := \overline{\mathbf{x}} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(g)). \quad (7.7)$$

  Let $C_{\mathbf{v}}$ and $C_{\mathsf{g},\mathbf{v}}$ be arithmetic and Boolean circuits specified in Fact 1 and Equation 7.6, respectively. Further define $C_{\mathsf{g}} := (\ldots, C_{\mathsf{g},\mathbf{v}}, \ldots)$.

  1. P_G, P_E locally runs $\mathsf{EvalKey}, \mathsf{EvalTag}$, respectively, to obtain additive shares $\langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle$ and $\langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle$ for all $\mathbf{v} \in \{0,1\}^{\ell_x}$.

$$P_G : \langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0 \leftarrow \mathsf{EvalKey}(\mathsf{aHMAC.pd}, C_{\mathbf{v}}, \langle s\overline{\mathbf{x}}\rangle_0), \qquad \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0 \leftarrow 0$$
$$P_E : \langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{aHMAC.pd}, C_{\mathbf{v}}, \langle s\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}), \quad \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1 \leftarrow C_{\mathbf{v}}(\overline{\mathbf{x}}).$$

  2. P_G, P_E locally runs $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$, respectively, to obtain additive shares $\langle s\overline{z}\rangle$ and $\langle \overline{z}\rangle$.

$$P_G : (\langle s\overline{z}\rangle_0, \langle \overline{z}\rangle_0) \leftarrow \mathsf{ExtEval}_0(\mathsf{HSS.pd}_{\mathsf{sk}}, C_{\mathsf{g}}, \left\{\langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0, \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0\right\}_{\mathbf{v}})$$
$$P_E : (\langle s\overline{z}\rangle_1, \langle \overline{z}\rangle_1) \leftarrow \mathsf{ExtEval}_1(\mathsf{HSS.pd}_{\mathsf{sk}}, C_{\mathsf{g}}, \left\{\langle s \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1, \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1\right\}_{\mathbf{v}}).$$

- P_G sends a bit $b := \langle \overline{z}\rangle_0 \bmod 2$ to P_E, who can then locally recover $\overline{z}$.

$$\overline{z} := \langle \overline{z}\rangle_1 - b \bmod 2.$$

Figure 7.3: 2PC subprotocol for $O(\log \lambda)$-ary Boolean gates.

Protocol $\mathsf{LBoolCircEval}^{C,\mathsf{Pai}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$. It uses the following ingradients:

- aHMAC *leveled* evaluation procedures $\mathsf{EvalKey}^{d_{\text{Ind}}}, \mathsf{EvalTag}^{d_{\text{Ind}}}$ for bounded depth computations by $d_{\text{Ind}} = O(\log\log\lambda)$ [a] over bounded integers by $B = 2$, and public data generation procedure $\mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}$ under Paillier groups; (See Lemma 7.4;)
- HSS evaluation procedures $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$ and public data generation procedure $\mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}$ under paillier groups; (See Lemma 7.5;)
- a $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ in NC1.

Inputs: P_G holds a vector $\mathbf{x} \in \{0,1\}^{\ell_x}$, while P_E holds notinog.

Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \{0,1\}^{\ell_z}$.

- Init : Let $d_C = \text{Depth}(C)$, and $d = d_C \cdot d_{\text{Ind}}$.

  1. P_G sends public data pd to the evaluator P_E.

$$
\begin{aligned}
&\zeta := 2,\ \mathsf{pp} = (N, \zeta) \leftarrow \mathsf{Pai.Gen}(1^\lambda, 1^\zeta),\\
&\mathbf{s} \leftarrow [N]^{d+1},\ \mathbf{k} \leftarrow [N]^{d_C},\ \mathsf{sk} \leftarrow \{0,1\}^\lambda\\
&\quad // \text{ For short, write } s^{(t)} = \mathbf{s}[t \cdot d_{\text{Ind}}],\ s^{(t)}_{\mathsf{end}} = \mathbf{s}[(t+1)\cdot d_{\text{Ind}} - 1].\\
&\quad \forall j \in [d], \quad \mathsf{aHMAC.pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}[j], \mathbf{s}[j+1]),\\
&\quad \forall t \in [d_C], \quad \mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}} \leftarrow \mathsf{aHMAC}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, s^{(t)}_{\mathsf{end}}, \mathbf{k}[t]),\\
&\quad \forall t \in [d_C], \quad \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}} \leftarrow \mathsf{HSS}^{\mathsf{Pai}}.\mathsf{pd}(\mathsf{pp}, \mathbf{k}[t], \mathsf{sk}\|\mathsf{Bits}(s^{(t+1)})),\\
&\mathsf{pd} := \left(\{\mathsf{aHMAC.pd}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}}, \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}}\}_{t\in[d_C]}\right).
\end{aligned}
\tag{7.10}
$$

  2. Let $s = \mathbf{s}[0]$. P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle s\overline{\mathbf{x}}\rangle_1$ to P_E as in $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 7.1).

---

[a] d_{Ind} is the depth of the indicator arithmetic circuit over $O(\log\lambda)$ inputs (Fact 1).

Figure 7.4: Leveled 2PC for Boolean circuits under Paillier groups, the Init.

Protocol $\mathsf{LBoolCircEval}^{C,\mathsf{Pai}}$ **Continued**

- Eval : P_G, P_E evaluate gates $\mathsf{g} \in C$ (at depth t) in the topological order while maintaining the following invariant. (We write $s^{(t)} = \mathbf{s}[t \cdot d_{\mathrm{Ind}}]$ for short.)

  1. P_G, P_E jointly hold additive shares $\langle s^{(t)}\overline{\mathbf{x}}_{\mathsf{g}}\rangle$, where $\overline{\mathbf{x}}_{\mathsf{g}}$ are masked input wire values to the gate g as in $\mathsf{BoolCircEval}^C$ (Equation 7.4).
  2. P_E holds the masked wire values $\overline{\mathbf{x}}_{\mathsf{g}}$.

  To evaluate the gate g, P_G, P_E call the sub-protocol $\mathsf{LBoolGateEval}$.

$$(P_G : \langle s^{(t+1)}\overline{z}_{\mathsf{g}}\rangle_0), (P_E : \langle s^{(t+1)}\overline{z}_{\mathsf{g}}\rangle_1, \overline{z}_{\mathsf{g}}) \leftarrow \mathsf{LBoolGateEval}^{C,\mathsf{g}}\left((P_G : \mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}_{\mathsf{g}}\rangle_0), (P_E : \mathsf{pd}, \langle s^{(t)}\overline{\mathbf{x}}_{\mathsf{g}}\rangle_1, \overline{\mathbf{x}}_{\mathsf{g}})\right).$$

  Then, for every gate g' (at depth $t' > t+1$) taking z as an input, P_G, P_E obtain shares $\langle s^{(t')}\overline{z}\rangle$ through local computations.

$$\mathsf{diff} := (t' - t - 1) \cdot d_{\mathrm{Ind}},\ \mathsf{pd}_{\mathsf{diff}} := \{\mathsf{aHMAC.pd}^{((t+1)\cdot d_{\mathrm{Ind}}+j)}\}_{j \in [\mathsf{diff}+1]}$$
$$P_G : \langle s^{(t')}\overline{z}\rangle_0 \leftarrow \mathsf{EvalKey}_0^{\mathsf{diff}}(\mathsf{pd}_{\mathsf{diff}}, C_{\mathsf{id}}, \langle s^{(t+1)} \cdot \overline{z}\rangle_0),$$
$$P_E : \langle s^{(t')}\overline{z}\rangle_1 \leftarrow \mathsf{EvalTag}_1^{\mathsf{diff}}(\mathsf{pd}_{\mathsf{diff}}, C_{\mathsf{id}}, \langle s^{(t+1)} \cdot \overline{z}\rangle_1, \overline{z}),$$

  where C_{id} (with depth $= \mathsf{diff}$) computes the identity function.

- Final : The same as $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 7.2).

Figure 7.5: Leveled 2PC for Boolean circuits under Paillier groups, the $\mathsf{Eval}, \mathsf{Final}$ phases.

Sub-protocol $\mathsf{LBoolGateEval}^{C,\mathsf{g}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean gate $\mathsf{g} \in C$.

Inputs: P_G, P_E both hold public data $\mathsf{pd} = \{\mathsf{aHMAC.pd}^{(j)}\}, \{\mathsf{aHMAC.pd}_{\mathbf{k}}^{(t)}, \mathsf{HSS.pd}_{\mathsf{sk},\mathbf{s}}^{(t)}\}$ (as defined in Equation 7.10), and jointly hold additive shares $\langle s^{(t)}\overline{\mathbf{x}}\rangle$, where $\overline{\mathbf{x}} \in \{0,1\}^{\ell_x}$ is a masked input vector. P_E additionally holds the vector $\overline{\mathbf{x}}$.

Outputs: P_G, P_E jointly output additive shares $\langle s^{(t+1)}\overline{z}\rangle$, where $\overline{z} \in \{0,1\}$ is the masked output. P_E additionally outputs the bit $\overline{z}$.

- P_G, P_E obtain additive shares $\langle s^{(t)}\overline{z}\rangle$ and $\langle \overline{z}\rangle$ through local computations, where $\overline{z}$ is defined as in $\mathsf{BoolGateEval}$ (Equation 7.7). Let $C_{\mathbf{v}}$ and $C_{\mathsf{g},\mathbf{v}}$ be arithmetic and Boolean circuits specified in Fact 1 and Equation 7.6, respectively. Further define $C_{\mathsf{g}} := (\ldots, C_{\mathsf{g},\mathbf{v}}, \ldots)$, and $C_{\mathsf{g}}^{(i)} : \{0,1\}^{\ell_x} \times \{0,1\}^{\lceil \log \ell_x \rceil} \to \{0,1\}^{2^{\ell_x} \cdot \lceil \log \ell \rceil}$.

$$C_{\mathsf{g}}^{(i)}(\mathsf{sk}, s^{(t+1)}) = (\ldots, C_{\mathsf{g},\mathbf{v}}(\mathsf{sk}) \cdot \mathsf{Bits}(s^{(t+1)})[i], \ldots)_{\mathbf{v} \in \{0,1\}^{\ell_x}}.$$

  1. P_G, P_E locally runs $\mathsf{EvalKey}^{d_{\mathrm{Ind}}}, \mathsf{EvalTag}^{d_{\mathrm{Ind}}}$, respectively, to obtain additive shares $\langle \mathbf{k}[t] \cdot C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle$ and $\langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle$ for all $\mathbf{v} \in \{0,1\}^{\ell_x}$.

$$\mathsf{pd}_{\mathrm{Ind}} := \{\mathsf{aHMAC.pd}^{(t \cdot d_{\mathrm{Ind}} + j)}\}_{j \in [d_{\mathrm{Ind}}]} \cup \{\mathsf{aHMAC.pd}_{\mathbf{k}}^{(t)}\}$$
$$P_G : \langle \mathbf{k}[t] C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0 \leftarrow \mathsf{EvalKey}^{d_{\mathrm{Ind}}}(\mathsf{pd}_{\mathrm{Ind}}, C_{\mathbf{v}}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_0), \qquad \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0 \leftarrow 0$$
$$P_E : \langle \mathbf{k}[t] C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1 \leftarrow \mathsf{EvalTag}^{d_{\mathrm{Ind}}}(\mathsf{pd}_{\mathrm{Ind}}, C_{\mathbf{v}}, \langle s^{(t)}\overline{\mathbf{x}}\rangle_1, \overline{\mathbf{x}}), \quad \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_1 \leftarrow C_{\mathbf{v}}(\overline{\mathbf{x}}).$$

  2. P_G, P_E locally runs $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$, respectively, to obtain additive shares $\langle s^{(t+1)}\overline{z}\rangle$ and $\langle \overline{z}\rangle$. ($P_E$'s computation is analogous P_G's.)

$$P_G : (_, \langle \overline{z}\rangle_0) \leftarrow \mathsf{ExtEval}_0(\mathsf{HSS.pd}_{\mathsf{sk},\mathbf{s}}^{(t)}, C_{\mathsf{g}}, \{\langle \mathbf{k}[t] C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0, \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0\}_{\mathbf{v}}),$$
$$(_, \langle \overline{z} \cdot \mathsf{Bits}(s^{(t+1)})[i]\rangle_0)$$
$$\leftarrow \mathsf{ExtEval}_0(\mathsf{HSSpd}_{\mathsf{sk},\mathbf{s}}^{(t)}, C_{\mathsf{g}}^{(i)}, \{\langle \mathbf{k}[t] C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0, \langle C_{\mathbf{v}}(\overline{\mathbf{x}})\rangle_0\}_{\mathbf{v}}),$$
$$\langle s^{(t+1)}\overline{z}\rangle_0 \leftarrow \mathsf{BitComp}\left(\langle \overline{z} \cdot \mathsf{Bits}(s^{(t+1)})[i]\rangle_0\right) \text{ over } \mathbb{Z}.$$

Figure 7.6: Leveled 2PC for Boolean gates.

Sub-protocol $\mathsf{LBoolGateEval}^{C,\mathbf{g}}$, Cont.

- P_G sends a bit $b := \langle \overline{z} \rangle_0 \bmod 2$ to P_E, who can then locally recover $\overline{z}$.

$$\overline{z} := \langle \overline{z} \rangle_1 - b \bmod 2.$$

Figure 7.7: Leveled 2PC for Boolean gates, continued.

Protocol $\mathsf{BoolCircEval}^{C,\mathsf{Lat}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$. It uses the following ingradients:

- public parameters $\mathsf{pp}^{\mathsf{Lat}} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ specified in Section 7.5.3;
- aHMAC evaluation procedures $\mathsf{EvalKey}, \mathsf{EvalTag}$ over bounded integers by $B = 2$, and public data generation procedure $\mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}$ under lattices; (See Lemma 7.12;)
- HSS evaluation procedures $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$ and public data generation procedure $\mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}$ under lattices; (See Lemma 7.14;)
- a $\mathsf{PRF} : \{0,1\}^{\lambda} \times \{0,1\}^{\lambda} \to \{0,1\}$ in NC1.

Inputs: P_G holds a vector $\mathbf{x} \in \{0,1\}^{\ell_x}$, while P_E holds notinog.

Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \{0,1\}^{\ell_z}$.

- Init :

  1. P_G sends public data pd to the evaluator P_E.

$$s \leftarrow \mathcal{D}_{\mathsf{sk}},\ \mathsf{sk} \leftarrow \{0,1\}^{\lambda}$$
$$\mathsf{pd} := \big(\mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, s, s), \mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, s, \mathsf{sk})\big). \tag{7.15}$$

  2. P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle s\overline{\mathbf{x}} \rangle_1$ to P_E.

$$\overline{\mathbf{x}} = \mathbf{x} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(C)),$$
$$\langle s\overline{\mathbf{x}} \rangle_1 := s\overline{\mathbf{x}} + \langle s\overline{\mathbf{x}} \rangle_0 \text{ (over } \mathcal{R}\text{), where } \langle s\overline{\mathbf{x}} \rangle_0 \leftarrow \mathcal{R}^{\ell_x}_{\lambda^{\omega(1)}}.$$

- $\mathsf{Eval}, \mathsf{Final}$ phases are the same as $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 7.2).

Figure 7.8: 2PC for Boolean circuits under lattices.

Protocol $\mathsf{LBoolCircEval}^{C,\mathsf{Lat}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$. It uses the following ingradients:

- public parameters $\mathsf{pp}^{\mathsf{Lat}} = (\mathcal{R}, p, q, \mathcal{D}_{\mathsf{err}}, \mathcal{D}_{\mathsf{sk}})$ specified in Section 7.5.3;
- aHMAC *leveled* evaluation procedures $\mathsf{EvalKey}^{d_{\text{Ind}}}, \mathsf{EvalTag}^{d_{\text{Ind}}}$ for bounded depth computations by $d_{\text{Ind}} = O(\log\log\lambda)$ over bounded integers by $B = 2$, and public data generation procedure $\mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}$ under lattices; (See Lemma 7.13;)
- HSS evaluation procedures $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$ and public data generation procedure $\mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}$ under lattices; (See Lemma 7.14;)
- a $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ in NC1.

Inputs: P_G holds a vector $\mathbf{x} \in \{0,1\}^{\ell_x}$, while P_E holds notinog.

Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \{0,1\}^{\ell_z}$.

- Init : Let $d_C = \text{Depth}(C)$, and $d = d_C \cdot d_{\text{Ind}}$.

  1. P_G sends public data pd to the evaluator P_E.

$$\mathbf{s} \leftarrow \mathcal{D}_{\mathsf{sk}}^{d+1},\ \mathbf{k} \leftarrow \mathcal{D}_{\mathsf{sk}}^{d_C},\ \mathsf{sk} \leftarrow \{0,1\}^\lambda$$

$$\text{// For short, write } s^{(t)} = \mathbf{s}[t \cdot d_{\text{Ind}}],\ s_{\mathsf{end}}^{(t)} = \mathbf{s}[(t+1)\cdot d_{\text{Ind}} - 1].$$

$$\forall j \in [d],\quad \mathsf{aHMAC.pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, \mathbf{s}[j], \mathbf{s}[j+1]),$$

$$\forall t \in [d_C],\quad \mathsf{aHMAC.pd}_{\mathbf{k}}^{(t)} \leftarrow \mathsf{aHMAC}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, s_{\mathsf{end}}^{(t)}, \mathbf{k}[t]), \tag{7.18}$$

$$\forall t \in [d_C],\quad \mathsf{HSS.pd}_{\mathsf{sk},\mathbf{s}}^{(t)} \leftarrow \mathsf{HSS}^{\mathsf{Lat}}.\mathsf{pd}(\mathsf{pp}^{\mathsf{Lat}}, \mathbf{k}[t], \mathsf{sk}\|\mathsf{Bits}(s^{(t+1)})),$$

$$\mathsf{pd} := \left(\{\mathsf{aHMAC.pd}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.pd}_{\mathbf{k}}^{(t)}, \mathsf{HSS.pd}_{\mathsf{sk},\mathbf{s}}^{(t)}\}_{t\in[d_C]}\right).$$

  2. Let $s = \mathbf{s}[0]$. P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle s\overline{\mathbf{x}}\rangle_1$ to P_E as in $\mathsf{BoolCircEval}^{C,\mathsf{Lat}}$ (Figure 7.8).

- $\mathsf{Eval}, \mathsf{Final}$ phases are the same as $\mathsf{LBoolCircEval}^{C,\mathsf{Pai}}$ (Figure 7.5).

Figure 7.9: Leveled 2PC for Boolean circuits under lattices.

Protocol $\mathsf{BoolCircEval}^{C,\mathsf{Pri}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$. It uses the following ingradients:

- aHMAC evaluation procedures, with error bound $\delta = 1/(\mathrm{poly}(\lambda) \cdot |C|)$, $\mathsf{EvalKey}, \mathsf{EvalTag}$ over bounded integers by $B = 2$, and public data generation procedure $\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}$ under prime-order groups; (See Lemma 7.7;)
- HSS evaluation procedures $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$ and public data generation procedure $\mathsf{HSS}^{\mathsf{EG}}.\mathsf{pd}$ under ElGamal; (See Lemma 7.9;)
- a $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ in NC1.

Inputs: P_G holds a vector $\mathbf{x} \in \{0,1\}^{\ell_x}$, while P_E holds notinog.

Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \{0,1\}^{\ell_z}$.

- Init :

  1. P_G sends public data pd to the evaluator P_E.

$$\begin{aligned} &\mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda), \\ &s \leftarrow \mathbb{Z}_p,\ \mathbf{s} := \mathsf{Bits}(s),\ \mathsf{sk} \leftarrow \{0,1\}^\lambda \\ &\mathsf{pd} := \big(\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, s), \mathsf{HSS}^{\mathsf{EG}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}, \mathsf{sk})\big). \end{aligned} \tag{7.23}$$

  2. P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_1$ to P_E.

$$\begin{aligned} \overline{\mathbf{x}} &= \mathbf{x} \oplus \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(C)), \\ \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_1 &:= \mathbf{s} \otimes \overline{\mathbf{x}} + \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_0 \text{ (over } \mathbb{Z}), \\ &\quad \text{where } \langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_0 \leftarrow [\lambda^{\omega(1)}]^{\lceil 3 \log p \rceil \times \ell_x}. \end{aligned}$$

- $\mathsf{Eval}, \mathsf{Final}$ phases are the same as $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 7.2), except for syntactical changes from using dot products $\cdot$ when multiplying with a scalar s to using tensor products $\otimes$ when multiplying with a vector $\mathbf{s}$.

Figure 7.10: 2PC for Boolean circuits under prime-order groups.

Protocol $\mathsf{LBoolCircEval}^{C,\mathsf{Pri}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate a Boolean circuit $C : \{0,1\}^{\ell_x} \to \{0,1\}^{\ell_z}$. It uses the following ingradients:

- aHMAC *leveled* evaluation procedures, with error bound $\delta = 1/(\text{poly}(\lambda) \cdot |C|)$, $\mathsf{EvalKey}^{d_{\text{Ind}}}, \mathsf{EvalTag}^{d_{\text{Ind}}}$ for bounded depth computations by $d_{\text{Ind}} = O(\log\log\lambda)$ over bounded integers by $B = 2$, and public data generation procedure $\mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}$ under lattices; (See Lemma 7.8;)
- HSS evaluation procedures $\mathsf{ExtEval}_0, \mathsf{ExtEval}_1$ and public data generation procedure $\mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{pd}$ under BHHO; (See Lemma 7.10;)
- a $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ in NC1.

Inputs: P_G holds a vector $\mathbf{x} \in \{0,1\}^{\ell_x}$, while P_E holds notinog.

Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \{0,1\}^{\ell_z}$.

- Init : Let $d_C = \text{Depth}(C)$, and $d = d_C \cdot d_{\text{Ind}}$.

  1. P_G sends public data pd to the evaluator P_E.

$$
\begin{aligned}
&\mathsf{pp} = (G, p, g) \leftarrow \mathsf{Pri.Gen}(1^\lambda),\ \mathbf{K} \leftarrow \{0,1\}^{d_C \times \lceil 3\log p \rceil},\ \mathsf{sk} \leftarrow \{0,1\}^\lambda \\
&\mathbf{S} \in \{0,1\}^{(d+1)\times\lceil \log p \rceil} \text{ where } s_j \leftarrow \mathbb{Z}_p,\ \mathbf{S}[j] := \mathsf{Bits}(s_j), \\
&// \text{ For short, write } \mathbf{s}^{(t)} = \mathbf{S}[t \cdot d_{\text{Ind}}],\ \mathbf{s}^{(t)}_{\mathsf{end}} = \mathbf{S}[(t+1)\cdot d_{\text{Ind}} - 1]. \\
&\forall j \in [d], \quad \mathsf{aHMAC.pd}^{(j)} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{S}[j], \mathbf{S}[j+1]), \\
&\forall t \in [d_C], \quad \mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}} \leftarrow \mathsf{aHMAC}^{\mathsf{Pri}}.\mathsf{pd}(\mathsf{pp}, \mathbf{s}^{(t)}_{\mathsf{end}}, \mathbf{K}[t]), \qquad (7.26) \\
&\forall t \in [d_C], \quad \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}} \leftarrow \mathsf{HSS}^{\mathsf{BHHO}}.\mathsf{pd}(\mathsf{pp}, \mathbf{K}[t], \mathsf{sk} \| \mathsf{Bits}(\mathbf{s}^{(t+1)})), \\
&\mathsf{pd} := \big(\{\mathsf{aHMAC.pd}^{(j)}\}_{j\in[d]}, \{\mathsf{aHMAC.pd}^{(t)}_{\mathbf{k}}, \mathsf{HSS.pd}^{(t)}_{\mathsf{sk},\mathbf{s}}\}_{t\in[d_C]}\big).
\end{aligned}
$$

  2. Let $\mathbf{s} = \mathbf{S}[0]$. P_G sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle \mathbf{s} \otimes \overline{\mathbf{x}} \rangle_1$ to P_E as in $\mathsf{BoolCircEval}^{C,\mathsf{Pri}}$ (Figure 7.10).

Figure 7.11: Leveled 2PC for Boolean circuits under prime-order groups.

Protocol $\mathsf{LBoolCircEval}^{C,\mathsf{Pri}}$, Cont.

- $\mathsf{Eval}, \mathsf{Final}$ phases are the same as $\mathsf{LBoolCircEval}^{C,\mathsf{Pai}}$ (Figure 7.5) except for syntactical changes from using dot products $\cdot$ when multiplying with a scalar s to using tensor products $\otimes$ when multiplying with a vector $\mathbf{s}$.

Figure 7.12: Leveled 2PC for Boolean circuits under prime-order groups, continued.

Protocol $\mathsf{ArithCircEval}^{C,\mathsf{Pai}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate an crithmetic circuit over $\mathbb{Z}_R$, $C : \mathbb{Z}_R^{\ell_x} \to \mathbb{Z}_R^{\ell_z}$. It uses the same ingradients as $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 7.1), except with a different PRF:

- a PRF : $\{0,1\}^\lambda \times \{0,1\}^\lambda \to [R^2 \cdot \lambda^{\omega(1)}]$ in NC1.

Inputs: P_G holds a vector $\mathbf{x} \in \mathbb{Z}_R^{\ell_x}$, while P_E holds notinog.

Outputs: P_G outputs nothing, while P_E outputs a vector $\mathbf{z} \in \mathbb{Z}_R^{\ell_z}$.

- Init : P_G sends public data pd to P_E computed in the same way as $\mathsf{BoolCircEval}^{C,\mathsf{Pai}}$ (Figure 7.1), then sends masked inputs $\overline{\mathbf{x}}$ and additive shares $\langle s\mathsf{Bits}(\overline{\mathbf{x}})\rangle_1$ to P_E.

$$\overline{\mathbf{x}}[i] = \mathbf{x}[i] + r^{(i)} \bmod R,\ \langle s\mathsf{Bits}(\overline{\mathbf{x}})\rangle := s\mathsf{Bits}(\overline{\mathbf{x}}) + \langle s\mathsf{Bits}(\overline{\mathbf{x}})\rangle_0 \text{ over } \mathbb{Z},$$
$$\text{where } r^{(i)} \leftarrow \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(C)[i]),\ \langle s\mathsf{Bits}(\overline{\mathbf{x}})\rangle_0 \leftarrow [N\lambda^{\omega(1)}]^{\ell_x \cdot \lceil \log R \rceil}.$$

- Eval : P_G, P_E evaluate gates $\mathsf{g} \in C$ in the topological order while maintaining the following invariant:

  1. P_G, P_E jointly hold additive shares $\langle s\mathsf{Bits}(\overline{\mathbf{x}}_\mathsf{g})\rangle$, where $\overline{\mathbf{x}}_\mathsf{g}$ are masked input wire values to the gate g

  $$\overline{\mathbf{x}}_\mathsf{g}[i] = \mathbf{x}_\mathsf{g}[i] + r^{(i)} \bmod R \text{ where } r^{(i)} \leftarrow \mathsf{PRF}(\mathsf{sk}, \mathsf{InWires}(\mathsf{g})[i]). \tag{7.32}$$

  2. P_E holds the masked wire values $\overline{\mathbf{x}}_\mathsf{g}$.

  To evaluate the gate g, P_G, P_E jointly call the sub-protocol $\mathsf{ArithGateEval}$.

$$(P_G : \langle s\mathsf{Bits}(\overline{z}_\mathsf{g})\rangle_0), (P_E : \langle s\mathsf{Bits}(\overline{z}_\mathsf{g}\rangle_1, \overline{z}_\mathsf{g}))$$
$$\leftarrow \mathsf{ArithGateEval}^{C,\mathsf{g}}\left((P_G : \mathsf{pd}, \langle s\mathsf{Bits}(\overline{\mathbf{x}}_\mathsf{g})\rangle_0), (P_E : \mathsf{pd}, \langle s\mathsf{Bits}(\overline{\mathbf{x}}_\mathsf{g})\rangle_1, \overline{\mathbf{x}}_\mathsf{g})\right)$$

- Final : P_G sends masks $\mathsf{PRF}(\mathsf{sk}, \mathsf{OutWire}(\mathsf{g})) \bmod R$ on all output gates $\mathsf{g} \in C$ to P_E, who can then recovers the output $\mathbf{z}$ by removing the masks $\bmod R$.

Figure 7.13: 2PC for arithmetic circuits with large modulus.

Sub-protocol $\mathsf{ArithGateEval}^{C,\mathsf{g}}$

The protocol runs between a garbler P_G and an evaluator P_E, to evaluate an arithmetic gate ($+$ or $\times$) $\mathsf{g} \in C$.

Inputs: P_G, P_E both hold public data $\mathsf{pd} = (\mathsf{aHMAC.pd}, \mathsf{HSS.pd}_{\mathsf{sk}})$ (as defined in Equation 7.3), and jointly hold additive shares $\langle s\mathsf{Bits}(\overline{x})\rangle$, $\langle s\mathsf{Bits}(\overline{y})\rangle$, where $\overline{x}, \overline{y} \in \mathbb{Z}_R$ are masked inputs. P_E additionally holds the values $\overline{x}$, $\overline{y}$.

Outputs: P_G, P_E jointly output additive shares $\langle s\mathsf{Bits}(\overline{z})\rangle$, where $\overline{z} \in \mathbb{Z}_R$ is the masked output. P_E additionally holds the value $\overline{z}$.

- Let $\mathsf{CRT}, \mathsf{CRT}^{-1}$ be functions defined in Equation 7.33, and C^{CRT}, C^{InvCRT} be Boolean circuits implementing them (Equation 7.34).
- P_G, P_E obtain additive shares $\langle s\mathsf{Bits}(\mathsf{CRT}(\overline{x}))\rangle$ through local computations:

$$P_G :\{\langle s\mathsf{Bits}(\overline{x}_i)\rangle_0\}, \leftarrow \mathsf{EvalKey}(\mathsf{aHMAC.pd}, C^{\mathsf{CRT}}, \langle s\mathsf{Bits}(\overline{x})\rangle_0),$$
$$P_E :\{\langle s\mathsf{Bits}(\overline{x}_i)\rangle_1\} \leftarrow \mathsf{EvalTag}(\mathsf{aHMAC.pd}, C^{\mathsf{CRT}}, \langle s\mathsf{Bits}(\overline{x})\rangle_1, \overline{x}),$$

  where $\overline{x}_i := \overline{x} \bmod p_i$. Similarly obtain shares of $\langle s\mathsf{Bits}(\mathsf{CRT}(\overline{y}))\rangle$.
- $\forall i \in [\ell]$, P_G, P_E apply $\mathsf{BoolGateEval}'$ over the shares $\langle s\mathsf{Bits}(\overline{x}_i)\rangle$, $\langle s\mathsf{Bits}(\overline{y}_i)\rangle$.

$$\begin{aligned}(P_G : \langle s\mathsf{Bits}(\overline{z}_i)\rangle_0), (P_E : \langle s\mathsf{Bits}(\overline{z}_i)\rangle_0, \overline{z}_i)\\ \leftarrow \mathsf{BoolGateEval}'^{C,\mathsf{g}}\big((P_G : \mathsf{pd}, \langle s\mathsf{Bits}(\overline{x}_i, \overline{y}_i)\rangle_0,\\ (P_E : \mathsf{pd}, \langle s\mathsf{Bits}(\overline{x}_i, \overline{y}_i)\rangle_1, \overline{x}_i, \overline{y}_i), \big)\end{aligned}$$

  where $\mathsf{BoolGateEval}'$ is a slight variant of $\mathsf{BoolGateEval}$ (Figure 7.3) as explained in Section 7.6.1.

Figure 7.14: 2PC for arithmetic gates.

Sub-protocol $\mathsf{ArithGateEval}^{C,\mathsf{g}}$, Cont.

- P_G, P_E obtain additive shares $\langle s\mathsf{Bits}(\overline{z}')\rangle$ where $\overline{z}' := \mathsf{CRT}^{-1}(\{\overline{z}_i\})$ through local computations.

$$P_G : \langle s\mathsf{Bits}(\overline{z}')\rangle_0, \leftarrow \mathsf{EvalKey}(\mathsf{aHMAC.pd}, C^{\mathsf{InvCRT}}, \{\langle s\mathsf{Bits}(\overline{z}_i)\rangle_0\}),$$
$$P_E : \langle s\mathsf{Bits}(\overline{z}')\rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{aHMAC.pd}, C^{\mathsf{InvCRT}}, \{\langle s\mathsf{Bits}(\overline{z}_i)\rangle_1\}, \{\overline{z}_i\}),$$

Then obtain additive shares $\langle s\mathsf{Bits}(\overline{z})\rangle$ where $\overline{z} := \overline{z}' \bmod R$ through local computations.

$$P_G : \langle s\mathsf{Bits}(\overline{z})\rangle_0, \leftarrow \mathsf{EvalKey}(\mathsf{aHMAC.pd}, \bmod R, \langle s\mathsf{Bits}(\overline{z}')\rangle_0),$$
$$P_E : \langle s\mathsf{Bits}(\overline{z})\rangle_1 \leftarrow \mathsf{EvalTag}(\mathsf{aHMAC.pd}, \bmod R, \langle s\mathsf{Bits}(\overline{z}')\rangle_1, \overline{z}').$$

Figure 7.15: 2PC for arithmetic gates, continued.

BIBLIOGRAPHY

[AAB15] Benny Applebaum, Jonathan Avron, and Christina Brzuska. Arithmetic cryptography: Extended abstract. In Tim Roughgarden, editor, *ITCS 2015: 6th Conference on Innovations in Theoretical Computer Science*, pages 143–151, Rehovot, Israel, January 11–13, 2015. Association for Computing Machinery.

[AARV17] Benny Applebaum, Barak Arkis, Pavel Raykov, and Prashant Nalini Vasudevan. Conditional disclosure of secrets: Amplification, closure, amortization, lower-bounds, and separations. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 727–757, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.

[AB09] Shweta Agrawal and Dan Boneh. Homomorphic MACs: MAC-based integrity for network coding. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 2009: 7th International Conference on Applied Cryptography and Network Security*, volume 5536 of *Lecture Notes in Computer Science*, pages 292–305, Paris-Rocquencourt, France, June 2–5, 2009. Springer Berlin Heidelberg, Germany.

[ABF+19] Benny Applebaum, Amos Beimel, Oriol Farràs, Oded Nir, and Naty Peter. Secret-sharing schemes for general and uniform access structures. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 441–471, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

[ABG+24] Amit Agarwal, Elette Boyle, Niv Gilboa, Yuval Ishai, Mahimna Kelkar, and

Yiping Ma. Compressing unit-vector correlations via sparse pseudorandom generators. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part VIII*, volume 14927 of *Lecture Notes in Computer Science*, pages 346–383, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[ABI+23] Benny Applebaum, Amos Beimel, Yuval Ishai, Eyal Kushilevitz, Tianren Liu, and Vinod Vaikuntanathan. Succinct computational secret sharing. In Barna Saha and Rocco A. Servedio, editors, *55th Annual ACM Symposium on Theory of Computing*, pages 1553–1566, Orlando, FL, USA, June 20–23, 2023. ACM Press.

[ABNP20] Benny Applebaum, Amos Beimel, Oded Nir, and Naty Peter. Better secret sharing via robust conditional disclosure of secrets. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *52nd Annual ACM Symposium on Theory of Computing*, pages 280–293, Chicago, IL, USA, June 22–26, 2020. ACM Press.

[ACC+19] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption standard. Cryptology ePrint Archive, Report 2019/939, 2019.

[ACG24] Abtin Afshar, Jiaqi Cheng, and Rishab Goyal. Leveled fully-homomorphic signatures from batch arguments. Cryptology ePrint Archive, Report 2024/931, 2024.

[ADOS22] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology –*

CRYPTO 2022, Part IV, volume 13510 of *Lecture Notes in Computer Science*, pages 421–452, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[AHI11] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In Bernard Chazelle, editor, *ICS 2011: 2nd Innovations in Computer Science*, pages 45–60, Tsinghua University, Beijing, China, January 7–9, 2011. Tsinghua University Press.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th Annual Symposium on Foundations of Computer Science*, pages 166–175, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.

[AIK05] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 260–274. IEEE Computer Society, 2005.

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 120–129, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 166–184, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer:

How to sell digital goods. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 119–135, Innsbruck, Austria, May 6–10, 2001. Springer Berlin Heidelberg, Germany.

[AJS17] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation for Turing machines: Constant overhead and amortization. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 252–279, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.

[AL19] Leonard Assouline and Tianren Liu. Multi-party PSM, revisited. Cryptology ePrint Archive, Report 2019/657, 2019.

[AL21] Martin R. Albrecht and Russell W. F. Lai. Subtractive sets over cyclotomic rings - limits of Schnorr-like arguments over lattices. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 519–548, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[AMN$^+$18] Nuttapong Attrapadung, Takahiro Matsuda, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Constrained PRFs for NC^1 in traditional groups. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 543–574, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[AN21] Benny Applebaum and Oded Nir. Upslices, downslices, and secret-sharing with complexity of 1.5^n. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in*

Computer Science, pages 627–655, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[App16] Benny Applebaum. Garbling XOR gates "for free" in the standard model. *Journal of Cryptology*, 29(3):552–576, July 2016.

[App17] Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 1–44. Springer International Publishing, 2017.

[ARS+15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.

[ARS24] Damiano Abram, Lawrence Roy, and Peter Scholl. Succinct homomorphic secret sharing. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 301–330, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[Att14] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 557–577, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.

[BCG+17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In Bhavani M.

Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 2105–2122, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[BCG+18] Nir Bitansky, Ran Canetti, Sanjam Garg, Justin Holmgren, Abhishek Jain, Huijia Lin, Rafael Pass, Sidharth Telang, and Vinod Vaikuntanathan. Indistinguishability obfuscation for RAM programs and succinct randomized encodings. *SIAM J. Comput.*, 47(3):1123–1210, 2018.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 387–416, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 896–912, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[BCLN16] Joppe W. Bos, Craig Costello, Patrick Longa, and Michael Naehrig. Selecting elliptic curves for cryptography: an efficiency and security analysis. *Journal of Cryptographic Engineering*, 6(4):259–286, November 2016.

[BCM+19] Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. Garbled neural networks are practical. *IACR Cryptol. ePrint Arch.*, page 338, 2019.

[BDGM20] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate

iO from homomorphic encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 79–109, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

[BF11] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 149–168, Tallinn, Estonia, May 15–19, 2011. Springer Berlin Heidelberg, Germany.

[BFR13] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 863–874, Berlin, Germany, November 4–8, 2013. ACM Press.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 533–556, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 501–519, Buenos Aires, Argentina, March 26–28, 2014. Springer Berlin Heidelberg, Germany.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier

for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 509–539, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany.

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 163–193, Paris, France, April 30 – May 4, 2017. Springer, Cham, Switzerland.

[BGI+18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *ITCS 2018: 9th Innovations in Theoretical Computer Science Conference*, volume 94, pages 21:1–21:21, Cambridge, MA, USA, January 11–14, 2018. Leibniz International Proceedings in Informatics (LIPIcs).

[BGIK22] Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 121–151, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 439–448, Portland, OR, USA, June 14–17, 2015. ACM Press.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005: 2nd Theory of Cryptography*

Conference, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341, Cambridge, MA, USA, February 10–12, 2005. Springer Berlin Heidelberg, Germany.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325, Cambridge, MA, USA, January 8–10, 2012. Association for Computing Machinery.

[BHHO08] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 108–125, Santa Barbara, CA, USA, August 17–21, 2008. Springer Berlin Heidelberg, Germany.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012: 19th Conference on Computer and Communications Security*, pages 784–796, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[BIKK14] Amos Beimel, Yuval Ishai, Ranjit Kumaresan, and Eyal Kushilevitz. On the cryptographic complexity of the worst functions. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 317–342, San Diego, CA, USA, February 24–26, 2014. Springer Berlin Heidelberg, Germany.

[BKN18] Amos Beimel, Eyal Kushilevitz, and Pnina Nissim. The complexity of multiparty PSM protocols and related models. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of

Lecture Notes in Computer Science, pages 287–318, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

[BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 3–33, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

[BKS+21] Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe D. M. de Souza, and Vinodh Gopal. Intel HEXL: Accelerating homomorphic encryption with intel AVX512-IFMA52. Cryptology ePrint Archive, Report 2021/420, 2021.

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 500–532, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

[BLLL23] Marshall Ball, Hanjun Li, Huijia Lin, and Tianren Liu. New ways to garble arithmetic circuits. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 3–34, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[BLW17] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In Serge Fehr, editor, *PKC 2017: 20th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 10175 of *Lecture Notes in Computer Science*, pages 494–524, Amsterdam, The Netherlands, March 28–31, 2017. Springer Berlin Heidelberg, Germany.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd Annual ACM Symposium on Theory of Computing*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[BMR16] Marshall Ball, Tal Malkin, and Mike Rosulek. Garbling gadgets for Boolean and arithmetic circuits. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 565–577, Vienna, Austria, October 24–28, 2016. ACM Press.

[BMZ19] James Bartusek, Fermi Ma, and Mark Zhandry. The distinction between fixed and random generators in group-based assumptions. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 801–830, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

[BRS03] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75, St. John's, Newfoundland, Canada, August 15–16, 2003. Springer Berlin Heidelberg, Germany.

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 264–302, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryp-

tion from (standard) LWE. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic PRFs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 1–30, Warsaw, Poland, March 23–25, 2015. Springer Berlin Heidelberg, Germany.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 280–300, Bengalore, India, December 1–5, 2013. Springer Berlin Heidelberg, Germany.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

[CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained PRFs for NC^1 from LWE. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 446–476, Paris, France, April 30 – May 4, 2017. Springer, Cham, Switzerland.

[CCH+24] Mingyu Cho, Woohyuk Chung, Jincheol Ha, Jooyoung Lee, Eun-Gyeol Oh, and Mincheol Son. FRAST: tfhe-friendly cipher based on random s-boxes. *IACR Trans. Symmetric Cryptol.*, 2024(3):1–43, 2024.

[CCKK21] Jung Hee Cheon, Wonhee Cho, Jeong Han Kim, and Jiseung Kim. Adventures in crypto dark matter: Attacks and fixes for weak pseudorandom functions. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12711 of *Lecture Notes in Computer Science*, pages 739–760, Virtual Event, May 10–13, 2021. Springer, Cham, Switzerland.

[CEMY09] Seung Geol Choi, Ariel Elbaz, Tal Malkin, and Moti Yung. Secure multi-party computation minimizing online rounds. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 268–286, Tokyo, Japan, December 6–10, 2009. Springer Berlin Heidelberg, Germany.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic MACs for arithmetic circuits. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 336–352, Athens, Greece, May 26–30, 2013. Springer Berlin Heidelberg, Germany.

[CFGN14] Dario Catalano, Dario Fiore, Rosario Gennaro, and Luca Nizzardo. Generalizing homomorphic MACs for arithmetic circuits. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 538–555, Buenos Aires, Argentina, March 26–28, 2014. Springer Berlin Heidelberg, Germany.

[CFOR12] Alfonso Cevallos, Serge Fehr, Rafail Ostrovsky, and Yuval Rabani. Unconditionally-secure robust secret sharing with compact shares. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 195–208, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany.

[CFT22] Dario Catalano, Dario Fiore, and Ida Tucker. Additive-homomorphic functional commitments and applications to homomorphic signatures. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part IV*, volume 13794 of *Lecture Notes in Computer Science*, pages 159–188, Taipei, Taiwan, December 5–9, 2022. Springer, Cham, Switzerland.

[CHHK25] Geoffroy Couteau, Carmit Hazay, Aditya Hegde, and Naman Kumar. $\omega(1/\lambda)$-rate boolean garbling scheme from generic groups. Cryptology ePrint Archive, Report 2025/268, 2025.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 39–53, Taormina, Sicily, Italy, March 19–21, 2012. Springer Berlin Heidelberg, Germany.

[CMPR23] Geoffroy Couteau, Pierre Meyer, Alain Passelègue, and Mahshid Riahinia. Constrained pseudorandom functions from homomorphic secret sharing. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 194–224, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[CNs07] Jan Camenisch, Gregory Neven, and abhi shelat. Simulatable adaptive oblivious transfer. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 573–590, Barcelona, Spain, May 20–24, 2007. Springer Berlin Heidelberg, Germany.

[Cou19] Geoffroy Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume

11477 of *Lecture Notes in Computer Science*, pages 473–503, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 502–534, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[CVW18] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 577–607, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[CWYY23] Hongrui Cui, Xiao Wang, Kang Yang, and Yu Yu. Actively secure half-gates with minimum overhead under duplex networks. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 35–67, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[DGI+19] Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 3–32, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

[DILO22] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of

Lecture Notes in Computer Science, pages 57–87, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, Cheju Island, South Korea, February 13–15, 2001. Springer Berlin Heidelberg, Germany.

[DKK18] Itai Dinur, Nathan Keller, and Ohad Klein. An optimal distributed discrete log protocol with applications to homomorphic secret sharing. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 213–242, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[DKL+23] Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 417–446, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[DKN+20] Alex Davidson, Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively secure constrained pseudorandom functions in the standard model. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 559–589, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

[DLL25] Marian Dietz, Hanjun Li, and Huijia Lin. TinyLabels: How to compress garbled

circuit input labels, efficiently. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025, Part VI*, volume 15606 of *Lecture Notes in Computer Science*, pages 245–274, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.

[FGJS17] Nelly Fazio, Rosario Gennaro, Tahereh Jafarikhah, and William E. Skeith, III. Homomorphic secret sharing from paillier encryption. In Tatsuaki Okamoto, Yong Yu, Man Ho Au, and Yannan Li, editors, *ProvSec 2017: 11th International Conference on Provable Security*, volume 10592 of *Lecture Notes in Computer Science*, pages 381–399, Xi'an, China, October 23–25, 2017. Springer, Cham, Switzerland.

[FKN94a] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th Annual ACM Symposium on Theory of Computing*, pages 554–563, Montréal, Québec, Canada, May 23–25, 1994. ACM Press.

[FKN94b] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 554–563. ACM, 1994.

[FLLL24] Ximing Fu, Mo Li, Shihan Lyu, and Chuanyi Liu. Bit-fixing correlation attacks on goldreich's pseudorandom generators. *IACR Cryptol. ePrint Arch.*, page 1594, 2024.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 191–219, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 464–479, Singer Island, Florida, October 24–26, 1984. IEEE Computer Society Press.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482, Santa Barbara, CA, USA, August 15–19, 2010. Springer Berlin Heidelberg, Germany.

[GHKW17] Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. A generic approach to constructing and proving verifiable random functions. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 537–566, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in*

Computer Science, pages 465–482, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany.

[GIKM00] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 60(3):592–629, 2000.

[GJM03] Philippe Golle, Stanislaw Jarecki, and Ilya Mironov. Cryptographic primitives enforcing communication and storage complexity. In Matt Blaze, editor, *FC 2002: 6th International Conference on Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 120–135, Southampton, Bermuda, March 11–14, 2003. Springer Berlin Heidelberg, Germany.

[GKP+13a] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run Turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[GKP+13b] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 555–564, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 567–578, Denver, CO, USA, October 12–16, 2015. ACM Press.

[GLNP18] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. *Journal of Cryptology*, 31(3):798–844, July 2018.

[GM94] Torbjörn Granlund and Peter L. Montgomery. Division by invariant integers using multiplication. In Vivek Sarkar, Barbara G. Ryder, and Mary Lou Soffa, editors, *Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation (PLDI), Orlando, Florida, USA, June 20-24, 1994*, pages 61–72. ACM, 1994.

[GN25] Jian Guo and Wenjie Nan. Efficient mixed garbling from homomorphic secret sharing and GGM-tree. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025, Part VI*, volume 15606 of *Lecture Notes in Computer Science*, pages 152–181, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

[GOS18] Sanjam Garg, Rafail Ostrovsky, and Akshayaram Srinivasan. Adaptive garbled RAM from laconic oblivious transfer. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 515–544, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[GR05] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005: 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 803–815, Lisbon, Portugal, July 11–15, 2005. Springer Berlin Heidelberg, Germany.

[GRR+16] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. MPC-friendly symmetric key primitives. In Edgar R. Weippl, Stefan

Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 430–443, Vienna, Austria, October 24–28, 2016. ACM Press.

[GS18a] Sanjam Garg and Akshayaram Srinivasan. Adaptively secure garbling with near optimal online complexity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 535–565, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

[GS18b] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 468–499. Springer, 2018.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[GU24] Romain Gay and Bogdan Ursu. On instantiating unleveled fully-homomorphic signatures from falsifiable assumptions. In Qiang Tang and Vanessa Teague, editors, *PKC 2024: 27th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 14601 of *Lecture Notes in Computer Science*, pages 74–104, Sydney, NSW, Australia, April 15–17, 2024. Springer, Cham, Switzerland.

[GVW15a] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523, Santa Barbara, CA, USA, August 16–20, 2015. Springer Berlin Heidelberg, Germany.

[GVW15b] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 469–477, Portland, OR, USA, June 14–17, 2015. ACM Press.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 301–320, Bengalore, India, December 1–5, 2013. Springer Berlin Heidelberg, Germany.

[Hea24] David Heath. Efficient arithmetic in garbled circuits. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part V*, volume 14655 of *Lecture Notes in Computer Science*, pages 3–31, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[HIKR23] Shai Halevi, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. Additive randomized encodings and their applications. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 203–235, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[HK20] David Heath and Vladimir Kolesnikov. Stacked garbling - garbled circuit proportional to longest execution path. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of

Lecture Notes in Computer Science, pages 763–792, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

[HK21] David Heath and Vladimir Kolesnikov. One hot garbling. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 574–593, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[HKKW19] Dennis Hofheinz, Akshay Kamath, Venkata Koppula, and Brent Waters. Adaptively secure constrained pseudorandom functions. In Ian Goldberg and Tyler Moore, editors, *FC 2019: 23rd International Conference on Financial Cryptography and Data Security*, volume 11598 of *Lecture Notes in Computer Science*, pages 357–376, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019. Springer, Cham, Switzerland.

[HKT11] Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 89–98, San Jose, CA, USA, June 6–8, 2011. ACM Press.

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th Annual Symposium on Foundations of Computer Science*, pages 415–434, Santa Cruz, CA, USA, November 6–9, 2023. IEEE Computer Society Press.

[HO12] Brett Hemenway and Rafail Ostrovsky. Extended-DDH and lossy trapdoor functions. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 627–643, Darmstadt, Germany, May 21–23, 2012. Springer Berlin Heidelberg, Germany.

[HVDH21] David Harvey and Joris Van Der Hoeven. Integer multiplication in time o (n log n). *Annals of Mathematics*, 193(2):563–617, 2021.

[HW15] Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015: 6th Conference on Innovations in Theoretical Computer Science*, pages 163–172, Rehovot, Israel, January 11–13, 2015. Association for Computing Machinery.

[IK97] Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *Fifth Israel Symposium on Theory of Computing and Systems, ISTCS 1997, Ramat-Gan, Israel, June 17-19, 1997, Proceedings*, pages 174–184. IEEE Computer Society, 1997.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st Annual Symposium on Foundations of Computer Science*, pages 294–304, Redondo Beach, CA, USA, November 12–14, 2000. IEEE Computer Society Press.

[IKM+13] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 600–620, Tokyo, Japan, March 3–6, 2013. Springer Berlin Heidelberg, Germany.

[ILL24a] Yuval Ishai, Hanjun Li, and Huijia Lin. Succinct homomorphic MACs from groups and applications. Cryptology ePrint Archive, Paper 2024/2073, 2024.

[ILL24b] Yuval Ishai, Hanjun Li, and Huijia Lin. Succinct partial garbling from groups and applications. Cryptology ePrint Archive, Paper 2024/2073, 2024.

[ILL25] Yuval Ishai, Hanjun Li, and Huijia Lin. A unified framework for succinct garbling

from homomorphic secret sharing. Cryptology ePrint Archive, Paper 2025/442, 2025.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594, Amsterdam, The Netherlands, February 21–24, 2007. Springer Berlin Heidelberg, Germany.

[IW14] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *ICALP 2014: 41st International Colloquium on Automata, Languages and Programming, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 650–662, Copenhagen, Denmark, July 8–11, 2014. Springer Berlin Heidelberg, Germany.

[JLL23] Aayush Jain, Huijia Lin, and Ji Luo. On the optimal succinctness and efficiency of functional encryption and attribute-based encryption. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 479–510, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd Annual ACM Symposium on Theory of Computing*, pages 60–73, Virtual Event, Italy, June 21–25, 2021. ACM Press.

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In Barna Saha and Rocco A. Servedio, editors, *55th Annual ACM Symposium on Theory of Computing*, pages 1545–1552, Orlando, FL, USA, June 20–23, 2023. ACM Press.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for Turing machines with unbounded memory. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 419–428, Portland, OR, USA, June 14–17, 2015. ACM Press.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 440–457, Santa Barbara, CA, USA, August 17–21, 2014. Springer Berlin Heidelberg, Germany.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science*, pages 364–373, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 669–684, Berlin, Germany, November 4–8, 2013. ACM Press.

[KRRW18] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 365–391, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[KS08a] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors,

ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498, Reykjavik, Iceland, July 7–11, 2008. Springer Berlin Heidelberg, Germany.

[KS08b] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[KY18] Ilan Komargodski and Eylon Yogev. Another step towards realizing random oracles: Non-malleable point obfuscation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 259–279, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

[Lip05] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In Jianying Zhou, Javier López, Robert H. Deng, and Feng Bao, editors, *Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings*, volume 3650 of *Lecture Notes in Computer Science*, pages 314–328. Springer, 2005.

[LL24] Hanjun Li and Tianren Liu. How to garble mixed circuits that combine boolean and arithmetic computations. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 331–360, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[LLL22] Hanjun Li, Huijia Lin, and Ji Luo. ABE for circuits with constant-size secret keys and adaptive security. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part I*, volume 13747 of

Lecture Notes in Computer Science, pages 680–710, Chicago, IL, USA, November 7–10, 2022. Springer, Cham, Switzerland.

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 329–346, Providence, RI, USA, March 28–30, 2011. Springer Berlin Heidelberg, Germany.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany.

[LV18] Tianren Liu and Vinod Vaikuntanathan. Breaking the circuit-size barrier in secret sharing. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th Annual ACM Symposium on Theory of Computing*, pages 699–708, Los Angeles, CA, USA, June 25–29, 2018. ACM Press.

[LVW18] Tianren Liu, Vinod Vaikuntanathan, and Hoeteck Wee. Towards breaking the exponential barrier for general secret sharing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 567–596, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

[LWYY24] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. Garbled circuits with 1 bit per gate. Cryptology ePrint Archive, Paper 2024/1988, 2024.

[LWYY25] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. BitGC: Garbled circuits with 1 bit per gate. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025, Part VII*, volume 15607 of *Lecture Notes*

in Computer Science, pages 437–466, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

[MORS24] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret sharing. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024: 22nd Theory of Cryptography Conference, Part IV*, volume 15367 of *Lecture Notes in Computer Science*, pages 71–97, Milan, Italy, December 2–6, 2024. Springer, Cham, Switzerland.

[MORS25] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Silent circuit relinearisation: Sublinear-size (boolean and arithmetic) garbled circuits from DCR. Cryptology ePrint Archive, Report 2025/245, 2025.

[MT10] Ueli M. Maurer and Stefano Tessaro. A hardcore lemma for computational indistinguishability: Security amplification for arbitrarily weak PRGs with optimal stretch. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 237–254, Zurich, Switzerland, February 9–11, 2010. Springer Berlin Heidelberg, Germany.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 129–139. ACM, 1999.

[OS07] Rafail Ostrovsky and William E. Skeith, III. A survey of single-database private information retrieval: Techniques and applications (invited talk). In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 393–411, Beijing, China, April 16–20, 2007. Springer Berlin Heidelberg, Germany.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 678–708, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer Berlin Heidelberg, Germany.

[PS18] Chris Peikert and Sina Shiehian. Privately constraining and programming PRFs, the LWE way. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 10770 of *Lecture Notes in Computer Science*, pages 675–701, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Cham, Switzerland.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267, Tokyo, Japan, December 6–10, 2009. Springer Berlin Heidelberg, Germany.

[QWW21] Willy Quach, Brent Waters, and Daniel Wichs. Targeted lossy functions and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 424–453, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In David S. Johnson, editor,

Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washington, USA, pages 73–85. ACM, 1989.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[RR21] Mike Rosulek and Lawrence Roy. Three halves make a whole? Beating the half-gates lower bound for garbled circuits. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 94–124, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 687–717, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[SEA23] Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL, January 2023. Microsoft Research, Redmond, WA.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15, 1997. Springer Berlin Heidelberg, Germany.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov,

editors, *ACM CCS 2010: 17th Conference on Computer and Communications Security*, pages 463–472, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.

[Ste98] Julien P. Stern. A new efficient all-or-nothing disclosure of secrets protocol. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology – ASIACRYPT'98*, volume 1514 of *Lecture Notes in Computer Science*, pages 357–371, Beijing, China, October 18–22, 1998. Springer Berlin Heidelberg, Germany.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443, Paris, France, May 26–28, 2010. Springer Berlin Heidelberg, Germany.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 475–484, New York, NY, USA, May 31 – June 3, 2014. ACM Press.

[Wee14] Hoeteck Wee. Dual system encryption via predicate encodings. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 616–637, San Diego, CA, USA, February 24–26, 2014. Springer Berlin Heidelberg, Germany.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 21–37, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[WW24] Hoeteck Wee and David J. Wu. Succinct functional commitments for circuits from k-Lin. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part II*, volume 14652 of *Lecture Notes in Computer Science*, pages 280–310, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

[Zha22] Mark Zhandry. To label, or not to label (in generic groups). In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part III*, volume 13509 of *Lecture Notes in Computer Science*, pages 66–96, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.

[ZYZL19] Shuoyao Zhao, Yu Yu, Jiang Zhang, and Hanlin Liu. Valiant's universal circuits revisited: An overall improvement and a lower bound. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 401–425, Kobe, Japan, December 8–12, 2019. Springer, Cham, Switzerland.