Tradeoffs in Cryptography

Ashrujit Ghoshal

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

Doctor of Philosophy

University of Washington

2023

Reading Committee:

Stefano Tessaro, Chair

Huijia Lin

Ilan Komargodski

Program Authorized to Offer Degree:
Computer Science & Engineering

©Copyright 2023
Ashrujit Ghoshal

University of Washington

Abstract

Tradeoffs in Cryptography

Ashrujit Ghoshal

Chair of the Supervisory Committee:
Stefano Tessaro
Department of Computer Science & Engineering

The primary objective of provable security is to characterize, as precisely as possible, the resources an attacker needs to compromise the security of a cryptographic system. While the time complexity of an attacker has traditionally been the key factor considered, actual attackers choose to invest in various other resources such as memory, preprocessing, parallelism etc. Moreover, an attacker may also exploit tradeoffs between these various resources in order to compromise security. This thesis develops new theory that characterizes the tradeoffs between various resources required to compromise the security of a cryptographic system. It makes progress in the two following directions:

1. **Memory-tightness**: Firstly, this thesis develops a toolbox of new techniques for giving memory-tight reductions. Earlier works on memory-tightness gave some generic elementary techniques to make reductions memory-tight, but they were somewhat limited in their applicability. The work here addresses this issue and invents more involved techniques to make certain types of reductions memory-tight. Secondly, this thesis gives new results on the impossibility of making certain reductions memory-tight. While prior impossibility results in other settings were known, this thesis proves a memory-tightness impossibility for a scheme with algebraic structure for the first time.

2. **Time-space tradeoffs**: This thesis continues the recent line of work on time-space tradeoffs in cryptography. Firstly, it proves new results that characterize time-space tradeoffs for mechanisms underlying the most popular cryptographic hash functions. Secondly, it gives a new hashing mechanism that provides optimal security against these time-space tradeoff attacks. Finally, it addresses the question of how much time is needed to produce the advice for preprocessing attacks.

TABLE OF CONTENTS

Page

List of Figures v

List of Tables xii

Chapter 1: Introduction 1

Part I: Memory-Tightness: New Techniques and Impossibilities 8

Chapter 2: Memory-Tightness: Introduction, Prior Work and Contributions 9
2.1 New Techniques for Memory-tight Reductions: Prior Work and Our Contributions 9
2.1.1 Technique 1: Adversary Memory Dependent Memory-Tight Reductions 9
2.1.2 Technique 2: Memory-Tight Reductions using Randomness Programming 12
2.2 Impossibility Results for Memory-tightness: Prior Work and Our Contributions 17
2.2.1 Impossibility 1: Memory-Tightness of Hashed ElGamal 18
2.2.2 Impossibility 2: A Generic Memory-Tightness Impossibility for INDR + CTXT $\implies$ AE 19

Chapter 3: Adversary Memory Dependent Memory-Tight Reductions 21
3.1 Definitions 21
3.1.1 Cryptographic Background 22
3.2 Nonce-based Encryption and Memory-boundedness 24
3.2.1 Indistinguishability From Random (INDR) Security 25
3.2.2 Security Beyond Confidentiality 27
3.2.3 Security of the CAU Encryption Scheme 28
3.3 Memory-tight Reductions for Cryptographic Channels 31
3.3.1 Syntax and Security Notions 31
3.3.2 Confidentiality and Integrity Imply Authenticated Encryption 32
3.3.3 AE Security of a TLS 1.3-like Channel 37
3.4 Proof of Thm. 3.3.4 ($\mathsf{NCH}[\mathsf{CAU}[\mathsf{E},\mathsf{H}]]$ is CTXT secure) 40

Chapter 4: Hiding in Plain Sight: Memory-tight proofs using Randomness Programming 45
4.1 Preliminaries 45
4.1.1 Computational Model 45
4.1.2 Cryptographic Background 46
4.2 Adversaries With Access to Random Functions 47

4.3 Multi-challenge Security of Digital Signature Schemes . . . 49
4.3.1 Syntax and Security . . . 50
4.3.2 Multi-Challenge Security for Extractable Message Distributions . . . 52
4.3.3 Proof of Theorem 4.3.1 (1UFCMA⇒mUFRMA) . . . 54
4.3.4 Applications and Weakly Extractable Variant . . . 56
4.3.5 mSUFCMA Security of RSA-PFDH . . . 60
4.4 AE Security of Encrypt-then-PRF . . . 66
4.4.1 Security Result . . . 67
4.5 Chosen Ciphertext Security of Public Key Encryption . . . 70
4.5.1 Left-or-right CCA Security of PKE . . . 71
4.5.2 Indistinguishable from Random CCA Security of PKE . . . 79
4.6 Extended Discussion of Oracle Adversaries . . . 87
4.7 KEM/DEM (Application Requiring AE/$CCA-m) . . . 89
4.7.1 Secret-key KEM/DEM . . . 90
4.7.2 Public-key KEM/DEM . . . 95

Chapter 5: Memory-Tightness of Hashed ElGamal . . . 103
5.1 Preliminaries . . . 106
5.1.1 Generic Group Model . . . 106
5.1.2 Compression Lemma . . . 109
5.1.3 Polynomials and Schwartz-Zippel Lemma . . . 109
5.1.4 Key Encapsulation Mechanism (KEM) . . . 109
5.2 Result and Proof Outline . . . 110
5.3 Proof of Theorem . . . 113
5.3.1 Adversary $\mathcal{A}$ against ODH . . . 113
5.3.2 The Shuffling Games . . . 116
5.3.3 Proof of Lemma 5.2.4 . . . 121
5.3.4 Memory Lower Bound when $|\mathcal{Z}| \geqslant l$ (Proof of Lemma 5.2.5) . . . 137
5.3.5 Proof of Lemma 5.3.9 . . . 142
5.4 Conclusions . . . 151
5.5 Proof Sketch of Correctness of Decoding in Section 5.3.5 . . . 152
5.6 Rewinding: Conjecture and Obstacles . . . 155

Chapter 6: A Generic Memory-Tightness Impossibility for INDR + CTXT $\implies$ AE . . . 156
6.1 Negative Results for Memory-tight AE Reductions . . . 157
6.1.1 Memory Lower Bound for Straightline Reductions . . . 161
6.1.2 Memory Lower Bound for Full-Rewinding Reductions . . . 164
6.2 Proof of Theorem 6.1.8 (Impossibility for Full-Rewinding Reductions) . . . 165
6.3 Analysis of Brute-Force Adversary's Advantage . . . 170
6.4 Proofs of Lemma 6.1.6, Lemma 6.2.1, and Lemma 6.2.3 (INDR Reductions) . . . 171

6.4.1 Proof of Lemma 6.1.6 . . . 171
6.4.2 Proof of Lemma 6.2.1 . . . 172
6.4.3 Proof of Lemma 6.2.3 . . . 174
6.5 Proof of Lemma 6.1.7 . . . 174
6.6 Proof of Lemma 6.2.4 (Compression Argument) . . . 175

Part II: Time-Space Tradeoffs: Attacks, Limitations and Feasibility . . . 178

Chapter 7: Time-Space Tradeoffs: Introduction, Prior Work and Our Contributions . . . 179
7.1 Time-Space Tradeoffs for Finding Short Collisions in Merkle-Damgård . . . 181
7.1.1 Our Results . . . 183
7.1.2 Discussion . . . 185
7.2 Time-Space Tradeoffs for Short Collisions in Sponge . . . 186
7.2.1 Our Results . . . 189
7.3 Optimal security of Keyed Hash Functions: Avoiding Time-Space Tradeoffs for Finding Collisions 192
7.3.1 Our Results . . . 195
7.4 The Query Complexity of Preprocessing Attacks . . . 202

Chapter 8: Time-Space Tradeoffs for Bounded-Length Collisions in Merkle-Damgård Hashing . . 208
8.1 Preliminaries . . . 215
8.2 The Framework: Reducing the Problem to a Multi-instance Collision Finder . . . 217
8.3 Proving the STB Conjecture for $B \in O(1)$ . . . 219
8.3.1 The Compression Argument . . . 224
8.3.2 Handling Cases 1 to 4 . . . 227
8.3.3 Handling Case 5 . . . 228
8.3.4 Handling Case 6 . . . 232
8.3.5 Description of our Encoding and Decoding Procedures of Theorem 9.3.5 . . . 241
8.3.6 The Encoding Procedure . . . 241
8.3.7 The Decoding Procedure . . . 247
8.4 Proving the STB Conjecture for $SB \ll T$ . . . 251
8.4.1 Changes for Theorem 8.4.1 . . . 257

Chapter 9: Time-Space Tradeoffs for Sponge Hashing: Attacks and Limitations for Short Collisions 258
9.1 Preliminaries . . . 265
9.2 Attacks . . . 267
9.2.1 Generic Attack for B-Block Collisions . . . 267
9.2.2 Preprocessing Attack for $B = 1$ . . . 271
9.2.3 Time-Space Tradeoffs for Inverting a Restricted Permutation . . . 275
9.3 Impossibility Results . . . 284
9.3.1 Advantage Upper Bound for B=1 . . . 284
9.3.2 Advantage Upper Bound for $B = 2$ . . . 286

9.3.3 Encoding and decoding procedure pseudocodes for Theorem 9.3.5 . . . 298

Chapter 10: Optimal Security for Keyed Hash Functions: Avoiding Time-Space Tradeoffs for Finding Collisions . . . 309
10.1 Preliminaries . . . 314
10.2 Merkle-Damgård Framework with a Keyed Inner Hash . . . 318
10.3 Instantiating the Inner Hash: Standard MD . . . 320
10.4 Instantiating the Inner Hash: Two-Level Merkle Tree . . . 323
10.4.1 The [AGL22] Framework . . . 323
10.4.2 Two-Level Merkle Tree . . . 324
10.4.3 Variable-Input Length Hash from Two-Level Merkle Trees . . . 332
10.5 Merkle Tree Framework with a Keyed Inner Hash . . . 333

Chapter 11: The Query-Complexity of Preprocessing Attacks . . . 338
11.1 Preliminaries . . . 338
11.2 Offline-Online Tradeoffs and the Role of Salting . . . 339
11.2.1 A general framework for offline-online attacks . . . 339
11.2.2 The power of salting . . . 342
11.3 Offline-Online Security of Salted Random oracles . . . 344
11.3.1 Pre-image-resistance of a salted random oracle . . . 344
11.3.2 Collision-resistance of a salted random oracle . . . 345
11.3.3 Attacks . . . 349
11.3.4 Strategies for offline-only attacks . . . 358
11.4 Offline-Online Security of Two-Block Merkle-Damgård . . . 359
11.4.1 Pre-image-resistance of two-block Merkle-Damgård . . . 360
11.4.2 Collision-resistance of two-block Merkle-Damgård . . . 365
11.4.3 Proof of Lemma 11.4.7 . . . 372
11.4.4 Proof of Lemma 11.4.8 . . . 374
11.4.5 Proof of Lemma 11.4.9 . . . 375
11.4.6 Proof of Lemma 11.4.10 . . . 377
11.4.7 Proof of Lemma 11.4.11 . . . 380
11.4.8 Proof of Lemma 11.4.12 . . . 383
11.4.9 Proof of Lemma 11.4.13 . . . 387
11.4.10 Proof of Lemma 11.4.14 . . . 399
11.5 Upper Bounding Offline-Online advantage via the Multi-Instance Approach . . . 420

Bibliography . . . 423

LIST OF FIGURES

Figure Number Page

2.1 Memory-tight reductions we provide. A 1 vs. an m prefix indicates whether one or many challenge queries are allowed. A ✓ vs. an ✗ indicates whether the reduction is tight with respect to that complexity metric. Reductions lacking tightness multiply running time/advantage by $O(q)$ or add $O(q)$ to the memory complexity, where q is the number of queries. An x vs. a w indicates whether the coins underlying the distribution of messages can be extracted from the message. RDS is randomization of any digital signature scheme by padding input messages with randomness. RSA-PFDH is probabilistic full-domain hash with RSA. EtP is the Encrypt-then-PRF AE construction. 16

3.1 Security games for PRF and PRP security of E and the switching lemma. 23

3.2 Security game for AXU security of H. 23

3.3 Syntax of nonce-based encryption scheme. 25

3.4 Games defining INDR, CTXT-w, and AE-w security of NE for $w \in \{1, 2, 3\}$. 26

3.5 Encryption scheme CAU parameterized by function family E (typically a block cipher) and hash function H. In the code, $\text{pad}(N) = N \,\|\, 0^m \,\|\, 1$ for the appropriate choice of m and $M_1 \dots M_\ell \leftarrow_n M$ splits M into n-bit blocks. 29

3.6 **Left:** Syntax of channel algorithms. **Right:** Channel correctness game. 31

3.7 Games defining the INDR, CTXT, and AE security of a channel. 33

3.8 Information theoretic game in which $\mathcal{A}$ tries to remember a δ bit sequence in an L-bit random string. 33

3.9 Hybrid games for proof of Theorem 3.3.2. Highlighted code is only included in highlighted games. Boxed code is only included in boxed games. 36

3.10 INDR adversaries for proof of Theorem 3.3.2. Highlighting indicates code that is only used by adversary $\mathcal{A}'_\delta$. 37

3.11 Algorithms of channel NCH[NE] constructed from encryption scheme NE. 38

3.12 Games G_0 and G_1 introducing $\mathsf{bad}_{\mathsf{forge}}$ for proof of Thm. 3.3.4. Highlighted code is only included in G_1. 40

3.13 Games used to analyze $\mathsf{bad}_{\mathsf{forge}}$ for proof of Thm. 3.3.4. Boxed or highlighted code is only included in correspondingly indicated game. 42

3.14 PRF adversaries for proof of Thm. 3.3.4. Highlighted code is only used by the highlighted adversary. 43

3.15 Adversary against the AXU security of H. 44

4.1 Security game capturing the pseudorandomness of function family F. 47

4.2 Syntax of digital signature scheme. 50

4.3 Security games capturing the unforgeability of a digital signature scheme. 51

4.4 Adversary $\mathcal{A}_{\mathsf{u}}$ used in proof of Theorem 4.3.1. . . . 53
4.5 Hybrid games used in proof of Theorem 4.3.1. . . . 54
4.6 Reduction adversary used in proof of Theorem 4.3.3. . . . 58
4.7 Hybrid games used in proof of Theorem 4.3.3. . . . 59
4.8 RSA one-wayness security game. . . . 60
4.9 Digital signature scheme $\mathsf{RSA} = \mathsf{RSA\text{-}PFDH}[\mathsf{R}, \mathsf{rl}]$. . . . 61
4.10 Adversary for Theorem 4.3.4. Highlighting shows where it halts with the specified output. . . . 62
4.11 Hybrid games H_0 through H_3 used in proof of Theorem 4.3.4. . . . 64
4.12 Hybrid games L_0 through L_2 used in proof of Theorem 4.3.4. Grey highlighting indicates where L_0 differs from H_3. . . . 65
4.13 Adversaries used for proof of Theorem 4.4.1. . . . 68
4.14 Hybrid games for proof of Theorem 4.4.1. . . . 69
4.15 Syntax of a public key encryption scheme PKE. . . . 71
4.16 Game defining CCA-w security of PKE for $w \in \{\mathtt{m}, \diamond, \perp\}$. . . . 72
4.17 Adversary $\mathcal{A}_1$ for Theorem 4.5.1. . . . 73
4.18 First set of hybrids H^b_h used in the proof of Theorem 4.5.1. . . . 76
4.19 Second set of hybrids used in the proof of Theorem 4.5.1. . . . 77
4.20 Final set of hybrids for the proof of Theorem 4.5.1. . . . 78
4.21 Game defining \$CCA-w security of PKE for $w \in \{\mathtt{m}, \diamond, \perp\}$. . . . 79
4.22 Adversary $\mathcal{A}_1$ for Theorem 4.5.2. . . . 81
4.23 First set of hybrids used for proof of Theorem 4.5.2. Highlighting indicates modifications in H^b_0 from $\mathsf{G}^{\$\mathsf{cca}\text{-}\mathtt{m}}_{\mathsf{PKE},b}$. . . . 82
4.24 Second set of hybrids used for proof of Theorem 4.5.2. Highlighting indicates modifications from H^b_3. . . . 84
4.25 Final set of hybrids for the proof of Theorem 4.5.2. The highlighted line is new with respect to the code of $\mathsf{H}^{k,b}$. . . . 86
4.26 Nonce-based AE scheme $\mathsf{KD}[\mathsf{NE}_{\mathsf{k}}, \mathsf{NE}_{\mathsf{d}}]$ constructed from nonce-based AE schemes NE_{k} and NE_{d} via the KEM/DEM paradigm. . . . 90
4.27 Game defining multi-user AE security . . . 91
4.28 First set of hybrids used for proof of Theorem 4.7.1. $\mathbb{I}$ denotes the identity function. . . . 92
4.29 Adversary $\mathcal{B}_a$ for Theorem 4.7.1. . . . 93
4.30 Second set of hybrids used for proof of Theorem 4.7.1. . . . 93
4.31 Adversary $\mathcal{C}_a$ for Theorem 4.7.1. . . . 94
4.32 Game defining multi-user \$CCA security of nonce-based encryption. . . . 95
4.33 A public-key scheme $\mathsf{KD}[\mathsf{PKE}, \mathsf{NE}]$ constructed from a public key encryption scheme PKE and a nonce-based encryption scheme NE. . . . 96
4.34 First set of hybrids used for proof of Theorem 4.7.2. $\mathbb{I}$ denotes the identity function. . . . 97
4.35 Adversary $\mathcal{B}_c$ for Theorem 4.7.2. . . . 98
4.36 Second set of hybrids used for proof of Theorem 4.7.2. . . . 99
4.37 Adversary $\mathcal{C}_a$ for Theorem 4.7.2. . . . 100

4.38 Third set of hybrids used for proof of Theorem 4.7.2. Hybrids after M_4 are mostly undoing prior transitions. . . . 101

4.39 Adversary $\mathcal{E}_c$ for Theorem 4.7.2. . . . 102

5.1 Games for ODH and SDH assumptions . . . 108

5.2 The adversary $\mathcal{A}$ . . . 116

5.3 Game G_1. We use the phrase $\mathcal{R}_1, \mathcal{R}_2$ win G_1 to mean $\mathsf{G}_1 \Rightarrow \mathsf{true}$. We shall use this convention for all games in the chapter that output boolean values. . . . 118

5.4 Adversary $\mathcal{B}$ . . . 119

5.5 Games $\mathsf{G}_2, \mathsf{G}_3$. The $\mathsf{Eval}, \mathsf{O}_\mathsf{v}$ oracles in $\mathsf{G}_2, \mathsf{G}_3$ are identical to those in G_1 and hence we do not rewrite it here. The newly introduced changes compared to G_1 are highlighted. The statement within the thinner box is present only in G_3 and the statement within the thicker box is present only in G_2. . . . 121

5.6 The permutation game $\mathbb{PG}$ being played by adversary $\mathcal{A}$ is denoted by $\mathbb{PG}(\mathcal{A})$ . . . 122

5.7 Encoding and decoding π using $\mathcal{A}$ . . . 123

5.8 Subroutines $\mathsf{PopulateSetsEval}, \mathsf{PopulateSetsO}_\mathsf{v}$ . . . 124

5.9 G_4 introduces additional bookkeeping. The newly introduced changes compared to G_2 are highlighted. . . . 125

5.10 G_5 lazily samples σ. The newly introduced changes compared to G_4 are highlighted. . . . 127

5.11 In G_7 we remove the restriction on t_j's. The main procedure of $\mathsf{G}_6, \mathsf{G}_7$ are identical to G_5 and the only changes in the oracles are the $\mathsf{AssignPoly}$ procedure, hence we write only the $\mathsf{AssignPoly}$ procedure. The changes in the procedure from that in G_5 have been highlighted. The boxed statement is present only in G_6. . . . 128

5.12 Games $\mathsf{G}_8, \mathsf{G}_9$ introduce new bad events. We remove σ from these games. The changes from G_7 have been highlighted. The boxed statements are present only in G_8. . . . 129

5.13 In games $\mathsf{G}_{10}, \mathsf{G}_{11}$ $\mathsf{p}_\mathbf{a}$ has been replaced by $\Lambda^{-1}(\mathbf{a})$ for labels $\mathbf{a}$ and t_j's are removed. The procedure $\mathsf{AssignPoly}$ is no longer written separately, its code is written inline instead. In G_{11}, i_j's, v are sampled with the restriction that the set $\{1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdot v\}$ has cardinality $2k + 2$. The changes from G_9 have been highlighted. The boxed statements are present only in G_{11}. . . . 131

5.14 Game G_{11} has been written more compactly than the one in Figure 5.13 by removing redundant code. . . . 132

5.15 **Subroutine $\mathsf{PolyMultCheck}$ for simulating O_v.** In particular, $\mathsf{coefficient}(\mathsf{p}, M)$ returns the coefficient of the monomial M in the polynomial p. The sets S and $\mathcal{Z}'$ have no effect on the behavior, and are only used in the analysis of $\mathcal{D}$. The symbol ABORT indicates that $\mathcal{D}$ aborts and outputs $\perp$. . . . 133

5.16 Adversary $\mathcal{D}$ which plays the permutation game $\mathbb{PG}$. The changes in $\mathcal{D}$ compared to G_{11} have been highlighted. . . . 134

5.17 Games $\mathsf{G}_{12}, \mathsf{G}_{13}$. The statement within the thinner box is present only in G_{12} and the statement within the thicker box is present only in G_{13}. The newly introduced changes compared to G_3 are highlighted. . . . 138

5.18 The first part of the Encode procedure. If $|\mathcal{Z}| < l$, a fixed encoding D (the output of Encode for some fixed σ when $|\mathcal{Z}| \geqslant l$) is returned. The function T.append(tuple) appends tuple into T. The first inserted tuple has index 1 and the index increases by 1 for every subsequent insertion. $\mathsf{T}[i]$ returns the tuple at index i in T. The function $\mathsf{T}(\mathbf{a})$ returns pre-image expression A such that $(\mathbf{a}, A, *) \in \mathsf{T}$. The function $\mathsf{VarReduce}(\mathsf{T}, C, C')$ equates expressions C and C', expresses the variable with highest subscript in the equation (say X_j) in terms of other variables, and substitutes X_j throughout T. 143

5.19 Second part of the Encode procedure. 144

5.20 The first part of the Decode procedure. The function T.append(tuple) appends tuple into T. The first inserted tuple has index 1 and the index increases by 1 for every subsequent insertion. $\mathsf{T}[i]$ returns the tuple at index i in T. The function T.index(**a**) returns the index of the tuple of the form $(\mathbf{a}, *, *)$ in T and the function $\mathsf{T}(\mathbf{a})$ returns pre-image expression A such that $(\mathbf{a}, A, *) \in \mathsf{T}$. The function $\mathsf{VarReduce}(\mathsf{T}, C, C')$ equates expressions C and C', expresses the variable with highest subscript in the equation (say X_j) in terms of other variables, and substitutes X_j throughout T. 145

5.21 The second part of the Decode procedure. 146

6.1 Syntax of a black-box reduction $\mathcal{R}$ running AE-1 adversary $\mathcal{A}$. We represent the oracles $\mathcal{R}$ has access to collectively as O. 158

6.2 Information theoretic game played by adversary $(\mathcal{D}_1, \mathcal{D}_2)$. 159

6.3 Game $\mathsf{G}^{\mathsf{bf\text{-}it\text{-}chl}}_{u,m,P}$ and adversary $\mathcal{D}'_1$ used in the proof of Lemma 6.1.2. 160

6.4 Games G^1 and G^0_b for $b \in \{0, 1\}$. Highlighted code is only included in G^1. 162

6.5 Adversaries against the AE-1 security of NE. Boxed code is only included in $\mathcal{S}$. Highlighted code is only included in $\mathcal{A}$ and $\mathcal{B}$. 163

6.6 Adversaries against the AE-1 security of NE used for proof of Thm. 6.1.8. Highlighted code is only included in $\mathcal{A}$. Boxed code is only included in $\mathcal{S}$. Adversaries $\mathcal{B}$ (used by $\mathcal{A}$) and $\mathcal{E}$ (used by $\mathcal{W}$) are repeated from Fig. 6.5. 166

6.7 Sequence of games used in proof Theorem 6.1.8. On the left, boxed code is *not* included in G^1 and highlighted code is only included in h^2. On the right, highlighted code is included in both games (indicating where they differ from G^1) and boxed code is only included in G^3. . . 167

6.8 Adversary $(\mathcal{D}_1, \mathcal{D}_2)$ used to prove Lemma 6.2.3. 169

6.9 Adversaries $\mathcal{C}_1$ and $\mathcal{C}_2$ used for Lemma 6.1.6. The highlighted code is present only in $\mathcal{C}_1$ and the boxed code is present only in $\mathcal{C}_2$. 171

6.10 Adversaries $\mathcal{C}_1$ and $\mathcal{C}_2$ for Lemma 6.2.1 and adversary $\mathcal{C}_3$ for Lemma 6.2.3. Highlighted code is only included in $\mathcal{C}_1$ and boxed code is only included in $\mathcal{C}_2$. 173

6.11 Adversary $(\mathcal{D}_1, \mathcal{D}_2)$ used in the proof of Lemma 6.1.7. 174

6.12 Procedures Encode and Decode. Highlighting indicates interesting code changes from G^2. . . . 176

7.1 A summary of the attacks and advantage upper bounds for finding B-block collisions for the Sponge hash function. All bounds are given ignoring $\mathrm{poly}(c, r)$ terms. We note that the attack for $B = T$ is implicitly claimed in [CDG18] based on [CDGS18]. 192

7.2 Our framework for building keyed hash functions based on the Merkle-Damgård construction with a keyed inner compression function. 196

7.3 Our framework for building keyed hash functions based on the Merkle tree construction. . . . 196

7.4 The construction H_1 underlying Theorem 7.3.1 given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a > 2n$. 198
7.5 The construction H_2 underlying Theorem 7.3.2 given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a > n$. The gray dotted boxes represent the compress function, instantiated with the Merkle-Damgård construction, that uses $\mathsf{k} \in \{0,1\}^n$ as the key and $y_{i-1}\|m_i \in \{0,1\}^{2n}$ as the message. 199
7.6 The construction H_3 underlying Theorem 7.3.3 given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a > 3n/2$. The gray dotted boxes represent the compress function, instantiated with a two-block Merkle-Damgård construction, that uses $\mathsf{k} \in \{0,1\}^n$ as the key and $y_{i-1}\|m_i \in \{0,1\}^{2a-2n}$ as the message. The $2a-2n$ bit message is split evenly into the first and second call to h, indicated in the figure by a diamond. 200
7.7 The construction H_4 underlying Theorem 7.3.4 given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a \geqslant 2n$. The gray dotted boxes represent the compress function, instantiated with a two-level Merkle tree, that uses $\mathsf{k} \in \{0,1\}^n$ in each leaf and $y_{i-1}\|m_i \in \{0,1\}^{2a-2n}$ as the message. The $2a-2n$ bit message is split evenly between the two leaves of the Merkle tree, indicated by a diamond in the figure. We require $a \geqslant 2n$ so that both outputs from the leaves can be fed into the next layer of the Merkle tree. 201

8.1 A mouse structure. For ease of visual representation we do not draw the nodes and edges of the graph, instead represent it as a continuous structure. 211
8.2 An example of a "hard" mouse structure. The NEW queries are drawn in red, REPEATEDUSED queries are drawn in blue and the REPEATEDUNUSED queries are drawn in green. 213
8.3 The bounded-length collision resistance game of salted MD hash in the AI-ROM, denoted $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}$. 216
8.4 A mouse structure. For ease of visual representation we do not draw the nodes and edges of the graph, instead represent it as a continuous structure. 223
8.5 Cases 1 to 4. The NEW queries are drawn in red, the REPEATEDUSED queries are drawn in blue, and the REPEATEDUNUSED queries are drawn in green. The black dashed lines indicate zero or more queries of any type. 227
8.6 Mouse structure for case 5 229
8.7 Mouse structures categorized in case 6. The NEW queries are drawn in red, and the REPEATEDUNUSED queries are drawn in green. The black dashed lines indicate zero or more NEW or REPEATEDUNUSED queries. 234

9.1 An illustration of the preprocessing phase of the generic attack. In red, we depict the components that are part of the output of the preprocessing phase. In blue we see the collisions that will be outputted in the online phase if some chain is hit. Notice that we denote $f(\cdot) := \Pi(0,\cdot)[2]$. 259
9.2 Partite representation of $\Pi(\cdot)$ and $\Pi^{-1}(0,\cdot)$. Notice that Π is a permutation and thus forms a perfect matching while $\Pi^{-1}(0,\cdot)$ is not a permutation and in expectation a random image will have several pre-images. 260
9.3 The bounded-length collision resistance game of salted sponge based hash in the AI-RPM, denoted $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}$. 266
9.4 Adversary $\mathcal{A}$ used in the proof of Theorem 9.2.1. 269
9.5 The bounded-length collision resistance game of salted sponge based hash in the BF-RPM, denoted $\mathsf{G}^{\mathsf{bf\text{-}cr}}_{c,r,B}$. 285

10.1 The collision resistance game $G^{\mathsf{ai\text{-}cr}}_{g^h}$ in AI-ROM for a function g^h based on a random oracle $h : \{0,1\}^a \to \{0,1\}^n$. The construction g^h has a parameter κ associated with it, where κ is the bit length of k used in g. 316
10.2 The keyed Merkle-Damgård construction $\mathsf{KMD}^h\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ given any underlying function $h\colon \{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$, where κ is the key length, n is the output length, and s is the message block size. 318
10.3 The standard Merkle-Damgård construction with fixed input message length b $\mathsf{FMD}^h_b\colon \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$ given any underlying function $h\colon \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^n$, where n is the key and output length, and s is the message block size. 321
10.4 The multi-instance game $G^{\mathsf{mi\text{-}cr}}_{g^h,S}$, where $\mathcal{B}$ is a uniform adversary with oracle access to the function h. 323
10.5 The two-level, keyed Merkle tree construction $\mathsf{2MT}^h_b\colon \{0,1\}^\kappa \times \{0,1\}^b \to \{0,1\}^n$ with fixed input message length b given any underlying function $h\colon \{0,1\}^a \to \{0,1\}^n$, where $a \geqslant \max(2n+2, \kappa+3)$ and $b \leqslant 2a - 2\kappa - 4$. h_1, h_2, h_3 are all domain-separated using the first two bits of h to encode $1, 2, 3$, respectively. 325
10.6 The construction KMT^h given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a \geqslant \max(\kappa+2, 2n+1)$. Let $s = a - \kappa - 1$. The inputs to h are padded with appropriate number of zeros if the size of the input is less than $\kappa + s + 1$ bits. I is the identity function that is used to "push" a value up the tree until it needs to be merged. 334
10.7 The Merkle tree construction with a keyed inner hash. 335
10.8 Recursive collision search procedure used in the proof of Theorem 10.5.2 336

11.1 Offline-Online security games for an original game G. Left: the unsalted case. Right: the salted case. Here, π is meant to be sampled from a salted ideal primitive $\mathcal{I}_s$, and $\pi_a(\cdot) = \pi(a, \cdot)$, i.e., the primitive with salt fixed to a. 341
11.2 Left: Oracle Game PR^{h} for preimage-resistance of oracle h. Right: Oracle Game CR^{h} for collision-resistance of oracle h. 345
11.3 Oracle game $\mathsf{2\text{-}PR\text{-}MD}^{\mathrm{h}}_n$ formalizing the pre-image-resistance of the two-block MD construction. 360
11.4 Game H used in the analysis of pre-image-resistance of two-block MD. The events introduced in this game are marked in red. 362
11.5 Oracle game $\mathsf{2\text{-}CR\text{-}MD}^{\mathrm{h}}_n$ formalizing collision-resistance of two-block MD. 366
11.6 The structure of the six different types of two-block MD collisions in the query graph. The nodes in the query graph are labelled with values in $\{0,1\}^n$, and there is an edge (a, a') labelled with M if the adversary made a query $\mathrm{h}(a, M) = a'$. We omit the node and edge labels for simplicity. 367
11.7 Game H used in the analysis of collision-resistance of two-block MD against offline-online adversaries. The events introduced in this game are marked in red. 369
11.8 Illustrating successor queries of a salt in the query graph. The queries marked in red to a salt a are successor queries for a. 379
11.9 Illustrating meeting queries of a salt in the query graph. The queries marked in red to a salt a are meeting queries for a. 382
11.10 Illustrating successor-hitting queries of a salt in the query graph. The queries marked in red to a salt a are successor-hitting queries for a. 386
11.11 The different categories of triangles. The edge colored red is the last of the three queries made. 388

11.12The zigzag structure for a salt pair (a, a') in the query graph. 395
11.13The different categories of diamonds. The edge colored red is the last of the four queries made.400
11.14The hexagon structure for a salt pair (a', a') in the query graph. 405
11.15The different categories of hexagons. The edge colored red is the last of the six queries made. 408
11.16Oracle Game s-mi-k-CR^{h} capturing multi-instance salted collision-resistance of oracle h 421

LIST OF TABLES

Table Number Page

4.1 Summary of the $\mathcal{F}$-oracle adversaries obtained by the results in this chapter. 87

7.1 A summary of our results as well as the standard MD construction for reference. The advantage of a construction is given in terms of the probability of (S, T)-attackers to find collisions relative to a random key. The efficiency is measured in terms of the number of calls to h that maps a-bit inputs to n when processing a b-bit input. The assumptions column specifies conditions on various parameters. 202

10.1 Upper bounds on probability of the cases 329

ACKNOWLEDGMENTS

Thank you to my primary advisor, Stefano Tessaro. I am very grateful to Stefano for being patient with me when I started and for pushing me towards higher standards when I required it. I am thankful also for the advice he was always ready to give about things beyond research. Thank you to my co-advisor, Rachel Lin. Even though I did not work on research with Rachel, her feedback about my talks and general advice and encouragement was very valuable to me. Thank you to Ilan Komargodski. Ilan was my mentor during my two internships at NTT Research. I am very grateful to Ilan for introducing me to a research area which became one of the main focuses of this thesis and for his constant encouragement and positivity. Thank you to Anup Rao and Cynthia Vinzant for agreeing to be part of my PhD supervising committee. Thank you to Debdeep Mukhopadhyay, my undergraduate thesis advisor, for sowing the seeds of interest in research in me and being a mentor to me over all these years.

Thank you to all my co-authors: Stefano Tessaro, Ilan Komargodski, Joseph Jaeger, Cody Freitag and Riddhi Ghosal. This thesis is based primarily on my works with them in [GT20, GJT20, GGJT22, GK22, FGK22, FGK23, GT23].

Thank you to all the other members of the crypto group at UW during my time here: Binyi Chen, Pratik Soni, Ben Terner, Ji Luo, Xihu Zhang, Anna Kornfeld Simpson, Tianren Liu, Chenzhi Zhu, Hanjun Li, Marshall Ball, Champ Chairattana-Apirom, Yao-Ching Hseih, Marian Dietz and Andrea Coladangelo. I am grateful to all of them for numerous conversations about cryptography and for a very supportive environment. I am especially grateful to Pratik for being a mentor who I could always turn for advice to. I am grateful to Ji and Xihu for being my companions for large parts of my PhD journey. I am also grateful to all the visitors to the group for helping make our weekly reading group an interactive shared learning experience.

Thank you to the PhD advising team at the Allen School. They did everything in their powers to make all the bureaucratic work go very smoothly during my time here.

Thank you to all my friends in the Allen School, Seattle, friends I made during my internships at NTT Research and friends who were there both at the beginning and the end of my PhD journey. Trying to name all of them is a futile exercise, so I will highlight only some of them. I am extremely grateful to Siddharth Iyer, Ben Lee and Siva Natarajan Ramamomoorthy for being amazing friends and huge sources of support throughout my PhD. At the very end of my PhD, I suffered an unfortunate and sudden medical emergency.

During this extremely challenging period, I had unflinching support from several friends: Sudheesh Singanamalla, Priyal Suneja, Tapan Chugh,Siddharth Iyer, Ben Lee, Siva Natarajan Ramamomoorthy, Arani Roy, Venkatesh Potluri, Sahil Verma, Reshabh Sharma, Gantavya Bhatt, Raghav Somani, Ashish Sharma, Sebastian Santy. I cannot thank them enough for all the help and care they gave me at this time.

Thank you to the more senior folks in the cryptography community who were always welcoming and gave me some very valuable advice during conversations I had with them during visits, conferences or internships. I am grateful to Mihir Bellare, Yevgeniy Dodis, Elette Boyle, Hoeteck Wee, Mark Zhandry, Elaine Shi, Aayush Jain, David Wu, Brent Waters, Jeremiah Blocki for such conversations.

Thank you to my family for always being very supportive. Thank you to my aunt Saptaparnee Mondal, uncle Saptarshi Mondal for being a family to me in the US. Lastly and most importantly, thank you to my parents *Maa* (Pampa Ghoshal) and *Baba* (Ashok Kumar Ghoshal) to whom this thesis is dedicated. I can never put in words the magnitude of my gratitude towards them for all they have done for me. Thank you for believing in me more than I ever did and providing so much love, care and encouragement at all times.

DEDICATION

To Maa and Baba

Chapter 1

INTRODUCTION

Designing secure cryptographic systems is a challenging task. No amount of testing against individual vulnerabilities can give enough confidence because there might always be a new vulnerability that is discovered in the future. This has led to the rise of the subfield of *provable security* which advocates relying on mathematical proofs to show that cryptographic systems provide security guarantees under plausible computational assumptions, e.g., factoring large integers is hard. This approach has been extremely successful over the years, and provable security is now seen as a necessary requirement for real-world cryptographic systems.

One main focus of provable security is to give a *concrete* characterization of the resources needed by an adversary to break a cryptographic system. This is important because these characterizations inform our choices of parameters for cryptographic systems that get deployed in the real world. In general, setting larger parameters means better security guarantees at the cost of reduced efficiency. This is why we aim for as precise characterizations as possible, because then we can set parameters in a way that guarantees us good security guarantees without unnecessarily sacrificing efficiency.

The most exact of these characterizations are given in the three following steps: **(1)** a model for an adversary is defined formally in a way which closely captures the *strongest* capabilities of any attack that can be done in the real world **(2)** a specific adversary in the aforementioned model that breaks the cryptographic system is described **(3)** a mathematical proof is provided which shows that this adversary is optimal, i.e., any other adversary with lesser amount of resources, fails.

Breadth of adversarial resources. Traditionally, the running time of an adversary has been the main resource that has been almost exclusively considered while giving security proofs in cryptography. There are mainly two reasons for this. First, one can obtain reasonable upper bounds on the running time of any plausible attacks from the computing powers of the fastest known processors, and accordingly set parameters of the cryptographic systems to ensure its security. Secondly, considering more resources makes the proofs significantly harder and the techniques for handling them had not developed enough.

However, restricting provable security to only take into account the running time of an adversary paints an incomplete picture of the real world. Real world attacks can choose to invest in or can be constrained by various other resources, e.g., memory, preprocessing, parallelism, availability of randomness, etc. The

traditional guarantees in provable security are agnostic to the use of these resources and assume no constraints on these other resources when providing guarantees about the *best possible* attacks. However, in many cases, when one or more of these resources are limited in quantity, the *best possible* attacks become significantly slower. For instance, memory is often a bottleneck for attacks, and there are several assumed-to-be-hard problems in cryptography for which the attacks which have no limitations on the amount of available memory significantly outperform attacks that only have some limited amount of memory. In turn, this means that if a reasonable upper bound on the amount of memory available to an attack is known, one could potentially set parameters more optimistically. This would ensure that the cryptographic system is more efficient without sacrificing its security.

Another angle: Resource tradeoffs. This can be viewed from another angle as well– real world attacks *can choose to* trade the amount of running time with resources such as preprocessing, memory, randomness, parallelism etc. In some cases this is necessary, e.g., if an adversary is trying to attack a cryptographic system for which the underlying secret resets at regular intervals, any useful attack can not run in time that is longer than the interval. When this is the case, the adversary can instead choose to invest in other resources. In other cases it can make this choice for a different reason– for example, certain attacks can be significantly sped up by using preprocessing advice, which is extremely expensive to compute but reusable. Therefore, if the adversary has to solve many instances of some hard problem, the large preprocessing cost amortizes over the number of instances.

The first example of an adversary exploiting a tradeoff between preprocessing advice and running time appeared in a seminal work [Hel80] by Hellman. Hellman explored the problem of function inversion where, for a function f, an adversary is given $y = f(x)$ as input where x is a randomly sampled point in the domain of f. The adversary needs to compute x' such that $f(x') = y$. This, in particular, captures the hardness of inverting one-way functions, a very fundamental problem in cryptography.

Hellman demonstrated that an adversary might be able to invest a significant amount of effort in *preprocessing* on the function f, and compute some "advice" about f. This advice helps in significantly reducing the *online* time, i.e., the time required by the adversary to complete the inversion *after* gets the point y as input. Even though the preprocessing cost might be very large, if the adversary needs to invert f on many different y's, the preprocessing cost gets amortized. The tradeoff between the amount of preprocessing advice and running time is popularly known as a "time-space tradeoff", where space refers to the storage required for the preprocessing.

In addition to this, modeling adversaries in this fashion captures non-uniform attacks, which is generally the chosen model of adversaries in theoretical cryptography. There are several technical reasons behind this choice, one of them being there are certain examples where non-uniformity is necessary for composition

within larger systems [GK96].

Goals and barriers. My thesis work is focused on extending the framework of provable security to consider additional adversarial resources. In particular, the additional resources that I consider in my thesis work are **(1)** memory and **(2)** pre-processing advice. My work has focused on developing new theory towards precisely characterizing the tradeoffs of time-complexity with these two resources. In several cases, these tradeoffs have connections to questions in areas such as combinatorics, complexity theory, information theory, etc., e.g., prior progress in time-memory tradeoffs [Din20] used connections to communication complexity. In many cases, these connections lead to major obstacles due to pre-existing known barriers in these areas, e.g., [CK19] showed that any improvement in proving limitations on best possible attacks on function inversion, would imply circuit lower bounds that have been longstanding open problems.

Contributions. More concretely, the contributions of my thesis work can be summarized into the two following broad directions.

1. **Memory-tightness: New techniques and impossibilities**: Cryptographic security proofs proceed via *reductions*, which show that given an adversary that can break some cryptographic algorithm, one can construct another adversary that can solve some well-known hard problem efficiently. Traditionally, security reductions have taken delicate care of relating the running time of the two adversaries, but ignored the relationship between the memory usage by the two adversaries. However, it is extremely important to take the relative memory usage into account because there are time-memory tradeoffs for many well-known hard problems in cryptography, i.e., these problems can be solved significantly faster when more memory is available. My thesis work has contributed towards correcting this oversight regarding memory. In [GT20], [GJT20] and [GGJT22], we *revisited the paradigm of cryptographic reductions to also consider memory as a resource of the adversary.* In particular, these results helped determine whether time-memory tradeoffs are preserved by certain commonly used security reductions.

   In more detail, in [GT20] we studied the memory-tightness of security reductions in public-key cryptography, focusing in particular on Hashed ElGamal. We proved that any *straightline* (i.e., without rewinding) black-box reduction needs memory which grows linearly with the number of queries of the adversary it has access to, as long as this reduction treats the underlying group generically. This made progress towards proving a conjecture by Auerbach *et al.* (CRYPTO 2017), and is also the first lower bound on memory-tightness for a concrete cryptographic scheme (as opposed to generalized reductions across security notions).

   In [GJT20], we initiated the study of the provable security of authenticated encryption (AE) in the memory-bounded setting. Prior works [TT18, JT19, Din20] had focused on confidentiality, and looked

at schemes for which tradeoffs between the attacker's memory and its data complexity are inherent. Here, we asked whether these results and techniques can be lifted to the full AE setting, which additionally asks for integrity.

We showed both positive and negative results. On the positive side, we provided tight memory-sensitive bounds for the security of GCM and its generalization, CAU [BT16]. Our bounds apply to a restricted case of AE security which abstracts the deployment within protocols like TLS, and rely on a new memory-tight reduction to corresponding restricted notions of confidentiality and integrity. In particular, our reduction uses an amount of memory which linearly depends on that of the given adversary, as opposed to only imposing a constant memory overhead as in earlier works (Auerbach et al., CRYPTO '17). On the negative side, we showed that a large class of black-box reductions cannot generically lift confidentiality and integrity security to a joint definition of AE security in a memory-tight way.

In [GGJT22], we addressed the fact that despite the importance of memory-tight reduction, only a few techniques to achieve memory-tightness were known and impossibility results in prior works showed that even basic, textbook reductions cannot be made memory-tight. We introduced a new class of memory-tight reductions which leverage random strings in the interaction with the adversary to hide state information, thus shifting the memory costs to the adversary. We exhibited this technique with several examples. We gave memory-tight proofs for digital signatures allowing many forgery attempts when considering randomized message distributions or probabilistic RSA-FDH signatures specifically. We proved security of the authenticated encryption scheme Encrypt-then-PRF with a memory-tight reduction to the underlying encryption scheme. By considering specific schemes or restricted definitions we avoid generic impossibility results of [ACFK17] and [GJT20]. As a further case study, we considered the textbook equivalence of CCA-security for public-key encryption for one or multiple encryption queries. We showed two qualitatively different memory-tight versions of this result, depending on the considered notion of CCA security.

In Part I of this thesis, we cover this line of work. We give a more detailed overview of our results on this line of work in Chapter 2. Further, we provide the details of the results in [GT20] in Chapter 5, the details of the results in [GJT20] in Chapters 3 and 6 and the details of the results in [GGJT22] in Chapter 4.

2. **Time-space tradeoffs: attacks and limitations**: A number of prior works [Hel80, Oec03, FN99, Unr07, DTT10, DGK17, CDGS18, CK18, CDG18] have focused on the time-space tradeoffs in many different contexts of cryptography. However, even after all these works, there remained numerous hard-

to-resolve theoretical gaps in our understanding of time-space tradeoffs. Our works in [GK22, FGK22] have contributed to *significantly improving the understanding of the time-space tradeoffs* for mechanisms underlying cryptographic hash functions, one of the most widely adopted cryptographic algorithms in practice. These include the SHA-2 and SHA-3 standards [SHA]. Further, our work in [FGK23] constructs a new cryptographic hashing mechanisms that is optimally resitant to these time-space tradeoff attacks. Finally, our work in [GT23] studies the actual cost of the preprocessing phase of these attacks.

In more detail, in [GK22], we studied the power of preprocessing adversaries in finding bounded-length collisions in the widely used Merkle-Damgård (MD) hashing in the random oracle model. Our goal is to characterize the advantage of such adversaries in finding a B-block collision in an MD hash function constructed using the random oracle with range size N as the compression function (given a random salt). The answer to this question is completely understood for very large values of B (essentially $\Omega(T)$) as well as for $B = 1, 2$. For $B \approx T$, [CDGS18] gave matching upper and lower bounds of $\tilde{\Theta}(ST^2/N)$. Akshima et al [ACDW20] observed that the attack of [CDGS18] could be adapted to work for any value of $B > 1$, giving an attack with advantage $\tilde{\Omega}(STB/N + T^2/N)$. Unfortunately, they could only prove that this attack is optimal for $B = 2$. Their proof involves a compression argument with exhaustive case analysis and, as they claim, a naive attempt to generalize their bound to larger values of B (even for $B = 3$) would lead to an explosion in the number of cases needed to be analyzed, making it unmanageable. With the lack of a more general upper bound, they formulated the *STB conjecture*, stating that the best-possible advantage is $\tilde{O}(STB/N+T^2/N)$ for any $B > 1$. In this work, we confirmed the STB conjecture in many new parameter settings. For instance, in one result, we show that the conjecture holds for all constant values of B, significantly extending the result of [ACDW20]. Further, using combinatorial properties of graphs, we were able to confirm the conjecture even for super constant values of B, as long as some restriction is made on S. For instance, we confirmed the conjecture for all $B \leqslant T^{1/4}$ as long as $S \leqslant T^{1/8}$. Technically, we developed structural characterizations for bounded-length collisions in MD hashing that allow us to give a compression argument in which the number of cases needed to be handled does not explode.

In [FGK22], we studied time-space tradeoffs in the context of sponge hashing while is a novel alternative to the popular Merkle-Damgård hashing design. The sponge construction has become increasingly popular in various applications, perhaps most notably, it underlies the SHA-3 hashing standard. Sponge hashing is parametrized by two numbers, r and c (bitrate and capacity, respectively), and by a fixed-size permutation on $r + c$ bits. In this work, we studied the collision resistance of sponge hashing instantiated with a random permutation by adversaries with arbitrary S-bit auxiliary advice input

about the random permutation that make T online queries. Recent work by Coretti et al [CDG18] showed that such adversaries can find collisions (with respect to a random c-bit initialization vector) with advantage $\Theta(ST^2/2^c + T^2/2^r)$. lthough the above attack formally breaks collision resistance in some range of parameters, its practical relevance is limited since the resulting collision is very long (on the order of T blocks). Focusing on the task of finding *short* collisions, we studied the complexity of finding a B-block collision for a given parameter $B \geqslant 1$. We gave several new attacks and limitations. Most notably, we give a new attack that outperforms the previously-known best attack. To the best of our knowledge, this was the first natural application for which sponge hashing is *provably less secure* than the corresponding instance of Merkle-Damgård hashing. Our attack relies on a novel connection between single-block collision finding in sponge hashing and the well-studied function inversion problem. We also give a general attack that works for any $B \geqslant 2$ and has advantage $\Omega(STB/2^c + T^2/2^{\min\{r,c\}})$, adapting an idea of Akshima et al. [ACDW20]. We complemented the above attacks with bounds on the best possible attacks. Specifically, we proved that there is a qualitative jump in the advantage of best possible attacks for finding unbounded-length collisions and those for finding very short collisions. Most notably, we prove (via a highly non-trivial compression argument) that the above attack is optimal for $B = 2$ in some range of parameters.

In our work in [FGK23], we studied how to key hash functions that gives optimal security against time-space tradeoff attacks. The most well-known methods for designing variable-input length hash function families from a fixed idealized function are the Merkle-Damgård and Sponge designs. The former underlies the SHA-1 and SHA-2 constructions and the latter underlies SHA-3. Unfortunately, recent works [CDGS18, CDG18] show non-trivial time-space tradeoff attacks for finding collisions for both. Thus, this forces a parameter blowup (i.e., efficiency loss) for reaching a certain desired level of security. We asked whether it is possible to build families of keyed hash functions which are *provably* resistant to any non-trivial time-space tradeoff attacks for finding collisions, without incurring significant efficiency costs. We presented several new constructions of keyed hash functions that are provably resistant to any non-trivial time-space tradeoff attacks for finding collisions. Our constructions provide various tradeoffs between their efficiency and the range of parameters where they achieve optimal security for collision resistance. Our main technical contribution wad proving optimal security bounds for converting a hash function with a fixed-sized input to a keyed hash function with (potentially larger) fixed-size input. We then used this keyed function as the underlying primitive inside the standard Merkle-Damgård and Merkle tree constructions. We strongly believe that this paradigm of using a keyed inner hash function in these constructions is the right one, for which non-uniform security had not been analyzed prior to this work.

In our work in [GT23], we addressed the fact that while a large number of works prove lower bounds on time-space tradeoffs in preprocessing attacks, the question of how much *time* is needed to produce this advice is often received less attention. We contend that is question is equally important, and often highly non-trivial. In this work, we presented lower bounds on the complexity of preprocessing attacks that depend on both offline and online time. As in the case of time-space tradeoffs, we focused in particular on settings with ideal primitives, where both the offline and online time-complexities are approximated by the number of queries to the given primitive. We gave generic results that highlight the benefits of salting to generically increase the offline costs of preprocessing attacks. The majority of our work covers focused on *salted* hash functions. In particular, we provided a fairly involved analysis of the pre-image-and collision-resistance security of the (two-block) Merkle-Damgård construction in our model.

In Part II of this thesis, we cover this line of work. We give a more detailed overview of our results on this line of work in Chapter 7. Further, we provide the details of the results in [GK22, FGK22, FGK23, GT23] in Chapters 8 to 11 respectively.

Part I

MEMORY-TIGHTNESS: NEW TECHNIQUES AND IMPOSSIBILITIES

Chapter 2

MEMORY-TIGHTNESS: INTRODUCTION, PRIOR WORK AND CONTRIBUTIONS

Security proofs rely on *reductions*, i.e., they show how to transform an adversary $\mathcal{A}$ breaking a scheme into an adversary $\mathcal{B}$ solving some underlying assumed-to-be-hard problem. Generally, the reduction ought to be *tight* – the resources used by $\mathcal{B}$, as well as the attained advantage, should be as close as possible to those of $\mathcal{A}$. Indeed, the more resources $\mathcal{B}$ needs, or the smaller its advantage, the weaker the reduction becomes.

Auerbach *et al.* [ACFK17] were the first to explicitly point out that *memory* resources have been ignored in reductions, and that this leads to a loss of quality in security results. Indeed, it is conceivable that $\mathcal{A}$'s memory is naturally bounded (say, at most 2^{64} bits), and the underlying problem is very sensitive to memory. For example, the best-known algorithm for discrete logarithms in a 4096-bit prime field runs in time (roughly) 2^{156} using memory 2^{80}. With less memory, the best algorithm is the generic one, requiring time $\Theta(\sqrt{p}) \approx 2^{2048}$. Therefore, if $\mathcal{B}$ also uses memory at most 2^{64}, we can infer a larger lower bound on the necessary time complexity for $\mathcal{A}$ to break the scheme, compared to the case where $\mathcal{B}$ uses 2^{100} bits instead.

One should therefore target reductions that are *memory-tight*, i.e., the memory usage of $\mathcal{B}$ is similar to that of $\mathcal{A}$. In this thesis, we continue the ongoing line of works on memory-tightness in two fronts: **(1)** coming up with new classes of techniques to make reductions memory-tight **(2)** proving impossibilities about the memory-tightness of certain reductions. We elaborate on our two contributions in Sections 2.1 and 2.2.

2.1 New Techniques for Memory-tight Reductions: Prior Work and Our Contributions

Auerbach et al. [ACFK17] gave several simple techniques that could be used to make reductions memory-tight. While these techniques are elementary, there are many reductions which cannot be made memory-tight using these techniques. More advanced techniques to make reductions memory-tight had been left unexplored. In this thesis, we introduce two new classes of techniques for making reductions memory-tight. In Sections 2.1.1 and 2.1.2 we summarize and contextualize our two new techniques.

2.1.1 Technique 1: Adversary Memory Dependent Memory-Tight Reductions

The starting point of our first technique is the observation that existing memory-tight reductions in the literature which are (near) "memory-less", i.e., the reduction adds a small memory overhead, *independent* of the memory of the adversary. We observe that if the size of the additional memory used by the reduction

depends linearly on the adversary's memory complexity, the total memory of the resulting reduction adversary is still linear in the size of the memory of the original adversary. Such a reduction would still be memory-tight. We give a memory tight reduction that has this aforementioned property. First off, we provide the necessary background and context for this new memory-tight reduction.

[JT19, Din20] had initiated the study of time-memory tradeoffs for symmetric encryption. More specifically, they had analyzed INDR security, a notion which measures the indistinguishability of ciphertexts from truly random ones, of nonce-based encryption. For example, consider a toy scheme SE based on a block cipher E with block length n which encrypts $M \in \{0,1\}^n$ with key K as

$$\mathsf{SE.E}(K, N, M) = \mathrm{E}_K(N) \oplus M \ .$$

Here, N is the nonce and INDR security should hold as long as no two messages are encrypted with the same nonce. One can show that for every adversary $\mathcal{A}$ with time, data, and memory complexities t, q, and S, respectively,

$$\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{SE}}(\mathcal{A}) \leqslant O\left(\frac{q \cdot S \cdot \log(q)}{2^n}\right) + \mathsf{Adv}^{\mathsf{prp}}_{\mathrm{E}}(\mathcal{B}) \ , \tag{2.1}$$

where $\mathcal{B}$ is an adversary against the security of E as a pseudorandom permutation (PRP), which has time and memory complexities (roughly) t and S, respectively, and makes q queries. In particular, if $S < 2^{n/2}$, then SE achieves beyond-birthday security $q > 2^{n/2}$ with respect to data complexity.

However, INDR security is rarely sufficient on its own – we want *fully secure* AE schemes which also satisfy (ciphertext) *integrity* (or CTXT security, for short). Following [RS06], we adopt a *single* AE security definition that incorporates *both* INDR and CTXT, by measuring indistinguishability of two oracle pairs $(\textsc{Enc}_b, \textsc{Dec}_b)$ for $b \in \{0,1\}$. For $b = 1$, $\textsc{Enc}_1$ returns real ciphertexts, and $\textsc{Dec}_1$ decrypts properly. For $b = 0$, instead, $\textsc{Enc}_0$ returns random ciphertexts, and $\textsc{Dec}_0$ decrypts only previous outputs from $\textsc{Enc}_0$. It is important to use a combined definition, as it captures settings such as chosen-ciphertext attacks and padding-oracle attacks [Vau02], which use a decryption oracle to break confidentiality.

Lifting tradeoffs. We want to prove a bound analogous to that of (2.1) for AE security, preserving in particular the existing space-time tradeoff. The usual approach is to prove INDR and CTXT *individually*, and then combine them to show AE security. This makes sense because (1) we *know* how to prove tight tradeoffs for INDR security, and (2) we may be able to prove stronger bounds on CTXT easily, even without memory restrictions. The classical statement (originally in [RS06]) is that for every adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{ae}}_{\mathsf{SE}}(\mathcal{A}) \leqslant \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{SE}}(\mathcal{B}) + \mathsf{Adv}^{\mathsf{ctxt}}_{\mathsf{SE}}(\mathcal{C}) \ ,$$

for suitable adversaries $\mathcal{B}$ and $\mathcal{C}$, with similar time and query complexities as those of $\mathcal{A}$. However, this is only helpful towards our goal if the reduction is *memory-tight*, in the sense Auerbach et al. [ACFK17], i.e., $\mathcal{B}$

and $\mathcal{C}$'s memory costs must not noticeably exceed those of $\mathcal{A}$. This is fundamental to preserve a time-memory tradeoff like the one from (2.1).

Unfortunately, the standard proof is not memory-tight with respect to the INDR adversary $\mathcal{B}$, as it needs to simulate $\textsc{Dec}_0$ which requires remembering prior ciphertexts.

We provide a novel memory-tight reduction for the common case where AE is used to establish a secure communication *channel*, as in TLS. The key point is that in this setting, only certain restricted adversarial interactions can occur in the AE security game, i.e.:

1. Nonces are *implicit* – they are incremented as a counter.

2. The receiver *aborts upon the first decryption failure*. In particular, messages *need* to be delivered in the same order as they are encrypted.

Our memory-tight reduction is for an abstraction of this setting we refer to as a *channel*. We apply our reduction to prove (tight) memory-sensitive bounds for a channel instantiated with the CAU scheme by Bellare and Tackmann [BT16], an abstraction of GCM [MV04].

The security game. When restricting AE security to this setting, we can assume that the adversary $\mathcal{A}$ can encrypt messages $M_1, M_2, \ldots$ and obtains ciphertexts $C_1, C_2, \ldots$ via an *encryption oracle* $\textsc{Enc}_b$, for $b \in \{0,1\}$. When $b = 1$, the C_i's are actual encryptions of the M_i's (with increasing nonces), whereas when $b = 0$, they are truly random ciphertexts. The adversary is also given access to a *decryption* oracle $\textsc{Dec}_b$. If $b = 1$, this just applies the decryption algorithm of the AE scheme, using increasing nonces. If decryption fails, $\textsc{Dec}_b$ responds to this and any future queries with $\perp$. For $b = 0$, the oracle responds with $M_1, M_2, \ldots$ as long as it is supplied the ciphertexts $C_1, C_2, \ldots$ *in the order they have been produced by* $\textsc{Enc}_0$. If the ciphertexts come in the wrong order, $\textsc{Dec}_0$ responds to this and any future queries with $\perp$. The goal here is to distinguish $(\textsc{Enc}_0, \textsc{Dec}_0)$ and $(\textsc{Enc}_1, \textsc{Dec}_1)$.

Proof idea. In this channel setting, to obtain a memory-tight reduction from AE security to CTXT and INDR security, we first use CTXT security to replace the oracles $(\textsc{Enc}_1, \textsc{Dec}_1)$ with $(\textsc{Enc}_1, \textsc{Dec}_0)$. (This step is easily seen to be memory-tight.) Next, we aim to use INDR security to replace $\textsc{Enc}_1$ with $\textsc{Enc}_0$. The catch here is that when doing so, we need to simulate the $\textsc{Dec}_0$ oracle in the INDR security game (which does not provide one). Again, this seems to require remembering all prior ciphertexts, thus preventing memory-tightness.

A key observation, however, is that ciphertexts are only accepted when arriving with the right order. For this reason, we will show (via an information-theoretic argument) that our reduction only needs to store the δ oldest ciphertexts which have not been delivered yet, for some δ – the key point here is that δ can be

chosen to depend (roughly linearly) on the memory of the adversary used by the reduction, so the overall memory of the constructed adversary is of the same magnitude of that of the AE adversary.

This is in contrast to existing memory-tight reductions in the literature which are (near) "memory-less", i.e., the reduction adds a small memory overhead, *independent* of the memory of the adversary. Our reduction is the first example where the reduction uses memory in addition to that of the adversary, but the size of this memory is bounded in terms of the adversary's memory complexity.

Application to CAU. We apply our memory-tight reduction to show bounds for CAU (and hence GCM) in the communication channel setting. We refer to the resulting channel as NCH, and it is based on a block cipher E. We show that for every adversary $\mathcal{A}$, there exists $\mathcal{B}$ such that

$$\mathsf{Adv}^{\mathsf{ch\text{-}ae}}_{\mathsf{NCH}}(\mathcal{A}) \leqslant 4 \cdot \mathsf{Adv}^{\mathsf{prp}}_{\mathrm{E}}(\mathcal{B}) + O\left(\frac{pqS}{2^n}\right) , \tag{2.2}$$

where $O(\cdot)$ hides a small constant, q and S are the data and memory complexities of $\mathcal{A}$, and p is an upper bound on the length of ciphertexts. Further, $\mathcal{B}$ makes $q \cdot p$ queries, and has time complexity similar to that of $\mathcal{A}$. Instrumental to our result here is Dinur's Switching Lemma [Din20]. The main challenge is to prove a bound for CTXT security – our proof relies once again on similar techniques to our memory-tight reduction.

In Chapter 3, we provide the full details of this result.

2.1.2 Technique 2: Memory-Tight Reductions using Randomness Programming

For our second technique, we identify a class of examples which admit a new kind of memory-tight reductions. Our approach relies on the availability of random strings exchanged between the adversary and the security game, and which the reduction can leverage to encode state which can be recovered from later queries of the adversary, without the need to store this information locally, and thus saving memory. (In particular, the burden of keeping this information remains on the adversary, which needs to reproduce this random string for this state information to be relevant.) We refer to this approach as *randomness programming*.

As a motivating example, consider the standard UFCMA security notion for signatures. It is defined via a game where the attacker, given the verification key vk, obtains signatures for chosen messages $m_1, m_2, \ldots$, after which it outputs a candidate message-signature pair (m^*, σ^*), and wins if m^* was not signed before, and σ^* is valid for m^*. When ignoring memory, this notion is *tightly* equivalent to one (which we refer to as mUFCMA) that allows for an arbitrary number of "forgery attempts" for pairs (m^*, σ^*), and the adversary wins if one of them succeeds in the above sense. This is convenient: we generally target mUFCMA, but only need to deal with proving the simpler UFCMA notion.

The classical reduction transforms any mUFCMA adversary into a roughly equally efficient UFCMA adversary, which wins with the same probability, by (1) simulating forgery queries using the verification

key, and (2) outputting the first forgery query (m^*, σ^*) which validates and such that m^* is fresh. This reduction is however *not* memory-tight, as we need to ensure the freshness of m^*, which requires remembering the previously signed messages. ACFK [ACFK17] prove that this is in some sense necessary, by showing that a (restricted) class of reductions cannot be memory-tight via a reduction to streaming lower bounds.

Our idea: Efficient tagging. To illustrate our new technique, which we refer to as *efficient tagging*, imagine now that we only use the signature scheme to sign *random* messages $m_1, \ldots, m_q \leftarrow\!\!\$\ \{0,1\}^\ell$, and consider a corresponding variant of mUFCMA security, which we want to reduce to (plain) UFCMA security. This, intuitively, does not seem to help resolve the above issue, because random messages are hardest to compress.

However, what is important here is that the reduction is responsible for simulating the random messages, and can simulate them in special ways, and *program* them so that they encode state information. For instance, assume that the reduction has access to an *injective* random function $f : [q] \to \{0,1\}^\ell$, with inverse f^{-1}, which can be simulated *succinctly* from a short key as a pseudorandom object. Then, the reduction to UFCMA can set $m_i \leftarrow f(i)$ for the i-th query, and upon simulating a forgery query for (m^*, σ^*), the reduction checks whether $f^{-1}(m^*) \in [q]$ to learn whether m^* is a fresh signing query or not.

Of course, the simulation is not perfect: The original m_i's are not necessarily distinct (this can be handled via the classical "switching lemma"). Also, the reduction could miss a valid forgery if the adversary outputs m_i *before* it is given to the adversary, but this again only occurs with small probability.

Inefficient tagging and non-time-tight reductions. In the above example, we can efficiently check that $f^{-1}(m^*) \in [q]$. However, in some cases we may not – again, consider an example where the messages to be signed are sampled as $m_i \leftarrow h(r_i)$, where h is a hard-to-invert function and r_i is random. Then, we could adapt our proof above by setting $m_i \leftarrow h(f(i))$, but now, to detect a prior signing query, we would have to check whether $m^* = h(f(i))$ for some $i \in [q]$, and this can only be done in linear time. The resulting UFCMA adversary runs in time $t' = t + \Theta(q_F \cdot q)$, where t is the running time of the original adversary and q_F is the number of forgery attempts. For example, if $q \approx q_F \approx t$, the reduction is not time tight, and the adversary runs in time $t' = O(t^2)$.

Are non-time-tight reductions useless? It turns out that such non-time-tight reductions can still be helpful to infer that breaking a scheme requires memory, although this ultimately depends on the concrete security of the problem targeted by the reduction. Say, for example, a reduction for a given scheme transforms a successful adversary running in time t and using memory s into an adversary running in time t^2 and using memory s breaking discrete logarithms over $\mathbb{F}_p$, for a 4096-bit prime p. It turns out that if we have fewer than 2^{78} bits of memory, no known discrete logarithm algorithm is better than a generic one (i.e. runs in time better than 2^{2048}), which means that our non-time-tight reduction is still sufficient to infer security for

any $s < 2^{78}$ as long as $t < 2^{1024}$.

MESSAGE ENCODING. At the highest level, what happens is that the reduction is in control of certain random values which we can exploit to hide state information which can later be uniquely recovered, since triggering a situation where the reduction needs to remember requires the adversary to actually give back to the reduction this value. In the above, this state information is simple, namely whether the query is old or not. But as we will show below, the paradigm can be used to store complex information – we refer to this technique as *message encoding*, and discuss an example below.

A NEW VIEWPOINT: $\mathcal{F}$-ORACLE ADVERSARIES. In our technique described above we needed access to a large random injection, which we argue can be simulated pseudorandomly. Prior works have similarly used PRFs to pseudorandomly simulate random oracles [ACFK17, Ber11] with low memory. The fact that one needs to decide how to simulate such objects when stating a memory-tight reduction is rather inconvenient: different instantiations seemingly lead to quantitatively different reductions, although this fact does not appear to be a reflection of any particular reality. In this chapter, we propose (and advocate for) what we believe to be the "right" viewpoint: Our reductions are stated in terms of $\mathcal{F}$-oracle adversaries where $\mathcal{F}$ is a set of functions and such an adversary expects oracle access to a random $f \in \mathcal{F}$. Then, a memory-tightness theorem is obtained in one of two ways, by either (1) applying a generic lemma stating that f can be instantiated in low memory using an $\mathcal{F}$-pseudorandom function, or (2) assuming that the use of f does not functionally increase the success chances of the adversary because f is independent of the problem instance being solved (this is provably the case for some information theoretic problems). In particular, (1) is more conservative than (2), but it is very likely that (2) is also a viable approach which leads to cleaner result – indeed, we do not expect any of the considered memory-sensitive problems to become easier given access to an oracle from any natural class $\mathcal{F}$ – e.g., Factoring does not become easier given access to a random injection.

Our Results

We now move to an overview of our results (summarized in Fig. 2.1) which exemplify different applications of the tagging and message-encoding techniques.

MULTI-CHALLENGE SECURITY OF DIGITAL SIGNATURES. Our first results consider the security of digital signatures in the face of multiple forgery attempts (i.e., challenge queries), generalizing the examples discussed above. We work with a notion we refer to as UFRMA (unforgeability under randomized message attack). This notion is parameterized by a *message distribution* D and when the attacker makes a signing query for m it receives a signature of $m' = \mathsf{D}(m; r)$ for a random r. If m and r can be extracted from m', giving the notion xUFRMA (or mxUFRMA for many forgery attempts), we can generalize our efficient tagging approach above by having the reduction to UFCMA choose $r = f(m, i)$ where each $f(m, \cdot)$ is a random

injection. This setting can capture, e.g., the signatures used in key exchange protocols like TLS 1.3 where the server signs a transcript which includes a random 256-bit nonce. A version of our inefficient tagging example works when only m can be extracted from m' (wUFRMA); we pick $r = f(m, i)$ and in verification of a forgery query perform the linear time check of whether $m^* = \mathsf{D}(m; f(m, i))$ for some $i \in [q]$. This setting captures places where the message to be signed includes a fresh public key or ciphertext. This includes, for example, the use of signatures for signing certificates, in some key exchange protocols, and in signcryption.

We further prove mUFCMA security for particular schemes. First, we can randomize any digital signature scheme DS (obtaining a scheme we call RDS) by signing $m \,\|\, r$ for random r chosen by the signing algorithm and including r as part of the signature. An immediate implication of our mxUFCRA result is a tight reduction from the mUFCMA security of RDS to the UFCMA security of the underlying scheme. One particular instantiation of RDS is Probabilistic Full Domain Hash with RSA ($\mathsf{RSA\text{-}PFDH}$) which was introduced by Coron [Cor02] to provide a variant of Full Domain Hash [BR93] with an (advantage-) tighter security proof. Using our efficient tagging technique we obtain a fully tight proof of the *strong* mUFCMA security of $\mathsf{RSA\text{-}PFDH}$ from the RSA assumption.

In independent and concurrent work, Diemert, Gellert, Jager, and Lyu [DGJL21] studied the mUFCMA security of digital signature schemes. They also considered the RDS construction, proving that if DS can be proven strong UFCMA1 secure[1] with a restricted class of "canonical" memory-tight reductions then there is a memory-tight reduction for the strong mUFCMA security of RDS. This complements our result, showing memory-tight strong mUFCMA security of RDS based on a restricted class of schemes while our result proves memory-tight plain mUFCMA security based on any plain UFCMA scheme. They apply their RDS result to establish tight proofs for the strong mUFCMA security of $\mathsf{RSA\text{-}PFDH}$ (matching our direct proof in Theorem 4.3.4). as well as schemes based on lossy identification schemes and pairings.

Authenticated Encryption Security. Ghoshal, Jaeger, and Tessaro [GJT20] had recently observed that in the context of authenticated encryption (AE), it is difficult to lift confidentiality of the scheme, in terms of INDR security, to full AE security, when additionally assuming ciphertext integrity, if we want to do so in a memory-tight way. This is well motivated, as several works establish tight time-memory tradeoffs for INDR security [TT18, JT19, Din20, DTZ20, SOS20], which we would like to lift to their AE security. The difficulty in the proof is that the INDR reduction must simulate a decryption oracle which rejects all ciphertexts *except those forwarded from an encryption query.* Recognizing these forwarded ciphertexts seems to require remembering state.

Here, we give a different take and show that for specific schemes – in particular, those obtained by adding

[1] The suffix '1' indicates a variant of UFCMA security in which the adversary can only obtain a single signature per message. The security game always returning the same signature if the adversary repeats signature queries.

Assumption	Scheme	Result	Time	Memory	Advantage	New Technique
1UFCMA	Any	mxUFRMA	✓	✓	✓	Efficient tagging
	RDS	mUFCMA	✓	✓	✓	Efficient tagging
	Any	mwUFRMA	✗	✓	✓	Inefficient tagging
RSA	RSA-PFDH	mSUFCMA	✓	✓	✓	Efficient tagging
(PRF, INDR)	EtP	AE	(✓,✓)	(✗,✓)	(✓,✓)	Efficient tagging
1CCA	Any	mCCA	✗	✓	✗	Inefficient tagging
1$CCA	Any	m$CCA	✓	✓	✗	Message encoding

Figure 2.1: Memory-tight reductions we provide. A 1 vs. an m prefix indicates whether one or many challenge queries are allowed. A ✓ vs. an ✗ indicates whether the reduction is tight with respect to that complexity metric. Reductions lacking tightness multiply running time/advantage by $O(q)$ or add $O(q)$ to the memory complexity, where q is the number of queries. An x vs. a w indicates whether the coins underlying the distribution of messages can be extracted from the message. RDS is randomization of any digital signature scheme by padding input messages with randomness. RSA-PFDH is probabilistic full-domain hash with RSA. EtP is the Encrypt-then-PRF AE construction.

integrity via a PRF, following the lines of [Rog04, BN08, NRS14] – a memory-tight reduction *can* be given. Our INDR reduction is applied after arguing that the PRF looks like a random function f and thus forgeries are unlikely to occur. It uses f in a version of our efficient tagging technique to identify whether a ciphertext queried to decryption is fresh.[2]

Chosen ciphertext security: One to many. A classical textbook result for public-key encryption shows that CCA-security against a *single* encryption query (1CCA) implies security against *multiple* queries (mCCA), with a quantitative advantage loss accounting to the number of such queries. ACFK [ACFK17] claim, incorrectly, that the associated reduction from 1CCA to mCCA is easy to make memory-tight, but this appears to be an oversight: No such reduction is known, and here we use our techniques to recover a memory-tight version of this result.

Let us consider concretely the "left-or-right" formulation of 1CCA/mCCA-security: The reduction from 1CCA to mCCA, given an adversary $\mathcal{A}$, picks a random $i \leftarrow\!\!\$\, [q]$ (where q is the number of encryption queries) and simulates the multi-query challenger to $\mathcal{A}$ by answering its first $i-1$ encryption queries with an encryption of the left message, whereas the last $q-i$ queries are answered by encrypting the right

[2]Ghoshal et al. [GJT20] in fact described three variants of AE with different conventions for how decryption responds to non-fresh queries. By our results, memory-tight reductions to INDR are possible for two of the three variants.

message. Only the answer to the i-th query is answered by the single-query challenger. A problem arises when simulating the decryption queries: Indeed, we need to guarantee that a decryption query for *any* of the challenge ciphertexts $c_1^*, \ldots, c_q^*$ returns an error $\perp$, yet this suggests that we seemingly need to remember the extra challenge ciphertexts c_j^* for $j \neq i$.

We will resolve this in two ways. First, we give a new memory-tight reduction using the inefficient tagging method, with the same advantage loss as the original textbook reduction. Our reduction is non-time-tight, so may not be suitable for all situations. The main idea here is that we use the *randomness* used to generate the challenge ciphertext as our tag.

To obtain a reduction which is also tight with respect to time, we resort to the observation that changing to a stronger (but still commonly achieved) definition of CCA-security allows for different memory-tight reductions. We give in particular a memory-tight *and* time-tight reduction (with the usual factor q advantage loss) from the notion of 1\$CCA-`m` security to the notion of m\$CCA-`m` security. These are variants of CCA security where (1) encryption queries are with respect to a *single* message, and return either the encryption of the message, or a random, independent ciphertext, and (2) decryption queries on a challenge ciphertext c_i^* returns the associated message.

Our reduction uses the full power of our message encoding approach, simulating random ciphertexts in a careful way which allows for recovering the associated challenge plaintext.

A FEW REMARKS. The above results on CCA security show us that the ability to give a memory-tight reduction is strongly coupled with definitional choices. In particular, different equivalent approaches to modeling the decryption oracle in the memory unbounded regime may not be equivalent in the memory-bounded setting. This means in particular that we need to exercise more care in choosing the right definition. We believe, for example, that the approach taken in m\$CCA-`m` security is the more "natural" one (as it does not require artificially blocking the output of the decryption oracle, by always returning a message), but there may be contexts where other definitional choices are favored.

Another important lesson learnt from our AE result is that impossibility results, such as those in [ACFK17, WMHT18, GT20, GJT20], do not preclude positive results in form of memory-tight reductions, either by leveraging the structure of specific schemes, or by considering restricted security notions.

In Chapter 4, we describe this technique and the related results in detail.

2.2 Impossibility Results for Memory-tightness: Prior Work and Our Contributions

The work of Auerbach et al. [ACFK17], and a follow-up by Wang et al. [WMHT18], showed *negative* results with respect to memory-tightness, i.e., they proved that certain reductions cannot be memory tight). These lower bounds are tailored at general notions (e.g., single- to multi-challenge reductions), and lower bounds

follow from a natural connection with classical frequency problems on streams. This thesis tackles the more ambitious question of proving impossibility of memory-tight reductions for *concrete schemes*. We summarize these contributions in Sections 2.2.1 and 2.2.2.

2.2.1 Impossibility 1: Memory-Tightness of Hashed ElGamal

Motivated by a concrete open question posed in [ACFK17], we consider the CCA-security of Hashed ElGamal. In its KEM variant, the scheme is based on a cyclic group $G = \langle g \rangle$ – the secret key sk is a random element from $\mathbb{Z}_{|G|}$, whereas the public key is $\mathsf{pk} = g^{sk}$. Then, encapsulation produces a ciphertext-key pair

$$C \leftarrow g^r \ , \quad K \leftarrow \mathsf{H}(\mathsf{pk}^r) \ .$$

for $r \leftarrow \mathbb{Z}_{|G|}$ and a hash function $\mathsf{H} : G \to \{0,1\}^\ell$. Decapsulation occurs by computing $K \leftarrow \mathsf{H}(C^{sk})$.

The CCA-security of Hashed ElGamal in the random-oracle model was proved by Abdalla, Bellare, and Rogaway [ABR01] based on the *Strong Diffie-Hellman* (SDH) assumption (also often called GapDH), and we briefly review the proof.[3] First, recall that in the SDH assumption, the attacker is asked to compute g^{uv} from g^u and g^v, given additionally access to a *decision* oracle O_v which on input $h, y \in G$, tells us whether $h^v = y$.

The reduction sets the Hashed ElGamal public-key to $\mathsf{pk} = g^v$ (setting implicitly $sk = v$), the challenge ciphertext to be $C^* = g^u$, and the corresponding key K^* to be a random string. Then, it simulates both the random oracle and the decapsulation oracle to the adversary $\mathcal{A}$ (which is run on inputs pk, C^* and K^*), until a random-oracle query for g^{uv} is made (this can be detected using the O_v oracle). The challenge is to simulate both oracles consistently: As the reduction cannot compute discrete logarithms, it uses the oracle O_v to detect whether a random-oracle query X and a decapsulation query C_i satisfy $\mathsf{O}_\mathsf{v}(C_i, X) = \mathsf{true}$, and, if this is the case, answers them with the same value.

This reduction requires memory to store all prior decapsulation and random-oracle queries. Unlike other reductions, the problem here is not to store the random-oracle output values (which could be compressed using a PRF), but the actual *inputs* to these queries, which are under adversarial control. This motivates the conjecture that a reduction using little memory does not exist, but the main challenge is of course to prove this is indeed the case.

Our result, in summary. We provide a *memory* lower bound for reductions that are *generic* with respect to the underlying group G. Specifically, we show the existence of an (inefficient) adversary $\mathcal{A}$ in *the generic group*

[3] Abdalla et al. [ABR01] do not phrase their paper in terms of the KEM/DEM paradigm [Sho01, CS03], which was introduced concurrently – instead, they prove that an intermediate assumption, called Oracle Diffie-Hellman (ODH), follows from SDH in the ROM. However, the ODH assumption is structurally equivalent to the CCA security of Hashed ElGamal KEM for one challenge ciphertext.

model (GGM) which breaks the CCA security of Hashed ElGamal via $O(k)$ random oracle/decapsulation queries, but such that no reduction using less than $k \cdot \lambda$ bits of memory can break the SDH assumption *even* with access to $\mathcal{A}$, where λ is the bit-size of the underlying group elements.

Our lower bound is strong in that it shows we do not even have a tradeoff between advantage and memory, i.e., if the memory is smaller than $k \cdot \lambda$, then the advantage is very small, as long as the reduction makes a polynomial number of queries to O_{v} and to the generic group oracle. It is however also important to discuss two limitations of our lower bound. The first one is that the reduction – which receives g, g^v in the SDH game – uses $\mathsf{pk} = g^v$ as the public key to the Hashed ElGamal adversary. The second one is that the reduction is straightline, i.e., it does not perform any rewinding.

We believe that our impossibility result would extend even when the reduction is not straightline. However, allowing for rewinding appears to be out of reach of our techniques. Nonetheless, we *do* conjecture a lower bound on the memory of $\Omega(k \log k)$ bits, and discuss the reasoning behind our conjecture in detail in Section 5.6.

We stress that our result applies to reductions in the GGM, but treats the adversary as a black box. This captures reductions which are black-box in their usage of the group and the adversary. (In particular, the reduction cannot see generic group queries made by the adversary, as in a GGM security proofs.) Looking at the GGM reduces the scope of our result. However, it is uncommon for reductions to depend on the specifics of the group, although our result can be bypassed for specific groups, e.g., if the group has a pairing.

In Chapter 5, we provide the details of this result.

2.2.2 Impossibility 2: A Generic Memory-Tightness Impossibility for INDR + CTXT $\implies$ AE

In the context of the memory-tight reduction that we discussed in Section 3, a meaningful question is whether we can give a memory-tight reduction beyond the setting of channels, and reduce AE security to INDR and CTXT security in the most general sense. We show that this is unlikely by giving impossibility results for black-box reductions.

We consider reductions to INDR and CTXT which are restricted, but note that all prior impossibility results on memory-tight reductions [ACFK17, WMHT18, GT20] make similar or stronger restrictions. In particular, we require the reductions to simulate their encryption oracles "faithfully" to an AE adversary, i.e., if they answer an encryption query with a ciphertext C, the same query (1) has been asked to the encryption oracle available to the reduction and (2) it has returned C. This restriction is natural, and we are not aware of any reductions evading it.

Straightline reductions. Our first result builds an (inefficient) adversary $\mathcal{A}$ against AE security which no straightline reduction can use to (1) break CTXT security (regardless of the memory available to the

reduction) or, more importantly, to (2) break INDR security (unless the reduction uses an amount of memory proportional to the query complexity of the adversary). Moreover, $\mathcal{A}$ uses little memory, and thus our result implies impossibility even for "weakly memory-tight reductions" which adapt their memory usage (such as the one we give here). This is unlike recent works [WMHT18, GT20], which only rule out reductions with memory independent of that of the adversary.

At a high level, $\mathcal{A}$ forces the reduction to complete a memory-hard task before being useful. If the reduction succeeds, $\mathcal{A}$ executes an (inefficient) procedure to break INDR security. (And importantly, this procedure does not help in breaking CTXT security!) More in detail, the first part of $\mathcal{A}$'s execution consists of *challenge rounds*. In each of these rounds, $\mathcal{A}$ encrypts random plaintexts $M_1, \ldots, M_u$, which result in ciphertexts $C_1, \ldots, C_u$, and also picks a random index $i^* \in [u]$. It then asks for the decryption of C_{i*}, and checks whether the response equals M_{i*}. If so, it moves to the next round, if not it aborts by doing something useless. Only if all rounds are successful $\mathcal{A}$ proceeds to break INDR security. We use techniques borrowed from the setting of random oracles with auxiliary input (AI-ROM) [CDGS18] to prove that the probability that all rounds are successful decays exponentially as long as the reduction's memory does not fit all of $M_1, \ldots, M_u$.

Full Rewinding. The restriction to straightline reductions seem too restrictive: After all, a reduction could (1) wait for a decryption query C_{i*}, then (2) rewind the adversary to re-ask $M_1, M_2, \ldots$ until M_{i*} is asked. The caveat is that our definition of INDR security does not allow for *re-asking* encryption queries (again, as pointed out above, such a notion would prevent us from using the results of [JT19, Din20]). Therefore, if we assume that all the reduction can do is remember (say) S plaintext-ciphertext pairs, the above adversary $\mathcal{A}$ will fail to pass a challenge round with probability at least $1 - S/u$.

Still, this does not mean that rewinding cannot help when allowing more general adversarial strategies. While handling arbitrary rewinding appears to be out of reach, we make partial progress by extending our proof (and our construction of $\mathcal{A}$) to show that "full" rewinding (i.e., re-running $\mathcal{A}$ from the beginning) does not help. This is the same rewinding model considered in prior memory-tightness lower bounds [ACFK17]. However, in those results, one obtains a rewinding-memory tradeoffs (in that reducing memory would require more rewinding). Here, our result is absolute, in the sense that if memory is too small, no amount of rewinding can help.

In Chapter 5, we provide the details of this result.

Chapter 3

ADVERSARY MEMORY DEPENDENT MEMORY-TIGHT REDUCTIONS

This chapter presents the new memory-tight reduction technique where the memory of the reduction depends on the memory of the adversary that we introduced in Section 3. This chapter is based on our work in [GJT20].

The chapter is organized as follows: in Section 3.1, we introduce our notation, basic definitions and cover some cryptographic background necessary for the chapter. In Section 3.2, we recall the standard definitions for the security notions of nonce-based encryption. We point out several nuances while defining security in the memory bounded setting. We conclude the section by giving a time-memory tradeoff for the INDR security of CAU. In Section 3.3, we show that memory-tight reductions can be given for the combined confidentiality and integrity security of cryptographic channels. Using the result from Section 3.2, we prove the security of a channel based on CAU. The resulting channel can be viewed as (a simplification of) the channel obtained when using GCM in TLS 1.3.

3.1 Definitions

Let $\mathbb{N} = \{0, 1, 2, \dots\}$. For $D \in \mathbb{N}$, let $[D] = \{1, 2, \dots, D\}$. If S and S' are finite sets, then $\mathsf{Fcs}(S, S')$ denotes the set of all functions $F : S \to S'$ and $\mathsf{Perm}(S)$ denotes the set of all permutations on S. Picking an element uniformly at random from S and assigning it to s is denoted by $s \xleftarrow{\$} S$. The set of finite vectors with entries in S is S^* or $(S)^*$. Thus $\{0,1\}^*$ is the set of finite length strings.

If $x \in \{0,1\}^*$ is a string, then $|x|$ denotes its bitlength. If $n \in \mathbb{N}$ and $x \in \{0,1\}^*$, then $|x|_n = \max\{1, \lceil |x|/n \rceil\}$. We let $x_1 \dots x_\ell \xleftarrow{}_n x$ denote setting $\ell \leftarrow |x|_n$ and parsing x into ℓ blocks of length n (except x_ℓ which may have $|x_\ell| < n$). We let $x[:n]$ denote the first n bits of x and $x[i:n]$ denote the i-th (exclusive) through n-th (inclusive) bits of x. We adopt the convention that if $|x| < |x'|$ then $x \oplus x' = x \oplus x'[:|x|]$. The empty string is ε.

We will make use of queues which operate in first-in, first-out order. If Q is a queue then $Q.\mathsf{add}(M)$ adds M to the back of the queue and $M \leftarrow Q.\mathsf{dq}()$ removes the first element of the queue and assigns it to M. If the queue is empty, then M is assigned the value $\perp \notin \{0,1\}^*$ which is used to represent rejection or uninitialized values.

Algorithms are randomized when not specified otherwise. If $\mathcal{A}$ is an algorithm, then $y \leftarrow \mathcal{A}^{\mathrm{O}_1, \cdots}(x_1, \dots; r)$

denotes running $\mathcal{A}$ on inputs $x_1, \ldots$ with coins r and access to the oracles $\mathrm{O}_1, \ldots$ to produce output y. Performing this execution with a random r is denoted $y \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{O}_1,\cdots}(x_1, \ldots)$. The set of all possible outputs of $\mathcal{A}$ when run with inputs $x_1, \ldots$ is $[\mathcal{A}(x_1, \ldots)]$. The notation $y \leftarrow \mathrm{O}(x_1, \ldots)$ is used for calling oracle O with inputs $x_1, \ldots$ and assigning its output to y. (Note, the code run by the oracle is not necessarily deterministic.)

We make regular use of pseudocode games inspired by the code-based framework of [BR06]. Examples of games can be found in Fig. 9.3. We let $\mathsf{Pr}[\mathsf{G}]$ denote the probability that a game G outputs true. Booleans are implicitly initialized to false, integers to 0, and all other types to $\perp$.

Complexity conventions. When measuring the efficiency of an adversary we follow the standard convention used in studying memory-tightness [ACFK17] on measuring the local complexity of an adversary and not included the complexity of whatever game it interacts with. We primarily focus on the worst-case runtime (i.e. how much computation it performs in between making oracle queries) and memory complexity (i.e. how many bits of state it stores for local computation) of adversaries. Note that while these exclude the time and memory used within whatever oracles the adversary may call, we do include the time and memory used to write down an oracle query and receive the response.

3.1.1 Cryptographic Background

Function family. A function family is an efficiently computable function $\mathsf{F} : A \times B \to C$, where A, B, and C are sets. A hash function is a family of functions. We often write $F_K(\cdot)$ in place of $F(K, \cdot)$.

Pseudorandom function/permutation. Let $\mathrm{E} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^m$ be a function family. If $n = m$ and $\mathrm{E}_K(\cdot)$ is a permutation for each $K \in \{0,1\}^k$, then we say that E is a block-cipher. The primary security notions of interest for such functions are PRF and PRP security. The former is typically more useful in applications, but when E is a block-cipher we prefer to assume PRP security and use that to deduce PRF security.

These security notions are defined by games shown in Fig. 3.1. In $\mathsf{G}^{\mathsf{prp}}$, the adversary is given access to either $\mathrm{E}_K(\cdot)$ for a random key or a random permutation $P : \{0,1\}^n \to \{0,1\}^n$. Game $\mathsf{G}^{\mathsf{prf}}$ is defined similarly except a random function $F : \{0,1\}^n \to \{0,1\}^m$ is used in place of the permutation. For $x \in \{\mathsf{prp}, \mathsf{prf}\}$, we define the advantage of $\mathcal{A}$ by $\mathsf{Adv}^x_{\mathrm{E}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^x_{\mathrm{E},1}(\mathcal{A})] - \mathsf{Pr}[\mathsf{G}^x_{\mathrm{E},0}(\mathcal{A})]$.

Switching Lemma. A classic result in cryptography is the "switching lemma" which bounds how well an adversary can distinguish between a random function and a random permutation. Consider the game $\mathsf{G}^{\mathsf{sl}}_{D,b}$ shown in Fig. 11.6a. In it, the adversary is given oracle access to either a random function or a random permutation with domain/range $[D]$ and is trying to figure out which. We define $\mathsf{Adv}^{\mathsf{sl}}_D(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{\mathsf{sl}}_{D,b}(\mathcal{A})] - \mathsf{Pr}[\mathsf{G}^{\mathsf{sl}}_{D,b}(\mathcal{A})]$.

Game $\mathsf{G}^{\mathsf{prp}}_{\mathrm{E},b}(\mathcal{A})$	Game $\mathsf{G}^{\mathsf{prf}}_{\mathrm{E},b}(\mathcal{A})$	Game $\mathsf{G}^{\mathsf{sl}}_{D,b}(\mathcal{A})$
$K \xleftarrow{\$} \{0,1\}^k$	$K \xleftarrow{\$} \{0,1\}^k$	$F \xleftarrow{\$} \mathsf{Fcs}([D],[D])$
$P \xleftarrow{\$} \mathsf{Perm}(\{0,1\}^n)$	$F \xleftarrow{\$} \mathsf{Fcs}(\{0,1\}^n,\{0,1\}^m)$	$P \xleftarrow{\$} \mathsf{Perm}([D])$
$b' \xleftarrow{\$} \mathcal{A}^{\textsc{Eval}_b}$	$b' \xleftarrow{\$} \mathcal{A}^{\textsc{Eval}_b}$	$b' \xleftarrow{\$} \mathcal{A}^{\textsc{Eval}_b}$
Return $b' = 1$	Return $b' = 1$	Return $b' = 1$
Oracle $\textsc{Eval}_b(x)$	Oracle $\textsc{Eval}_b(x)$	Oracle $\textsc{Eval}_b(x)$
$y_1 \leftarrow \mathrm{E}_K(x)$	$y_1 \leftarrow \mathrm{E}_K(x)$	$y_1 \leftarrow F(x)$
$y_0 \leftarrow P(x)$	$y_0 \leftarrow F(x)$	$y_0 \leftarrow P(x)$
Return y_b	Return y_b	Return y_b

Figure 3.1: Security games for PRF and PRP security of E and the switching lemma.

Game $\mathsf{G}^{\mathsf{axu}}_{\mathrm{H}}(\mathcal{X})$

$((A_1,C_1),(A_2,C_2),Z) \xleftarrow{\$} \mathcal{X}$

$L \xleftarrow{\$} \{0,1\}^k$

If $(A_1,C_1) = (A_2,C_2)$ then return false

Return $\mathrm{H}_L(A_1,C_1) \oplus \mathrm{H}_L(A_2,C_2) = Z$

Figure 3.2: Security game for AXU security of H.

The classic switching lemma shows $\mathsf{Adv}^{\mathsf{sl}}_D(\mathcal{A}) \in O(q^2/D)$ where q is the number of queries made by $\mathcal{A}$. In general, bounding the memory-complexity of the attacker cannot be used to meaningfully improve this bound because a low-memory collision-finding attack (e.g., using Pollard's ρ-method [Pol75, QD90]) achieves advantage $\mathsf{Adv}^{\mathsf{sl}}_D(\mathcal{A}) \in \Omega(q^2/D)$. However, as originally observed by Jaeger and Tessaro we *can* obtain better results when restricting attention to adversaries that never repeat any queries.

Let $\mathsf{Adv}^{\mathsf{sl}}_D(q,S)$ denote the maximal value of $\mathsf{Adv}^{\mathsf{sl}}_D(\mathcal{A})$ for all $\mathcal{A}$ that are S-bounded and make q non-repeating queries to their oracle. Jaeger and Tessaro [JT19] showed that $\mathsf{Adv}^{\mathsf{sl}}_D(q,S) \leqslant \sqrt{Sq/D}$ under a combinatorial conjecture. Later, Dinur [Din20] improved this to show that $\mathsf{Adv}^{\mathsf{sl}}_D(q,S) \in O(Sq\log(q)/D)$.

An immediate application of the switching lemma is that if $\mathcal{A}$ is an S-bounded adversary which makes q non-repeating queries to its oracle, then $|\mathsf{Adv}^{\mathsf{prf}}_{\mathrm{E}}(\mathcal{A}) - \mathsf{Adv}^{\mathsf{prp}}_{\mathrm{E}}(\mathcal{A})| \leqslant \mathsf{Adv}^{\mathsf{sl}}_D(q,S)$ for any block-cipher E whose range has size D.

AXU HASH FUNCTION. Let $\mathrm{H} : \{0,1\}^k \times (\{0,1\}^* \times \{0,1\}^*) \to \{0,1\}^n$ be a hash function. Its almost XOR-universal (AXU) security is defined by the game $\mathsf{G}^{\mathsf{axu}}_{\mathrm{H}}$ shown in Fig. 3.2. In it, an adversary $\mathcal{X}$ attempts to guess the xor of the output of H on two distinct inputs of its choosing for a random key L. We define $\mathsf{Adv}^{\mathsf{axu}}_{\mathrm{H}}(\mathcal{X}) = \mathsf{Pr}[\mathsf{G}^{\mathsf{axu}}_{\mathrm{H}}(\mathcal{X})]$. Typically one makes use of a c-AXU hash which for all $\mathcal{X}$ satisfy $\mathsf{Adv}^{\mathsf{axu}}_{\mathrm{H}}(\mathcal{X}) \leqslant c \cdot (N_1 + N_2)/2^n$ where N_1 (resp. N_2) is the maximum block length of any A (resp. C) output by $\mathcal{X}$. Note this is unconditional, so we will not have to worry about memory complexity when reducing to AXU security.

3.2 Nonce-based Encryption and Memory-boundedness

In this section we recall known definitions and results for nonce-based encryption [Rog04]. We carefully consider how these change when we move to the memory-bounded setting. For example, as was previously noted by Auerbach, et al. [ACFK17], definitions which are tightly equivalent when the memory usage of adversaries is not bounded do not necessarily remain so with bounds on memory. So we will consider several variants of the definitions we are recalling and try to reason about which is the "correct" one to use. We additionally note some results which can be extended to give appealing time-memory tradeoffs in the memory-bounded setting and some for which this does not seem to be possible.

In Section 3.2.1, we discuss INDR security which measures the indistinguishability of ciphertexts from truly random ones. This security notion requires that the adversary be disallowed from repeating nonces. We discuss three conventions for capturing this which are tightly equivalent when ignoring memory restrictions, but observe they are no longer tightly equivalent with these restrictions. Based on these discussions, the rest of the chapter focuses on the restricted class of adversaries that will never repeat nonces in their queries to encryption oracles. In Section 3.2.2, we discuss CTXT (integrity of ciphertexts) and AE security (combined INDR and CTXT) security. For these, the adversary must be disallowed from trivially winning by forwarding ciphertexts from its encryption oracle to its decryption oracle. Again we discuss several conventions for this which are tightly equivalent when ignoring memory restrictions. Based on these discussions, the rest of the chapter will use the convention that if an adversary queries (N, C) to its decryption oracle after receiving C from an encryption query for (N, M), the oracle will respond with M.

Finally, in Section 3.2.3 we recall the CAU scheme by Bellare and Tackmann [BT16], an abstraction of GCM [MV04]. Following existing proofs [BT16, IOM12, MV04] and using [JT19, Din20], we show that INDR security of CAU can be proven by a memory-tight reduction to PRP security with an appealing time-memory tradeoff and we informally discuss why such reductions seem impossible for CTXT or AE security.

SYNTAX AND CORRECTNESS. A (nonce-based) encryption scheme NE is defined by algorithms $\mathsf{NE.K}$, $\mathsf{NE.D}$, and $\mathsf{NE.E}$. Additionally it is associated with message space $\mathsf{NE.M} \subseteq \{0,1\}^*$ and nonce space $\mathsf{NE.N}$.

The syntax of the algorithms is shown in Fig. 3.3. The key generation algorithm $\mathsf{NE.K}$ takes no input

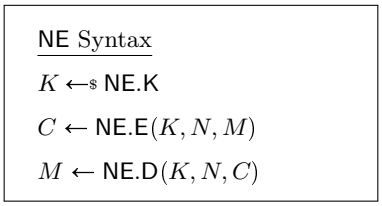


Figure 3.3: Syntax of nonce-based encryption scheme.

and returns key K. The encryption algorithm $\mathsf{NE.E}$ takes key K, nonce $N \in \mathsf{NE.N}$, and message $M \in \mathsf{NE.M}$. It returns ciphertext C. The decryption algorithm $\mathsf{NE.D}$ takes key K, nonce $N \in \mathsf{NE.N}$, and ciphertext C. It returns message $M \in \mathsf{NE.M} \cup \{\perp\}$. When $M = \perp$, the ciphertext is rejected as invalid.

We additionally assume there is a ciphertext-length function $\mathsf{NE.cl} : \mathbb{N} \to \mathbb{N}$ such that for any $K \in [\mathsf{NE.K}]$, $N \in \mathsf{NE.N}$, and $M \in \mathsf{NE.M}$ we have $|C| = \mathsf{NE.cl}(|M|)$ whenever $C \leftarrow \mathsf{NE.E}(K, N, M)$. Typically, a nonce-based encryption scheme also takes associated data as input which is authenticated during encryption. Associated data does not meaningfully effect our results, so we have omitted it for simplicity of notation.

Correctness of an encryption scheme requires for all $K \in [\mathsf{NE.K}]$, $N \in \mathsf{NE.N}$, and $M \in \mathsf{NE.M}$ that $\mathsf{NE.D}(K, N, \mathsf{NE.E}(K, N, M)) = M$.

3.2.1 Indistinguishability From Random (INDR) Security

The first security notion we will consider requires that ciphertexts output by the encryption scheme cannot be distinguished from ciphertexts chosen at random.

Definitions. Consider the game $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},b}$ shown in Fig. 3.4. Here an adversary $\mathcal{A}$ is given access to an encryption oracle Enc to which it can query a pair (N, M) and receive back either the encryption of message M with nonce N ($b = 1$) or a random string of the appropriate length ($b = 0$). The adversary outputs a bit trying to guess which of these two views it was given. We define $\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}(\mathcal{A})] - \mathsf{Pr}[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}(\mathcal{A})]$.

In defining security we must address how to handle the possibility of $\mathcal{A}$ making multiple queries with the same nonce. Encryption schemes are typically designed under the assumption that the same nonce will not be used multiple times and may become completely insecure in the face of such nonce repetition. The primary convention we will adopt is to restrict attention to adversaries that will never repeat nonces in their encryption queries. We use the phrase "nonce-respecting INDR" to refer to security with respect to such adversaries.

An alternate approach would be to modify the code of the game to respond appropriately to queries where nonces repeat. One version of this, which we will refer to as INDR-R, would restrict attention to

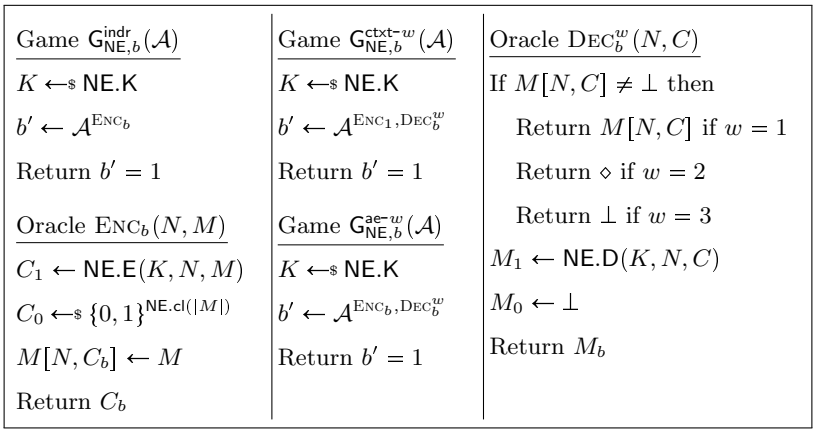


Figure 3.4: Games defining INDR, CTXT-w, and AE-w security of NE for $w \in \{1, 2, 3\}$.

adversaries that will only repeat nonces when they also repeat the message queried to encryption. For this the game would be modified to keep track of all encryption queries that have been made so far. When it receives a repeated (N, M) pair, it simply returns the same C that it returned last time it saw that pair. A second version of this, which we will refer to as INDR-B, makes no restriction on the queries of the adversary. Instead, the game is modified to return $\perp$ whenever the adversary makes a query with a nonce it has already used.

Discussion. When memory is not an issue, all of these variants would be equivalent. Proving this follows by noting that an adversary can just remember all prior queries it has made and thus never need to repeat. This proof strategy is no longer available to us when we want to preserve the memory usage of adversaries. We focus on nonce-respecting INDR because it hits the sweet spot of being strong enough for common applications, yet weak enough that we know how to give provable time-memory tradeoffs.

Because nonce-respecting INDR considers a strictly smaller class of adversaries than the other two and all of the games behave identically for this class of adversary it is tightly implied by the others. In fact, using ideas from [JT19, Din20] we can see that nonce-respecting INDR is strictly weaker. The toy encryption scheme SE considered in the introduction built from a block-cipher with block length n is vulnerable to low-memory collision-finding attacks with advantage $\Omega(q^2/2^n)$ in the INDR-R and INDR-B settings, but no attacks can have advantage better than $O(qs/2^n)$ in the nonce-respecting INDR setting. Here q and s refer to the number of queries and amount of memory used by the attackers, respectively. This underlies why the ideas of Jaeger and Tessaro [JT19] can be used to prove nonce-respecting INDR (but not INDR-R or INDR-B) time-memory tradeoffs for natural counter-mode based encryption schemes. In most common uses of nonce-based encryption the nonces are incremented as a counter or picked uniformly at random. In

the former case, nonces clearly never repeat so nonce-respecting INDR suffices (we will see this formally in Section 3.3). Nonces may repeat in the latter case, but we can follow [JT19, Din20] here and replace the uniform random values with random, non-repeating values so again nonce-respecting INDR suffices.

3.2.2 Security Beyond Confidentiality

INDR security only guarantees confidentiality of the messages against passive attackers. However, in practice, attackers may actively modify ciphertexts in transit. As such, it is important to consider security definition that take this into account. We will consider integrity definitions and authenticated encryption definitions which simultaneously asked for integrity and confidentiality.

DEFINITIONS. Consider the other two games shown in Fig. 3.4. We will first focus on $\mathsf{G}^{\mathsf{ae}\text{-}w}_{\mathsf{NE},b}$ which defines three variants of authenticated encryption security parameterized by $w \in \{1,2,3\}$. In this game, the adversary is given access to an encryption oracle and a decryption oracle. Its goal is to distinguish between a "real" and "ideal" world. In the real world ($b = 1$) the oracles uses NE to encrypt messages and decrypt ciphertexts. In the ideal world ($b = 0$) encryption returns random messages of the appropriate length and decryption returns $\perp$. For simplicity, we will restrict attention nonce-respecting adversaries which do not repeat nonces across encryption queries (as in nonce-respecting INDR security). Note there is no restriction placed on nonces used for decryption queries. Integrity of ciphertext security is defined by $\mathsf{G}^{\mathsf{ctxt}\text{-}w}_{\mathsf{NE},b}$ which behaves similarly except the adversary is always given access to the real encryption algorithm.

The decryption oracle needs to prevent trivial attacks. If the adversary receives C from a query of $\textsc{Enc}(N, M)$ and then queries $\textsc{Dec}(N, C)$ it would receive M in the real world and $\perp$ in the ideal world, making them easy to distinguish. We must adopt some convention for how the oracles behave when such a query is made to prevent this type of trivial attack. Towards this, the decryption oracle is parameterized by the value $w \in \{1,2,3\}$ corresponding to three different security notions. In all three, we use a table $M[\cdot,\cdot]$ to detect when the adversary forwards encryption queries on to its decryption oracle. When $w = 1$, the decryption oracle returns $M[N, C]$ in this case. When $w = 2$, it returns a special symbol $\diamond$. When $w = 3$, it returns the symbol $\perp$ which is also used by the encryption scheme to represent rejection. For $x \in \{\mathsf{ae}, \mathsf{ctxt}\}$ and $w \in \{1,2,3\}$ we define the advantage of an adversary $\mathcal{A}$ by $\mathsf{Adv}^{x\text{-}w}_{\mathsf{NE}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{x\text{-}w}_{\mathsf{NE},1}(\mathcal{A})] - \mathsf{Pr}[\mathsf{G}^{x\text{-}w}_{\mathsf{NE},0}(\mathcal{A})]$. The corresponding security notions are referred to as AE-w and CTXT-w.

DISCUSSION. When memory usage is not an issue, the choice of w does not matter. We can without loss of generality assume that the adversary never makes one of these trivial attack queries because it could simply store the table $M[\cdot,\cdot]$ for itself and simulate any such queries.[1] It's not clear that this equivalence holds if we do not assume that storing $M[\cdot,\cdot]$ is "free" for the adversary.

[1] Restricting attention to adversaries which never make trivial attack queries is, indeed, a fourth way one could define security.

The only memory-tight implication we are aware of between these is that security for $w = 2$ tightly implies security for $w = 3$. This follows because an adversary with access to DEC_b^2 can simulate DEC_b^3 with low memory. If DEC_b^2 returns $M = \diamond$ the adversary returns $\perp$, otherwise it does not modify M. All of the other implications we might want to show seem to require remembering all prior encryption queries to properly simulate DEC.

Ultimately, for heuristic reasons, we believe that $w = 1$ is the "correct" choice and will focus on it in our later sections. The typical motivation behind chosen-ciphertext security notions is that in practice an attacker can often observe the behavior of the decrypting party to learn something about the message they received. There is no reason to think an attacker should only be able to do that for ciphertexts that have been modified, but not ciphertexts that have been unmodified. This is best captured by $w = 1$. The $w = 2$ definition seems to posit that the adversary can distinguish between ciphertexts it forwarded on and ciphertexts that it modified (whether they were accepted or rejected) by observing the decrypting party's behavior. The $w = 3$ definition seems to posit that the adversary cannot learn anything about ciphertexts it forwards on unmodified, but can learn about other modified ciphertexts by observing the decrypting party's behavior.

REVISITING A CLASSIC RESULT. A classic result, which has been shown for numerous styles of encryption, is that confidentiality and integrity together imply authenticated encryption [RS06]. However, this becomes more difficult for nonce-based encryption when we consider memory-tightness.

The classic proof that INDR and CTXT-1 security imply AE-$\mathtt{m}$ security first replaces real decryption with $\perp$ via a reduction to CTXT-1 security and then replace real encryption with random using INDR security. However, in this second step the reduction adversary would have to simulate the oracle DEC_0^1 which seems to require storing the table $M[\cdot, \cdot]$.[2] This potentially requires using much more memory than the AE-$\mathtt{m}$ adversary, losing the benefit of time-memory tradeoffs for INDR-R. In this chapter we make it memory tight when restricting attention to secure channels which only accept ciphertexts if they are received in order. In Chapter 6, we give negative results showing that for nonce-based encryption this reduction cannot be made memory tight (using a black-box reduction).

3.2.3 Security of the CAU Encryption Scheme

We conclude this section by considering the specific encryption scheme CAU for which we can prove INDR security with a time-memory tradeoff. We will use this scheme in Section 3.3 to show a time-memory tradeoff for the authenticated encryption security of a channel instantiated with it.

One of the most widely deployed encryption schemes is Galois Counter-Mode (GCM) [MV04]. Bellare

[2] The standard reduction *would* be memory tight for $w = 3$.

Algorithm $\mathsf{CAU}[\mathrm{E},\mathrm{H}].\mathsf{E}(K,N,M)$	Algorithm $\mathsf{CAU}[\mathrm{E},\mathrm{H}].\mathsf{D}(K,N,T \,\|\, C)$
$Y \leftarrow \mathrm{pad}(N)$	$L \leftarrow E_K(0^n)$; $Y \leftarrow \mathrm{pad}(N)$
$M_1 \ldots M_\ell \leftarrow_n M$	$C_1 \ldots C_\ell \leftarrow_n C$
For $i = 1, ..., \ell$ do	$T' \leftarrow H_L(A, C) \oplus E_K(Y)$
$\quad C_i \leftarrow M_i \oplus E_K(Y + i)$	If $T \neq T'$ then return $\perp$
$C \leftarrow C_1 \ldots C_\ell$	For $i = 1, ..., \ell$ do
$L \leftarrow E_K(0^n)$	$\quad M_i \leftarrow C_i \oplus E_K(Y + i)$
$T \leftarrow H_L(A, C) \oplus E_K(Y)$	$M \leftarrow M_1 \ldots M_\ell$
Return $T \,\|\, C$	Return M

Figure 3.5: Encryption scheme CAU parameterized by function family E (typically a block cipher) and hash function H. In the code, $\mathrm{pad}(N) = N \,\|\, 0^m \,\|\, 1$ for the appropriate choice of m and $M_1 \ldots M_\ell \leftarrow_n M$ splits M into n-bit blocks.

and Tackmann [BT16] generalized it to the scheme CAU which constructs an encryption scheme from a block cipher E and hash function H. Using the techniques of Jaeger and Tessaro [JT19] we obtain a proof of security for its nonce-respecting INDR security with an appealing time-memory tradeoff.

Construction. We recall the CAU construction of an encryption scheme. Fix a key length $\mathsf{klCAU} \in \mathbb{N}$, a block length $n = \mathsf{CAU.bl} \in \mathbb{N}$, and a nonce length $\mathsf{CAU.nl} < \mathsf{CAU.bl}$. Then let E be a function family with $\mathrm{E} : \{0,1\}^{\mathsf{klCAU}} \times \{0,1\}^{\mathsf{CAU.bl}} \to \{0,1\}^{\mathsf{CAU.bl}}$ and H be a function family with $\mathrm{H} : \{0,1\}^{\mathsf{CAU.bl}} \times (\{0,1\}^* \times \{0,1\}^*) \to \{0,1\}^{\mathsf{CAU.bl}}$. The scheme constructed from E and H is denoted $\mathsf{CAU}[\mathrm{E},\mathrm{H}]$. Its message space $\mathsf{CAU}[\mathrm{E},\mathrm{H}].\mathsf{M}$ is the set of all strings of length at most $n \cdot (2^{n-\mathsf{CAU.nl}} - 1)$ and its nonce space $\mathsf{CAU}[\mathrm{E},\mathrm{H}].\mathsf{N}$ is the set $\{0,1\}^{\mathsf{CAU.nl}}$.

The algorithms of $\mathsf{CAU}[\mathrm{E},\mathrm{H}]$ are shown in Fig. 3.5. The code uses $\mathrm{pad}(\cdot)$ to denote the padding function which on input N outputs $N \,\|\, 0^{n-\mathsf{CAU}[\mathrm{E},\mathrm{H}].\mathsf{nl}-1} \,\|\, 1$. Since our simplified notation does not use associated data we instead assume there is a fixed associated data string A used with every message.

The encryption algorithm parses the input message into ℓ blocks of length n (except for the last, which may be shorter) and pads the nonce to a string Y of length n. It encrypts the message using counter-mode encryption with $Y+1$ as the first counter. This gives it a partial ciphertext C. The authentication is inspired by a Carter-Wegman MAC. A key L for the hash function is obtained as $L \leftarrow E_K(0^n)$. This key is used to compute the tag T as $T \leftarrow H_L(A, C) \oplus E_K(Y)$ and then $T \,\|\, C$ is the full ciphertext output by encryption.

The decryption algorithm parses the input ciphertext as $T \,\|\, C$. It computes the correct tag T' for C by setting $L \leftarrow E_K(0^n)$ and $T \leftarrow H_L(A, C) \oplus E_K(Y)$ (as was done in encryption). If $T \neq T'$ the ciphertext is

rejected by returning $M = \perp$. Otherwise the message M is obtained by counter-mode decrypting C.

INDR SECURITY OF CAU. The following theorem formalizes that CAU is nonce-respecting INDR secure assuming E is a secure PRF.

Theorem 3.2.1. *Let $\mathcal{A}$ be an adversary against the nonce-respecting* INDR *security of $\mathsf{CAU}[\mathrm{E},\mathrm{H}]$ that makes at most q oracle queries, each at most $p \cdot \mathsf{CAU.bl}$ bits long. Then we can construct a $\mathcal{A}_{\mathsf{p}}$ such that*

$$\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{CAU}[\mathrm{E},\mathrm{H}]}(\mathcal{A}) \leqslant \mathsf{Adv}^{\mathsf{prf}}_{\mathrm{E}}(\mathcal{A}_{\mathsf{p}}) \ .$$

Adversary $\mathcal{A}_{\mathsf{p}}$ has runtime essentially that of $\mathcal{A}$, makes at most $q(p+1)+1$ queries to its oracle, has memory/time complexity essentially that of $\mathcal{A}$ and never repeats queries to its oracle.

It is important that $\mathcal{A}_{\mathsf{p}}$ never repeats queries because it allows us to apply the time-memory switching lemma from Section 11.4.3. This gives us roughly,

$$\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{CAU}[\mathrm{E},\mathrm{H}]}(\mathcal{A}) \in \mathsf{Adv}^{\mathsf{prp}}_{\mathrm{E}}(\mathcal{A}_{\mathsf{p}}) + O(S \cdot pq \cdot \log(pq)/2^n)$$

where S is a bound on the memory complexity of $\mathcal{A}$. For variants other than nonce-respecting INDR it would not be clear how to prevent $\mathcal{A}_{\mathsf{p}}$ from repeating queries without storing the prior queries of $\mathcal{A}$.

Proof. [Sketch] One constructs $\mathcal{A}_{\mathsf{p}}$ to first set $L \leftarrow \textsc{Eval}(0^n)$. Then it runs $\mathcal{A}$ and simulates encryption queries by running $\mathsf{CAU.E}$ while using its EVAL oracle in place of E_K. It does not recompute L each time because it has already computed it. Its final output is whatever $\mathcal{A}$ outputs. One can verify that the view of $\mathcal{A}$ when simulated by $\mathcal{A}_{\mathsf{p}}$ is "real" encryptions when $b = 1$ and random strings when $b = 0$, so the claimed advantage bound follows. ■

CTXT/AE SECURITY OF CAU. It does not appear to be possible to give a similar time-memory tradeoff for the CTXT or AE security of CAU. The standard analysis of either of these first uses PRF security to replace the output of E with random. It then argues that the adversary's view is independent of the $\mathrm{H}_L(A,C)$ values produced in encryption so that it can apply the security of H. For $x = \mathsf{ae}$ or $x = \mathsf{ctxt}$ this would give a bound of the form,

$$\mathsf{Adv}^{x\text{-}1}_{\mathsf{CAU}[\mathrm{E},\mathrm{H}]}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{prf}}_{\mathrm{E}}(\mathcal{A}_{\mathsf{p}}) + \mathsf{Adv}^{\mathsf{axu}}_{\mathrm{H}}(\mathcal{X}) \ .$$

However, this PRF adversary $\mathcal{A}_{\mathsf{p}}$ needs to to simulate a decryption oracle to $\mathcal{A}$. The natural ways of doing this (remembering all prior encryption queries or using EVAL to run decryption) either require significant use of memory or repeating queries to EVAL. This prevents us from applying the switching lemmas of [JT19, Din20] to get appealing time-memory tradeoffs when E is a PRP.

In Section 3.3.3, we will use a new technique for memory-tight reductions to prove that using CAU in a channel can provide (the channel equivalent of) CTXT security (and thus AE security from Section 3.3.2).

$\underline{\mathsf{CH}\text{ Syntax}}$	$\underline{\text{Game } \mathsf{G}^{\mathsf{ch\text{-}corr}}_{\mathsf{CH},b}(\mathcal{A})}$	$\underline{\text{Oracle } \textsc{EncDec}_b(M_0)}$
$(\sigma^s, \sigma^r) \leftarrow\!\!\$\ \mathsf{CH.Sg}$	$(\sigma^s, \sigma^r) \leftarrow\!\!\$\ \mathsf{CH.Sg}$	$(\sigma^s, C) \leftarrow\!\!\$\ \mathsf{CH.S}(\sigma^s, M_0)$
$(\sigma^s, C) \leftarrow\!\!\$\ \mathsf{CH.S}(\sigma^s, M)$	$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{EncDec}}$	$(\sigma^r, M_1) \leftarrow \mathsf{CH.R}(\sigma^r, C)$
$(\sigma^r, M) \leftarrow \mathsf{CH.R}(\sigma^r, C)$	Return $b' = 1$	Return M_b

Figure 3.6: **Left:** Syntax of channel algorithms. **Right:** Channel correctness game.

3.3 Memory-tight Reductions for Cryptographic Channels

In this section we show that memory-tight reductions can be given for the combined confidentiality and integrity security of cryptographic channels. These are a form of stateful encryption which provide the guarantee that messages cannot be duplicated or reordered, in addition to the typical confidentiality and integrity goals of encryption.

3.3.1 Syntax and Security Notions

Syntax and correctness. A (cryptographic) channel CH specifies algorithms $\mathsf{CH.Sg}$, $\mathsf{CH.S}$, and $\mathsf{CH.R}$ along with message space $\mathsf{CH.M} \subseteq \{0,1\}^*$. The syntax of these algorithms is shown in Fig. 3.6. The state generation algorithm $\mathsf{CH.Sg}$ takes no input. It returns sender state σ^s and receiver state σ^r. The sending algorithm $\mathsf{CH.S}$ takes a sender state σ^s and message $M \in \mathsf{CH.M}$. It returns updated sender state σ^s and a ciphertext C. The receiving algorithm $\mathsf{CH.R}$ takes a receiver state σ^r and a ciphertext C. It returns updated receiver state σ^r and a message $M \in \mathsf{CH.M} \cup \{\perp\}$. When $M = \perp$, this represents the receiver rejecting the message as invalid.

A channel is expected to never again return $M \neq \perp$ after if it has rejected a message. This models the behavior of protocols such as TLS which are assumed to be run over a reliable transport layer and has been the standard notion for channels since the work of Bellare, Kohno, and Namprempre [BKN04]. When a protocol (e.g. QUIC or DTLS) is run over an unreliable transport layer, then a *robust* channel is used instead [FGJ20]. We leave memory-tight proofs of security for robust channels as an interesting direction for future work.

We typically assume there is a ciphertext-length function $\mathsf{CH.cl} : \mathbb{N} \to \mathbb{N}$ such that for any $M \in \mathsf{CH.M}$ and state σ^s, we have $\mathsf{Pr}[|C| = \mathsf{CH.cl}(|M|) : (\sigma^s, C) \leftarrow\!\!\$\ \mathsf{CH.S}(\sigma^s, M)] = 1$.

Correctness requires that if the receiver is given the ciphertexts sent by the sender in order and without modification then the receiver will output the same sequence of messages that were sent. One way to formalize this is via the game $\mathsf{G}^{\mathsf{ch\text{-}corr}}_{\mathsf{CH},b}$ shown in Fig. 3.6. We define $\mathsf{Adv}^{\mathsf{ch\text{-}corr}}_{\mathsf{CH}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{\mathsf{ch\text{-}corr}}_{\mathsf{CH},1}(\mathcal{A})] - \mathsf{Pr}[\mathsf{G}^{\mathsf{ch\text{-}corr}}_{\mathsf{CH},0}(\mathcal{A})]$.

Perfect correctness requires that $\mathsf{Adv}^{\mathsf{ch\text{-}corr}}_{\mathsf{CH}}(\mathcal{A}) = 0$ for all (even unbounded) $\mathcal{A}$. This implies that the M_1 output by $\mathsf{CH.R}$ always equals M_0.

SECURITY DEFINITIONS. We consider indistinguishability from random, integrity of ciphertext, and authenticated encryption security for channels just like we did for nonce based encryption.

Authenticated encryption security of a channel CH is defined by game $\mathsf{G}^{\mathsf{ch\text{-}ae}}_{\mathsf{CH},b}$ defined in Fig. 3.7. In it the adversary is given access to an encryption oracle and a decryption oracle. The adversary's goal is to distinguish between a "real" and "ideal" world. In the real world ($b = 1$) the oracles use CH to encrypt messages and decrypt ciphertexts. In the ideal world ($b = 0$) encryption returns random messages of the appropriate length and decryption returns $\perp$. In both worlds, as long as the adversary's queries to decryption have consisted of the outputs of encryption in the correct order, the oracles are considered in sync and decryption just returns the appropriate message that was queried to encryption.[3] After the first time the adversary queries something else, the oracles are out of sync and will never be in sync again (so DEC will always return M_b).

Authenticated encryption security is a combined confidentiality and integrity notion. We can also define separate notions. INDR security is defined by the game $\mathsf{G}^{\mathsf{ch\text{-}indr}}_{\mathsf{CH},b}$ which is the same as $\mathsf{G}^{\mathsf{ch\text{-}ae}}_{\mathsf{CH},b}$ except the adversary is only given oracle access to $\textsc{Enc}_b$. CTXT security is defined by the game $\mathsf{G}^{\mathsf{ch\text{-}ctxt}}_{\mathsf{CH},b}$ which is the same as $\mathsf{G}^{\mathsf{ch\text{-}ae}}_{\mathsf{CH},b}$ except the adversary is given oracle access to $\textsc{Enc}_1$ and $\textsc{Dec}_b$. These games are given explicitly in Fig. 3.7. We define the advantage of $\mathcal{A}$ by $\mathsf{Adv}^{x}_{\mathsf{CH}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{x}_{\mathsf{CH},1}(\mathcal{A})] - \mathsf{Pr}[\mathsf{G}^{x}_{\mathsf{CH},0}(\mathcal{A})]$ for $x \in \{\mathsf{ch\text{-}ae}, \mathsf{ch\text{-}indr}, \mathsf{ch\text{-}ctxt}\}$.

3.3.2 Confidentiality and Integrity Imply Authenticated Encryption

We will show that INDR security plus CTXT security imply AE security using a memory-tight reduction. While the normal proof that INDR and CTXT security suffice to imply AE security is not particularly difficult, it uses a non-memory tight reduction to INDR security. Making the proof memory tight will require more involved analysis.

INFORMATION THEORETIC LEMMA. Before proceeding to the proof, we first will provide a simple information theoretic lemma that will be a useful subcomponent of that proof. Consider the game $\mathsf{G}^{\mathsf{it}}_{L,\delta}$ shown in Fig. 3.8. In it, an adversary is given a length L string R and tries to choose an index i for which it is able to remember the next δ-bits of the string using state σ. We say that an adversary $(\mathcal{A}_1, \mathcal{A}_2)$ is S-bounded if $|\sigma| = S$ always. We define $\mathsf{Adv}^{\mathsf{it}}_{L,\delta}(\mathcal{A}_1, \mathcal{A}_2) = \mathsf{Pr}[\mathsf{G}^{\mathsf{it}}_{L,\delta}(\mathcal{A}_1, \mathcal{A}_2)]$.

[3] This matches the convention of CTXT-1 and AE-$\mathtt{m}$ for encryption schemes. We believe it to be "correct" for the same reasons discussed for those definitions.

Game $\mathsf{G}^{\mathsf{ch\text{-}ae}}_{\mathsf{CH},b}(\mathcal{A})$	Game $\mathsf{G}^{\mathsf{ch\text{-}ctxt}}_{\mathsf{CH},b}(\mathcal{A})$	Oracle $\textsc{Dec}_b(C)$
$\mathsf{sync} \leftarrow \mathsf{true}$	$\mathsf{sync} \leftarrow \mathsf{true}$	$(\sigma^r, M_1) \leftarrow \mathsf{CH.R}(\sigma^r, C)$
$(\sigma^s, \sigma^r) \leftarrow\!\!\$\ \mathsf{CH.Sg}$	$(\sigma^s, \sigma^r) \leftarrow\!\!\$\ \mathsf{CH.Sg}$	$M_0 \leftarrow \bot$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Enc}_b, \textsc{Dec}_b}$	$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Enc}_1, \textsc{Dec}_b}$	$M' \leftarrow \mathbf{M}.\mathsf{dq}()$
Return $b' = 1$	Return $b' = 1$	$C' \leftarrow \mathbf{C}.\mathsf{dq}()$
Game $\mathsf{G}^{\mathsf{ch\text{-}indr}}_{\mathsf{CH},b}(\mathcal{A})$	Oracle $\textsc{Enc}_b(M)$	If sync then
$\mathsf{sync} \leftarrow \mathsf{true}$	$(\sigma^s, C_1) \leftarrow\!\!\$\ \mathsf{CH.S}(\sigma^s, M)$	If $C = C'$ then
$(\sigma^s, \sigma^r) \leftarrow\!\!\$\ \mathsf{CH.Sg}$	$C_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{CH.cl}(\lvert M \rvert)}$	Return M'
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Enc}_b}$	$\mathbf{M}.\mathsf{add}(M)$; $\mathbf{C}.\mathsf{add}(C_b)$	$\mathsf{sync} \leftarrow \mathsf{false}$
Return $b' = 1$	Return C_b	Return M_b

Figure 3.7: Games defining the INDR, CTXT, and AE security of a channel.

Game $\mathsf{G}^{\mathsf{it}}_{L,\delta}(\mathcal{A}_1, \mathcal{A}_2)$

$R \leftarrow\!\!\$\ \{0,1\}^L$

$(i, \sigma) \leftarrow\!\!\$\ \mathcal{A}_1(R)$

$r \leftarrow\!\!\$\ \mathcal{A}_2(i, \sigma, R[:i-1])$

Return $r = R[i : i+\delta]$

Figure 3.8: Information theoretic game in which $\mathcal{A}$ tries to remember a δ bit sequence in an L-bit random string.

Lemma 3.3.1. *Let $L, \delta, S \in \mathbb{N}$. Let $(\mathcal{A}_1, \mathcal{A}_2)$ be an S-bounded adversary. Then*

$$\mathsf{Adv}^{\mathsf{it}}_{L,\delta}(\mathcal{A}_1, \mathcal{A}_2) \leqslant L \cdot 2^S / 2^\delta \ .$$

Proof. Let $L, \delta, S, \mathcal{A}_1, \mathcal{A}_2$ be defined as in the theorem statement. Without loss of generality we can assume that $\mathcal{A}_1$ and $\mathcal{A}_2$ are deterministic. Then for any fixed choice of i and σ, the probability that $\mathcal{A}_2(i, \sigma, R[:i-1]) = R[i : i+\delta]$ will be exactly $1/2^\delta$. Then we can calculate as follows.

$$\begin{aligned}
\mathsf{Pr}[\mathsf{G}^{\mathsf{it}}_{L,n}(\mathcal{A}_1, \mathcal{A}_2)] &\leqslant \mathsf{Pr}_R[\exists i, \sigma \text{ s.t. } R[i : i+\delta] = \mathcal{A}_2(i, \sigma, R[:i-1])] \\
&\leqslant \sum_{i,\sigma} \mathsf{Pr}[R[i : i+\delta] = \mathcal{A}_2(i, \sigma, R[:i-1])] \\
&= \sum_{i,\sigma} 1/2^\delta \leqslant L \cdot 2^S / 2^\delta \ .
\end{aligned}$$

The last inequality follows from there being at most $L \cdot 2^{\delta}$ choices for (i, σ).

■

SECURITY RESULT. Now we can proceed to our security result showing that AE security can be implied by INDR and CTXT security in a memory-tight manner. The technical crux of the result is the reduction adversary $\mathcal{A}_\delta$ which simulates the view of an AE adversary $\mathcal{A}$ to attack the INDR security of the channel. In our theorem statement this reduction adversary is parameterized by a variable δ which determines how much local memory it uses. Using Lemma 3.3.1, our concrete advantage bound is expressed in terms of δ and establishes that the reduction can be successful with this value not much larger than the local memory of $\mathcal{A}$.

Theorem 3.3.2. *Let* CH *be a cryptographic channel. Let* $\mathcal{A}$ *be an adversary with memory complexity* S *and making at most* q *queries to its* ENC *oracle, each of which returns a ciphertext of length at most* x*. Then for any* $\delta \in \mathbb{N}$ *we can build an adversary* $\mathcal{A}_\delta$ *(described in the proof) such that*

$$\mathsf{Adv}^{\mathsf{ch\text{-}ae}}_{\mathsf{CH}}(\mathcal{A}) \leqslant \mathsf{Adv}^{\mathsf{ch\text{-}ctxt}}_{\mathsf{CH}}(\mathcal{A}) + 2 \cdot \mathsf{Adv}^{\mathsf{ch\text{-}indr}}_{\mathsf{CH}}(\mathcal{A}_\delta) + 2q \cdot x \cdot 2^S/2^\delta \ .$$

Adversary $\mathcal{A}_\delta$ *has running time approximately that of* $\mathcal{A}$ *and uses about* $S + 2\delta$ *bits of state.*

Setting $\delta = S + \log(qx) + \kappa$ makes the last term about $1/2^\kappa$ while limiting the memory usage of $\mathcal{A}_\delta$ to only $2S + 2\log(qx) + 2\kappa$.

The standard way of proving that INDR security and CTXT security imply AE security would first use CTXT security to transition from a world in which $\mathcal{A}$ is given oracle access to $(\text{ENC}_1, \text{DEC}_1)$ to a world in which $\mathcal{A}$ is given oracle access to $(\text{ENC}_1, \text{DEC}_0)$. Then INDR security would be used to transition to $\mathcal{A}$ being given oracle access to $(\text{ENC}_0, \text{DEC}_0)$. The issue in our setting with this proof arises in the second step. The INDR reduction adversary needs to simulate DEC_0 for $\mathcal{A}$. The natural way of doing so requires storing the entirety of the tables $\mathbf{M}$ and $\mathbf{C}$ which means that $\mathcal{A}_\delta$ may use much more memory than $\mathcal{A}$.

Our proof of Thm. 3.3.2 follows this same general proof flow, but uses a more involved analysis for the reduction to INDR security. In particular, we make use of the following insight: If $\mathcal{A}$ has memory complexity S but cannot distinguish the ciphertexts it sees from random (because of INDR security), then from Lemma 3.3.1 it cannot remember many more than S of the ciphertext bits that it has received from ENC but not yet forwarded to DEC.

If $\mathcal{A}$ ever queries a ciphertext which is not the next ciphertext in $\mathbf{C}$, then DEC_0 oracle will never again return anything other than $\perp$. Because we can assume that $\mathcal{A}$ will be unable to remember too many bits of ciphertext, we can just have our reduction adversary $\mathcal{A}_\delta$ remember a few more bits of ciphertext than $\mathcal{A}$ can. If the total length of ciphertext that $\mathcal{A}$ has received from its encryption oracle, but not forwarded on to its

decryption oracle ever exceeds the amount that $\mathcal{A}_\delta$ will store, then $\mathcal{A}_\delta$ assumes $\mathcal{A}$ must have forgotten some intermediate ciphertext before that point, allowing the reduction to cease storing future ciphertexts because sync will be false before that point.

Proof. We will construct INDR adversaries $\mathcal{A}'_\delta$, $\mathcal{A}''_\delta$, and S-bounded adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and show that

$$\mathsf{Adv}^{\mathsf{ch\text{-}ae}}_{\mathsf{CH}}(\mathcal{A}) \leqslant \mathsf{Adv}^{\mathsf{ch\text{-}ctxt}}_{\mathsf{CH}}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{ch\text{-}indr}}_{\mathsf{CH}}(\mathcal{A}'_\delta) + 2 \cdot \mathsf{Adv}^{\mathsf{it}}_{q\cdot x,\delta}(\mathcal{A}_1, \mathcal{A}_2) + \mathsf{Adv}^{\mathsf{ch\text{-}indr}}_{\mathsf{CH}}(\mathcal{A}''_\delta)\ .$$

The stated theorem then follows by applying Lemma 3.3.1 and constructing the adversary $\mathcal{A}_\delta$ which runs either $\mathcal{A}'_\delta$ or $\mathcal{A}''_\delta$ (chosen at random) and outputs whatever that adversary does. The resulting $\mathcal{A}_\delta$ will satisfy the efficiency constraints stated in the theorem statement. We will prove this bound via a sequence of transformations that slowly change $\mathsf{G}^{\mathsf{ch\text{-}ae}}_{\mathsf{CH},1}$ to $\mathsf{G}^{\mathsf{ch\text{-}ae}}_{\mathsf{CH},0}$.

CTXT TRANSITION. Let $\mathsf{G}_0 = \mathsf{G}^{\mathsf{ch\text{-}ae}}_{\mathsf{CH},1}(\mathcal{A})$ and $\mathsf{G}_1 = \mathsf{G}^{\mathsf{ch\text{-}ctxt}}_{\mathsf{CH},0}(\mathcal{A})$. Because $\mathsf{G}^{\mathsf{ch\text{-}ae}}_{\mathsf{CH},1}(\mathcal{A})$ and $\mathsf{G}^{\mathsf{ch\text{-}ctxt}}_{\mathsf{CH},1}(\mathcal{A})$ are identical games we have that $\mathsf{Pr}[\mathsf{G}_0] - \mathsf{Pr}[\mathsf{G}_1] = \mathsf{Adv}^{\mathsf{ch\text{-}ctxt}}_{\mathsf{CH}}(\mathcal{A})$.

TRANSITION TO LIMITED MEMORY GAME. Next we want to transition to a version of G_1 that stores a bounded amount of local state. Consider the games G_2 and G_3 shown in Fig. 3.9. The tables $\mathbf{M}_2$ and $\mathbf{C}_2$ track the messages and ciphertexts as in the real game. Because of this $\mathsf{Pr}[\mathsf{G}_1] = \mathsf{Pr}[\mathsf{G}_2]$.

In the transition to G_3 we are going to stop using these tables and instead solely rely on the tables $\mathbf{M}$ and $\mathbf{C}$. With these tables, if the total number of bits of ciphertexts that would be stored in $\mathbf{C}$ exceeds δ then we permanently stop adding elements to these tables – we assume that the adversary will cause sync to be set to false at some point earlier in the game. Note that up until this point the tables $(\mathbf{M}_2, \mathbf{C}_2)$ and $(\mathbf{M}, \mathbf{C})$ are used identically. The two games only differ in the boxed code in DEC which returns M_2 if the adversary has queried a ciphertext stored in $\mathbf{C}_2$ that was not stored in $\mathbf{C}$. Hence, these games are identical-until-bad so the Fundamental Lemma of Game Playing [BR06] gives,

$$\mathsf{Pr}[\mathsf{G}_3] - \mathsf{Pr}[\mathsf{G}_2] \leqslant \mathsf{Pr}[\mathsf{G}_3 \text{ sets } \mathsf{bad}].$$

We want to apply Lemma 3.3.1 to bound the probability that bad is set. To do so we need to be able to treat the ciphertexts as random strings. Thus we defer the analysis of the probability that it occurs until after applying INDR security.

INDR TRANSITION. Now consider the game G_4. It is identical to G_3 except that the ciphertexts returned by ENC are chosen at random instead of using CH. We can transition to this game using a reduction to INDR security. It is important here that our reduction adversary will not need to use too much memory because of the way that we have limited the memory needed for G_3.

Consider the adversaries $\mathcal{A}_\delta$ and $\mathcal{A}'_\delta$ shown in Fig. 3.10. Highlighted code is only included in the latter adversary.

Games $\boxed{\mathsf{G}_2}$, G_3, G_4, $\boxed{\mathsf{G}_5}$	Oracle $\textsc{Enc}(M)$	Oracle $\textsc{Dec}(C)$
$\mathsf{flag} \leftarrow \mathsf{true}$	$(\sigma^s, C) \leftarrow\!\!\$\ \mathsf{CH.S}(\sigma^s, M)$	$M' \leftarrow \mathbf{M}.\mathsf{dq}();\ M_2 \leftarrow \mathbf{M}_2.\mathsf{dq}()$
$\mathsf{sync} \leftarrow \mathsf{true}$	$C \leftarrow\!\!\$\ \{0,1\}^{\mathsf{CH.cl}(\lvert M\rvert)}$	$C' \leftarrow \mathbf{C}.\mathsf{dq}();\ C_2 \leftarrow \mathbf{C}_2.\mathsf{dq}()$
$(\sigma^s, \sigma^r) \leftarrow\!\!\$\ \mathsf{CH.Sg}$	$\mathbf{M}_2.\mathsf{add}(M);\ \mathbf{C}_2.\mathsf{add}(C)$	If sync then
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Enc},\textsc{Dec}}$	If flag then	If $C = C'$ then
Return $b' = 1$	If $\lvert\lvert\mathbf{C}\rvert\rvert + \lvert C\rvert < \delta$ then	Return M'
	$\mathbf{M}.\mathsf{add}(M);\ \mathbf{C}.\mathsf{add}(C)$	Elif $C = C_2$ then
	Else	$\mathsf{bad} \leftarrow \mathsf{true}$
	$\mathsf{flag} \leftarrow \mathsf{false}$	$\boxed{\text{Return } M_2}$
	Return C	$\mathsf{sync} \leftarrow \mathsf{false}$
		Return $\perp$

Figure 3.9: Hybrid games for proof of Theorem 3.3.2. Highlighted code is only included in highlighted games. Boxed code is only included in boxed games.

Adversary $\mathcal{A}'_\delta$ uses its $\textsc{Enc}$ oracle to present $\mathcal{A}$ with a view identical to G_3 if $b = 1$ and identical to G_4 if $b = 0$. Note here that the tables $(\mathbf{M}_2, \mathbf{C}_2)$ do not effect the view of $\mathcal{A}$ in either of these game, allowing $\mathcal{A}_\delta$ not to have to store them. We have that $\mathsf{Pr}[\mathsf{G}^{\mathsf{ch\text{-}indr}}_{\mathsf{CH},1}(\mathcal{A}'_\delta)] = \mathsf{Pr}[\mathsf{G}_3]$ and $\mathsf{Pr}[\mathsf{G}^{\mathsf{ch\text{-}indr}}_{\mathsf{CH},0}(\mathcal{A}'_\delta)] = \mathsf{Pr}[\mathsf{G}_4]$. In other words, $\mathsf{Adv}^{\mathsf{ch\text{-}indr}}_{\mathsf{CH}}(\mathcal{A}'_\delta) = \mathsf{Pr}[\mathsf{G}_3] - \mathsf{Pr}[\mathsf{G}_4]$.

Adversary $\mathcal{A}''_\delta$ instead uses its INDR oracle to simulate the view of $\mathcal{A}$, but returns 1 if the flag bad would have been set. Because this can only be set by the first ciphertext not stored in $\mathbf{C}$ we only need to be able to simulate the games up until that point. So we store this extra ciphertext and put an $*$ in $\mathbf{C}$ so that in $\textsc{Dec}$ we know when we have reached the relevant point. We have that $\mathsf{Pr}[\mathsf{G}^{\mathsf{ch\text{-}indr}}_{\mathsf{CH},1}(\mathcal{A}''_\delta)] = \mathsf{Pr}[\mathsf{G}_3 \text{ sets } \mathsf{bad}]$ and $\mathsf{Pr}[\mathsf{G}^{\mathsf{ch\text{-}indr}}_{\mathsf{CH},0}(\mathcal{A}''_\delta)] = \mathsf{Pr}[\mathsf{G}_4 \text{ sets } \mathsf{bad}]$. In other words, $\mathsf{Pr}[\mathsf{G}_3 \text{ sets } \mathsf{bad}] \leqslant \mathsf{Adv}^{\mathsf{ch\text{-}indr}}_{\mathsf{CH}}(\mathcal{A}''_\delta) + \mathsf{Pr}[\mathsf{G}_4 \text{ sets } \mathsf{bad}]$.

Final transition. The final transition is from G_4 to G_5. These two games are identical-until-bad as can be seen in $\textsc{Dec}$. Because of this we have that

$$\mathsf{Pr}[\mathsf{G}_4] - \mathsf{Pr}[\mathsf{G}_5] \leqslant \mathsf{Pr}[\mathsf{G}_4 \text{ sets } \mathsf{bad}].$$

Using all of $\mathbf{M}_2$ and $\mathbf{C}_2$ instead of just $\mathbf{M}$ and $\mathbf{C}$ makes G_5 identical to $\mathsf{G}^{\mathsf{ch\text{-}ae}}_{\mathsf{CH},0}$.

Bounding probability of bad. We conclude by bounding the probability G_4 sets bad via a reduction to our information theoretic analysis. Consider the S-bounded $(\mathcal{A}_1, \mathcal{A}_2)$ that behaves as follows. First, $\mathcal{A}_1$ internally simulates the view of $\mathcal{A}$ in G_4 using the coins for $\mathcal{A}$ which maximize the probability of bad and using the bits of R as the ciphertext bits returned by encryption. If $\mathcal{A}$ causes flag to be set to false, $\mathcal{A}_1$ will

Adversary $\mathcal{A}'^{\text{Enc}}_{\delta}$	Oracle SimEnc(M)	Oracle SimDec(C)
flag $\leftarrow$ true	flag $\leftarrow$ true	$M' \leftarrow \mathbf{M}.\mathsf{dq}()$; $C' \leftarrow \mathbf{C}.\mathsf{dq}()$
sync $\leftarrow$ true	sync $\leftarrow$ true	If sync then
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{SimEnc},\text{SimDec}}$	$C \leftarrow\!\!\$\ \text{Enc}(M)$	If $C = C'$ then
Return $b' = 1$	If flag then	Return M'
	If $\|\|\mathbf{C}\|\| + \|C\| < \delta$ then	Elif $C' = *$ and $C = C^*$ then
Adversary $\mathcal{A}''^{\text{Enc}}_{\delta}$	$\mathbf{M}.\mathsf{add}(M)$; $\mathbf{C}.\mathsf{add}(C)$	**abort**(1)
flag $\leftarrow$ true	Else	sync $\leftarrow$ false
sync $\leftarrow$ true	$\mathbf{C}.\mathsf{add}(*)$; $C^* \leftarrow C$	Return $\perp$
$\mathcal{A}^{\text{SimEnc},\text{SimDec}}$	flag $\leftarrow$ false	
Return 0	Return C	

Figure 3.10: INDR adversaries for proof of Theorem 3.3.2. Highlighting indicates code that is only used by adversary $\mathcal{A}'_{\delta}$.

halt and output the current state of $\mathcal{A}$ as σ with i chosen so the next δ bits of $\mathbf{C}$ and c are the values of R for $\mathcal{A}_2$ to guess.

Then $\mathcal{A}_2$ will resume executing $\mathcal{A}$ using σ. When $\mathcal{A}$ makes encryption queries it will just make up its own responses. When $\mathcal{A}$ makes a decryption query for a ciphertext C then $\mathcal{A}_2$ will concatenate it into its guess r. It just assumes this was the correct next ciphertext that should have been stored in $\mathbf{C}$ (otherwise $\mathcal{A}$ would fail in setting bad). To determine which M to return for this query, $\mathcal{A}_2$ re-runs $\mathcal{A}$ from the beginning using the same coins $\mathcal{A}_1$ used. It uses its given prefix of R and the current value of r to respond to encryption queries until it reaches the encryption query corresponding to the current decryption query. Whatever message $\mathcal{A}$ queried for this encryption query is then returned for the decryption query. Once r is δ bits long, $\mathcal{A}_2$ outputs that as its guess.

We can see that when bad would be set in G_4, the view of $\mathcal{A}$ is perfectly simulated up until that point and $\mathcal{A}_2$ will guess r correctly. This gives us $\mathsf{Pr}[\mathsf{G}_4 \text{ sets } \mathsf{bad}] \leqslant \mathsf{Adv}^{\mathsf{it}}_{q\cdot x,\delta}(\mathcal{A}_1, \mathcal{A}_2)$ as desired.

Combining all the bounds we have shown completes the proof. ■

3.3.3 AE Security of a TLS 1.3-like Channel

We have shown that the AE security of a channel can be reduced to its constituent INDR and CTXT security in a way that preserves memory complexity. This is, of course, only meaningful if we have channels for which we can give provable time-memory tradeoffs for their INDR and CTXT security. Using the ideas of Jaeger

$\underline{\mathsf{NCH[NE].Sg}}$	$\underline{\mathsf{NCH[NE].S}((K,N),M)}$	$\underline{\mathsf{NCH[NE].R}((K,N),C)}$
$K \xleftarrow{\$} \mathsf{NE.K}$	$N \leftarrow N+1$	If $N = \bot$ then
$N \xleftarrow{\$} \mathsf{NE.N}$	$C \leftarrow \mathsf{NE.E}(K,N,M)$	Return $((\bot,\bot),\bot)$
Return $((K,N),(K,N))$	Return $((K,N),C)$	$N \leftarrow N+1$
		$M \leftarrow \mathsf{NE.D}(K,N,C)$
		If $M = \bot$ then
		Return $((\bot,\bot),\bot)$
		Return $((K,N),M)$

Figure 3.11: Algorithms of channel NCH[NE] constructed from encryption scheme NE.

and Tessaro [JT19] it is easy to give such examples for INDR security.

Using the ideas from proof of Thm. 3.3.2 we will prove the security of a channel based on GCM (or more generally CAU). The resulting channel can be viewed as a (simplified) version of the channel obtained by using GCM in TLS 1.3.

The construction. The construction we consider is a straightforward construction of a channel from a nonce-based encryption scheme NE by using a counter for the nonce. The INDR security of this channel follows easily from the nonce-respecting INDR security of NE. Proving integrity of the channel from the integrity of NE is possible, but of limited applicability since we do not have examples of encryption schemes with proven time-memory tradeoffs for integrity. We will instead only show integrity for the specific case that NE = CAU.

The channel NCH[NE] is parameterized by an encryption scheme NE. It has NCH[NE].M = NE.M. We assume that NE.N can be interpreted as a cyclic group written using additive notation. Its algorithms are shown in Fig. 3.11. State generation sets the state of both parties equal to a shared random key and nonce. Encryption increments the nonce and uses NE to encrypt the message with the current nonce. Decryption increments the nonce and uses NE to decrypt the ciphertext with the current nonce. If the ciphertext is rejected ($M = \bot$), the receiver will replace its state with $\bot$'s. Henceforth it will reject all ciphertexts it receives (via the first line which checks if $N = \bot$ already holds.

INDR Security. The INDR security of NCH[NE] follows easily from nonce-respecting INDR security of NE. This is captured by the following theorem.

Theorem 3.3.3. *Let $\mathcal{A}$ be an adversary against the* INDR *security of* NCH[NE] *that makes less than* |NE.N|

oracle queries. Then we can construct $\mathcal{B}$ such that

$$\mathsf{Adv}^{\mathsf{ch\text{-}indr}}_{\mathsf{NCH[NE]}}(\mathcal{A}) \leqslant \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{B}) \ .$$

Adversary $\mathcal{B}$ has complexity comparable to that of $\mathcal{A}$ and is nonce-respecting.

Proof. [Sketch] Adversary $\mathcal{B}$ picks N at random and then starts executing $\mathcal{A}$. Whenever $\mathcal{A}$ makes a $\textsc{Enc}(M)$ query, $\mathcal{B}$ increments N, queries $C \leftarrow \textsc{Enc}(N, M)$, and returns C to $\mathcal{A}$. Adversary $\mathcal{B}$ outputs whatever $\mathcal{A}$ does. Verifying the claims made about this adversary is straightforward. ■

CTXT Security. For CTXT security we need to focus our attention on the particular construction of $\mathsf{NCH[NE]}$ obtained when using the encryption scheme $\mathsf{NE} = \mathsf{CAU[E,H]}$ for some function families E and H.

In our proof, we will take advantage of the fact that the adversary can essentially only make a single forgery attempt. If it fails at this attempt, then the state of the decryption algorithm can be erased and it will henceforth always return $\perp$. Because CAU uses a Carter-Wegman style MAC we have to first use the PRF security of E to hide the values of $H_L(A, C)$ used in encryption queries. To get our desired state-aware results we need to make sure that our PRF reduction does not use much more memory than the original adversary. This creates an issue similar to what we saw in Section 3.3 where it can be difficult to simulate the values returned by Dec. This issue is resolved by adjusting the proof technique used to establish Thm. 3.3.2 where we exploit the fact that ciphertexts look random to assume that $\mathcal{A}$ cannot remember too many ciphertexts.

Theorem 3.3.4. *Let $\mathsf{NE} = \mathsf{CAU[E,H]}$ for some* E *and* H. *Let $\mathcal{A}$ be a nonce-respecting adversary against the* CTXT *security of $\mathsf{NCH[NE]}$ with memory complexity S that makes at most $q \leqslant 2^{\mathsf{CAU.nl}} - 1$ encryption queries, each of which returns a ciphertext of length at most x. Then for any $\delta \in \mathbb{N}$ we can construct an adversary $\mathcal{A}_{\mathsf{p}}$ such that*

$$\mathsf{Adv}^{\mathsf{ch\text{-}ctxt}}_{\mathsf{NCH[NE]}}(\mathcal{A}) \leqslant 2 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathrm{E}}(\mathcal{A}_{\mathsf{p}}) + \mathsf{Adv}^{\mathsf{axu}}_{\mathrm{H}}(\mathcal{X}) + q \cdot x \cdot 2^S/2^\delta \ .$$

Adversary $\mathcal{A}_{\mathsf{p}}$ has running time approximately that of $\mathcal{A}$ and uses about $S + 2\delta$ bits of state. It makes at most $q(x/n + 2) + 1$ non-repeating queries to its oracle.

The proof is given in Section 3.4. As with Thm. 3.2.1, the PRF adversary we give never repeats queries so we can apply the switching lemma to obtain a bound using PRP security of E. Here it is important that the memory of $\mathcal{A}_{\mathsf{p}}$ is not much more than that of $\mathcal{A}$. Assuming $|A| < p$, setting $\delta \approx S + n$, and assuming $S > n$ we can combine all of our theorems so far to obtain a bound of

$$\mathsf{Adv}^{\mathsf{ch\text{-}ae}}_{\mathsf{NCH[NE]}}(\mathcal{A}) \leqslant 4 \cdot \mathsf{Adv}^{\mathsf{prf}}_{\mathrm{E}}(\mathcal{B}) + O\left(\frac{Spq \log(pq) + c(p+q)}{2^n}\right)$$

for a $\mathcal{B}$ with comparable efficiency to $\mathcal{A}$ and assuming H is c-AXU.

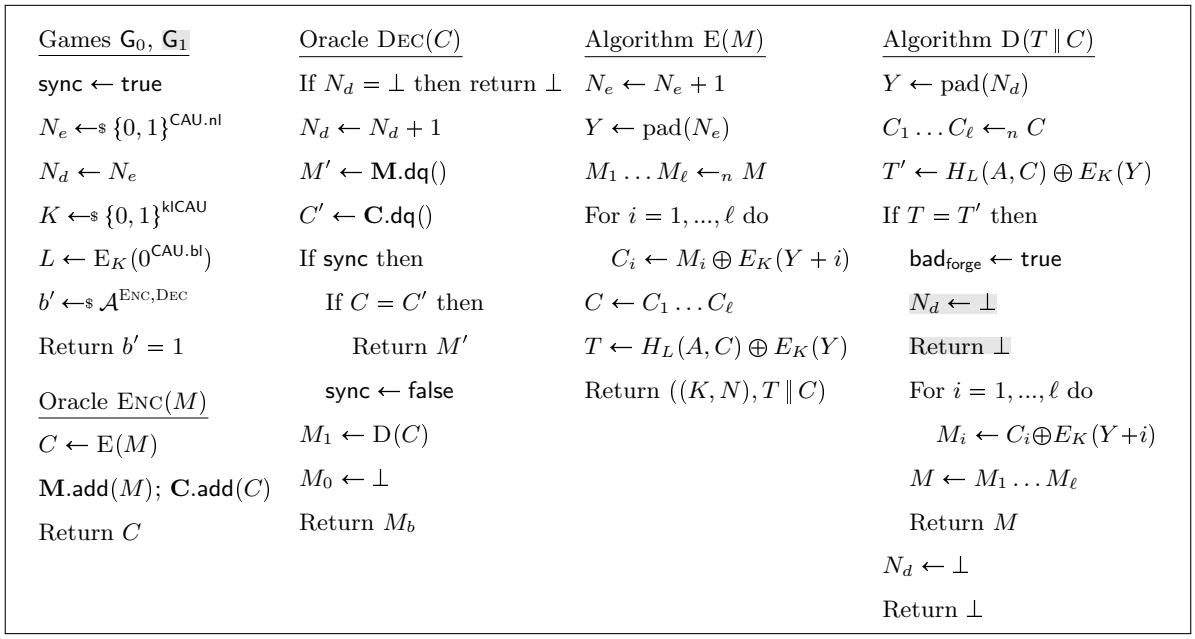


Figure 3.12: Games G_0 and G_1 introducing $\mathsf{bad}_{\mathsf{forge}}$ for proof of Thm. 3.3.4. Highlighted code is only included in G_1.

3.4 Proof of Thm. 3.3.4 (NCH[CAU[E, H]] is CTXT secure)

In this section we provide a proof of Thm. 3.3.4 which gave a concrete bound on the CTXT security of $\mathsf{NCH}[\mathsf{NE}]$ for $\mathsf{NE} = \mathsf{CAU}[\mathrm{E}, \mathrm{H}]$.

Proof. Consider the game G_0 shown in Fig. 3.12. It was obtained by plugging the code of $\mathsf{NCH}[\mathsf{NE}]$ into $\mathsf{G}^{\mathsf{ae}}_{\mathsf{NCH}[\mathsf{NE}],1}$ and making some minor organizational changes (e.g. L is computed once for all time and encryptions/decryptions are performed in the separate algorithms E and D). The game G_1 is the same except that when a forgery would occur (i.e. when $\mathsf{bad}_{\mathsf{forge}}$ is set) the oracle D sets the nonce to $\perp$ and returns $\perp$ anyway. Equivalently, this game always returns $\perp$ once sync is false. Because of this we have that G_1 is identical to $\mathsf{G}^{\mathsf{ae}}_{\mathsf{NCH}[\mathsf{NE}],0}$. Thus we have that

$$\mathsf{Adv}^{\mathsf{ch\text{-}ctxt}}_{\mathsf{NCH}[\mathsf{NE}]}(\mathcal{A}) = \Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_1] \leqslant \Pr[\mathsf{G}_1 \text{ sets } \mathsf{bad}_{\mathsf{forge}}].$$

The rest of the proof is dedicated to bounding this probability.

Transitioning to random. Recall that CAU's authentication is inspired by a Carter-Wegman MAC. Thus we ultimately expect to reduce security to the AXU security of H. However, to do so we first will need to replace the output of E with random so that the key L looks random and the encryption queries before the

forgery attempt have not revealed anything about L. Naturally, this consists of a reduction to the PRF security of E, but this is somewhat difficult in our setting. We'd like the reduction to be memory tight and not repeat PRF queries. This makes it difficult to simulate DEC while $\mathsf{sync} = \mathsf{true}$.

Ultimately, the approach we will take here closely mirrors that used in our proof of Thm. 3.3.2. We will assume that the adversary $\mathcal{A}$ cannot remember too many bits of ciphertext at a time because the ciphertexts should look random to it. If there are ever more than that many bits of ciphertext that $\mathcal{A}$ has received from ENC, but not forwarded to DEC then we can stop remembering additional ciphertext bits in our table and assume the sender and receiver will be out of sync before $\mathcal{A}$ has forwarded on all the ciphertext bits.

To start this analysis consider the game h_0 shown in Fig. 3.13 for which we claim $\mathsf{Pr}[\mathsf{G}_1 \text{ sets } \mathsf{bad}_{\mathsf{forge}}] = \mathsf{Pr}[\mathrm{h}_0]$. Note that h_0 returns $\mathsf{bad}_{\mathsf{forge}}$ rather than something depending on the bit guessed by $\mathcal{A}$. To obtain h_0 from G_1 we have made a few syntactic changes that do not effect the probability $\mathsf{bad}_{\mathsf{forge}}$ is set. First, the use of tables surrounding sync has been re-organized to use two copies of the table as was done in proving Thm. 3.3.2. The flag sync has been removed because we will simply abort whenever it would be changed. Next, all executions of E_K has been forwarded to an oracle RF which simply runs E. Finally, we have added code to DEC to make the game abort and output $\mathsf{bad}_{\mathsf{forge}}$ rather than return $\perp$ and continue execution (after a failed attempt to set $\mathsf{bad}_{\mathsf{forge}}$ in G_1 the nonce N_d would be set to $\perp$, preventing $\mathsf{bad}_{\mathsf{forge}}$ from ever being set again in the future).

The game h_1 is the analog of G_3 in the proof of Thm. 3.3.2. It differs from h_0 by never using $\mathbf{M}_2$ to determine the output of DEC, thereby limiting how much memory is needed to simulate it. Games h_0 and h_1 are identical-until-bad so we have that

$$\mathsf{Pr}[\mathrm{h}_0] - \mathsf{Pr}[\mathrm{h}_1] \leqslant \mathsf{Pr}[\mathrm{h}_1 \text{ sets } \mathsf{bad}].$$

PRF TRANSITION. Having moved to a game that can be simulated with low memory we can now apply PRF security to replace the output of E with random. Consider the game h_2. It differs from h_1 in that the highlighted code has been removed from RF which now implements a (lazily sampled) random function. In Fig. 3.14, we give PRF adversaries $\mathcal{A}'_{\mathsf{p}}$ and $\mathcal{A}''_{\mathsf{p}}$ satisfying

$$\mathsf{Pr}[\mathrm{h}_1] - \mathsf{Pr}[\mathrm{h}_2] \leqslant \mathsf{Adv}^{\mathsf{prf}}_{\mathrm{E}}(\mathcal{A}'_{\mathsf{p}}) \text{ and } \mathsf{Pr}[\mathrm{h}_1 \text{ sets } \mathsf{bad}] - \mathsf{Pr}[\mathrm{h}_2 \text{ sets } \mathsf{bad}] \leqslant \mathsf{Adv}^{\mathsf{prf}}_{\mathrm{E}}(\mathcal{A}''_{\mathsf{p}}).$$

These adversaries simulate the view of $\mathcal{A}$ in h_1 or h_2 by using its EVAL oracle whenever E would be executed, the first outputting 1 whenever $\mathsf{bad}_{\mathsf{forge}}$ would be set and the second outputting 1 whenever bad would be set. They store one extra ciphertext C^* beyond what $\mathbf{C}$ stores to know when bad occurs. We want to be careful that neither adversary repeats queries to EVAL. This is immediate for $\mathcal{A}''$. When $\mathcal{A}'$ checks if the tag of a ciphertext is correct in SIMDEC we have to be careful not to create a repeat query. This would be possible

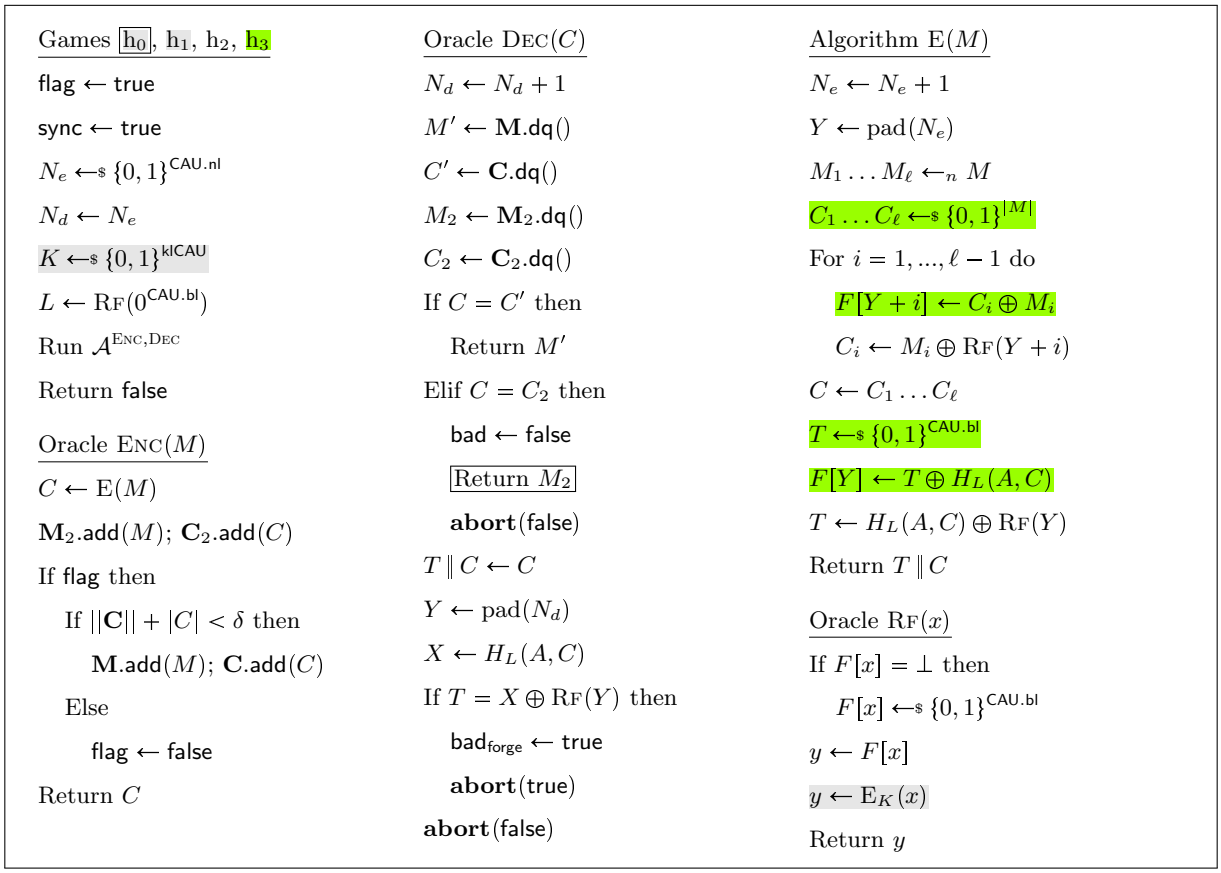


Figure 3.13: Games used to analyze $\mathsf{bad}_{\mathsf{forge}}$ for proof of Thm. 3.3.4. Boxed or highlighted code is only included in correspondingly indicated game.

if $N_e \geqslant N_d$, but in that case we have the corresponding ciphertext stored from which we can re-derive the appropriate output of EVAL (referred to as Z).

REWRITING TRANSITION. Our next transition (to h_3) is merely a conceptual change which does not effect the behavior at the game. This gives

$$\mathsf{Pr}[\mathrm{h}_2] - \mathsf{Pr}[\mathrm{h}_3] = 0 \text{ and } \mathsf{Pr}[\mathrm{h}_2 \text{ sets } \mathsf{bad}] - \mathsf{Pr}[\mathrm{h}_3 \text{ sets } \mathsf{bad}] = 0.$$

The change occurs solely in E where, rather than creating the output ciphertext by xor-ing things with entries of the table F (via RF), we instead pick these outputs at random and use them to define F to match. Note that this results in a uniform distribution for F either way as long as F had not already been defined at that point. This is necessarily the case; E never repeats queries to RF because N_e is always always increasing and if DEC makes a query to RF, then the game will abort immediately afterwards.

Adversaries $\mathcal{A}_{\mathsf{p}}^{\prime\textsc{Eval}}$, $\mathcal{A}_{\mathsf{p}}^{\prime\prime\textsc{Eval}}$

$\mathsf{flag} \leftarrow \mathsf{true}$
$N_e \leftarrow\!\!\$\ \{0,1\}^{\mathsf{CAU.nl}}$
$N_d \leftarrow N_e$
$L \leftarrow \textsc{Eval}(0^{\mathsf{CAU.bl}})$
Run $\mathcal{A}^{\textsc{SimEnc},\textsc{SimDec}}$
Return 0

Algorithm $\mathrm{E}(M)$

$N_e \leftarrow N_e + 1$
$Y \leftarrow \mathrm{pad}(N_e)$
$M_1 \ldots M_\ell \leftarrow_n M$
For $i = 1, \ldots, \ell - 1$ do
  $Z_i \leftarrow \textsc{Eval}(Y + i)$
  $C_i \leftarrow M_i \oplus Z_i$
$C \leftarrow C_1 \ldots C_\ell$
$X \leftarrow H_L(A, C)$
$T \leftarrow X \oplus \textsc{Eval}(Y)$
Return $T \,\|\, C$

Oracle $\textsc{SimEnc}(M)$

$C \leftarrow \mathrm{E}(M)$
If flag then
  If $||\mathbf{C}|| + |C| < \delta$ then
    $\mathbf{M}.\mathsf{add}(M)$
    $\mathbf{C}.\mathsf{add}(C)$
  Else
    $\mathbf{C}.\mathsf{add}(*)$
    $C^* \leftarrow C$
    $\mathsf{flag} \leftarrow \mathsf{false}$
Return C

Oracle $\textsc{SimDec}(C)$

$N_d \leftarrow N_d + 1$
$M' \leftarrow \mathbf{M}.\mathsf{dq}()$; $C' \leftarrow \mathbf{C}.\mathsf{dq}()$
If $C = C'$ then return M'
Elif $C' = *$ and $C = C^*$ then
  $\mathbf{abort}(1)$
  $\mathbf{abort}(0)$
$\mathbf{abort}(0)$
$Y \leftarrow \mathrm{pad}(N_d)$
$T \,\|\, C \leftarrow C$
$X \leftarrow H_L(A, C)$
If $C' = *$ then $C' \leftarrow C^*$
If $C' \neq \perp$ then
  $T' \,\|\, C' \leftarrow C'$
  $X' \leftarrow H_L(A, C')$
  $Z \leftarrow T' \oplus X'$
Else
  $Z \leftarrow \textsc{Eval}(Y)$
If $T = X \oplus Z$ then $\mathbf{abort}(1)$
$\mathbf{abort}(0)$

Figure 3.14: PRF adversaries for proof of Thm. 3.3.4. Highlighted code is only used by the highlighted adversary.

Bounding probability of bad. At this point we are in a game where the adversary is given ciphertexts which are picked uniformly at random. To set bad is must at some point remember more than δ bits of ciphertext that it delayed sending to Dec. In the exact same way we saw in the proof of Thm. 3.3.2 we can create an $(\mathcal{A}_1, \mathcal{A}_2)$ that succeeds in $\mathsf{G}^{\mathsf{it}}_{q\cdot x,\delta}$ with the same probability that bad is set, giving

$$\mathsf{Pr}[\mathrm{h}_3 \text{ sets } \mathsf{bad}] \leqslant \mathsf{Adv}^{\mathsf{it}}_{q\cdot x,\delta}(\mathcal{A}_1, \mathcal{A}_2).$$

This advantage is then bounded by $q \cdot x \cdot 2^S/2^\delta$ using Lemma 3.3.1.

Final bound. Finally we can conclude the proof by analyzing $\mathsf{Pr}[\mathrm{h}_3]$. This reduces naturally to the AXU security of the hash function H. Note that Enc returns random bits, so when $\mathcal{A}$ makes its one attempt to set $\mathsf{bad}_{\mathsf{forge}}$ its view is independent of L. Assuming $N_e \geqslant N_d$ (the $N_e < N_d$ case is strictly harder) at the time of

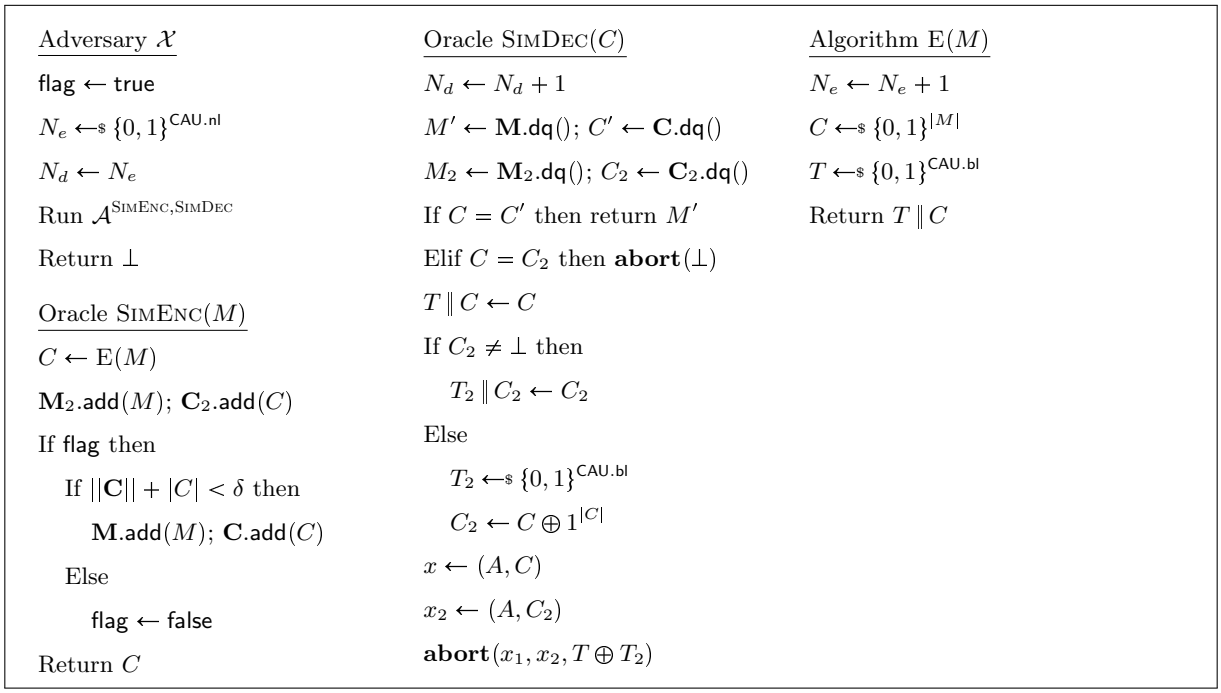


Figure 3.15: Adversary against the AXU security of H.

this attempt, the adversary has seen a random tag T and random C from which $F[\mathsf{pad}(N_d)]$ was defined to equal $T \oplus H_L(A, C)$. Letting $T^* \,\|\, C$ represent its final query, it will win if $T^* = H_L(A, C^*) \oplus F[\mathrm{pad}(N_d)]$. Plugging in for F and re-arranging gives $T^* \oplus T = H_L(A, C^*) \oplus H_L(A, C)$. The adversary $\mathcal{X}$ shown in Fig. 3.15 captures this idea. It simulates the view of $\mathcal{A}$ with random strings until a query is made to Dec that has the potential of setting $\mathsf{bad}_{\mathsf{forge}}$, in which case it provides (In the case that $N_d > N_e$ it just picks T_2 at random and sets C_2 to anything other than C. This gives the correct probability distribution because $\mathcal{A}$'s view would have been independent of $F[\mathsf{pad}(N_d)]$ at that point and it would have needed to guess $H_L(A, C^*) \oplus F[\mathrm{pad}(N_d)]$ to set the flag.)

The above reasoning tells us that

$$\mathsf{Pr}[\mathrm{h}_3] = \mathsf{Adv}^{\mathsf{axu}}_{\mathrm{H}}(\mathcal{X}).$$

Completing the proof is then a simply matter of combining all of the equations we have established. The PRF adversary $\mathcal{A}$ is defined to randomly choose either $\mathcal{A}'_{\mathsf{p}}$ and $\mathcal{A}''_{\mathsf{p}}$ (each with probability 1/2) and then run the adversary it chose, outputting whatever that adversary does. ■

Chapter 4

HIDING IN PLAIN SIGHT: MEMORY-TIGHT PROOFS USING RANDOMNESS PROGRAMMING

In this chapter, we elaborate on a new technique of giving memory-tight reductions that we introduced in Section 2.1.2. This chapter is based on our work in [GGJT22].

The following is an outline of this chapter: section 4.1 introduces notation, our computational model, and basic cryptographic background. Section 4.2 discusses our convention of using $\mathcal{F}$-oracle adversaries. Section 4.3 gives our memory-tight reduction for digital signature schemes when many forgery attempts are allowed. In particular, the generic results are in Section 4.3.2, while the result specific to RSA-PFDH is in Section 4.3.5. Section 4.4 proves the security of Encrypt-then-PRF with a memory-tight reduction to the INDR security of the encryption scheme. Section 4.5 gives our results relating the one- and many-challenge query variants of CCA security. In particular, Section 4.5.1 gives our result for the traditional "left-vs.-right" notion and Section 4.5.2 gives our result for the "indistinguishable from random" variant.

4.1 Preliminaries

Let $\mathbb{N} = \{0, 1, \dots\}$ and $[n] = \{1, \dots, n\}$ for $n \in \mathbb{N}$. If $x \in \{0,1\}^*$ is a string, then $|x|$ denotes its length in bits. If S is a set, then $|S|$ denotes its size. We let $x \,\|\, y \,\|\, \dots$ denote an encoding of the strings $x, y, \dots$ from which the constituent strings can be unambiguously recovered. We identify bitstrings with integers in the standard way.

FUNCTIONS. Let T be a set (called the tweak set) and for each $t \in T$ let D_t and R_t be sets. Then $\mathsf{Fcs}(T, D, R)$ denotes the set of all f such that for each $t \in T$, $f(t, \cdot)$ is a function from D_t to R_t. Similarly, $\mathsf{Inj}(T, D, R)$ denotes the set of all f such that for each $t \in T$, $f(t, \cdot)$ is an injection from D_t to R_t. When D_t or R_t are independent of the choice of t we may omit the subscript.

If $f \in \mathsf{Inj}(T, D, R)$, then its inverse f^{-1} is defined by $f^{-1}(t, f(t, x)) = x$ for all (t, x) and $f^{-1}(t, y) = \perp$ for $y \notin f(t, D_t)$. For such f we let $f^{\pm}$ denote the function defined by $f^{\pm}(+, x) = f(x)$ and $f^{\pm}(-, x) = f^{-1}(x)$. We let $\mathsf{Inj}^{\pm}(T, D, R) = \{f^{\pm} : f \in \mathsf{Inj}(T, D, R)\}$.

4.1.1 Computational Model

PSEUDOCODE. We regularly use pseudocode inspired by the code-based framework of [BR06]. We think of algorithms as randomized RAMs when not specified otherwise. If $\mathcal{A}$ is an algorithm, then $y \leftarrow \mathcal{A}^{\mathrm{O}_1, \cdots}(x_1, \dots; r)$

denotes running $\mathcal{A}$ on inputs $x_1, \ldots$ with coins r and access to the oracles $\mathrm{O}_1, \ldots$ to produce output y. When the coins are implicit we write $\leftarrow\!\!\$$ in place of $\leftarrow$ and omit r.

We let $x \leftarrow\!\!\$ \mathcal{D}$ denote sampling x according to the distribution $\mathcal{D}$. If $\mathcal{D}$ is a set, we overload notation and let $\mathcal{D}$ also denote the uniform distribution over elements of $\mathcal{D}$. The domain of $\mathcal{D}$ is denoted by $[\mathcal{D}]$.

Security notions are defined via games; for an example see Fig. 9.3. The probability that G outputs true is denoted $\mathsf{Pr}[\mathsf{G}]$. In proofs we sometimes define a sequence of "hybrid" games in one figure, using comments of the form "$//\mathsf{H}_{[i,j)}$." A line of code commented thusly is only included in the hybrids H_k for $i \leqslant k < j$. (We are of course referring only to values of $k \in \mathbb{N}$.) By this convention to identify the differences between H_{k-1} and H_k one looks for comments $\mathsf{H}_{[i,k)}$ (code no longer included in the k-th hybrid) and $\mathsf{H}_{[k,j)}$ (code new to the k-th hybrid).

We let $\perp$ be a special symbol used to indicate rejection. If we do not explicitly include $\perp$ in a set, then $\perp$ is not contained in that set. If $\perp$ is an input to a function or algorithm, then we assume its output is $\perp$. We do not distinguish between $\perp$ and tuples $(\perp, \ldots, \perp)$. Algorithms cannot query $\perp$ to their oracles.

Complexity measures. To measure the complexity of algorithms we follow the conventions of measuring their local complexity, not including the complexity of whatever oracles they interact with. Local complexity was preferred by Auerbach et al. [ACFK17] for analyzing memory-limited adversaries so that analysis can be agnostic to minor details of security definitions' implementations. We focus on worst-case runtime $\mathbf{Time}(\mathcal{A})$ and memory complexity $\mathbf{Mem}(\mathcal{A})$ (i.e. how many bits of state it stores for local computation). These exclude the internal complexity of oracles queried by $\mathcal{A}$, but include the time and memory used to write the query and receive the response. If $\mathcal{A}$ expects access to n oracles then we let $\mathbf{Query}(\mathcal{A}) = (q_1, \ldots, q_n)$ where q_i is an upper bound on the number of queries to its i-th oracle. (Here we index from left to right, so for $\mathcal{A}^{\mathrm{O}_1,\ldots,\mathrm{O}_n}$ the i-th oracle is O_i.) If S is a scheme, then $\mathbf{Time}(\mathsf{S})$ and $\mathbf{Mem}(\mathsf{S})$ are the sums of the corresponding complexities over all of its algorithms. If G is a game, then we define $\mathbf{Time}(\mathsf{G})$ and $\mathbf{Mem}(\mathsf{G})$ to *exclude* the complexity of any adversaries embedded in the game.

4.1.2 Cryptographic Background

Ideal models. Some schemes we look at may be proven secure in ideal models (e.g. the random oracle or ideal cipher models). To capture this we can think of a scheme S as specifying a set of functions $\mathsf{S.I}$. At the beginning of a security game a function h will be sampled from this set. The adversary and all algorithms of S are given oracle access to h.

Function families. A family of functions F specifies, for each $K \in \mathsf{F.K}$, an efficiently computable function $\mathsf{F}_K \in \mathsf{F.F}$. We refer to $\mathsf{F.F}$ as the function space of F. Pseudorandom (PR) security of F is captured by the game defined in Fig. 4.1. It measures how F with a random key can be distinguished from a random function

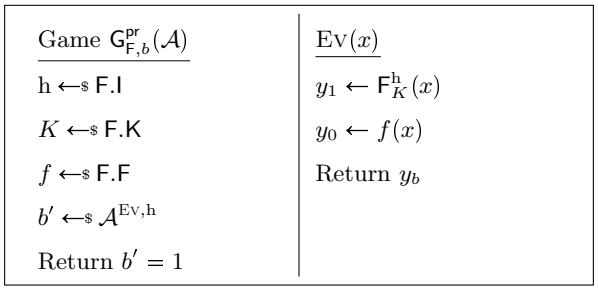


Figure 4.1: Security game capturing the pseudorandomness of function family F.

in $\mathsf{F.F}$ via oracle access. We define $\mathsf{Adv}^{\mathsf{pr}}_{\mathsf{F}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{\mathsf{pr}}_{\mathsf{F},1}(\mathcal{A})] - \mathsf{Pr}[\mathsf{G}^{\mathsf{pr}}_{\mathsf{F},0}(\mathcal{A})]$. The standard notions of (tweakable) pseudorandom functions/injections/permutations or strong injections/permutations are captured by appropriate choices of $\mathsf{F.F}$.

Switching lemma. We restate the Switching Lemma in the notation of this chapter.

Lemma 4.1.1 (Switching Lemma)**.** *Fix T, D, and R. Let $N = \min_{t \in T} |R_t|$. Then for any adversary $\mathcal{A}$ with $q = \mathbf{Query}(\mathcal{A})$ we have that*

$$|\mathsf{Pr}[\mathcal{A}^f \Rightarrow 1] - \mathsf{Pr}[\mathcal{A}^g \Rightarrow 1]| \leqslant 0.5 \cdot q^2/N.$$

The probabilities are measured over the coins of $\mathcal{A}$, the uniform choice of f from $\mathsf{Fcs}(T, D, R)$, and the uniform choice of g from $\mathsf{Inj}(T, D, R)$.

Recent papers [JT19, Din20, SOS20] have given improved versions of the switching lemma for adversaries with bounded memory complexity, as long as it does not repeat oracle queries. In our application of the switching lemma the adversary's memory complexity is too large for these bounds to provide any improvement.

Other primitives. We recall relevant syntax and security definitions for digital signatures, nonce-based encryption, and public key encryption schemes in the sections where we consider them (Sections 4.3, 4.4, and 4.5 respectively).

4.2 Adversaries With Access to Random Functions

This chapter proposes and adopts what we consider to be a better formalism to deal with memory-tight reductions. Namely, all of our reductions will require access to some variety of large random functions which it will query on a small number of inputs (specifically uniformly random functions and invertible random injections). That is, our reduction adversaries can be written in the form shown of the left below, for some

set of functions $\mathcal{F}$ and algorithm $\mathcal{A}_2$. (On the right is a pseudorandom version of $\mathcal{A}$ which we will discuss momentarily.)

Adversary $\mathcal{A}^{\mathrm{O}}(in)$	Adversary $\mathcal{A}_{\mathsf{F}}^{\mathrm{O}}(in)$
$f \xleftarrow{\$} \mathcal{F}$	$K \xleftarrow{\$} \mathsf{F.K}$
$out \xleftarrow{\$} \mathcal{A}_2^{\mathrm{O},f}(in)$	$out \xleftarrow{\$} \mathcal{A}_2^{\mathrm{O},\mathsf{F}_K}(in)$
Return out	Return out

We refer to such an $\mathcal{A}$ as an $\mathcal{F}$-oracle adversary. In this section we will generally discuss such adversaries, rather than separately providing the discussion for such adversaries each time we apply them.

The time and memory complexity of any $\mathcal{F}$-oracle adversary must include the complexity of sampling, storing, and evaluating f. This will be significant if $\mathcal{F}$ is large. However, as we will argue, this additional state and time should be assumed to not significantly increase the advantage of $\mathcal{A}$. As such, we will define the *reduced complexity* of $\mathcal{A}$ by

$$\mathbf{Time}^*(\mathcal{A}) = \mathbf{Time}(\mathcal{A}_2) \text{ and } \mathbf{Mem}^*(\mathcal{A}) = \mathbf{Mem}(\mathcal{A}_2).$$

Later we state theorems in terms of reduced complexity.

Pseudorandom replacement. The most conservative justification of $\mathcal{F}$-oracle adversaries is to bound how much the oracle can help by replacing it with a pseudorandom version. This was the approach taken by Auerbach et al. [ACFK17] when they used pseudorandom functions for purposes such as emulating random oracles and storing the coins required by an adversary with low memory, and has been adopted by follow-up work [WMHT18, Bha20, DGJL21]. If F is a function family with $\mathsf{F.F} = \mathcal{F}$, then the adversary $\mathcal{A}_{\mathsf{F}}$ we gave above does exactly this. It replaces $\mathcal{A}_2$'s oracle access to f with access to F_K for a random K. The following lemma is straightforward.

Lemma 4.2.1. *Let $\mathcal{A}$ be an $\mathcal{F}$-oracle adversary for a game G. Then for any function family F with $\mathsf{F.F} = \mathcal{F}$ we can define a pseudorandomness adversary $\mathcal{A}_{\mathsf{i}}$ such that*

$$\mathsf{Pr}[\mathsf{G}(\mathcal{A})] \leqslant \mathsf{Pr}[\mathsf{G}(\mathcal{A}_{\mathsf{F}})] + \mathsf{Adv}^{\mathsf{pr}}_{\mathsf{F}}(\mathcal{A}_{\mathsf{i}}), \quad \mathbf{Time}(\mathcal{A}_{\mathsf{i}}) = \mathbf{Time}^*(\mathcal{A}) + \mathbf{Time}(\mathsf{G}(\mathcal{A})),$$
$$\mathbf{Query}(\mathcal{A}_{\mathsf{i}}) = q, \text{ and} \qquad \mathbf{Mem}(\mathcal{A}_{\mathsf{i}}) = \mathbf{Mem}^*(\mathcal{A}) + \mathbf{Mem}(\mathsf{G}(\mathcal{A})).$$

Here q is an upper bound on the number of queries $\mathcal{A}_2$ makes to its second oracle.

Note that the complexity of $\mathcal{A}_{\mathsf{F}}$ is given by $\mathbf{Time}(\mathcal{A}_{\mathsf{F}}) = \mathbf{Time}^*(\mathcal{A}) + q \cdot \mathbf{Time}(\mathsf{F})$ and $\mathbf{Mem}(\mathcal{A}_{\mathsf{F}}) = \mathbf{Mem}^*(\mathcal{A}) + \mathbf{Mem}(\mathsf{F})$. Thus the existence of an appropriate pseudorandom F ensures that the memory and time complexity excluded by $\mathbf{Time}^*$ and $\mathbf{Mem}^*$ cannot significantly aid an adversary. In the use of this technique by Auerbach et al. [ACFK17] the reduction $\mathcal{A}_{\mathsf{i}}$ was memory-tight. Note this is not strictly

necessary as long as we are willing to assume the existence of F with sufficient security as a function of attackers' time and query complexities without regard to memory complexity.

We could have combined Lemma 4.2.1 with any of our coming theorems to obtain bounds in terms of **Time** and **Mem**, rather than their reduced version. However we find the use of reduced complexity cleaner as it simplifies our theorems, allowing us to focus on the conceptual core of the proofs without having to repeat the rote step of replacing random objects with pseudorandom ones.

When combining the lemma with a theorem, game G would correspond to the security game played by the reduction adversary. For our theorems, that game will have low time and memory overhead over that of $\mathcal{A}$, so the application of the lemma would be time- and memory-tight. That said, the tightness of this is less important than the tightness of the other components of the theorem we would apply it to. Note that the definition of $\mathcal{A}_{\mathsf{i}}$ is *independent* of the choice of F. Consequently, we can always choose F with a very high security threshold to counteract any looseness in the lemma. In Section 4.6, we summarize the $\mathcal{F}$ used in our theorems and how they could be pseudorandomly instantiated.

Assumed independence. As a second observation why the storage of f may not help $\mathcal{A}$, note that f is completely "independent" of the problem $\mathcal{A}$ is trying to solve (as specified by *in* and the behavior of O). In various settings it seems likely that such independent state does not help. For example, it would be very surprising (or even a breakthrough) to show a better factoring or lattice algorithm given access to a random function f from a natural set. Indeed, cryptanalytic work often makes use of random oracles without significant comment (from which other types of random functions can be constructed).

Information theoretic settings. In some information theoretic settings, the "independence" of f from the problem can be made rigorous. Information theoretic results are typically depending only on the query complexity of the attacker or its memory usage, *ignoring code size*. In such settings, we expect bounds of the form $\mathsf{Adv}(\mathcal{A}) \leqslant \varepsilon(\mathbf{Mem}(\mathcal{A}), \mathbf{Query}(\mathcal{A}))$ for some function ε. Because this bound *does not* depend on the code size of $\mathcal{A}$, if $\mathcal{A}$ is an $\mathcal{F}$-oracle adversary we should be able to prove $\mathsf{Adv}(\mathcal{A}) \leqslant \varepsilon(\mathbf{Mem}^*(\mathcal{A}), \mathbf{Query}(\mathcal{A}))$ by a coin-fixing argument in which we fix the random choice of function ahead of time and embed it in the description of the adversary. This is, for example, the case for the recent time-memory tradeoffs shown for distinguishing between a random function and a random injection without repeating queries [JT19, Din20, SOS20]. A coin-fixing readily shows that these tradeoffs hold when using $\mathbf{Mem}^*(\mathcal{A})$ in place of $\mathbf{Mem}(\mathcal{A}), \mathbf{Query}(\mathcal{A})$.

4.3 Multi-challenge Security of Digital Signature Schemes

In the context of memory-tightness, the security of digital signature schemes has been considered in several works [ACFK17, WMHT18, DGJL21]. The standard security notion for signatures asks the attacker, given

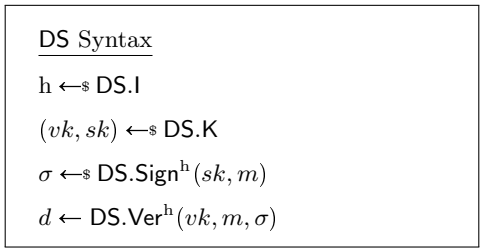


Figure 4.2: Syntax of digital signature scheme.

examples, to come up with a forged signature on a fresh message. A straightforward proof shows (in the standard setting where memory efficiency is not a concern) that the security notion is equivalent whether the attacker is allowed one or many forgery attempts. However, Auerbach et al. [ACFK17] proved an impossibility result showing that a (certain form of black-box) reduction cannot be time, memory, and advantage tight. The difficulty faced by the reduction is in distinguishing between when the adversary has produced a novel forgery and when it is simply repeating a signature that it was given.

In this section we show a few ways that security against many forgery attempts (i.e., multiple challenges) can be proven to follow from security against a single forgery (i.e., a single challenge) in a memory-tight manner. Our first results consider a variant definition of digital signature security we introduce (called UFRMA) in which the adversary has only partial control over the messages being signed. Using our new techniques, we show that single challenge UFCMA security implies multi-challenge UFRMA security in a memory-tight manner (for some practically relevant distributions over messages). We also consider the security of the RSA full domain hash digital signature scheme. Auerbach et al. [ACFK17] gave a memory-, but not advantage-tight proof of the security of the standard version of this scheme in the single challenge setting. By considering a probabilistic variant of the scheme introduced by Coron [Cor02] we are able to provide a memory-, time-, and advantage-tight proof of the many-forgery SUFCMA security of the variant.

4.3.1 Syntax and Security

Digital signature syntax. A digital signature scheme DS specifies a key generation algorithm $\mathsf{DS.K}$, a signing algorithm $\mathsf{DS.Sign}$, and a verification algorithm $\mathsf{DS.Ver}$. The syntaxes of these algorithms are shown in Fig. 4.2. We capture ideals models by providing $\mathsf{DS.Sign}$ and $\mathsf{DS.Ver}$ with oracle access to a function h drawn at random from the set $\mathsf{DS.I}$. When relevant we let $\mathsf{DS.M}$ denote the set of messages it accepts. The verification and signing keys are respectively denoted by vk and sk. The message to be signed is m, the signature produced is σ, and the decision is $d \in \{\mathsf{true}, \mathsf{false}\}$. Correctness requires $\mathsf{DS.Ver}^{\mathrm{h}}(vk, m, \sigma) = \mathsf{true}$

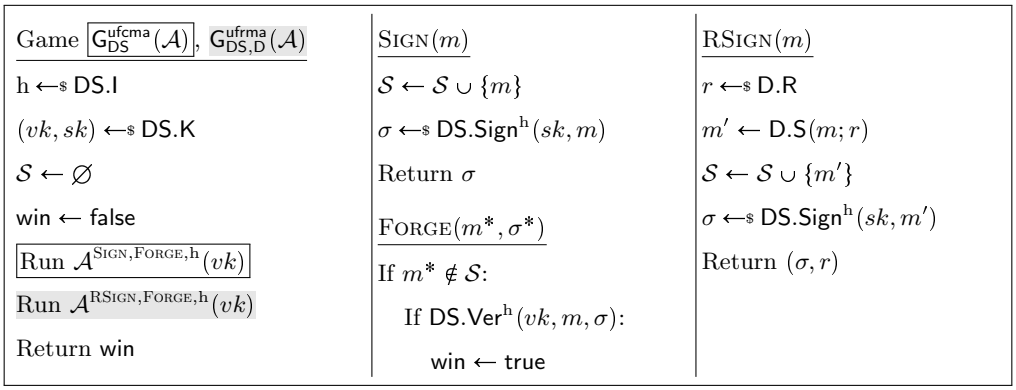


Figure 4.3: Security games capturing the unforgeability of a digital signature scheme.

for all $\mathrm{h} \in \mathsf{DS.I}$, all $(vk, sk) \in [\mathsf{DS.K}]$, all $m \in \mathsf{DS.M}$, and all $\sigma \in [\mathsf{DS.Sign}^{\mathrm{h}}(sk, m)]$.

Message distribution syntax. One of the security notions we consider for digital signature schemes will be parameterized by a message distribution via which the adversary is given incomplete control over the messages which are signed. A message distribution D specifies sampling algorithm $\mathsf{D.S}$ which samples an output message m' based on parameters m given as input (written $m' \leftarrow\!\!\$\ \mathsf{D.S}(m)$). The parameters m must be drawn from a set $\mathsf{D.M}$, which we typically leave implicit. When making the randomness of the sampling algorithm explicit we let $\mathsf{D.R}$ be the set from which its randomness is drawn and write $m' \leftarrow \mathsf{D.S}(m; r)$. If there exists an extraction algorithm $\mathsf{D.X}$ such that $\mathsf{D.X}(\mathsf{D.S}(m; r)) = (m, r)$ for all m, r then we say D is *extractable*. If $\mathsf{D.X}(\mathsf{D.S}(m; r)) = m$ for all m, r then we say D is *weakly extractable*. We assume that $\mathsf{D.X}(m') = \perp$ if $m' \neq \mathsf{D.S}(m; r)$ for all m, r. We define the min-entropy of D as

$$\mathsf{D.H}_{\infty} = -\lg \max_{m} \mathsf{Pr}[r \leftarrow\!\!\$\ \mathsf{D.R} : \mathsf{D.S}(m; r) = m'] \ .$$

Unforgeability security. The unforgeability security notions we consider are defined in Fig. 4.3. The standard notion of UFCMA (unforgeability under chosen message attack) security is captured by $\mathsf{G}^{\mathsf{ufcma}}$ which includes the boxed but not the highlighted code, giving the adversary access to a regular signing oracle Sign. The goal of the adversary is to query Forge with a valid signature σ^* of a message m^* which was not previously included in a signing query (as stored by the set $\mathcal{S}$). We define $\mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{DS}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{\mathsf{ufcma}}_{\mathsf{DS}}(\mathcal{A})]$.

Our new security notion UFRMA (unforgeability under randomized message attack) is captured by the game $\mathsf{G}^{\mathsf{ufrma}}$ which is parameterized by a message distribution D. In this game the adversary is instead given access to the randomized signing oracle RSign where the message to be signed is chosen by D. Note that the coins used by D are returned to the adversary along with the signature. Otherwise this game matches

that of UFCMA security. We define $\mathsf{Adv}^{\mathsf{ufrma}}_{\mathsf{DS},\mathsf{D}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{\mathsf{ufrma}}_{\mathsf{DS},\mathsf{D}}(\mathcal{A})]$.

We will relate the advantage of attacks making only a single forgery attempt and those making many such attempts. When wanting to make the distinction explicit we prefix the abbreviation of a security notion with an 'm' or '1'. Strong UFCMA security, denoted SUFCMA, is captured by modifying $\mathsf{G}^{\mathsf{ufcma}}$ to store the tuple (σ, m) in $\mathcal{S}$ in Sign and checking $(m^*, \sigma^*) \notin \mathcal{S}$ in Forge. We denote this by $\mathsf{G}^{\mathsf{sufcma}}$ and the corresponding advantage by $\mathsf{Adv}^{\mathsf{sufcma}}$. We define SUFRMA, $\mathsf{G}^{\mathsf{sufrma}}$, and $\mathsf{Adv}^{\mathsf{sufrma}}$ analogously. We write xUFRMA when assuming that D is extractable and wUFRMA when assuming it is weakly extractable.

4.3.2 Multi-Challenge Security for Extractable Message Distributions

The first applications we show for our techniques are generic methods of tightly implying security of a digital signature scheme against multiple forgery attempts (i.e., multi-challenge security). Recall that Auerbach et al. [ACFK17] gave a lower bound showing that a black-box reduction proving that single UFCMA security implies many UFCMA cannot be made memory-tight and time-tight. We avoid this in two ways; first by considering mUFRMA, rather than mUFCMA, security and then by considering a particular choice of digital signature scheme.

High-level idea. The primary difficulty of a tight proof that 1UFCMA security implies mUFCMA security is that a successful mUFCMA attacker may have made many Forge queries which verify correctly, one of which is a valid forgery and the rest of which were just forwarded from its Sign oracle. A 1UFCMA reduction must then somehow be able to identify which of the queries is the true forgery so it can forward this to its own Forge oracle.

The technical core of the coming proof for mUFRMA is that our reduction adversary will use the random coins of the message distribution D to signal things to its future self. In particular, when $\mathcal{A}_{\mathsf{r}}$ makes a query $\mathrm{RSign}(m)$, the reduction will choose coins for $\mathsf{D.S}$ via $r \leftarrow f(m, i)$ where i is a counter which is incremented with each query and f is a random tweakable function/injection. The coins then act as a sort of authentication tag for m. On a later $\mathrm{Forge}(m^*, \sigma^*)$ query, if $m^* = \mathsf{D.S}(m; r)$ where $r = f(m, i)$ for some $i \in [q_{\mathrm{Sign}}]$ the reduction can safely assume this message was signed by an earlier RSign query.

When D is fully extractable, we can perform the requisite check for Forge by having f be an injection. We extract m and r from m^* and then compute $i \leftarrow f^{-1}(m, r)$. This is the strategy used in Theorem 4.3.1. If we assume only that D is weakly extractable, we can extract m if D has a sufficient amount of entropy, and then individually check if $\mathsf{D.S}(m; f(m, i))$ holds for each choice of i. This reduction strategy, used in Theorem 4.3.3, obtains the same advantage at the cost of an extra runtime being needed to iterate over the possible choices of i in Forge.

Adversary $\mathcal{A}_{\mathsf{u}}^{\textsc{Sign},\textsc{Forge},\mathrm{h}}(vk)$	$\textsc{SimRSign}(m)$	$\textsc{SimForge}(m^*,\sigma^*)$
$i \leftarrow 0$	$i \leftarrow i+1$	$(m,r) \leftarrow \mathsf{D.X}(m^*)$
$f \leftarrow\!\!\$\ \mathsf{Inj}^{\pm}(\mathsf{DS.M},[q_{\textsc{Sign}}],\mathsf{D.R}))$	$r \leftarrow f(m,i)$	If $f^{-1}(m,r) \notin [q_{\textsc{Sign}}]$:
Run $\mathcal{A}_{\mathsf{r}}^{\textsc{SimRSign},\textsc{SimForge},\mathrm{h}}(vk)$	$m' \leftarrow \mathsf{D.S}(m;r)$	If $\mathsf{DS.Ver}^{\mathrm{h}}(vk,m^*,\sigma^*)$:
	$\sigma \leftarrow \textsc{Sign}(m')$	Query $\textsc{Forge}(m^*,\sigma^*)$
	Return (σ,r)	Halt execution

Figure 4.4: Adversary $\mathcal{A}_{\mathsf{u}}$ used in proof of Theorem 4.3.1.

Extractable Message Distribution. If the message distribution D is extractable, the following theorem captures that 1UFCMA security tightly implies mUFRMA security. The proof makes use of our efficient tagging technique.

Theorem 4.3.1 (1UFCMA $\Rightarrow$ mxUFRMA)**.** *Let* DS *be a digital signature scheme and* D *be an extractable message distribution. Let* $\mathcal{A}_{\mathsf{r}}$ *be an adversary with* $(q_{\textsc{Sign}},q_{\textsc{Forge}},q_{\mathrm{h}}) = \mathbf{Query}(\mathcal{A}_{\mathsf{r}})$ *and assume* $q_{\mathsf{Sign}} \leqslant 0.5|\mathsf{D.R}|$*. Let* $\mathcal{A}_{\mathsf{u}}$ *be the* $\mathsf{Inj}^{\pm}(\mathsf{DS.M},[q_{\textsc{Sign}}],\mathsf{D.R})$*-oracle adversary shown in Fig. 4.4. Then,*

$$\mathsf{Adv}^{\mathsf{ufrma}}_{\mathsf{DS},\mathsf{D}}(\mathcal{A}_{\mathsf{r}}) \leqslant \mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{u}}) + (0.5 \cdot q_{\textsc{Sign}}^2 + 2 \cdot q_{\textsc{Sign}} \cdot q_{\textsc{Forge}})/|\mathsf{D.R}|$$

$$\mathbf{Query}(\mathcal{A}_{\mathsf{u}}) = (q_{\textsc{Sign}}, 1, q_{\mathrm{h}} + q_{\textsc{Forge}} \cdot \mathbf{Query}(\mathsf{DS}))$$

$$\mathbf{Time}^*(\mathcal{A}_{\mathsf{u}}) = \mathbf{Time}(\mathcal{A}_{\mathsf{r}}) + q_{\textsc{Sign}} \cdot \mathbf{Time}(\mathsf{D}) + q_{\textsc{Forge}}(\mathbf{Time}(\mathsf{D}) + \mathbf{Time}(\mathsf{DS}))$$

$$\mathbf{Mem}^*(\mathcal{A}_{\mathsf{u}}) = \mathbf{Mem}(\mathcal{A}_{\mathsf{r}}) + \mathbf{Mem}(\mathsf{D}) + \mathbf{Mem}(\mathsf{DS}) + \lg(q_{\textsc{Sign}}).$$

This is time-tight because $\mathbf{Time}(\mathcal{A}_{\mathsf{r}}) \in \Omega(q_{\mathsf{Sign}} + q_{\textsc{Forge}})$ must hold and $\mathbf{Time}(\mathsf{D})$ and $\mathbf{Time}(\mathsf{DS})$ will be small. This is memory-tight because $\mathbf{Mem}(\mathsf{D})$, $\mathbf{Mem}(\mathsf{DS})$, and $\lg(q_{\textsc{Sign}})$ will be small.

The main idea of $\mathcal{A}_{\mathsf{u}}$ is using the output of an invertible random injection f on the message and a counter as coins instead of sampling them uniformly at random when answering RSign queries. Since D is fully extractable, during a Forge query on m^*, we can extract $(m,r) \leftarrow \mathsf{D.X}(m^*)$ and use the fact that f is invertible to compute $f^{-1}(m,r)$ and check if the index is in $[q_{\textsc{Sign}}]$. This is used to avoid remembering $\mathcal{S}$. If $m^* \in \mathcal{S}$, and $(m,r) \leftarrow \mathsf{D.X}(m^*)$, then there exists $j \in [q_{\textsc{Sign}}]$ such that $r = f(m,j)$ – so the check passes. We can argue that if $m^* \notin \mathcal{S}$, our check is unlikely to pass. We give the formal proof of this theorem in Section 4.3.3. It applies the switching lemma to argue the use of f cannot be distinguished from honestly sampling r with advantage better than $0.5 \cdot q_{\textsc{Sign}}^2/|\mathsf{D.R}|$ and shows that the probability of falsely making the check pass is bounded by $2q_{\textsc{Sign}}q_{\textsc{Forge}}/|\mathsf{D.R}|$.

Games H_h for $0 \leqslant h \leqslant 4$

$\mathrm{h} \leftarrow\!\!\$\ \mathsf{DS.I}$

$(vk, sk) \leftarrow\!\!\$\ \mathsf{DS.K}$; $\mathcal{S} \leftarrow \varnothing$

$\mathsf{win} \leftarrow \mathsf{false}$

$i \leftarrow 0$; $I[\cdot] \leftarrow \varnothing$

$f \leftarrow\!\!\$\ \mathsf{Fcs}(\mathsf{DS.M}, [q_{\text{SIGN}}], \mathsf{D.R})$ //$\mathsf{H}_{[1,2)}$

$f \leftarrow\!\!\$\ \mathsf{Inj}^{\pm}(\mathsf{DS.M}, [q_{\text{SIGN}}], \mathsf{D.R})$ //$\mathsf{H}_{[2,\infty)}$

Run $\mathcal{A}_{\mathsf{r}}^{\text{RSIGN},\text{FORGE},\mathrm{h}}(vk)$

Return win

RSIGN(m)

$r \leftarrow\!\!\$\ \mathsf{D.R}$ //$\mathsf{H}_{[0,1)}$

$i \leftarrow i + 1$ //$\mathsf{H}_{[1,\infty)}$

$r \leftarrow f(m, i)$ //$\mathsf{H}_{[1,\infty)}$

$I[m] \leftarrow I[m] \cup \{i\}$ //$\mathsf{H}_{[3,4)}$

$m' \leftarrow \mathsf{D.S}(m; r)$

$\mathcal{S} \leftarrow \mathcal{S} \cup \{m'\}$ //$\mathsf{H}_{[0,3)}$

$\sigma \leftarrow\!\!\$\ \mathsf{DS.Sign}^{\mathrm{h}}(sk, m')$

Return (σ, r)

FORGE(m^*, σ^*)

$(m, r) \leftarrow \mathsf{D.X}(m^*)$

If $m^* \notin \mathcal{S}$: //$\mathsf{H}_{[0,3)}$

If $f^{-1}(m, r) \notin I[m]$: //$\mathsf{H}_{[3,4)}$

If $f^{-1}(m, r) \notin [q_{\text{SIGN}}]$: //$\mathsf{H}_{[4,\infty)}$

  If $\mathsf{DS.Ver}^{\mathrm{h}}(vk, m^*, \sigma^*)$:

    win $\leftarrow$ true

Figure 4.5: Hybrid games used in proof of Theorem 4.3.1.

We would not be able to use the technique in this proof to prove mxSUFRMA from 1SUFCMA in a memory-tight way. In particular, since the coins r of the message distribution are chosen before σ is known, our trick of using r to signal freshness of a forgery query does not work for a message-signature pair.

4.3.3 Proof of Theorem 4.3.1 (1UFCMA⇒mUFRMA)

Proof. We consider a sequence of hybrids H_0 through H_4 defined in Fig. 4.5. When examining these hybrids recall our conventions regarding "//$\mathsf{H}_{[i,j)}$" comments described in Sec. 4.1.1. Of these hybrids we will make the following claims, which establish the upper bound on the advantage of $\mathcal{A}_{\mathsf{r}}$ claimed in the proof.

1. $\mathsf{Pr}[\mathsf{G}^{\mathsf{ufrma}}_{\mathsf{DS,D}}(\mathcal{A}_{\mathsf{r}})] = \mathsf{Pr}[\mathsf{H}_0] = \mathsf{Pr}[\mathsf{H}_1]$
2. $\mathsf{Pr}[\mathsf{H}_1] \leqslant \mathsf{Pr}[\mathsf{H}_2] + 0.5 \cdot q^2_{\text{SIGN}}/|\mathsf{D.R}|$
3. $\mathsf{Pr}[\mathsf{H}_2] = \mathsf{Pr}[\mathsf{H}_3]$
4. $\mathsf{Pr}[\mathsf{H}_3] \leqslant \mathsf{Pr}[\mathsf{H}_4] + 2q_{\text{SIGN}}q_{\text{FORGE}}/|\mathsf{D.R}|$
5. $\mathsf{Pr}[\mathsf{H}_4] = \mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{u}})$

TRANSITION H_0 TO H_1. The hybrid H_0 is simply a copy of the game $\mathsf{G}^{\mathsf{ufrma}}$. (We also added code to initialize variables i and $I[\cdot]$ that will be used in later hybrids.) Hence $\mathsf{Pr}[\mathsf{G}^{\mathsf{ufrma}}(\mathcal{A}_{\mathsf{r}})] = \mathsf{Pr}[\mathsf{H}_0]$. In hybrid H_1, we

replace the random sampling of r for D in RSign with the output of a random function f applied to m, using a counter i to provide domain separation between different queries. This method of choosing r is equivalent, so $\mathsf{Pr}[\mathsf{H}_0] = \mathsf{Pr}[\mathsf{H}_1]$.

Transition H_1 to H_2. In hybrid H_2 we replace the random function with a random injection. This modifies the behavior of the game only in that values of r are guaranteed not to repeat across different signing queries that used the same message. There are at most q_{Sign} invocations of f, so the switching lemma (Lemma 11.4.7) tells us that $\mathsf{Pr}[\mathsf{H}_1] \leqslant \mathsf{Pr}[\mathsf{H}_2] + 0.5 \cdot q_{\text{Sign}}^2/|\mathsf{D.R}|$.

Transition H_2 to H_3. In hybrid H_3, we replace the check whether $m^* \notin \mathcal{S}$ in oracle Forge with a check if $f^{-1}(m, r) \notin I[m]$ where $(m, r) = \mathsf{D.X}(m^*)$. Here $I[\cdot]$ is a new table introduced into the game. In RSign, code was added which uses $I[m]$ to store each of the counter values for which $\mathcal{A}_{\mathsf{r}}$ made a signing query for m. Hence $f^{-1}(m, r)$ will be in $I[m]$ iff m^* is in $\mathcal{S}$ and so $\mathsf{Pr}[\mathsf{H}_2] = \mathsf{Pr}[\mathsf{H}_3]$.

Transition H_3 to H_4. In the final transition to hybrid H_4 we replace the Forge check $f^{-1}(m, r) \notin I[m]$ with $f^{-1}(m, r) \notin [q_{\text{Sign}}]$. This *does* change behavior if $\mathcal{A}_{\mathsf{r}}$ ever makes a successful forgery query for $m^* = \mathsf{D.S}(m; f(m, i))$ without its i-th signing query having used the message m. This would require guessing $f(m, i)$ for some $i \in [q_{\text{Sign}}] \backslash I[m]$. We can bound the probability of this ever occurring by a union bound over the Forge queries made by $\mathcal{A}_{\mathsf{r}}$. Consider the set $f(m, [q_{\text{Sign}}] \backslash I[m]) = \{f(m, i) : i \in [q_{\text{Sign}}] \backslash I[m]\}$. It has size at most q_{Sign}. Because f is a random injection it is uniform subset of the set $\mathsf{D.R} \backslash f(m, I[m])$ (which has size at least $|\mathsf{D.R}| - q_{\text{Sign}}$). Hence the probability of any particular query triggering this different behavior is at most $q_{\text{Sign}}/(|\mathsf{D.R}| - q_{\text{Sign}}) \leqslant 2q_{\text{Sign}}/|\mathsf{D.R}|$. Applying the union bound gives us $\mathsf{Pr}[\mathsf{H}_3] \leqslant \mathsf{Pr}[\mathsf{H}_4] + 2q_{\text{Sign}} \cdot q_{\text{Forge}}/|\mathsf{D.R}|$.

Reduction to UFCMA. We complete the proof using adversary $\mathcal{A}_{\mathsf{u}}$ from Fig. 4.4 which simulates hybrid H_4 and succeeds whenever $\mathcal{A}_{\mathsf{r}}$ would. The adversary $\mathcal{A}_{\mathsf{u}}$ samples f at random from $\mathsf{Inj}(\mathsf{DS.M}, [q_{\text{Sign}}], \mathsf{D.R})$. When run on input vk, it runs $\mathcal{A}_{\mathsf{r}}$ on the same input. It gives $\mathcal{A}_{\mathsf{r}}$ direct access to h. To simulate a query $\text{RSign}(m)$, it computes $m' \leftarrow \mathsf{D.S}(m; f(m, i))$, increments i, and queries $\text{Sign}(m')$, returning the result to $\mathcal{A}_{\mathsf{r}}$. On a query $\text{Forge}(m^*, \sigma^*)$, it computes $(m, r) \leftarrow \mathsf{D.X}(m^*)$. If $f^{-1}(r) \notin [q_{\text{Sign}}]$ and $\mathsf{DS.Ver}(vk, m^*, \sigma^*) = \mathsf{true}$ then it queries its own oracle with (m^*, σ^*) and halts. Otherwise it ignores the query.

If adversary $\mathcal{A}_{\mathsf{u}}$ ever makes a Forge query, it will succeed. It ensured that (m^*, σ^*) is verified correctly and $f^{-1}(r) \notin [q_{\text{Sign}}]$ ensures that it is has not previously made a Sign query for m^*. If $\mathcal{A}_{\mathsf{r}}$ would have succeeded in hybrid H_4, its winning query will cause $\mathcal{A}_{\mathsf{u}}$ to make a Forge query. Hence, we have $\mathsf{Pr}[\mathsf{H}_4] = \mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{u}})$.

The claimed complexity of $\mathcal{A}_{\mathsf{u}}$ is straightforward. Clearly it makes q_{Sign} queries to its signing oracle and 1 query to its forgery oracle. It forwards all of $\mathcal{A}_{\mathsf{r}}$'s queries to h and additionally may make queries when running $\mathsf{DS.Ver}$ in SimForge giving $q_{\mathsf{h}} + q_{\text{Forge}} \cdot \mathbf{Query}(\mathsf{DS.Ver})$ queries total. The time complexity of $\mathcal{A}_{\mathsf{u}}$ includes that of $\mathcal{A}_{\mathsf{r}}$, plus the time to execute $\mathsf{D.S}$ for each signing query, and the time to run $\mathsf{D.X}$ and $\mathsf{DS.Ver}$

for each forgery attempt. The memory complexity of $\mathcal{A}_{\mathsf{u}}$ includes that of $\mathcal{A}_{\mathsf{r}}$, plus the amount of memory required to run the algorithms $\mathsf{D.S}$, $\mathsf{D.X}$, and $\mathsf{DS.Ver}$ and to store the counter i. ■

4.3.4 Applications and Weakly Extractable Variant

We discuss some applications of Theorem 4.3.1. This includes scenarios where extractable message distributions are used and proving security of digital signature schemes when their messages are padded with randomness. Additionally, we give a variant of the theorem when the underlying message distribution is only weakly extractable. The resulting reduction is memory- but not time-tight.

EXAMPLE EXTRACTABLE DISTRIBUTIONS. The simplest extractable distribution does not accept parameters as input and simply outputs its randomness as the message. Security with respect to this is the standard notion of security against random message attacks which was originally introduced by Even, Goldreich, and Micali [EGM96].

Extractable distributions arise naturally when the messages being signed include random values. For example, protocols often include random nonces in messages that are signed. In TLS 1.3, for example, when the server is responding to the Client Hello Message it signs a transcript of the conversation up until that point which includes a 256-bit nonce just chosen by the server. We could think of the security for this setting being captured by an extractable distribution $\mathsf{D}_{\mathrm{tls}}$ that takes as input message parameter m that specifies all of the transcript other than the nonce and sets the nonce to its randomness $r \in \{0,1\}^{256}$.

PADDING SCHEMES WITH RANDOMNESS. Using Theorem 4.3.1, we can see that augmenting any digital signature scheme by appending auxiliary randomness will give us a memory-tight reduction from the mUFCMA security of the augmented scheme to the 1UFCMA security of the original scheme.

Let DS be a digital signature scheme and R be a set. We define $\mathsf{RDS}[\mathsf{DS},\mathsf{R}]$ as follows. The signing algorithm $\mathsf{RDS}[\mathsf{DS},\mathsf{R}].\mathsf{Sign}(sk,m)$ does the following: "$r \leftarrow\!\!\$\ \mathsf{R}$; Return $\mathsf{DS.Sign}(sk, m \,\|\, r) \,\|\, r$". The verification algorithm $\mathsf{RDS}[\mathsf{DS},\mathsf{R}].\mathsf{Ver}(vk,m,\sigma')$ does: "$\sigma \,\|\, r \leftarrow \sigma'$; Return $\mathsf{DS.Ver}(vk, m \,\|\, r, \sigma)$." We also define a related message distribution $\mathsf{RD}[\mathsf{R}]$ by $\mathsf{RD}[\mathsf{R}].\mathsf{R} = \mathsf{R}$ and $\mathsf{RD}[\mathsf{R}].\mathsf{S}(m;r) = m \,\|\, r$. Clearly it is extractable.

The following reduces the mUFCMA security of RDS to the mUFRMA security of DS. Theorem 4.3.1 can in turn be used to reduce this to the 1UFCMA security of DS. It also relates the mSUFCMA security of RDS to the mSUFRMA security of DS. We note this because if DS has unique signatures, then its mSUFRMA and mUFRMA security are identical and hence UFCMA security of DS implies mSUFCMA security of RDS in a memory-tight way.

Theorem 4.3.2. *Let* DS *be a digital signature scheme and* R *be a set. Then for any* $\mathcal{A}_{\mathsf{u}}$ *we can construct* $\mathcal{A}_{\mathsf{r}}$ *such that* $\mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{RDS}[\mathsf{DS},\mathsf{R}]}(\mathcal{A}_{\mathsf{u}}) = \mathsf{Adv}^{\mathsf{ufrma}}_{\mathsf{DS},\mathsf{RD}[\mathsf{R}]}(\mathcal{A}_{\mathsf{r}})$. *It additionally holds that* $\mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{RDS}[\mathsf{DS},\mathsf{R}]}(\mathcal{A}_{\mathsf{u}}) = \mathsf{Adv}^{\mathsf{sufrma}}_{\mathsf{DS},\mathsf{RD}[\mathsf{R}]}(\mathcal{A}_{\mathsf{r}})$. *Adversary* $\mathcal{A}_{\mathsf{r}}$ *has essentially the same complexity as* $\mathcal{A}_{\mathsf{u}}$.

Proof. [Sketch] The proof of this is straightforward. If $\mathcal{A}_\mathsf{u}$ queries $\textsc{Sign}(m)$, then $\mathcal{A}_\mathsf{r}$ queries $\textsc{Sign}(m)$ and receives (σ, r) and returns$\sigma \,\|\, r$ to $\mathcal{A}_\mathsf{u}$. If $\mathcal{A}_\mathsf{u}$ queries $\textsc{Forge}(m^*, \sigma^* \,\|\, r^*)$, then $\mathcal{A}_\mathsf{r}$ queries $\textsc{Forge}(m^* \,\|\, r^*, \sigma^*)$. Note that $\mathcal{A}_\mathsf{r}$ wins whenever $\mathcal{A}_\mathsf{u}$ would. ■

In independent and concurrent work, Diemert, Gellert, Jager, and Lyu [DGJL21] also considered RDS, proving that if DS can be proven SUFCMA1 secure (in this notion the game records its responses to signature queries and repeats them if the adversary repeats a query) with a restricted class of "canonical" memory-tight reductions, then there is a memory-tight reduction for the mSUFCMA security of RDS. This complements our results as they use a more restrictive assumption to prove mSUFCMA while we use a generic assumption to prove mUFCMA.

Weakly extractable message distribution. If D is only weakly extractable (but still has high entropy), then we can prove a variant of Theorem 4.3.1 with a less efficient reduction. (In particular, the running time of the reduction has an additional term of $q_{\textsc{Forge}} \cdot q_{\textsc{Sign}} \cdot \mathbf{Time}(\mathsf{D.S})$.) This difference arises because rather than extracting r and computing $j \leftarrow f^{-1}(m, r)$ in $\textsc{Forge}$ we instead need to iterate over the possible values of $f(m, j)$ to check for consistency. Thus the proof is an instance of our inefficient tagging technique.

Theorem 4.3.3 (1UFCMA ⇒ mwUFRMA)**.** *Let* DS *be a digital signature scheme and* D *be a weakly extractable message distribution. Let* $\mathcal{A}_\mathsf{r}$ *be an adversary with* $(q_{\textsc{Sign}}, q_{\textsc{Forge}}, q_\mathrm{h}) = \mathbf{Query}(\mathcal{A}_\mathsf{r})$*. Define the* $\mathsf{Fcs}(\mathsf{DS.M}, [q_{\textsc{Sign}}], \mathsf{D.R})$*-oracle adversary* $\mathcal{A}_\mathsf{u}$ *as shown in Fig. 4.6. Then.*

$$\mathsf{Adv}^{\mathsf{ufrma}}_{\mathsf{DS},\mathsf{D}}(\mathcal{A}_\mathsf{r}) \leqslant \mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{DS}}(\mathcal{A}_\mathsf{u}) + q_{\textsc{Sign}} \cdot q_{\textsc{Forge}} \cdot 2^{-\mathsf{D.H}_\infty}$$

$$\mathbf{Query}(\mathcal{A}_\mathsf{u}) = (q_{\textsc{Sign}}, 1, q_\mathrm{h} + q_{\textsc{Forge}} \cdot \mathbf{Query}(\mathsf{DS}))$$

$$\mathbf{Time}^*(\mathcal{A}_\mathsf{u}) = \mathbf{Time}(\mathcal{A}_\mathsf{r}) + q_{\textsc{Sign}} \cdot \mathbf{Time}(\mathsf{D}) + q_{\textsc{Forge}}(q_{\textsc{Sign}}\mathbf{Time}(\mathsf{D}) + \mathbf{Time}(\mathsf{DS}))$$

$$\mathbf{Mem}^*(\mathcal{A}_\mathsf{u}) = \mathbf{Mem}(\mathcal{A}_\mathsf{r}) + \mathbf{Mem}(\mathsf{D}) + \mathbf{Mem}(\mathsf{DS}) + \lg(q_{\textsc{Sign}}).$$

This running time is not time-tight because $\mathbf{Time}(\mathcal{A}_\mathsf{r}) \in O(q_{\mathsf{Sign}} + q_{\textsc{Forge}})$ may hold, while $\mathbf{Time}^*(\mathcal{A}_\mathsf{u}) \in \Omega(q_{\textsc{Forge}} \cdot q_{\textsc{Sign}})$. This is memory-tight because we expect $\mathbf{Mem}(\mathsf{D}) + \mathbf{Mem}(\mathsf{DS}) + \lg(q_{\textsc{Sign}})$ to be small. Recall that, as discussed in the introduction, such non-time-tight reductions may be useful when the best attack for the underlying problem with low memory requires significantly more running time than the best attack with high memory.

This theorem is useful when the messages being signed include values derived from randomness which are hard to invert to recover the underlying randomness. Examples of this include the signing of public keys, as is done for certificates or some key exchange protocols, and the signing of ciphertexts, as is done for signcryption. We could capture such settings with appropriate choices of D.

The main idea behind adversary $\mathcal{A}_\mathsf{u}$ is very similar to the idea behind the adversary in Theorem 4.3.1. Because we now only assume weak extractability, we extract m and then iterate over each choice of j to

$\underline{\text{Adversary } \mathcal{A}_{\mathsf{u}}^{\text{Sign},\text{Forge},\mathrm{h}}(vk)}$	$\underline{\text{SimRSign}(m)}$	$\underline{\text{SimForge}(m^*, \sigma^*)}$
$i \leftarrow 0$	$i \leftarrow i + 1$	$(m, r) \leftarrow \mathsf{D.X}(m^*)$
$f \leftarrow\!\!\$\ \mathsf{Fcs}(\mathsf{DS.M}, [q_{\text{Sign}}], \mathsf{D.R})$	$r \leftarrow f(m, i)$	If $\forall j \in [q_{\text{Sign}}], \mathsf{D.S}(m; f(m, j)) \neq m^*$:
Run $\mathcal{A}_{\mathsf{r}}^{\text{SimRSign},\text{SimForge},\mathrm{h}}(vk)$	$m' \leftarrow \mathsf{D.S}(m; r)$	If $\mathsf{DS.Ver}^{\mathrm{h}}(vk, m^*, \sigma^*)$:
	$\sigma \leftarrow \text{Sign}(m')$	Query $\text{Forge}(m^*, \sigma^*)$
	Return (σ, r)	Halt execution

Figure 4.6: Reduction adversary used in proof of Theorem 4.3.3.

check whether $\mathsf{D.S}(m; f(m, j)) = m^*$. Moreover, since we no longer need to invert f here, it suffices for f to be a random function. We give formal proof of Theorem 4.3.3.

Proof. We consider a sequence of hybrids H_0 through H_4 defined in Fig. 4.7. Of these hybrids we will make the following claims, which establish the upper bound on the advantage of $\mathcal{A}_{\mathsf{r}}$ claimed in the proof.

1. $\mathsf{Pr}[\mathsf{G}^{\mathsf{ufrma}}_{\mathsf{DS},\mathsf{D}}(\mathcal{A}_{\mathsf{r}})] = \mathsf{Pr}[\mathsf{H}_0] = \mathsf{Pr}[\mathsf{H}_1]$
2. $\mathsf{Pr}[\mathsf{H}_1] = \mathsf{Pr}[\mathsf{H}_2]$
3. $\mathsf{Pr}[\mathsf{H}_2] = \mathsf{Pr}[\mathsf{H}_3]$
4. $\mathsf{Pr}[\mathsf{H}_3] \leqslant \mathsf{Pr}[\mathsf{H}_4] + q_{\text{Sign}} \cdot q_{\text{Forge}} \cdot 2^{-\mathsf{D.H}_\infty}$
5. $\mathsf{Pr}[\mathsf{H}_4] = \mathsf{Adv}^{\mathsf{ufcma}}_{\mathsf{DS}}(\mathcal{A}_{\mathsf{u}})$

Transition to H_0. The hybrid H_0 is simply a copy of the game $\mathsf{G}^{\mathsf{ufrma}}$. (We also added code to initialize variables i and $I[\cdot]$ that will be used in later hybrids.) Hence $\mathsf{Pr}[\mathsf{G}^{\mathsf{ufrma}}(\mathcal{A}_{\mathsf{r}})] = \mathsf{Pr}[\mathsf{H}_0]$.

Transition H_0 to H_1. In hybrid H_1, we replace the random sampling of r for D in RSign with the output of a random function f applied to m, using a counter i to provide domain separation between different queries. This method of choosing r is equivalent, so $\mathsf{Pr}[\mathsf{H}_0] = \mathsf{Pr}[\mathsf{H}_1]$.

Transition H_1 to H_2. Hybrid H_2 is identical to hybrid H_1. So $\mathsf{Pr}[\mathsf{H}_1] = \mathsf{Pr}[\mathsf{H}_2]$. We include this redundant hybrid to maintain consistency of the hybrid numbers with the proof of Theorem 4.3.3.

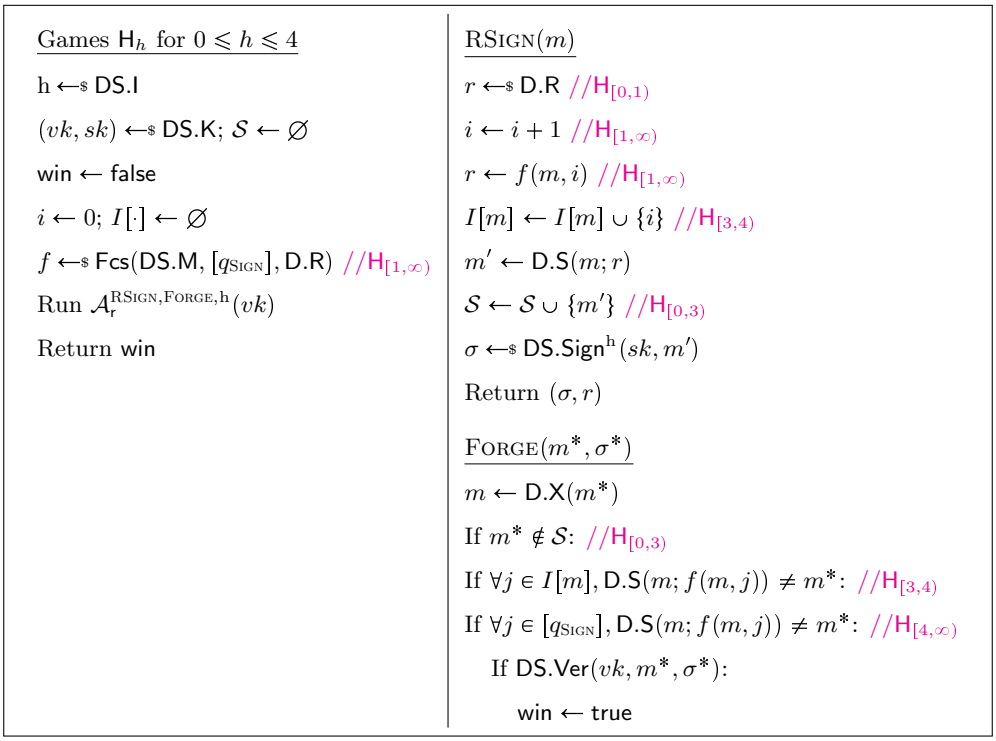


Figure 4.7: Hybrid games used in proof of Theorem 4.3.3.

TRANSITION H_2 TO H_3. In hybrid H_3, we replace the check whether $m^* \notin \mathcal{S}$ in oracle FORGE with a check if $\forall j \in I[m], \mathsf{D.S}(m; f(m, j)) \neq m^*$. Here $I[\cdot]$ is a new table introduced into the game. In RSIGN, code was added which uses $I[m]$ to store each of the counter values for which $\mathcal{A}_{\mathsf{r}}$ made a signing query for m. It is easy to see that if for all $j \in I[m]$, $\mathsf{D.S}(m; f(m, j)) \neq m^*$, then $m^* \notin \mathcal{S}$. Also if there exists $j \in I[m]$ such that $\mathsf{D.S}(m; f(m, j)) = m^*$, the j-th signing query was on m and hence $m^* \in \mathcal{S}$. Therefore the new check is equivalent to the replaced one and so $\mathsf{Pr}[\mathsf{H}_2] = \mathsf{Pr}[\mathsf{H}_3]$.

TRANSITION H_3 TO H_4. In the final transition to hybrid H_4 we replace the FORGE check $\forall j \in I[m]$, $\mathsf{D.S}(m; f(m, j)) \neq m^*$ with $\forall j \in [q_{\mathrm{SIGN}}], \mathsf{D.S}(m; f(m, j)) \neq m^*$. This *does* change behavior if $\mathcal{A}_{\mathsf{r}}$ ever makes a successful forgery attempt for $m^* = \mathsf{D.S}(m; f(m, j))$ without its j-th signing query having used the message m. Note that the view of the adversary would be independent of $f(m, j)$ in this case and hence $f(m, j)$ can be thought of as a value chosen uniformly at random from $\mathsf{D.R}$. Thus for every FORGE query and $j \in [q_{\mathrm{SIGN}}]$ the probability that $m^* = \mathsf{D.S}(m; f(m, j))$ is at most $2^{-\mathsf{D.H}_\infty}$. By an union bound over all values of j it follows that for every FORGE this happens with probability at most $q_{\mathrm{FORGE}} \cdot 2^{-\mathsf{D.H}_\infty}$. By a union bound over all FORGE queries we get that $\mathsf{Pr}[\mathsf{H}_3] \leqslant \mathsf{Pr}[\mathsf{H}_4] + q_{\mathrm{SIGN}} \cdot q_{\mathrm{FORGE}} \cdot 2^{-\mathsf{D.H}_\infty}$.

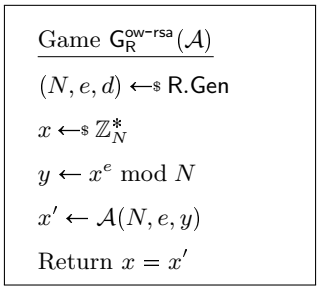


Figure 4.8: RSA one-wayness security game.

Reduction to UFCMA. We complete the proof by designing an adversary $\mathcal{A}_{\mathsf{u}}$ (see Fig. 4.6)). It is easy to see that adversary $\mathcal{A}_{\mathsf{u}}$ simulates hybrid H_4 to $\mathcal{A}_{\mathsf{r}}$ and succeeds whenever $\mathcal{A}_{\mathsf{r}}$ would. It follows that, $\mathsf{Pr}[\mathsf{H}_4] = \mathsf{Adv}_{\mathsf{DS}}^{\mathsf{ufcma}}(\mathcal{A}_{\mathsf{u}})$.

The claimed complexity of $\mathcal{A}_{\mathsf{u}}$ is straightforward. Clearly it makes q_{Sign} queries to its signing oracle and 1 query to its forgery oracle. It forwards all of $\mathcal{A}_{\mathsf{r}}$'s queries to h and additionally may make queries when running $\mathsf{DS.Ver}$ in SimForge giving $q_{\mathsf{h}} + q_{\text{Forge}} \cdot \mathbf{Query}(\mathsf{DS.Ver})$ queries total. The time complexity of $\mathcal{A}_{\mathsf{u}}$ includes that of $\mathcal{A}_{\mathsf{r}}$, plus the time to execute $\mathsf{D.S}$ for each signing query, and the time to run $\mathsf{D.X}$ at most q_{Sign} times and $\mathsf{DS.Ver}$ once for each forgery attempt. The memory complexity of $\mathcal{A}_{\mathsf{u}}$ includes that of $\mathcal{A}_{\mathsf{r}}$, plus the memory required to run the algorithms $\mathsf{D.S}$, $\mathsf{D.X}$, and $\mathsf{DS.Ver}$ and to store the counter i. ■

4.3.5 mSUFCMA Security of RSA-PFDH

We saw that augmenting any digital signature scheme by including extra randomness gives us a memory-tight reduction for the mUFCMA security of the augmented scheme from the 1UFCMA security of the original scheme in Theorem 4.3.2. Now we will consider a particular signature scheme, RSA-based Probabilistic Full-Domain Hash (RSA-PFDH) which was originally introduced by Coron [Cor02]. This is a digital signature scheme obtained by including extra randomness in the standard RSA-based Full Domain Hash (RSA-FDH) scheme. Theorem 4.3.2 gives us a memory-tight reduction from the mUFCMA security of RSA-PFDH to the 1UFCMA security of RSA-FDH, but the UFCMA security reduction of the latter to hardness of inverting the RSA permutation is not tight in terms of advantage. In this section, we use the efficient tagging trick to give a direct reduction from the mSUFCMA security of RSA-PFDH to the hardness of RSA which is tight in terms of both memory and advantage.

RSA trapdoor permutation. The RSA function defines a trapdoor permutation that is plausibly one-way. It is based on the observation that given modulus $N \in \mathbb{N}$ and an integer $e \geqslant 2$ relatively prime to $\phi(N)$

$\underline{\mathsf{RSA.K}}$	$\underline{\mathsf{RSA.Sign}^{\mathrm{h}}(sk, m)}$	$\underline{\mathsf{RSA.Ver}^{\mathrm{h}}(vk, m^*, \sigma^*)}$
$(N, e, d) \leftarrow\!\!\$\ \mathsf{R.Gen}$	$(N, d) \leftarrow sk$	$(N, e) \leftarrow vk$
$sk \leftarrow (N, d)$	$r \leftarrow\!\!\$\ \{0,1\}^{\mathsf{rl}}$	$z \,\|\|\, r \leftarrow \sigma^*$
$vk \leftarrow (N, e)$	$w \leftarrow \mathrm{h}(N, m \,\|\|\, r)$	$w \leftarrow z^e \bmod N$
Return (sk, vk)	$z \leftarrow w^d \bmod N$	Return $(w = \mathrm{h}(N, m^* \,\|\|\, r))$
	Return $z \,\|\|\, r$	

Figure 4.9: Digital signature scheme $\mathsf{RSA} = \mathsf{RSA\text{-}PFDH}[\mathsf{R}, \mathsf{rl}]$.

(where ϕ is Euler's totient function), exponentiation to the e-th power modulo N is a permutation on $\mathbb{Z}_N^*$. An RSA generator R specifies a generation algorithm $\mathsf{R.Gen}$ such that $\mathsf{R.Gen}$ returns (N, e, d) where N is an integer such that e is co-prime to N and $d = e^{-1} \bmod \phi(N)$. We assume N is always of a fixed bit-length $\mathsf{R.k}$.

Typically $N = pq$ for distinct $(\mathsf{R.k}/2)$-bit primes p and q. The one-wayness of an RSA generator R is defined by the game $\mathsf{G}_{\mathsf{R}}^{\mathsf{ow\text{-}rsa}}$ in Fig. 4.8. The game runs $\mathsf{R.Gen}$ to obtain (N, e, d), and then samples x uniformly at random from $\mathbb{Z}_N^*$. It computes $y = x^e \bmod N$ and runs the adversary on input (N, e, y). The adversary wins if it returns $x' = x$. The advantage of an adversary $\mathcal{A}$ against RSA generator R is $\mathsf{Adv}_{\mathsf{R}}^{\mathsf{ow\text{-}rsa}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}_{\mathsf{R}}^{\mathsf{ow\text{-}rsa}}(\mathcal{A})]$.

We let $\mathbf{Time}(\mathsf{R})$ denote the time required by $\mathsf{R.Gen}$ plus an upper bound on the time to compute multiplication or exponentiation by e in $\mathbb{Z}_N^*$ for any (N, e, d) output but R. We define $\mathbf{Mem}(\mathsf{R})$ analogously.

Full domain hash. Full Domain Hash (FDH) [BR93] is a digital signature scheme where the message m is first hashed using a hash function h onto the domain of a one-way trapdoor permutation f. Then the signature is $f^{-1}(\mathrm{h}(m))$. When instantiated with the RSA trapdoor permutation, it is known as RSA-FDH.

Assuming h is a random oracle, it can be proven [BR93, Cor00] that for every adversary $\mathcal{A}$ that makes q_{SIGN} queries to its signing oracle and achieves advantage ε against the UFCMA security of RSA-FDH, we can construct an adversary $\mathcal{B}$ that breaks RSA with advantage $\varepsilon' \approx q_{\mathrm{SIGN}} \cdot \varepsilon$. Auerbach et al. [ACFK17] showed how to make this reduction memory-tight.

In order to overcome the advantage loss factor of q_{SIGN}, Coron [Cor02] introduced Probabilistic FDH (PFDH) where a random salt r is hashed with the message m and the signature is $f^{-1}(\mathrm{h}(m \,\|\, r)) \,\|\, r$. Using Theorem 4.3.2, and the result of Auerbach et al. we can give a memory-tight reduction for the mUFCMA security of RSA-PFDH, but the reduction is not-tight in terms of advantage. Via a more direct proof we will obtain a reduction that is tight in all metrics.

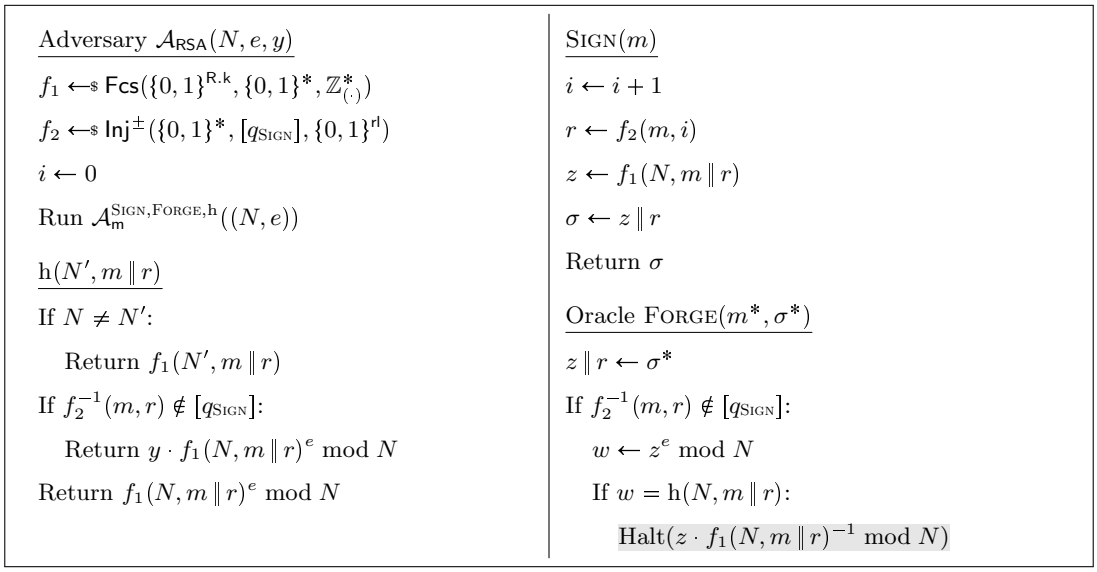


Figure 4.10: Adversary for Theorem 4.3.4. Highlighting shows where it halts with the specified output.

We let $\mathsf{RSA\text{-}PFDH}[\mathsf{R}, \mathsf{rl}]$ denote the instantiation of RSA-PFDH with a given RSA generator R and randomness length $\mathsf{rl} \in \mathbb{N}$. For compactness we typically define $\mathsf{RSA} = \mathsf{RSA\text{-}PFDH}[\mathsf{R}, \mathsf{rl}]$. Its algorithms are given in Fig. 4.9. They expect access to a random oracle $\mathrm{h} \in \mathsf{RSA\text{-}PFDH}[\mathsf{R}, \mathsf{rl}].\mathsf{I} = \mathsf{Fcs}(\{0,1\}^{\mathsf{R.k}}, \{0,1\}^*, \mathbb{Z}^*_{(\cdot)})$. Typical analysis of FDH constructions (e.g., [BR93, Cor00]) uses a single non-tweakable hash function with range Z_N^*. Thus h depends on N which is determined by the verification key used. This dependence does not make sense with our notation conventions for treating ideal models, so we instead represent h as a hash function tweaked by N. Here the $\mathbb{Z}^*_{(\cdot)}$ indicates that for $h \in \mathsf{RSA\text{-}PFDH}[\mathsf{R}, \mathsf{rl}].\mathsf{I}$, the function $h(N, \cdot)$ has a range of $\mathbb{Z}_N^*$. Thus $\mathsf{RSA\text{-}PFDH}[\mathsf{R}, \mathsf{rl}]$ is the same as the scheme considered by Coron, merely with different notational conventions.

SECURITY RESULT. The following result gives a memory-tight and advantage-tight reduction for the mSUFCMA security of RSA-PFDH.

Theorem 4.3.4 (mSUFCMA security of RSA-PFDH)**.** *Let* R *be an RSA generator. Let* $\mathsf{rl} \in \mathbb{N}$ *and assume* $\mathsf{rl} < \mathsf{R.k}$*. Let* $\mathsf{RSA} = \mathsf{RSA\text{-}PFDH}[\mathsf{R}, \mathsf{rl}]$*. Let* $\mathcal{A}_{\mathsf{m}}$ *be an adversary with* $(q_{\text{SIGN}}, q_{\text{FORGE}}, q_{\mathrm{h}}) = \mathbf{Query}(\mathcal{A}_{\mathsf{m}})$ *and*

assume $q_{\text{SIGN}} \leqslant 2^{\mathsf{rl}-1}$. *Let* $\mathcal{A}_{\mathsf{RSA}}$ *be the adversary defined in Fig. 4.10. Then,*

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{sufcma}}_{\mathsf{RSA}}(\mathcal{A}_{\mathsf{m}}) &\leqslant \mathsf{Adv}^{\mathsf{ow\text{-}rsa}}_{R}(\mathcal{A}_{\mathsf{RSA}}) + (0.5 \cdot q^2_{\text{SIGN}} + 2 \cdot q_{\text{SIGN}} \cdot q_{\text{FORGE}})/2^{\mathsf{rl}} \\
\mathbf{Time}^{*}(\mathcal{A}_{\mathsf{RSA}}) &= O(\mathbf{Time}(\mathcal{A}_{\mathsf{m}})) + O((q_{\mathrm{h}} + q_{\text{FORGE}})\mathbf{Time}(\mathsf{R})) \\
\mathbf{Mem}^{*}(\mathcal{A}_{\mathsf{RSA}}) &= O(\mathbf{Mem}(\mathcal{A}_{\mathsf{m}})) + O(\mathbf{Mem}(\mathsf{R})) + \lg(q_{\text{SIGN}}).
\end{aligned}$$

Adversary $\mathcal{A}_{\mathsf{RSA}}$ *is an* $\mathcal{F}$*-oracle adversary for*

$$\mathcal{F} = \mathsf{Fcs}(\{0,1\}^{\mathsf{R.k}}, \{0,1\}^*, \mathbb{Z}^*_{(\cdot)}) \times \mathsf{Inj}(\{0,1\}^*, [q_{\text{SIGN}}], \{0,1\}^{\mathsf{rl}}).$$

The advantage of the adversary $\mathcal{A}_{\mathsf{RSA}}$ is nearly the same as the advantage of $\mathcal{A}_{\mathsf{m}}$ if rl is chosen so that $0.5 \cdot q^2_{\text{SIGN}} + 2 \cdot q_{\text{SIGN}} \cdot q_{\text{FORGE}} \ll 2^{\mathsf{rl}}$ will hold. The reduced complexity of $\mathcal{A}_{\mathsf{RSA}}$ is nearly the same as the running time and memory of $\mathcal{A}_{\mathsf{m}}$. Therefore, the reduction is tight with respect to advantage, time, and memory.

Its main idea of $\mathcal{A}_{\mathsf{RSA}}$ is based around its simulation of h using the random function f_1. We have $\mathrm{h}(N', m \,\|\, r)$ simply return $f_1(N', m \,\|\, r)$ whenever $N' \neq N$ because these queries are not relevant for security. We want to return $f_1(N, m \,\|\, r)^e \bmod N$ for any $m \,\|\, r$ that will be signed in SIGN. This allows us to return $f_1(N, m \,\|\, r) \,\|\, r$ as the signature because then $\mathrm{h}(N, m \,\|\, r)^d \mod N = f_1(N, m \,\|\, r)$. Finally we want to embed our challenge y into all other h queries by returning $y \cdot f_1(N, m \,\|\, r)^e \bmod N$. Then given a forgery with respect to such a value (that is, given z such that $z^e \bmod N = \mathrm{h}(N, m \,\|\, r)$) we can see that $y^d \bmod N = z \cdot f_1(N, m \,\|\, r)^{-1} \bmod N$, solving the RSA game.

If memory was not a concern we could sample r at random in SIGN and remember each $m \,\|\, r$ used to respond appropriately to them in h and FORGE. To make things memory-tight we instead chose r as $f_2(m, i)$ for a random function f_2 and counter i. Then whenever we see $m \,\|\, r$ such that $f_2^{-1}(m, r) \in [q_{\text{SIGN}}]$ we assume this must have been used in a signing query and respond appropriately. In concurrent work, Diemert, Gellert, Jager, and Lyu [DGJL21] also give a time, memory, and advantage tight reduction for RSA-PFDH via a different proof.

We give the formal proof of Theorem 4.3.4.

Proof. We consider a sequence of hybrids H_0 through H_3 and L_0 through L_3 defined in Fig. 4.11 and 4.12. Of these hybrids we will make the following claims, which establish the upper bound on the advantage of $\mathcal{A}_{\mathsf{m}}$ claimed in the proof.

1. $\mathsf{Pr}[\mathsf{G}^{\mathsf{sufcma}}_{\mathsf{RSA}}(\mathcal{A}_{\mathsf{m}})] = \mathsf{Pr}[\mathsf{H}_0] = \mathsf{Pr}[\mathsf{H}_1] = \mathsf{Pr}[\mathsf{H}_2]$
2. $\mathsf{Pr}[\mathsf{H}_2] \leqslant \mathsf{Pr}[\mathsf{H}_3] + 0.5 \cdot q^2_{\text{SIGN}}/2^{\mathsf{rl}}$
3. $\mathsf{Pr}[\mathsf{H}_3] = \mathsf{Pr}[\mathsf{L}_0]$
4. $\mathsf{Pr}[\mathsf{L}_0] \leqslant \mathsf{Pr}[\mathsf{L}_1] + 2q_{\text{SIGN}} \cdot q_{\text{FORGE}}/2^{\mathsf{rl}}$
5. $\mathsf{Pr}[\mathsf{L}_1] = \mathsf{Pr}[\mathsf{L}_2]$
6. $\mathsf{Pr}[\mathsf{L}_2] = \mathsf{Adv}^{\mathsf{ow\text{-}rsa}}_{\mathsf{R}}(\mathcal{A}_{\mathsf{RSA}})$

TRANSITION TO H_0. Hybrid H_0 was obtained by embedding the code of RSA into $\mathsf{G}^{\mathsf{sufcma}}$ and rewriting h as an oracle that simply evaluates the random function f_1 chosen in the same way that h was sampled. Hence, $\mathsf{Pr}[\mathsf{G}^{\mathsf{sufcma}}_{\mathsf{RSA}}(\mathcal{A}_{\mathsf{m}})] = \mathsf{Pr}[\mathsf{H}_0]$.

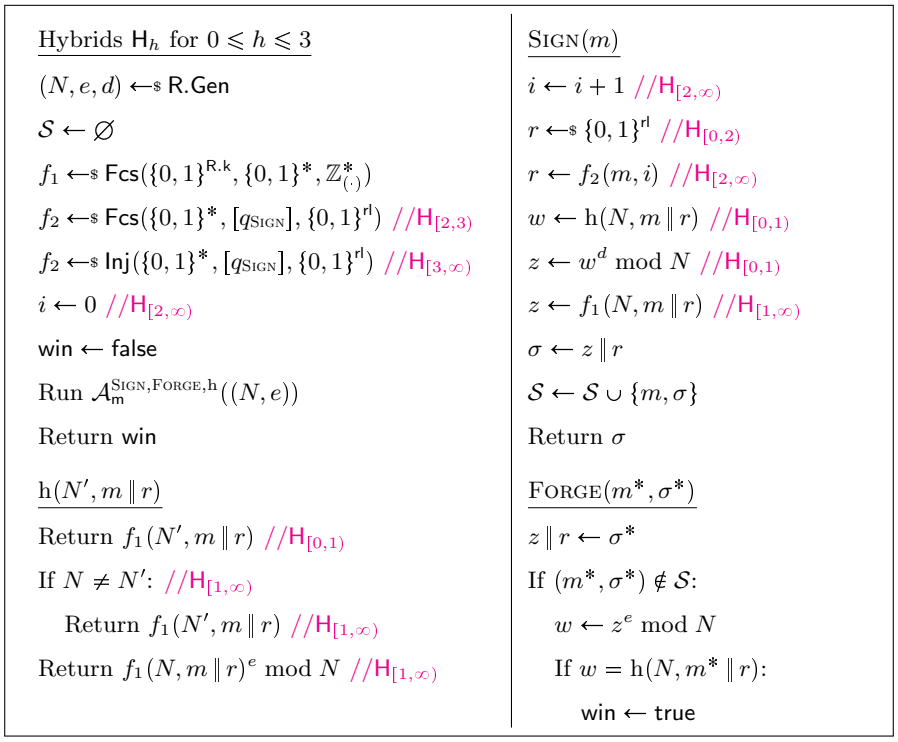


Figure 4.11: Hybrid games H_0 through H_3 used in proof of Theorem 4.3.4.

TRANSITION FROM H_0 TO H_1. In hybrid H_1, we assign $f_1(N, m \,\|\, r)^e \bmod N$ to $\text{h}(N, m \,\|\, r)$ instead of $f_1(N, m \,\|\, r)$. Because exponentiation by e us a permutation this is still a uniformly random element. Then for a SIGN query, we directly assign $f_1(N, m \,\|\, r)$ to z since $(f_1(N, m \,\|\, r)^e)^d \bmod N = f_1(N, m \,\|\, r)$. This does not change the distribution of the hash values or the signatures, so $\mathsf{Pr}[\mathsf{H}_0] = \mathsf{Pr}[\mathsf{H}_1]$.

TRANSITION FROM H_1 TO H_2. In hybrid H_2, we replace the random sampling of r during signing with the output of a random function f_2, using a counter i to provide domain separation between different SIGN queries. This method of choosing r is equivalent, so $\mathsf{Pr}[\mathsf{H}_1] = \mathsf{Pr}[\mathsf{H}_2]$.

TRANSITION FROM H_2 TO H_3. In hybrid H_3 we replace the random function f_2 with a random injection (tweaked by the message m). This modifies the behavior of the game only in that values of r are guaranteed not to repeat across different signing queries that used the same message. There are at most q_{SIGN} invocations of f_2, so the switching lemma (Lemma 11.4.7) tells us that $\mathsf{Pr}[\mathsf{H}_2] \leqslant \mathsf{Pr}[\mathsf{H}_3] + 0.5 \cdot q^2_{\text{SIGN}}/2^{\mathsf{rl}}$.

TRANSITION FROM H_3 TO L_0. Now consider L_0 shown in Fig. 4.12. We've used grey highlighting to indicate places where code was changed from H_3 to L_0. In hybrid L_0's oracle FORGE, we replace the check whether

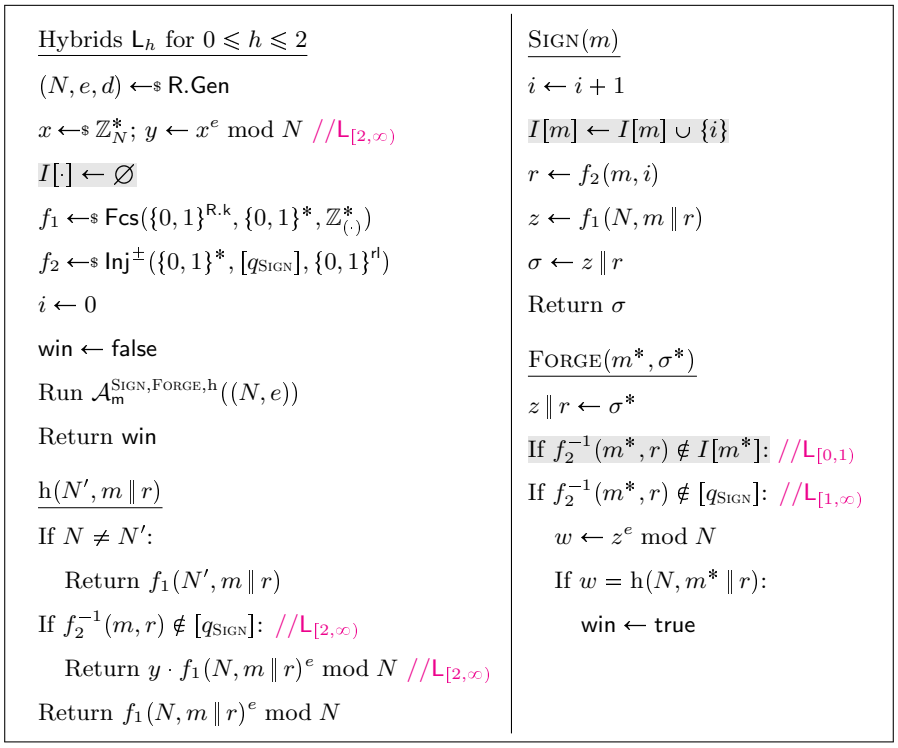


Figure 4.12: Hybrid games L_0 through L_2 used in proof of Theorem 4.3.4. Grey highlighting indicates where L_0 differs from H_3.

$(m^*, \sigma^*) \notin \mathcal{S}$ with a check whether $f^{-1}(m, r) \notin I[m]$. Here $I[\cdot]$ is a new table; code was added to SIGN which in $I[m]$ stores each of the counter values for which $\mathcal{A}_{\mathsf{m}}$ made a signing query for m. Note that $f^{-1}(m, r)$ is in $I[m]$ iff the $f^{-1}(m, r)$-th signing query was for m and returned $z' \,\|\, r$ for some z'. Hence, if $(m^*, \sigma^*) \in \mathcal{S}$, then $f^{-1}(m, r) \in I[m]$. If $(m^*, \sigma^*) \notin \mathcal{S}$ either $f^{-1}(m, r) \notin I[m]$ or $f^{-1}(m, r) \in I[m]$, but in the latter case it must be that $\sigma^* = z \,\|\, r$ for some z not equal to the z' returned by the signing query and so $z^e \neq \text{h}(N, m^* \,\|\, r)$. Hence these games are equivalent, giving $\mathsf{Pr}[\mathsf{H}_3] = \mathsf{Pr}[\mathsf{L}_0]$.

TRANSITION L_0 TO L_1. Next, in hybrid L_1 we replace the FORGE check $f^{-1}(m, r) \notin I[m]$ with $f^{-1}(m, r) \notin [q_{\text{SIGN}}]$. Detecting this change requires guessing $f_2(m^*, i)$ for some $i \in [q_{\text{SIGN}}] \backslash I[m]$. We can bound the probability of this ever occurring in L_0 by a union bound over the FORGE queries made by the adversary. Consider the set $f(m^*, [q_{\text{SIGN}}] \backslash I[m^*]) = \{f(m^*, i) : i \in [q_{\text{SIGN}}] \backslash I[m^*]\}$. It has size at most q_{SIGN}. Because f_2 is a random injection, this is uniform subset of the set $\{0,1\}^{\mathsf{rl}} \backslash f_2(m^*, I[m^*])$ (which has size at least $2^{\mathsf{rl}} - q_{\text{SIGN}}$). Hence the probability of any particular query being the first to trigger this different behavior is at most $q_{\text{SIGN}}/(2^{\mathsf{rl}} - q_{\text{SIGN}}) \leqslant 2q_{\text{SIGN}}/2^{\mathsf{rl}}$. Applying the union bound gives us $\mathsf{Pr}[\mathsf{L}_0] \leqslant \mathsf{Pr}[\mathsf{L}_1] + 2q_{\text{SIGN}} \cdot q_{\text{FORGE}}/2^{\mathsf{rl}}$.

TRANSITION FROM L_1 TO L_2. In hybrid L_2 we now begin the game by sampling x at random from $\mathbb{Z}_N^*$ and setting $y \leftarrow x^e \bmod N$. Our goal is to "embed" y into the responses to random oracle queries so that a successful forgery allows x to be recovered. In particular, we change the output of h whenever $f^{-1}(m,r) \notin [q_{\text{SIGN}}]$, now returning $y \cdot f_2(m \,\|\, r)^e \bmod N$. Note that multiplying the fixed element $y \in \mathbb{Z}_N^*$ by a uniformly random element still gives a uniformly random element. Because we only perform this modification for $f^{-1}(m,r) \notin [q_{\text{SIGN}}]$, it will not cause any inconsistency with SIGN where $f^{-1}(m,r) \in [q_{\text{SIGN}}]$ always holds. Hence the view of the adversary is unchanged and so $\mathsf{Pr}[\mathsf{L}_1] = \mathsf{Pr}[\mathsf{L}_2]$.

REDUCTION TO RSA. We complete the proof by arguing that $\mathcal{A}_{\mathsf{RSA}}$ perfectly simulates hybrid L_2 and succeeds whenever $\mathcal{A}_{\mathsf{m}}$ would. It is formally defined in Fig. 4.10. Examining its code, we can see that the code is basically identical to that of L_2, except it is given (N, e, y) as input rather than generating them locally. The grey highlighted code in FORGE shows where $\mathcal{A}_{\mathsf{RSA}}$ will halt early whenever $\mathcal{A}_{\mathsf{m}}$ would win. The check immediately before this ensures that $z^e \bmod N = y \cdot f_3(m \,\|\, r)^e \bmod N$. Hence $y = (z \cdot (f_3(m \,\|\, r))^{-1})^e \bmod N$, meaning that $z \cdot (f_3(m \,\|\, r))^{-1}$ is indeed the correct response. Hence $\mathsf{Pr}[\mathsf{L}_2] = \mathsf{Adv}_R^{\mathsf{ow\text{-}rsa}}(\mathcal{A}_{\mathsf{RSA}})$.

The complexity of $\mathcal{A}_{\mathsf{RSA}}$ follows from its description. The counter i is the additional $\lg(q_{\text{SIGN}})$ storage. The $\mathbf{Time}(\mathsf{R})$ and $\mathbf{Mem}(\mathsf{R})$ terms come from performing operations in $\mathbb{Z}_N^*$ for h and FORGE queries.

■

4.4 AE Security of Encrypt-then-PRF

In this section we look at the AE security (introduced in chapter 3) of a particular scheme which we refer to as Encrypt-then-PRF. Given a nonce-based encryption scheme NE that only has INDR security, one generic way to construct a new encryption scheme NE' which also achieves ciphertext integrity is to use a PRF and let the ciphertext of NE' be the concatenation of the ciphertext of NE and a tag which is the evaluation of the PRF on the ciphertext and the nonce.

We show that in the context of Encrypt-then-PRF, for two of the notions of AE security introduced in [GJT20], we can give a memory-tight reduction to the INDR security of the underlying encryption scheme and a non-memory-tight reduction to the security of the PRF. This shows that we can bypass the generic impossibility result of [GJT20] if we consider specific constructions of nonce-based authenticated encryption schemes. In more detail, the impossibility result of [GJT20] rules out lifting the INDR security of a scheme to full AE security in a memory tight way, when additionally assuming ciphertext integrity *for a generic scheme*. Here, we show that for the *specific* case of Encrypt-then-PRF schemes, lifting the INDR security of the encryption scheme to full AE security of Encrypt-then-PRF is possible in a memory-tight way, assuming security of the PRF.

ENCRYPT-THEN-PRF. In this section we consider the Encrypt-then-PRF construction of a nonce-based encryption scheme, due to Rogaway [Rog04]. Namprempre et al. [NRS14] gave a more extensive exploration of the many ways to construct an AEAD encryption scheme via generic composition. Given nonce-based encryption scheme NE and function family F, we define $\mathsf{EtP}[\mathsf{NE},\mathsf{F}]$ by the following algorithms. We refer to the t component of the ciphertext returned by $\mathsf{EtP}[\mathsf{NE},\mathsf{F}].\mathsf{E}$ as the "tag" below. When including associated data, it would be input to F.

$\underline{\mathsf{EtP}[\mathsf{NE},\mathsf{F}].\mathsf{K}}$	$\underline{\mathsf{EtP}[\mathsf{NE},\mathsf{F}].\mathsf{E}(K,n,m)}$	$\underline{\mathsf{EtP}[\mathsf{NE},\mathsf{F}].\mathsf{D}(K,n,c)}$
$K \xleftarrow{\$} \mathsf{NE.K}$	$(K,K') \leftarrow K$	$(K,K') \leftarrow K;\ (c',t) \leftarrow c$
$K' \xleftarrow{\$} \mathsf{F.K}$	$c' \leftarrow \mathsf{NE.E}(K,n,m)$	If $t = \mathsf{F}_{K'}(n,c')$:
Return (K,K')	$t \leftarrow \mathsf{F}_{K'}(n,c')$	Return $\mathsf{NE.D}(K,n,m)$
	Return (c',t)	Return $\perp$

Our security result will analyze the authenticated security of EtP assuming NE has ciphertexts indistinguishable from random ciphertexts and F is pseudorandom. We re-use the security notions defined in Chapter 3.

We argue that $w = \mathtt{m}$ is the "correct" definition because chosen ciphertext security is intended to capture the power of an adversary that can observe the behavior of a decrypting party. Both the $w = \perp$ and $w = \diamond$ definitions restrict what the adversary learns about this behavior when honestly generated ciphertexts are forwarded, which does not seem to model anything about real use of encryption. The $w = \mathtt{m}$ definition avoids this unnatural restriction.

We provide some technical context for this philosophical argument. In Section 4.7 we give memory-tight proofs for the security of encryption schemes constructed with the KEM/DEM paradigm with $w = \mathtt{m}$ and noting this does not seem possible for the other choices of w. In this section and Sec. 4.5 we prove the AE/CCA-w security of encryption schemes for differing choices of w. We view this as a general exploration of what results are possible with memory-tight proofs. A proof which works for some w, but not others helps build some understanding of how these notions related.

4.4.1 Security Result

Now we give a proof of the AE-$\diamond$ security of $\mathsf{EtP}[\mathsf{NE},\mathsf{F}]$. In particular we provide a memory-tight reduction to the INDR security of NE and a non-memory-tight reduction to the security of F. Such a result is useful if a time-memory tradeoff is known for NE and F is sufficiently secure even against high-memory attackers.

Theorem 4.4.1 (Security of EtP)**.** *Let* NE *be a nonce-based encryption scheme and* F *be a family of function with* $\mathsf{F.F} = \mathsf{Fcs}(\mathsf{NE.N}, \mathsf{NE.C}, \{0,1\}^\tau)$ *for* $\tau \in \mathbb{N}$*. Let* $\mathcal{A}_a$ *be an* AE-$\diamond$ *adversary with* $(q_{\text{ENC}}, q_{\text{DEC}}) = \mathbf{Query}(\mathcal{A}_a)$*.*

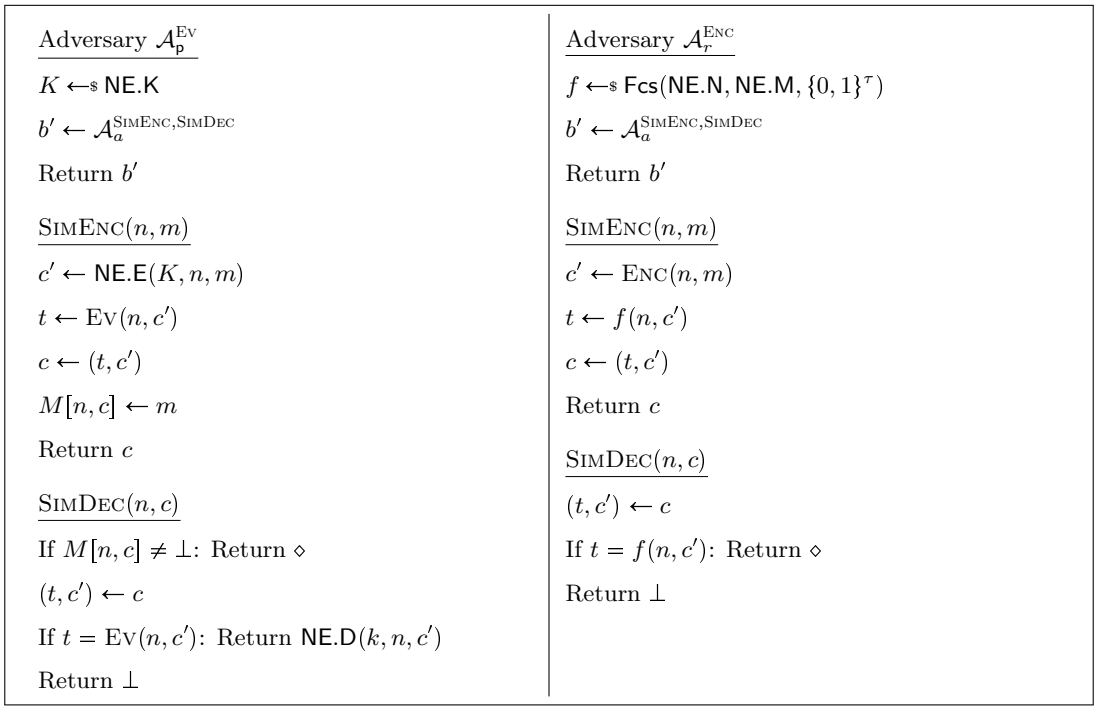


Figure 4.13: Adversaries used for proof of Theorem 4.4.1.

Define adversaries $\mathcal{A}_{\mathsf{p}}$ and $\mathcal{A}_r$ as shown in Fig. 4.13. Then,

$$\mathsf{Adv}^{\mathsf{ae\text{-}\diamond}}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}]}(\mathcal{A}_a) \leqslant \mathsf{Adv}^{\mathsf{pr}}_{\mathsf{F}}(\mathcal{A}_{\mathsf{p}}) + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{A}_r) + 2q_{\text{Dec}}/2^\tau$$

$$\begin{aligned}
&\mathbf{Query}(\mathcal{A}_{\mathsf{p}}) = q_{\text{Enc}} + q_{\text{Dec}} && \mathbf{Query}(\mathcal{A}_r) = q_{\text{Enc}} \\
&\mathbf{Time}(\mathcal{A}_{\mathsf{p}}) = \mathbf{Time}(\mathsf{G}^{\mathsf{ae\text{-}\diamond}}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}]}(\mathcal{A}_a)) && \mathbf{Time}^*(\mathcal{A}_r) = \mathbf{Time}(\mathcal{A}_a) \\
&\mathbf{Mem}(\mathcal{A}_{\mathsf{p}}) = \mathbf{Mem}(\mathsf{G}^{\mathsf{ae\text{-}\diamond}}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}]}(\mathcal{A}_a)) && \mathbf{Mem}^*(\mathcal{A}_r) = \mathbf{Mem}(\mathcal{A}_a).
\end{aligned}$$

Adversary $\mathcal{A}_r$ is an $\mathsf{F.F}$-oracle adversary.

The standard (not memory-tight) proof of the security of EtP begins identically to our proof; we start in $\mathsf{G}^{\mathsf{ae\text{-}\diamond}}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}],1}$ replace the use of F with a truly random function (using $\mathcal{A}_{\mathsf{p}}$) and then information theoretically argue that the attacker shall be incapable of creating any forgeries. In the standard proof we would transition to a game where the decryption oracle is exactly that of $\text{Dec}_0^\diamond$, i.e. it always returns $\bot$ when $M[n, c] = \bot$. Then we reduce to the security of NE to replace the generated ciphertexts with random. However this standard reduction will not be memory-tight because the attacker must store the table $M[\cdot, \cdot]$ to know

Games H_h for $0 \leqslant h \leqslant 4$	$\textsc{Enc}(n, m)$	$\textsc{Dec}(n, c)$
$K \leftarrow\!\!\$\ \mathsf{NE.K}$	$c' \leftarrow \mathsf{NE.E}(K, n, m)$ $//\mathsf{H}_{[0,3)}$	If $M[n, c] \neq \bot$: Return $\diamond$
$K' \leftarrow\!\!\$\ \mathsf{F.K}$ $//\mathsf{H}_{[0,1)}$	$c' \leftarrow\!\!\$\ \{0,1\}^{\mathsf{NE.cl}(\lvert m \rvert)}$ $//\mathsf{H}_{[3,\infty)}$	$(t, c') \leftarrow c$
$f \leftarrow\!\!\$\ \mathsf{Fcs}(\mathsf{NE.N}, \mathsf{NE.C}, \{0,1\}^\tau)$	$t \leftarrow \mathsf{F}_{K'}(n, c')$ $//\mathsf{H}_{[0,1)}$	If $t = F_{K'}(n, c')$: $//\mathsf{H}_{[0,1)}$
$//\mathsf{H}_{[1,\infty)}$	$t \leftarrow f(n, c')$ $//\mathsf{H}_{[1,\infty)}$	If $t = f(n, c')$: $//\mathsf{H}_{[1,\infty)}$
$b' \leftarrow \mathcal{A}_a^{\textsc{Enc},\textsc{Dec}}$	$c \leftarrow (t, c')$	$\mathsf{bad} \leftarrow \mathsf{true}$
Return $b' = 1$	$M[n, c] \leftarrow m$	Return $\mathsf{NE.D}(K, n, c')$ $//\mathsf{H}_{[0,2)}$
	Return c	Return $\diamond$ $//\mathsf{H}_{[2,4)}$
		Return $\bot$ $//\mathsf{H}_{[4,\infty)}$
		Return $\bot$

Figure 4.14: Hybrid games for proof of Theorem 4.4.1.

whether it should return $\diamond$ or $\bot$ when simulating decryption queries.[1] Instead we first transition to a world where F has been replaced by the random function f and $\textsc{Dec}$ always returns $\diamond$ when given a ciphertext with a correct tag. (Which we can do because either $M[n, c] \neq \bot$ held or the attacker managed to guess a random tag, which is unlikely.) Now we can make our INDR reduction memory-tight. It forwards encryption queries to its encryption oracle and then uses its own function f to create the tag. For decryption queries it checks $f(n, c') = t$, returning $\diamond$ if so and $\bot$ otherwise. Then we can finally conclude by switching to the decryption oracle $\textsc{Dec}_0^\diamond$ by arguing that noticing this change requires guessing a random tag.

It does not seem possible to extend this proof technique to AE-$\mathtt{m}$ security because the tag would be too short to embed values of m we need to remember.

We give the formal proof of Theorem 4.4.1.

Proof. We consider a sequence of hybrids H_0 through H_4 defined in Fig. 4.14. Of these hybrids we will make the following claims, which establish the upper bound on the advantage of $\mathcal{A}_a$ claimed in the proof.

1. $\mathsf{Pr}[\mathsf{H}_0] = \mathsf{Pr}[\mathsf{G}^{\mathsf{ae}\text{-}\diamond}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}],1}(\mathcal{A}_a)]$
2. $\mathsf{Pr}[\mathsf{H}_0] \leqslant \mathsf{Pr}[\mathsf{H}_1] + \mathsf{Adv}^{\mathsf{pr}}_{\mathsf{F}}(\mathcal{A}_{\mathsf{p}})$
3. $\mathsf{Pr}[\mathsf{H}_1] \leqslant \mathsf{Pr}[\mathsf{H}_2] + q_{\textsc{Dec}}/2^\tau$
4. $\mathsf{Pr}[\mathsf{H}_2] \leqslant \mathsf{Pr}[\mathsf{H}_3] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{A}_r)$
5. $\mathsf{Pr}[\mathsf{H}_3] \leqslant \mathsf{Pr}[\mathsf{H}_4] + q_{\textsc{Dec}}/2^\tau$
6. $\mathsf{Pr}[\mathsf{H}_4] = \mathsf{Pr}[\mathsf{G}^{\mathsf{ae}\text{-}\diamond}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}],0}(\mathcal{A}_a)]$

The claims regarding the complexities of the adversaries considered are clear from their code.

Transition to H_0. The hybrid H_0 was obtained by plugging the code of $\mathsf{EtP}[\mathsf{NE}, \mathsf{F}]$ into $\mathsf{G}^{\mathsf{ae}\text{-}\diamond}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}],1}(\mathcal{A}_a)$, so the first claim is clear.

[1] Note this *would* be memory-tight for AE-$\bot$ security.

TRANSITION H_0 TO H_1. In H_1 we replace each use of F with a random f sampled from $\mathsf{F.F}$. The reduction to the PR security of F is given by $\mathcal{A}_{\mathsf{p}}$ in Fig. 4.13. It simply simulates these hybrids for $\mathcal{A}_a$, using its EV oracle in place of F or f. The claimed bound follows (and is in fact an equality).

TRANSITION H_1 TO H_2. In H_2, we change the behavior of DEC. Now the oracle returns $\diamond$ when $M[n,c] = \perp$ and $t = f(n,c')$. Note that this is the only case in which the hybrids we are considering differ. In particular, they are identical-until-bad and so the Fundamental Lemma of Game Playing [BR06] gives $\mathsf{Pr}[\mathsf{H}_1] \leqslant \mathsf{Pr}[\mathsf{H}_2] + \mathsf{Pr}[\mathsf{H}_2 \text{ sets } \mathsf{bad}]$. Setting bad requires guessing a value $f(n,c')$ which is a uniform value in $\{0,1\}^\tau$. Hence by a union bound $\mathsf{Pr}[\mathsf{H}_2 \text{ sets } \mathsf{bad}] \leqslant q_{\text{DEC}}/2^\tau$, giving the claim.

TRANSITION H_2 TO H_3. In H_3, we replace the real encryption of m in ENC with a uniformly random c'. Consider the adversary $\mathcal{A}_r$ given in Fig. 4.13. It perfectly simulates hybrid H_2 to $\mathcal{A}_a$ when interacting with $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}$ and hybrid H_3 to $\mathcal{A}_a$ when interacting with $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}$. Note that $\mathcal{A}_r$ avoids storing the table M – this is possible because observe that $M[n,c] \neq \perp$ holds if and only if $f(n,c') = t$ also holds in $\mathsf{H}_2, \mathsf{H}_3$. Since the check $M[n,c] \neq \perp$ is the sole place in the code of hybrids $\mathsf{H}_2, \mathsf{H}_3$ where M affects execution, $\mathcal{A}_r$ replaces it with the equivalent check $f(n,c') = t$ and avoids storing M. So it follows that $\mathsf{Pr}[\mathsf{H}_2] \leqslant \mathsf{Pr}[\mathsf{H}_3] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{A}_r)$.

TRANSITION H_3 TO H_4. In H_4, we change the behavior of DEC. Now the oracle returns $\perp$ when $M[n,c] = \perp$ and $t = f(n,c')$. Using an identical-until-bad argument analogous to when we transitioned between H_1 and H_2 we get $\mathsf{Pr}[\mathsf{H}_1] \leqslant \mathsf{Pr}[\mathsf{H}_2] + q_{\text{DEC}}/2^\tau$ as desired.

FINAL TRANSITION. Finally we claim that the view of $\mathcal{A}_a$ in H_4 is identical to its view in $\mathsf{G}^{\mathsf{ae\text{-}\diamond}}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}],0}$, giving $\mathsf{Pr}[\mathsf{H}_4] = \mathsf{Pr}[\mathsf{G}^{\mathsf{ae\text{-}\diamond}}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}],0}(\mathcal{A}_a)]$. If $M[n,c] = \perp$ in DEC, then the oracle it always returns $\perp$ so we can ignore its use of f. This means the only use of f is in ENC and these uses never repeat inputs because the nonces do not repeat. Hence setting $t \leftarrow f(n,c)$ is equivalent to sampling t uniformly from $\{0,1\}^\tau$ which gives us exactly the view expected in $\mathsf{G}^{\mathsf{ae\text{-}\diamond}}_{\mathsf{EtP}[\mathsf{NE},\mathsf{F}],0}$.

■

4.5 *Chosen Ciphertext Security of Public Key Encryption*

Now we apply our techniques to give memory-tight reductions between single- and multi-challenge notions of chosen-ciphertext security. The standard reduction bounds the advantage of an adversary making q_{ENC} encryption queries by q_{ENC} times the advantage of an adversary making 1 query. The reduction requires memory linear in q_{ENC} and so is not memory-tight.[2] In Section 4.5.1, we consider the most common "left-or-right" definition of CCA security and introduce three different variants (mirroring the three notions for

[2]Auerbach et al. [ACFK17] stated that this reduction is memory-tight for both CPA and CCA security. While the former is correct, the latter depends on the definition of CCA. In personally communication with Auerbach et al. [ACFK21], they concurred that their claim was incorrect for their intended definition of CCA security ($w = \diamond$) but pointed out that it does work for an "exclusion" variant, $w = E$, which we will mention briefly.

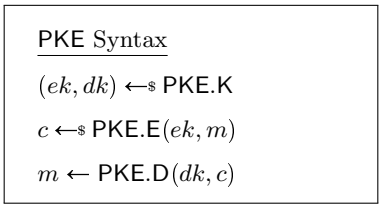


Figure 4.15: Syntax of a public key encryption scheme PKE.

AE security in Section 4.4). We give a memory-tight reduction between single- and multi-challenge security for two of the three variants ($\diamond$ and $\perp$), but the reduction is not time-tight. In Section 4.5.2, we look at the CCA security variant that requires ciphertexts be indistinguishable from random. We give a memory-tight and time-tight reduction between single- and multi-challenge security for all three variants of this notion.

PUBLIC KEY ENCRYPTION. A public key encryption scheme PKE specifies algorithms $\mathsf{PKE.K}$, $\mathsf{PKE.E}$, and $\mathsf{PKE.D}$. The syntax of these algorithms is shown in Fig. 4.15. The key generation algorithm $\mathsf{PKE.K}$ returns encryption key ek and decryption key dk. The encryption algorithm $\mathsf{PKE.E}$ encrypts message m with ek to produce a ciphertext c. We write $\mathsf{PKE.E}(ek, m; r)$ when making random coins $r \in \mathsf{PKE.R}$ explicit. The decryption algorithm decrypts c with dk to produce m. The decryption algorithm may output $m = \perp$ to indicate rejection.

Correctness requires that $\mathsf{PKE.D}(dk, c) = m$ for all $(ek, dk) \in [\mathsf{PKE.K}]$, all m, and all $c \in [\mathsf{PKE.E}(ek, m)]$. We define the min-entropy of PKE as

$$\mathsf{PKE.H}_{\mathscr{O}} = -\lg \max_{m,ek,c} \Pr[r \leftarrow\!\!\$\ \mathsf{PKE.R} : \mathsf{PKE.E}(ek, m; r) = c] \ .$$

4.5.1 Left-or-right CCA Security of PKE

LEFT-OR-RIGHT CCA SECURITY. In this section, we consider the left-or-right definition of CCA-security most commonly used in the literature. For $w \in \{\mathtt{m}, \diamond, \perp\}$ we denote this as CCA-w[3] and the corresponding security game $\mathsf{G}^{\mathsf{cca}\text{-}w}_{\mathsf{PKE},b}$ is defined in Fig. 4.16. The adversary gets the encryption key ek and has access to an encryption and a decryption oracle. The encryption oracle takes in messages m_0 and m_1 and encrypts m_b where b is the secret bit. The decryption oracle returns the decryption of a ciphertext, unless the ciphertext was previously returned by an encryption query. This is tracked by table M. When $w = \mathtt{m}$, the decryption oracle returns $M[c]$ which is m_1 from the earlier encryption query. When $w = \diamond$, it returns $\diamond$. When $w = \perp$, it returns $\perp$ which is also used by the encryption scheme to represent rejection. The advantage of an adversary

[3]The discussion in Section 4.4 about the choice to have three variants of the definitions is applicable here as well.

Game $\mathsf{G}^{\mathsf{cca}\text{-}w}_{\mathsf{PKE},b}(\mathcal{A})$	$\textsc{Enc}_b(m_0, m_1)$	$\textsc{Dec}^w(c)$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$	// $\lvert m_0\rvert = \lvert m_1\rvert$	If $M[c] \neq \bot$:
$b' \leftarrow \mathcal{A}^{\textsc{Enc}_b, \textsc{Dec}}(ek)$	$c_0 \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, m_0)$	Return $M[c]$ if $w = \mathtt{m}$
Return $b' = 1$	$c_1 \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, m_1)$	Return $\diamond$ if $w = \diamond$
	$M[c_b] \leftarrow m_1$	Return $\bot$ if $w = \bot$
	Return c_b	$m \leftarrow \mathsf{PKE.D}(dk, c)$
		Return m

Figure 4.16: Game defining CCA-w security of PKE for $w \in \{\mathtt{m}, \diamond, \bot\}$.

$\mathcal{A}$ against the CCA-w security of PKE is defined as $\mathsf{Adv}^{\mathsf{cca}\text{-}w}_{\mathsf{PKE}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{\mathsf{cca}\text{-}w}_{\mathsf{PKE},1}(\mathcal{A})] - \mathsf{Pr}[\mathsf{G}^{\mathsf{cca}\text{-}w}_{\mathsf{PKE},0}(\mathcal{A})]$.

The goal of this section is to relate the advantage of attacks making only a single encryption query and those making many such queries. When wanting to make the distinction explicit we may use the adjectives "many" and "single" or prefix the abbreviation of a security notion with an 'm' or '1'.

1CCA-$\diamond$ IMPLIES mCCA-$\diamond$. The following theorem gives a memory-tight reduction establishing that CCA-$\diamond$ security against adversaries making one encryption query implies security for an arbitrary number of queries. The proof makes use of our inefficient tagging technique. The reduction performs a hybrid over the encryption queries of the original adversary and is thus not advantage-tight.

Theorem 4.5.1 (1CCA-$\diamond \Rightarrow$ mCCA-$\diamond$)**.** *Let* PKE *be a public key encryption scheme. Let* $\mathcal{A}_{\mathsf{m}}$ *be an adversary with* $(q_{\textsc{Enc}}, q_{\textsc{Dec}}) = \mathbf{Query}(\mathcal{A}_{\mathsf{m}})$*. Define* $D_{(\cdot)}$ *by* $D_n = \{0,1\}^n \times [q_{\textsc{Enc}}]$*. Let* $\mathcal{A}_1$ *be the* $\mathsf{Fcs}(\mathbb{N}, D, \mathsf{PKE.R})$*-oracle adversary shown in Fig. 4.17. Then,*

$$\mathsf{Adv}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE}}(\mathcal{A}_{\mathsf{m}}) \leqslant q_{\textsc{Enc}} \cdot \mathsf{Adv}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE}}(\mathcal{A}_1) + 4 \cdot q_{\textsc{Enc}} \cdot q_{\textsc{Dec}} / 2^{\mathsf{PKE.H}_\infty}$$

$$\mathbf{Query}(\mathcal{A}_1) = (1, q_{\textsc{Dec}})$$

$$\mathbf{Time}^*(\mathcal{A}_1) = O(\mathbf{Time}(\mathcal{A}_{\mathsf{m}})) + q_{\textsc{Enc}}(q_{\textsc{Dec}} + 1)\mathbf{Time}(\mathsf{PKE})$$

$$\mathbf{Mem}^*(\mathcal{A}_1) = O(\mathbf{Mem}(\mathcal{A}_{\mathsf{m}})) + \mathbf{Mem}(\mathsf{PKE}) + \lg q_{\textsc{Enc}}.$$

The standard (non-memory-tight) reduction against 1CCA security picks an index $k \in [q_{\textsc{Enc}}]$ where $q_{\textsc{Enc}}$ is the number of encryption queries made by $\mathcal{A}_{\mathsf{m}}$. It runs $\mathcal{A}_{\mathsf{m}}$, simulating encryption queries as follows. For the first $k-1$ encryption queries, it answers with an encryption of m_1, for the k-th encryption query it forwards the query to its own encryption oracle, and the rest of the queries it answers with an encryption of m_0. To answer the decryption queries, the reduction returns $\diamond$ if it was ever queried the ciphertext for a previous encryption query. Otherwise, it forwards the query to its own decryption oracle. Finally, the

Adversary $\mathcal{A}_1^{\text{Enc},\text{Dec}}(ek)$	$\text{SimEnc}(m_0, m_1)$	$\text{SimDec}(c)$
$k \xleftarrow{\$} [q_{\text{Enc}}]$	$i \leftarrow i + 1$	If $c = c^*$: Return $\diamond$
$i \leftarrow 0$	For $d \in \{0,1\}$:	$m \leftarrow \text{Dec}(c)$
$D_{(\cdot)} \leftarrow \{0,1\}^{(\cdot)} \times [q_{\text{Enc}}]$	$r_d \leftarrow f(\lvert m_d \rvert, (m_d, i))$	If $m = \perp$: Return $\perp$
$f \xleftarrow{\$} \mathsf{Fcs}(\mathbb{N}, D, \mathsf{PKE.R})$	$c_d \leftarrow \mathsf{PKE.E}(ek, m_d; r_d)$	For $j \in [i]$ do:
$b' \leftarrow \mathcal{A}_{\mathsf{m}}^{\text{SimEnc},\text{SimDec}}(ek)$	If $i < k$: $c \leftarrow c_1$	If $m \in \{m_0^*, m_1^*\}$ and $j = k$:
Return b'	If $i = k$:	Skip to next j
	$c \leftarrow \text{Enc}(m_0, m_1)$	$r \leftarrow f(\lvert m \rvert, (m, j))$
	$c^* \leftarrow c$	If $\mathsf{PKE.E}(ek, m; r) = c$:
	$(m_0^*, m_1^*) \leftarrow (m_0, m_1)$	Return $\diamond$
	If $i > k$: $c \leftarrow c_0$	Return m
	Return c	

Figure 4.17: Adversary $\mathcal{A}_1$ for Theorem 4.5.1.

reduction adversary outputs whatever $\mathcal{A}_{\mathsf{m}}$ outputs. Standard hybrid analysis shows that if the advantage of $\mathcal{A}_{\mathsf{m}}$ is ε, then the advantage of the reduction adversary is $\varepsilon/q_{\text{Enc}}$. Simulating decryption queries required remembering all prior encryption queries and hence the reduction is not memory-tight.

We give an adversary $\mathcal{A}_1$ in Fig. 4.17 that is very similar to the reduction just described, but avoids remembering prior encryption queries. The main idea is that it picks the coins when encrypting m_0 or m_1 locally as the output of a random function f applied to the message and a counter. This allows $\mathcal{A}_1$ to detect whether a ciphertext c queried to the decryption oracle is one it answered to an earlier encryption query as follows: it first asks for the decryption of c from its own decryption oracle and receives m. Then it iterates over all counter values for which encryption queries have been made so far and checks if c was the encryption of m using the output of f on m and the counter as coins. If any of these checks succeed it returns $\diamond$, otherwise it returns m. If c was the answer of an encryption query $\mathcal{A}_1$ detects it successfully. The probability that $\mathcal{A}_1$ returns $\diamond$ for a decryption query when it should not is small.

Notice that the additional memory overhead for $\mathcal{A}_1$ is just that required to store a counter, run $\mathsf{PKE.E}$, and store (c^*, m_0^*, m_1^*). However, there is an *increase* in runtime by $q_{\text{Enc}} \cdot q_{\text{Dec}} \cdot \mathbf{Time}(\mathsf{PKE})$ because of the iteration over the counters to answer decryption queries. As discussed in the introduction, such reductions may be useful when the best attack for the underlying problem with low memory requires significantly more running time than the best attack with high memory.

EXTENSION TO CCA-$\perp$. We can prove the same result for CCA-$\perp$, using a very similar proof flow. Alternatively, Theorem 4.5.1 directly implies the same result for CCA-$\perp$. First off, 1CCA-$\perp$ implies 1CCA-$\diamond$ in a memory-tight way because an adversary with access to $\text{DEC}^{\perp}$ can simulate $\text{DEC}^{\diamond}$ by just remembering the ciphertext c^* returned for the single ENC query, returning $\diamond$ if c^* is queried to DEC, and otherwise forwarding the response of its own decryption oracle. We also noted above that mCCA-$\diamond$ implies mCCA-$\perp$ in a memory-tight way. Putting it together with Theorem 4.5.1 gives the desired result.

However, it does not seem possible to extend this proof technique to CCA-$\mathtt{m}$ security because if the adversary queries the decryption oracle on a ciphertext c which was an answer to a previous query for (m_0, m_1) the oracle needs to return m_1 even if c is an encryption of m_0. This seems to require memory to simulate.

OTHER VARIANTS (EXCLUSION AND PENALTY). In personal communication with Auerbach et al. [ACFK21] they pointed out two other variants of CCA security which were given in [BHK15]. For the sake of concreteness we will described them as based on CCA-$\perp$, but the way they are defined means we could just as well have started from CCA-$\mathtt{m}$ or CCA-$\diamond$. The first, which we will refer to as CCA-E (where 'E' stands for "exclusion") is defined the same as CCA-$\perp$ except we require security only for adversaries that will *never* make a decryption query $\text{DEC}(c)$ if c was ever returned by a prior encryption query. The standard hybrid argument is memory-tight for CCA-E. This follows immediately because we do not have to simulate decryption for forwarded ciphertexts.

For the second, which we will refer to as CCA-P (where 'P' stands for "penalty"), we can think of the adversary as being penalized at the end of the game if it even makes a decryption query $\text{DEC}(c)$ if c was ever returned by a prior encryption query. Let $\mathsf{G}^{\mathsf{cca\text{-}P}}_{\mathsf{PKE},1}$ be identical to $\mathsf{G}^{\mathsf{cca\text{-}\perp}}_{\mathsf{PKE},1}$, except the game returns false no matter what b' is if the adversary ever made such a query. Similarly, let $\mathsf{G}^{\mathsf{cca\text{-}P}}_{\mathsf{PKE},0}$ be identical to $\mathsf{G}^{\mathsf{cca\text{-}\perp}}_{\mathsf{PKE},0}$, except the game returns true no matter what b' is if the adversary ever made such a query. Then we define $\mathsf{Adv}^{\mathsf{cca\text{-}P}}_{\mathsf{PKE}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{\mathsf{cca\text{-}P}}_{\mathsf{PKE},1}(\mathcal{A})] - \mathsf{Pr}[\mathsf{G}^{\mathsf{cca\text{-}P}}_{\mathsf{PKE},0}(\mathcal{A})]$. It is not clear how to write a memory-tight hybrid argument for CCA-P.

The philosophical and technical arguments from Sec. 4.4 for why $w = \mathtt{m}$ may be "correct" apply similarly to argue in favor of it over $w = \text{E}$ and $w = \text{P}$. Additionally, $w = \text{E}$ seems particularly "weak" because it seems overly restrictive. Consider a low-memory attacker that has made a large number of encryption queries so far. It will be incredibly restricted in what decryption queries it can make because it is required to absolutely avoid any query that has a non-zero chance of being a ciphertext returned by a prior encryption given its current state. Note that even very naive adversaries are excluded, for example one that asks a random string of appropriate format to the decryption oracle after seeing some challenge ciphertexts.

TIME-TIGHTNESS IF MESSAGES DO NOT REPEAT. If we require that $\mathcal{A}_{\mathsf{m}}$ never repeats messages queried to ENC then we can make Theorem 4.5.1 time-tight as well. In that case, $\mathcal{A}_1$ would not need to use the counter i to ensure domain separation for f and so it would not have to use the loop inside SIMDEC. One setting where this suffices is if PKE is being used as a key-encapsulation mechanism. Then we can think of m_0 and m_1 being picked uniformly at random. Using the switching lemma we can switch to m_0 and m_1 being sampled without replacement, meaning the encryption queries do not repeat.[4] We next give the formal proof of Theorem 4.5.1.

Proof. We start by considering the hybrid games H^b_h for $0 \leqslant h \leqslant 2$ defined in Fig. 4.18. Of these we make the following claims for $b \in \{0,1\}$.

1. $\mathsf{Pr}[\mathsf{G}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE},b}] = \mathsf{Pr}[\mathsf{H}^b_0]$
2. $\mathsf{Pr}[\mathsf{H}^b_0] = \mathsf{Pr}[\mathsf{H}^b_1]$
3. $|\mathsf{Pr}[\mathsf{H}^b_1] - \mathsf{Pr}[\mathsf{H}^b_2]| \leqslant q_{\text{ENC}} \cdot q_{\text{DEC}} \cdot 2^{-\mathsf{PKE.H}_\infty}$

Combining the above claims, we get that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE}}(\mathcal{A}_{\mathsf{m}}) &= \mathsf{Pr}[\mathsf{G}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE},1}] - \mathsf{Pr}[\mathsf{G}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE},0}] = \mathsf{Pr}[\mathsf{H}^1_0] - \mathsf{Pr}[\mathsf{H}^0_0] \\ &= \mathsf{Pr}[\mathsf{H}^1_1] - \mathsf{Pr}[\mathsf{H}^1_0] \leqslant \mathsf{Pr}[\mathsf{H}^1_2] - \mathsf{Pr}[\mathsf{H}^0_2] + 2 \cdot q_{\text{ENC}} \cdot q_{\text{DEC}} \cdot 2^{-\mathsf{PKE.H}_\infty}. \end{aligned} \tag{4.1}$$

Next, we prove the claims.

TRANSITION TO H^b_0. The game H^b_0 was copied from the game $\mathsf{G}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE},b}$. We added variable i that counts the number of ENC queries and will be used for future hybrids and unrolled the encryption to make the sampling of coins explicit. It follows that $\mathsf{Pr}[\mathsf{G}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE},b}] = \mathsf{Pr}[\mathsf{H}^b_0]$.

TRANSITION FROM H^b_0 TO H^b_1. In game H^b_1, we replace the random sampling of r_0 and r_1 with the output of a random function f, using a counter i to provide domain separation between different queries. This method of choosing r is equivalent, so $\mathsf{Pr}[\mathsf{H}^b_0] = \mathsf{Pr}[\mathsf{H}^b_1]$.

TRANSITION FROM H^b_1 TO H^b_2. In game H^b_2, we stop using $M[\cdot]$ to keep track ciphertexts that were returned by ENC. Instead we first decrypt c to m and then iterating over $j \in [i]$ to check whether m encrypted with ek using randomness $r = f(|m|,(m,j))$ is c. Note that if $M[c] \neq \perp$ holds in H^b_1 then there will necessarily be such a j (in particular j being the value i held at the time of the query that set $M[c]$).

So H^b_1 and H^b_2 are identical unless the following bad event happens: there is a DEC query on c and for some $j \in [j]$, it holds that $c = \mathsf{PKE.E}(ek, m; f(|m|,(m,j)))$ and despite m was not m_b for the j-th query to

[4] Here the list of m_0 and m_1 to be queried can be specified by an oracle given to the adversary.

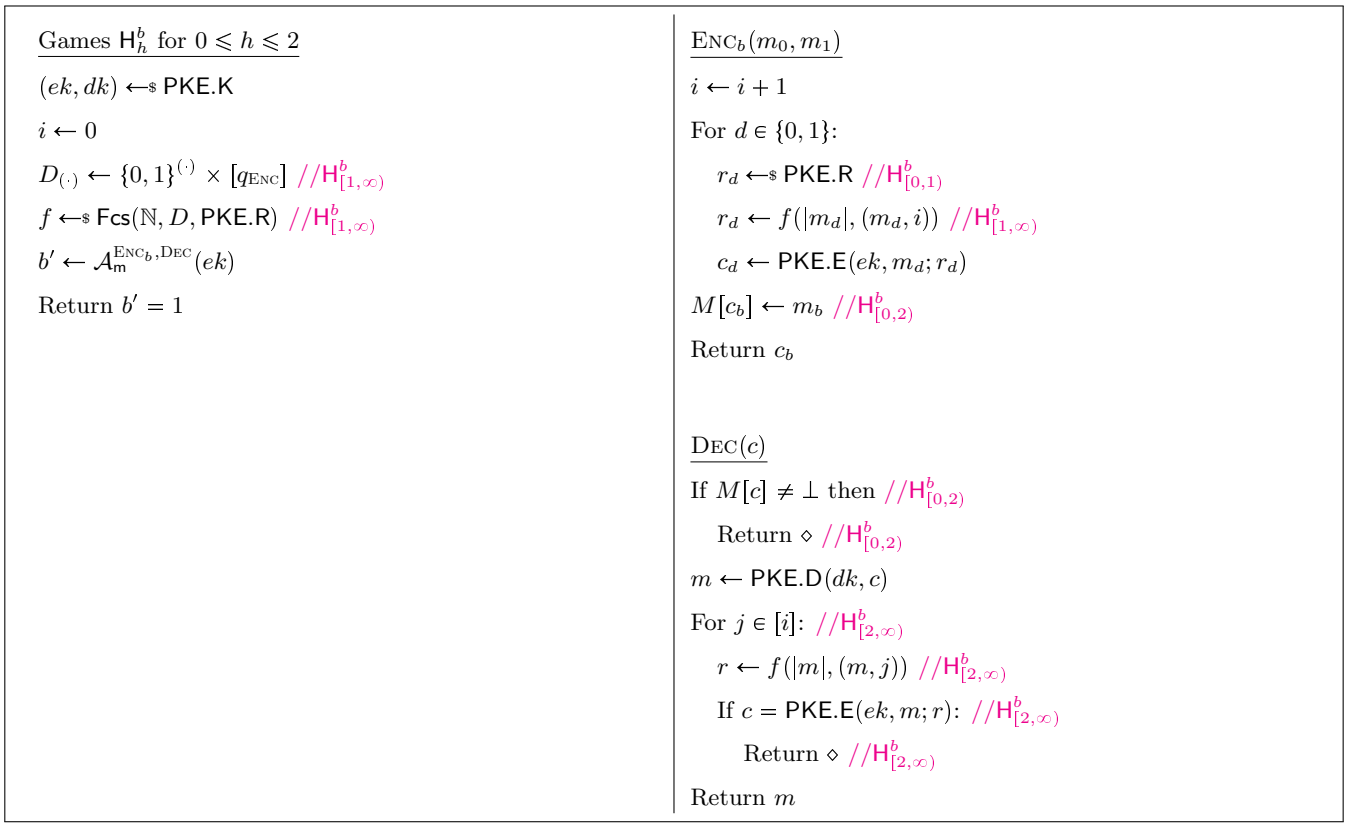


Figure 4.18: First set of hybrids H^b_h used in the proof of Theorem 4.5.1.

ENC. We can analyze the probability of this in H^b_1. Since m was not the j-th query to ENC, the view of the adversary was independent of $f(|m|, (m, j))$ at this time. By the min-entropy of PKE, the probability of this occurring for a given decryption query and j is at most $1/2^{\mathsf{PKE.H}_\infty}$. Taking a union bound over all $j \in [q_{\text{ENC}}]$ and decryption queries the overall probability is at most $q_{\text{ENC}} \cdot q_{\text{DEC}}/2^{\mathsf{PKE.H}_\infty}$. We could formalize this via the Fundamental Lemma of Game Playing [BR06], to get $|\mathsf{Pr}[\mathsf{H}^b_1] - \mathsf{Pr}[\mathsf{H}^b_2]| \leqslant q_{\text{ENC}} \cdot q_{\text{DEC}}/2^{\mathsf{PKE.H}_\infty}$ as desired.

TRANSITION TO $\mathsf{H}^{k,b}$ HYBRIDS. We next consider the hybrid games $\mathsf{H}^{k,b}$ for $(k, b) \in [q_{\text{ENC}}] \times \{0,1\}$ defined in Fig. 4.19. In these hybrids, ciphertexts for m_1 are returned when $i < k$, ciphertexts for m_0 are returned when $i > k$, and a ciphertext for m_b is returned when $i = k$. Note that at the extremes ($k = 1, b = 0$ and $k = q_{\text{ENC}}, b = 1$), $\mathcal{A}_{\mathsf{m}}$ will receive either all ciphertexts of m_0 or all ciphertexts of m_1. The decryption queries are answered as in H^b_2. So $\mathsf{H}^{1,0}$ perfectly matches H^0_2 and $\mathsf{H}^{q_{\text{ENC}},1}$ perfectly matches H^1_2. Hence combining with (4.1), we have

$$\mathsf{Adv}^{\mathsf{cca}^{\diamond}}_{\mathsf{PKE}}(\mathcal{A}_{\mathsf{m}}) \leqslant \mathsf{Pr}[\mathsf{H}^{q_{\text{ENC}},1}] - \mathsf{Pr}[\mathsf{H}^{1,0}] + 2 \cdot q_{\text{ENC}} \cdot q_{\text{DEC}} \cdot 2^{-\mathsf{PKE.H}_\infty} \ . \tag{4.2}$$

Hybrids $\mathsf{H}^{k,b}$	$\textsc{Enc}_b(m_0, m_1)$	$\textsc{Dec}(c)$
$//(k,b) \in [q_{\textsc{Enc}}] \times \{0,1\}$	$i \leftarrow i + 1$	$m \leftarrow \mathsf{PKE.D}(dk, c)$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$	For $d \in \{0,1\}$:	For $j \in [i]$:
$i \leftarrow 0$	$r_d \leftarrow f(\lvert m_d\rvert, (m_d, i))$	$r \leftarrow f(\lvert m\rvert, (m, j))$
$D_{(\cdot)} \leftarrow \{0,1\}^{(\cdot)} \times [q_{\textsc{Enc}}]$	$c_d \leftarrow \mathsf{PKE.E}(ek, m_d; r_d)$	If $c = \mathsf{PKE.E}(ek, m; r)$:
$f \leftarrow\!\!\$\ \mathsf{Fcs}(\mathbb{N}, D, \mathsf{PKE.R})$	If $i < k$: $c \leftarrow c_1$	Return $\diamond$
$b' \leftarrow \mathcal{A}_{\mathsf{m}}^{\textsc{Enc}_b, \textsc{Dec}}(ek)$	If $i = k$: $c \leftarrow c_b$	Return m
Return $b' = 1$	If $i > k$: $c \leftarrow c_0$	
	Return c	

Figure 4.19: Second set of hybrids used in the proof of Theorem 4.5.1.

In general, in $\mathsf{H}^{k,b}$ the first $k+b-1$ encryption queries use m_1 and the rest use m_0; so $\mathsf{Pr}[\mathsf{H}^{k,1}] = \mathsf{Pr}[\mathsf{H}^{k+1,0}]$ holds. Hence

$$\begin{aligned}\mathsf{Pr}[\mathsf{H}^{q_{\textsc{Enc}},1}] - \mathsf{Pr}[\mathsf{H}^{1,0}] &= \mathsf{Pr}[\mathsf{H}^{q_{\textsc{Enc}},1}] - \mathsf{Pr}[\mathsf{H}^{1,0}] + \sum_{k \in [q_{\textsc{Enc}}-1]} \mathsf{Pr}[\mathsf{H}^{k,1}] - \mathsf{Pr}[\mathsf{H}^{k+1,0}] \\ &= \sum_{k \in [q_{\textsc{Enc}}]} \mathsf{Pr}[\mathsf{H}^{k,1}] - \mathsf{Pr}[\mathsf{H}^{k,0}]. \end{aligned} \tag{4.3}$$

$\textsc{Transition to}\ \mathsf{H}_h\ \textsc{hybrids}$. Next, consider the games shown in Fig. 4.25. Of these we make the following claims.

1. $2\mathsf{Pr}[\mathsf{H}_0] - 1 = (1/q_{\textsc{Enc}}) \sum_{k \in [q_{\textsc{Enc}}]} \mathsf{Pr}[\mathsf{H}^{k,1}] - \mathsf{Pr}[\mathsf{H}^{k,0}]$

2. $\mathsf{Pr}[\mathsf{H}_0] \leqslant \mathsf{Pr}[\mathsf{H}_1] + q_{\textsc{Dec}} \cdot 2^{-\mathsf{PKE.H}_\infty}$

3. $\mathsf{Pr}[\mathsf{H}_1] = \mathsf{Pr}[\mathsf{H}_2]$

4. $2\mathsf{Pr}[\mathsf{H}_2] - 1 = \mathsf{Adv}^{\mathsf{cca}^{-\diamond}}_{\mathsf{PKE}}(\mathcal{A}_1)$

Hybrids H_h for $0 \leqslant h \leqslant 2$

$(k, b) \leftarrow\!\!\$\ [q_{\text{ENC}}] \times \{0,1\}$

$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$

$i \leftarrow 0$

$D_{(\cdot)} \leftarrow \{0,1\}^{(\cdot)} \times [q_{\text{ENC}}]$

$f \leftarrow\!\!\$\ \mathsf{Fcs}(\mathbb{N}, D, \mathsf{PKE.R})$

$b' \leftarrow \mathcal{A}_{\mathsf{m}}^{\text{ENC}_b, \text{DEC}}(ek)$

Return $b' = b$

DEC(c)

If $c = c^*$: Return $\diamond$ //$\mathsf{H}_{[1,\infty)}$

$m \leftarrow \mathsf{PKE.D}(dk, c)$

For $j \in [i]$ do:

  If $m \in \{m_0^*, m_1^*\}$ and $j = k$: //$\mathsf{H}_{[1,\infty)}$

    Skip to next j //$\mathsf{H}_{[1,\infty)}$

  $r \leftarrow f(|m|, (m, j))$

  If $\mathsf{PKE.E}(ek, m; r) = c$:

    Return $\diamond$

Return m

ENC$_b(m_0, m_1)$

$i \leftarrow i + 1$

For $d \in \{0,1\}$:

  $r_d \leftarrow f(|m_d|, (m_d, i))$

  $c_d \leftarrow \mathsf{PKE.E}(ek, m_d; r_d)$

If $i < k$:

  $c \leftarrow c_1$

If $i = k$:

  $c \leftarrow c_b$ //$\mathsf{H}_{[0,1)}$

  $r \leftarrow f(|m_b|, (m_b, i))$ //$\mathsf{H}_{[1,2)}$

  $r \leftarrow\!\!\$\ \mathsf{PKE.R}$ //$\mathsf{H}_{[2,\infty)}$

  $c \leftarrow \mathsf{PKE.E}(ek, m_b; r)$ //$\mathsf{H}_{[1,\infty)}$

  $c^* \leftarrow c$ //$\mathsf{H}_{[1,\infty)}$

  $(m_0^*, m_1^*) \leftarrow (m_0, m_1)$ //$\mathsf{H}_{[1,\infty)}$

If $i > k$:

  $c \leftarrow c_0$

Return c

Figure 4.20: Final set of hybrids for the proof of Theorem 4.5.1.

In the rest of the proof we address these one at a time. Putting them together along with (4.2) and (4.3) gives the bound claimed in the theorem statement via,

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE}}(\mathcal{A}_{\mathsf{m}}) &= 2 \cdot q_{\text{ENC}} \cdot q_{\text{DEC}} \cdot 2^{-\mathsf{PKE.H}_\infty} + \sum_{k \in [q_{\text{ENC}}]} \mathsf{Pr}[\mathsf{H}^{k,1}] - \mathsf{Pr}[\mathsf{H}^{k,0}] \\
&= 2 \cdot q_{\text{ENC}} \cdot q_{\text{DEC}} \cdot 2^{-\mathsf{PKE.H}_\infty} + q_{\text{ENC}}(2\mathsf{Pr}[\mathsf{H}_0] - 1) \\
&\leqslant 2 \cdot q_{\text{ENC}} \cdot q_{\text{DEC}} \cdot 2^{-\mathsf{PKE.H}_\infty} + q_{\text{ENC}}(2\mathsf{Pr}[\mathsf{H}_2] + 2q_{\text{DEC}} \cdot 2^{-\mathsf{PKE.H}_\infty} - 1) \\
&= \mathsf{Adv}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE}}(\mathcal{A}_1) + 4q_{\text{ENC}} \cdot q_{\text{DEC}} \cdot 2^{-\mathsf{PKE.H}_\infty}.
\end{aligned}$$

TRANSITION TO H_0. Game H_0 is identical to $\mathsf{H}^{k,b}$ with (k, b) chosen at random and with the game returning true if $\mathcal{A}_{\mathsf{m}}$ correctly guessed b. Standard calculations by conditioning on all the possible values of (k, b) gives the claim.

Game $\mathsf{G}^{\$\mathsf{cca}\text{-}w}_{\mathsf{PKE},b}(\mathcal{A})$	$\textsc{Enc}_b(m)$	$\textsc{Dec}^w(c)$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$	$c_1 \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, m)$	If $M[c] \neq \bot$:
$b' \leftarrow \mathcal{A}^{\textsc{Enc}_b, \textsc{Dec}^w}(ek)$	$c_0 \leftarrow\!\!\$\ \mathsf{PKE.C}(ek, \lvert m\rvert)$	Return $M[c]$ if $w = \mathtt{m}$
Return $b' = 1$	$M[c_b] \leftarrow m$	Return $\diamond$ if $w = \diamond$
	Return c_b	Return $\bot$ if $w = \bot$
		$m \leftarrow \mathsf{PKE.D}(dk, c)$
		Return m

Figure 4.21: Game defining \$CCA-w security of PKE for $w \in \{\mathtt{m}, \diamond, \bot\}$.

Transition from H_0 to H_1. We make three changes to transition to H_1. First, when $i = k$ in Enc we store the two messages queried as m_0^* and m_1^* along with the ciphertext returned as c^*. Then on a Dec query, $\diamond$ is returned immediately if $c = c^*$. Finally in Dec we skip over the iteration of $j = k$ if $m \in \{m_0^*, m_1^*\}$. When $m = m_b^*$ this does not change anything because the $c = c^*$ check will have covered that case.

So this changes the behavior of the oracle only if $m = m_{1-b}^* \neq m_b^*$, $c = \mathsf{PKE.E}(ek, m; f(|m|, (m, k)))$, in H_0 it would return $\diamond$ while in H_1 it may return m. Note then that in H_1 the view of the adversary is completely independent of this $f(|m|, (m, k))$. So we can think of the adversary having Dec attempts to guess a ciphertext generated with uniformly random coins. Hence the probability of this sort of query in H_1 is at most $q_{\textsc{Dec}} \cdot 2^{-\mathsf{PKE.H}_\infty}$. This gives $\mathsf{Pr}[\mathsf{H}_0] \leqslant \mathsf{Pr}[\mathsf{H}_1] + q_{\textsc{Dec}} \cdot 2^{-\mathsf{PKE.H}_\infty}$.

Transition from H_1 to H_2. The only change in H_2 is that in Enc for $i = k$, the randomness is sampled uniformly at random instead of by evaluating f. Because $f(|m_b|, (m_b^*, k))$ is used nowhere else, this does not change the behavior of the game. It follows that $\mathsf{Pr}[\mathsf{H}_1] = \mathsf{Pr}[\mathsf{H}_2]$.

Adversary $\mathcal{A}_1$. Finally we can see that our adversary $\mathcal{A}_1$ (defined in Fig. 4.17) perfectly simulates the view of $\mathcal{A}_\mathtt{m}$ in H_2. It was obtained by copying the code of H_2 and then modifying it to query its Enc and Dec oracle as appropriate. It follows that $\mathsf{Pr}[\mathsf{H}_2] = 0.5\mathsf{Pr}[\mathsf{G}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE},1}(\mathcal{A}_1)] + 0.5(1 - \mathsf{Pr}[\mathsf{G}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE},0}(\mathcal{A}_1)])$. Hence, $2\mathsf{Pr}[\mathsf{H}_2] - 1 = \mathsf{Adv}^{\mathsf{cca}\text{-}\diamond}_{\mathsf{PKE}}(\mathcal{A}_1)$.

Adversary $\mathcal{A}_1$'s extra running time comes from using $\mathsf{PKE.E}$ in SimEnc and in the loop in SimDec. Its extra memory is that required for running $\mathsf{PKE.E}$, for storing i, and for storing (c^*, m_0^*, m_1^*). ■

4.5.2 Indistinguishable from Random CCA Security of PKE

We saw in the previous section that we could have a memory-tight reduction from mCCA-$\diamond$ to 1CCA-$\diamond$; however, the reduction is not tight with respect to running time. In this section, we show that for a different

formalization of CCA security, we can indeed have a memory-tight and time-tight reduction between many- and single-challenge variants.

CIPHERTEXT AND ENCRYPTION KEY SPACE. Before describing the indistinguishable from random formalization of CCA security, we need to make some assumptions on PKE. We define the encryption keyspace by $\mathsf{PKE.Ek} = \{ek : (ek, dk) \in \mathsf{PKE.K}\}$. We assume for each $ek \in \mathsf{PKE.Ek}$ and allowed message length $n \in \mathbb{N}$ there is a set $\mathsf{PKE.C}(ek, n)$ such that $\mathsf{PKE.E}(ek, m; r) \in \mathsf{PKE.C}(ek, |m|)$ always holds. Let $\mathsf{PKE.C}^{-1}(ek, c)$ returns n such that $c \in \mathsf{PKE.C}(ek, n)$. Correctness implies that $\mathsf{PKE.C}(ek, n)$ and $\mathsf{PKE.C}(ek, n')$ are disjoint for $n \neq n'$.

INDISTINGUISHABLE FROM RANDOM CIPHERTEXT CCA SECURITY. The security notion we will consider in this section is captured by the game $\mathsf{G}^{\$\mathsf{cca}\text{-}w}$ shown in Fig. 4.21. It requires that ciphertexts output by the encryption scheme cannot be distinguished from ciphertexts chosen at random even given access to a decryption oracle. The adversary gets the encryption key ek and has access to an encryption oracle ENC and a decryption oracle DEC. The adversary needs to distinguish the following real and ideal worlds: in the real world, a query to ENC with a message m returns an encryption of m under ek, while in the ideal world, the same query returns a uniformly random element of $\mathsf{PKE.C}(ek, |m|)$. The decryption oracle $\textsc{Dec}^w$ acts exactly as the corresponding oracle in $\mathsf{G}^{\mathsf{cca}\text{-}w}$.[5] The advantage of an adversary $\mathcal{A}$ against the \$CCA-w security of PKE is defined as $\mathsf{Adv}^{\$\mathsf{cca}\text{-}w}_{\mathsf{PKE}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{\$\mathsf{cca}\text{-}w}_{\mathsf{PKE},1}(\mathcal{A})] - \mathsf{Pr}[\mathsf{G}^{\$\mathsf{cca}\text{-}w}_{\mathsf{PKE},0}(\mathcal{A})]$. If $\mathsf{PKE.E}(ek, m; \cdot)$ is injective, then this is exactly identical to the standard CCA notion for $\mathsf{PKE.C}(ek, n) = \{\mathsf{PKE.E}(ek, m; r) : |m| = n, r \in \mathsf{PKE.R}\}$.

1\$CCA-m IMPLIES m\$CCA-m. The following theorem captures a memory-tight reduction establishing that 1\$CCA-m security implies m\$CCA-m security. The proof makes use of our message encoding technique.

Theorem 4.5.2 (1\$CCA-m $\Rightarrow$ m\$CCA-m)**.** *Let PKE be a public key encryption scheme. Let τ satisfy $|\mathsf{PKE.C}(ek, n)| \geqslant 2^n \cdot 2^\tau$ for all n, ek. Let $\mathcal{A}_\mathsf{m}$ be an adversary with $(q_{\textsc{Enc}}, q_{\textsc{Dec}}, q_\mathrm{h}) = \mathbf{Query}(\mathcal{A}_\mathsf{m})$ and assume $q_{\textsc{Enc}} + q_{\textsc{Dec}} \leqslant 0.5 \cdot 2^\tau$. Let $\mathcal{F} = \mathsf{Inj}^{\pm}(T, D, R)$ where T, D, and R are defined by $T = \mathbb{N} \times \mathsf{PKE.Ek}$, $D_{n,ek} = \{0,1\}^n \times [q_{\textsc{Enc}}]$ and $R_{n,ek} = \mathsf{PKE.C}(ek, n)$. Let $\mathcal{A}_1$ be the $\mathcal{F}$-oracle adversary defined in Fig. 4.22. Then,*

$$\mathsf{Adv}^{\$\mathsf{cca}\text{-}m}_{\mathsf{PKE}}(\mathcal{A}_\mathsf{m}) \leqslant q_{\textsc{Enc}} \cdot \mathsf{Adv}^{\$\mathsf{cca}\text{-}m}_{\mathsf{PKE}}(\mathcal{A}_1) + 8 q_{\textsc{Enc}} q_{\textsc{Dec}} / 2^\tau + 5 q_{\textsc{Enc}}^2 / 2^\tau$$

$$\mathbf{Query}(\mathcal{A}_1) = (1, q_{\textsc{Dec}}, q_\mathrm{h})$$

$$\mathbf{Time}^*(\mathcal{A}_1) = O(\mathbf{Time}(\mathcal{A}_\mathsf{m})) + q_{\textsc{Enc}} \mathbf{Time}(\mathsf{PKE})$$

$$\mathbf{Mem}^*(\mathcal{A}_1) = O(\mathbf{Mem}(\mathcal{A}_\mathsf{m})) + \mathbf{Mem}(\mathsf{PKE}) \lg q_{\textsc{Enc}}.$$

The standard (non-memory-tight) reduction against 1\$CCA security that runs an m\$CCA adversary $\mathcal{A}_\mathsf{m}$

[5] As mentioned, the discussion in Section 4.4 about the three variants definitions is applicable here as well. In Section 4.7 we give an example where we can prove CCA security of a KEM/DEM scheme in the memory restricted setting, but only if we use the $w = \mathtt{m}$ definition.

Adversary $\mathcal{A}_1^{\text{Enc},\text{Dec}}(ek)$	$\text{SimEnc}(m)$	$\text{SimDec}(c)$
$k \leftarrow\!\!\$\ [q_{\text{Enc}}]$	$i \leftarrow i + 1$	If $c = c^*$: Return m^*
$i \leftarrow 0$	$c_1 \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, m)$	$n \leftarrow \mathsf{PKE.C}^{-1}(ek, c)$
$f \leftarrow\!\!\$\ \mathsf{Inj}^{\pm}(T, D, R)$	$c_0 \leftarrow f((\lvert m\rvert, ek), (m, i))$	$(m, j) \leftarrow f^{-1}((n, ek), c)$
$b' \leftarrow \mathcal{A}_{\mathsf{m}}^{\text{SimEnc},\text{SimDec}}(ek)$	If $i < k$: $c \leftarrow c_1$	If $m \neq \perp$ and $k \leqslant j \leqslant i$:
Return b'	If $i = k$:	If $(m, j) = (m^*, k)$:
	$c \leftarrow \text{Enc}(m)$	Skip next line
	$(c^*, m^*) \leftarrow (c, m)$	Return m
	If $i > k$: $c \leftarrow c_0$	$m \leftarrow \text{Dec}(c)$
	Return c	Return m

Figure 4.22: Adversary $\mathcal{A}_1$ for Theorem 4.5.2.

works in a similar manner as the standard reduction from an 1CCA adversary and an mCCA adversary that we described in Section 4.5.1. Again here, simulating decryption queries requires remembering all the answers of the encryption queries, and hence the reduction is not memory-tight.

We give an adversary $\mathcal{A}_1$ in Fig. 4.22 that is very similar to the standard reduction, but avoids remembering all the answers of the encryption queries. The main idea here is picking the ciphertext c_0 as the output of a random injective function f evaluated on the message and a counter, instead of sampling it uniformly at random. This way of picking the c_0 allows $\mathcal{A}_1$ detect whether a ciphertext c queried to the decryption oracle was the answer to an earlier encryption query as follows: it first checks if the inverse of f on the ciphertext is defined (i.e., not $\perp$), it returns the message part of the inverse. Otherwise it asks for the decryption of the ciphertext to its own decryption oracle and returns the answer. Using our assumption on the size of $\mathsf{PKE.C}(ek, n)$, we can argue that except with small probability, $\mathcal{A}_1$ simulates the decryption oracle correctly. The additional memory overhead for $\mathcal{A}_1$ is only a counter. Moreover, there is no increase in the running time of $\mathcal{A}_1$ unlike the adversary in Theorem 4.5.1.

Extension to \$CCA-$\diamond$, \$CCA-$\perp$. We can prove the same result for \$CCA-$\diamond$, \$CCA-$\perp$ but the adversary would not be tight with respect to running time. The adversary in these cases would pick the coins for encrypting m (to compute c_1) like the adversary in Theorem 4.5.1. This would require iterating over counters to answer decryption queries and hence lead to looseness with respect to running time. We omit the theorems for these notions because they would not involve any new ideas beyond those presented in Theorems 4.5.1 and 4.5.2.

Games $\mathsf{H}_h^b(\mathcal{A})$ for $0 \leqslant h \leqslant 3$	$\textsc{Enc}_b(m)$	$\textsc{Dec}^{\mathtt{m}}(c)$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$	$i \leftarrow i + 1$	If $M[c] \neq \bot$ then //$\mathsf{H}^b_{[0,2)}$
$i \leftarrow 0$	$c_1 \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, m)$	Return $M[c]$ //$\mathsf{H}^b_{[0,2)}$
$f \leftarrow\!\!\$\ \mathsf{Inj}^{\pm}(T, D, R)$	$c_0 \leftarrow\!\!\$\ \mathsf{PKE.C}(ek, \lvert m\rvert)$ //$\mathsf{H}^b_{[0,1)}$	If $b = 0$ and $i \geqslant 1$:
$b' \leftarrow \mathcal{A}^{\textsc{Enc}_b, \textsc{Dec}^{\mathtt{m}}}(ek)$	$c_0 \leftarrow f((\lvert m\rvert, ek), (m, i))$ //$\mathsf{H}^b_{[1,\infty)}$	$n \leftarrow \mathsf{PKE.C}^{-1}(ek, c)$
Return $b' = 1$	$M[c_b] \leftarrow m$	$(m, j) \leftarrow f^{-1}((n, ek), c)$
	Return c_b	If $m \neq \bot$ and $j \leqslant i$:
		If $M[c] = \bot$ then:
		$\mathsf{bad} \leftarrow \mathsf{true}$
		Return m //$\mathsf{H}^b_{[3,\infty)}$
		Else
		Return m
		$m \leftarrow \mathsf{PKE.D}(dk, c)$
		Return m

Figure 4.23: First set of hybrids used for proof of Theorem 4.5.2. Highlighting indicates modifications in H_0^b from $\mathsf{G}^{\$\mathsf{cca}\text{-}\mathtt{m}}_{\mathsf{PKE},b}$.

We give the formal proof of Theorem 4.5.2.

Proof. We start by considering the hybrid games H_h^b defined in Fig. 4.23. In this and future games we define T, D, and R by $T = \mathbb{N} \times \mathsf{PKE.Ek}$, $D_{n,ek} = \{0,1\}^n \times [q_{\textsc{Enc}}]$ and $R_{n,ek} = \mathsf{PKE.C}(n, ek)$ Of these we make the following claims for $b \in \{0,1\}$.

1. $\mathsf{Pr}[\mathsf{G}^{\$\mathsf{cca}\text{-}\mathtt{m}}_{\mathsf{PKE},b}] = \mathsf{Pr}[\mathsf{H}_0^b]$
2. $|\mathsf{Pr}[\mathsf{H}_0^b] - \mathsf{Pr}[\mathsf{H}_1^b]| \leqslant 0.5 \cdot q_{\textsc{Enc}}^2/2^\tau$
3. $\mathsf{Pr}[\mathsf{H}_1^b] = \mathsf{Pr}[\mathsf{H}_2^b]$
4. $|\mathsf{Pr}[\mathsf{H}_2^b] - \mathsf{Pr}[\mathsf{H}_3^b]| \leqslant 2q_{\textsc{Dec}}q_{\textsc{Enc}}/2^\tau$

Combining the above claims, we get that

$$\begin{aligned}\mathsf{Adv}^{\$\mathsf{cca}\text{-}\mathtt{m}}_{\mathsf{PKE}}(\mathcal{A}_{\mathsf{m}}) &= \mathsf{Pr}[\mathsf{G}^{\$\mathsf{cca}\text{-}\mathtt{m}}_{\mathsf{PKE},1}] - \mathsf{Pr}[\mathsf{G}^{\$\mathsf{cca}\text{-}\mathtt{m}}_{\mathsf{PKE},0}] = \mathsf{Pr}[\mathsf{H}_0^1] - \mathsf{Pr}[\mathsf{H}_0^0] \\ &\leqslant \mathsf{Pr}[\mathsf{H}_3^1] - \mathsf{Pr}[\mathsf{H}_3^0] + q_{\textsc{Enc}}^2/2^\tau + 4q_{\textsc{Enc}}q_{\textsc{Dec}}/2^\tau. \qquad (4.4)\end{aligned}$$

Next, we prove the claims.

TRANSITION TO H_0^b. The game H_0^b was copied from the game $\mathsf{G}^{\$\mathsf{cca}\text{-}\mathtt{m}}_{\mathsf{PKE},b}$ with some code added that has been highlighted (note that the entire code inside the highlighted if statement is new). We added a variable i that counts the number of ENC queries and sample a random injective function f that we will be used for future hybrids. We add an if statement in DEC that checks if $b = 0$ and $i \geqslant 1$ and if true it computes $(m, j) \leftarrow f^{-1}((n, ek), c)$ where $n \leftarrow \mathsf{PKE.C}^{-1}(ek, c)$ and then if the check $m \neq \perp$ and $j \leqslant i$ succeeds, it sets a flag bad, otherwise returns m. We note that in H_0^b, the return statement never occurs because if $M[c] \neq \perp$, we would have returned before the execution of this if statement. Hence the code inside the highlighted if statement can be ignored in H_0^b. Therefore, there is no change in behavior in H_0^b compared to $\mathsf{G}^{\$\mathsf{cca}\text{-}\mathtt{m}}_{\mathsf{PKE},b}$. It follows that $\mathsf{Pr}[\mathsf{G}^{\$\mathsf{cca}\text{-}\mathtt{m}}_{\mathsf{PKE},b}] = \mathsf{Pr}[\mathsf{H}_0^b]$.

TRANSITION FROM H_0^b TO H_1^b. In game H_1^b, we replace the random sampling of c_0 with the output of the random injective function f, using the counter i to provide domain separation between different queries. We again note that in H_1^b, we would never return anything from inside the highlighted if statement, and hence the code inside it can be ignored. In particular that means the behavior of DEC is independent of f. So this modification in how we compute c_0 changes behavior only in ENC since the values of c_0 will never repeat in H_1^b unlike H_0^b. Hence, the switching lemma (Lemma 11.4.7) gives us $|\mathsf{Pr}[\mathsf{H}_0^b] - \mathsf{Pr}[\mathsf{H}_1^b]| \leqslant 0.5 \cdot q_{\text{ENC}}^2/2^\tau$ (since the image of f always has size at least 2^τ).

TRANSITION FROM H_1^b TO H_2^b. In DEC of game H_2^b, we stop returning $M[c]$ if it is not $\perp$ at the beginning. If $b = 1$ the behavior remains the same as the highlighted if statement fails and we return $m = \mathsf{PKE.D}(dk, c)$ which we would have returned in H_1^1 (not if $M[c] \neq \perp$ then $M[c] = \mathsf{PKE.D}(dk, c)$ holds in this game). When $b = 0$ and $M[c] \neq \perp$, then both H_1^0 and H_2^0 return $M[c] = f^{-1}((n, ek), c)$. If $b = 0$ and $M[c] = \perp$, then both H_1^b and H_2^b return $m = \mathsf{PKE.D}(dk, c)$. Hence the behavior of DEC is identical in H_1^b and H_2^b. So, $\mathsf{Pr}[\mathsf{H}_1^b] = \mathsf{Pr}[\mathsf{H}_2^b]$.

TRANSITION FROM H_2^b TO H_3^b. In game H_3^b, we return m if the bad flag gets set. Note that for a DEC query on c, bad is set in H_2 only if $M[c] = \perp$ and $f^{-1}((n, ek), c) \neq \perp$. This has no effect when $b = 1$. The probability that in H_2^0 that a given DEC query has a c such that $M[c] = \perp$ and $f^{-1}((n, ek), c) \neq \perp$ is at most $q_{\text{ENC}} 2^n/(2^{n+\tau} - q_{\text{ENC}}) \leqslant 2q_{\text{ENC}}/2^\tau$. This follows because there are $q_{\text{ENC}} \cdot 2^n$ values in the domain (and hence image) of f and the view of the adversary in H_2^b is dependent only on q_{ENC} points of f which are mapped to c' satisfying $M[c'] \neq \perp$ (i.e. those returned by ENC). Taking a union bound over all DEC queries, we get that bad is set with probability at most $2q_{\text{ENC}} \cdot q_{\text{DEC}}/2^\tau$. Since H_2^b and H_3^b are identical-until-bad, using the Fundamental Lemma of Game Playing [BR06], we get, for $b \in \{0, 1\}$, $|\mathsf{Pr}[\mathsf{H}_2^b] - \mathsf{Pr}[\mathsf{H}_3^b]| \leqslant 2q_{\text{ENC}} \cdot q_{\text{DEC}}/2^\tau$.

TRANSITION TO $\mathsf{H}^{k,b}$ HYBRIDS. We next consider the hybrid games $\mathsf{H}^{k,b}$ for $(k, b) \in [q_{\text{ENC}}] \times \{0, 1\}$ defined in Fig. 4.24 which have been derived by cleaning up (removing M and bad) and modifying the code of H_3^b.

Hybrids $\mathsf{H}^{k,b}$	$\textsc{Enc}(m)$	$\textsc{Dec}(c)$
$//(k,b) \in [q_{\textsc{Enc}}] \times \{0,1\}$	$i \leftarrow i+1$	If $i \geqslant k+b$:
$i \leftarrow 0$	$c_1 \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, m)$	$n \leftarrow \mathsf{PKE.C}^{-1}(ek, c)$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$	$c_0 \leftarrow f((\lvert m\rvert, ek), (m, i))$	$(m, j) \leftarrow f^{-1}((n, ek), c)$
$f \leftarrow\!\!\$\ \mathsf{Inj}(T, D, R)$	If $i < k$: $c \leftarrow c_1$	If $m \neq \perp$ and $k+b \leqslant j \leqslant i$:
$b' \leftarrow \mathcal{A}_{\mathsf{m}}^{\textsc{Enc},\textsc{Dec}}(ek)$	If $i = k$: $c \leftarrow c_b$	Return m
Return $b' = 1$	If $i > k$: $c \leftarrow c_0$	$m \leftarrow \mathsf{PKE.D}(dk, c)$
	Return c	Return m

Figure 4.24: Second set of hybrids used for proof of Theorem 4.5.2. Highlighting indicates modifications from H_3^b.

The modified code has been highlighted in Fig. 4.24. In $\textsc{Enc}$ of $\mathsf{H}^{k,b}$, ciphertexts c_0 and c_1 are computed as in H_3^b, but $\mathsf{H}^{k,b}$ returns c_1 when $i < k$, c_0 when $i > k$, and c_b when $i = k$. Note that at the extremes ($k = 1, b = 0$ and $k = q_{\textsc{Enc}}, b = 1$) $\mathcal{A}_{\mathsf{m}}$ will either always receive c_0 or always receive c_1. The decryption queries are answered as in H_3^b with some modifications – the $b = 0$ and $i \geqslant 1$ check is modified to $i \geqslant k+b$ and the check $j \leqslant i$ is modified to $k+b \leqslant j \leqslant i$.[6] Observe that $\textsc{Dec}$ queries in $\mathsf{H}^{1,0}$ will be answered identically as in H_3^0 because the condition $i \geqslant k+b$ in $\mathsf{H}^{1,0}$ is $i \geqslant 1$ and the condition $b = 0 \wedge i \geqslant 1$ is equivalent to $i \geqslant 1$ in H_3^0, and $1 \leqslant j \leqslant i$ is equivalent to $j \leqslant i$ since $j \geqslant 1$. Similarly the $\textsc{Dec}$ queries in $\mathsf{H}^{q_{\textsc{Enc}},1}$ will be answered identically as in H_3^1 because the condition $i \geqslant q_{\textsc{Enc}} + 1$ in $\mathsf{H}^{q_{\textsc{Enc}},0}$ is always false and the condition $b = 0 \wedge i \geqslant 1$ is always false in H_3^1, and the check $k+b \leqslant j \leqslant i$ is never executed in $\mathsf{H}^{q_{\textsc{Enc}},1}$ just like the check $j \leqslant i$ in H_3^1. So $\mathsf{H}^{1,0}$ perfectly matches H_3^0 and $\mathsf{H}^{q_{\textsc{Enc}},1}$ perfectly matches H_3^1. Hence combining with (4.4), we have

$$\mathsf{Adv}^{\$\mathsf{cca}\text{-}\mathtt{m}}_{\mathsf{PKE}}(\mathcal{A}_{\mathsf{m}}) \leqslant \mathsf{Pr}[\mathsf{H}^{1,1}] - \mathsf{Pr}[\mathsf{H}^{q_{\textsc{Enc}},0}] + (q_{\textsc{Enc}}^2 + 4 \cdot q_{\textsc{Enc}} \cdot q_{\textsc{Dec}})/2^{\tau} \ . \tag{4.5}$$

In general, in $\mathsf{H}^{k,b}$ the first $k+b-1$ encryption queries use c_1 and the rest use c_0; also in $\textsc{Dec}$, the checks involve $k+b$, making them identical in $\mathsf{H}^{k,1}$ and $\mathsf{H}^{k+1,0}$, so $\mathsf{Pr}[\mathsf{H}^{k,1}] = \mathsf{Pr}[\mathsf{H}^{k+1,0}]$ holds. Hence

$$\begin{aligned} \mathsf{Pr}[\mathsf{H}^{q_{\textsc{Enc}},1}] - \mathsf{Pr}[\mathsf{H}^{1,0}] &= \mathsf{Pr}[\mathsf{H}^{q_{\textsc{Enc}},1}] - \mathsf{Pr}[\mathsf{H}^{1,0}] + \sum_{k \in [q_{\textsc{Enc}}-1]} \mathsf{Pr}[\mathsf{H}^{k,1}] - \mathsf{Pr}[\mathsf{H}^{k+1,0}] \\ &= \sum_{k \in [q_{\textsc{Enc}}]} \mathsf{Pr}[\mathsf{H}^{k,1}] - \mathsf{Pr}[\mathsf{H}^{k,0}]. \end{aligned} \tag{4.6}$$

[6] The latter check already implies former, so in future games we remove the former.

TRANSITION TO H_h HYBRIDS. Next, consider the games shown in Fig. 4.25. Of these we make the following claims.

1. $2\mathsf{Pr}[\mathsf{H}_0] - 1 = (1/q_{\text{ENC}}) \sum_{k \in [q_{\text{ENC}}]} \mathsf{Pr}[\mathsf{H}^{k,1}] - \mathsf{Pr}[\mathsf{H}^{k,0}]$
2. $\mathsf{Pr}[\mathsf{H}_0] \leqslant \mathsf{Pr}[\mathsf{H}_1] + (q_{\text{ENC}} + 2q_{\text{DEC}})/2^\tau$
3. $\mathsf{Pr}[\mathsf{H}_1] \leqslant \mathsf{Pr}[\mathsf{H}_2] + q_{\text{ENC}}/2^\tau$
4. $2\mathsf{Pr}[\mathsf{H}_2] - 1 = \mathsf{Adv}^{\$\text{cca-m}}_{\mathsf{PKE}}(\mathcal{A}_1)$

In the rest of the proof we address these one at a time. Putting them together along with (4.5) and (4.3) gives the bound claimed in the theorem statement via the following calculation, where we let $\varepsilon = (q_{\text{ENC}}^2 + 4q_{\text{ENC}}q_{\text{DEC}})/2^\tau$,

$$\begin{aligned}
\mathsf{Adv}^{\$\text{cca-m}}_{\mathsf{PKE}}(\mathcal{A}_{\mathsf{m}}) &= \varepsilon + \sum_{k \in [q_{\text{ENC}}]} \mathsf{Pr}[\mathsf{H}^{k,1}] - \mathsf{Pr}[\mathsf{H}^{k,0}] \\
&= \varepsilon + q_{\text{ENC}}(2\mathsf{Pr}[\mathsf{H}_0] - 1) \\
&\leqslant \varepsilon + q_{\text{ENC}}(2\mathsf{Pr}[\mathsf{H}_1] + (2q_{\text{ENC}} + 4q_{\text{DEC}})/2^\tau - 1) \\
&\leqslant \varepsilon + q_{\text{ENC}}(2\mathsf{Pr}[\mathsf{H}_2] + (2q_{\text{ENC}} + 4q_{\text{DEC}})/2^\tau + 2q_{\text{ENC}}/2^\tau - 1) \\
&= \mathsf{Adv}^{\$\text{cca-m}}_{\mathsf{PKE}}(\mathcal{A}_1) + \varepsilon + 4q_{\text{ENC}}q_{\text{DEC}}/2^\tau + 4q_{\text{ENC}}^2/2^\tau \\
&= \mathsf{Adv}^{\$\text{cca-m}}_{\mathsf{PKE}}(\mathcal{A}_1) + 8q_{\text{ENC}}q_{\text{DEC}}/2^\tau + 5q_{\text{ENC}}^2/2^\tau
\end{aligned}$$

TRANSITION TO H_0. Game H_0 is identical to $\mathsf{H}^{k,b}$ with (k, b) chosen at random and with the game returning true if $\mathcal{A}_{\mathsf{m}}$ correctly guessed b. We added highlighted code to ENC which stores the message and ciphertext from when $i = k$ in preparation for the next hybrid. We removed the $i \geqslant k + b$ check in DEC because it is anyway implied by the $k + b \leqslant j \leqslant i$ check. Standard calculations by conditioning on all the possible values of (k, b) gives the claim.

TRANSITION FROM H_0 TO H_1. We make three changes to DEC to transition to H_1. First, it immediately returns m^* if given $c = c^*$. Secondly, we modify the check $k + b \leqslant j \leqslant i$ to $k \leqslant j \leqslant i$. Thirdly, we add an if statement inside the if statement checking $m \neq \perp$ and $k \leqslant j \leqslant i$, which checks if $(m, j) = (m^*, k)$ and skips returning m if that is the case.

Note that if $b = 0$, a DEC query on $c = c^*$ would never exhibit different behavior in H_1 compared to H_0 because it returns m^* in both cases. A DEC query on $c \neq c^*$ would never exhibit different behavior in H_1 compared to H_0 because in H_1 the condition to skip a line never gets triggered since $f^{-1}((n, ek), c) \neq$

Hybrids H_h	$\textsc{Enc}(m)$	$\textsc{Dec}(c)$
$//0 \leqslant h \leqslant 2$	$i \leftarrow i + 1$	If $c = c^*$: Return m^* $//\mathsf{H}_{[1,\infty)}$
$(k, b) \leftarrow\!\!\$\ [q_{\textsc{Enc}}] \times \{0, 1\}$	$c_1 \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, m)$	$n \leftarrow \mathsf{PKE.C}^{-1}(ek, c)$
$i \leftarrow 0$	$c_0 \leftarrow f((\lvert m\rvert, ek), (m, i))$	$(m, j) \leftarrow f^{-1}((n, ek), c)$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$	If $i < k$:	If $m \neq \bot$ and $k + b \leqslant j \leqslant i$: $//\mathsf{H}_{[0,1)}$
$f \leftarrow\!\!\$\ \mathsf{Inj}^{\pm}(T, D, R)$	$\quad c \leftarrow c_1$	If $m \neq \bot$ and $k \leqslant j \leqslant i$: $//\mathsf{H}_{[1,\infty)}$
$b' \leftarrow \mathcal{A}_{\mathsf{m}}^{\textsc{Enc},\textsc{Dec}}(ek)$	If $i = k$:	$\quad$ If $(m, j) = (m^*, k)$: $//\mathsf{H}_{[1,\infty)}$
Return $b' = b$	$\quad c_0 \leftarrow\!\!\$\ \mathsf{PKE.C}(ek, \lvert m\rvert)$ $//\mathsf{H}_{[2,\infty)}$	$\quad\quad$ Skip next line $//\mathsf{H}_{[1,\infty)}$
	$\quad c \leftarrow c_b$	$\quad$ Return m
	$\quad (c^*, m^*) \leftarrow (c, m)$	$m \leftarrow \mathsf{PKE.D}(dk, c)$
	If $i > k$:	Return m
	$\quad c \leftarrow c_0$	
	Return c	

Figure 4.25: Final set of hybrids for the proof of Theorem 4.5.2. The highlighted line is new with respect to the code of $\mathsf{H}^{k,b}$.

$f^{-1}((n, ek), c^*) = (m^*, k)$ (because f is injective). Since $b = 0$, the change from $k + b \leqslant j$ to $k \leqslant j$ has no effect.

For $b = 1$, a $\textsc{Dec}$ query on $c = c^*$ would exhibit different behavior in H_0 than in H_1 only if $f^{-1}((n, ek), c^*) \neq \bot$ in H_1 (recall that for $b = 1$, c^* is an actual encryption of m^*). Note that, up until c^* is defined in H_0, the view of the adversary is independent of f. So, the probability that $f^{-1}((n, ek), c^*) \neq \bot$ is at most $q_{\textsc{Enc}} 2^n / 2^{n+\tau} \leqslant q_{\textsc{Enc}} / 2^{\tau}$.

For $b = 1$, a $\textsc{Dec}$ query on $c \neq c^*$ would exhibit different behavior in H_1 compared to H_0 only if $f^{-1}((n, ek), c) = (m, j)$ and $m \neq \bot$ and $k = j$ holds in H_1. Because otherwise if $m = \bot$ or $k < j$, the outer if statements would not be true in either game or if $k > j$ then the skip condition (the only difference inside the outer if statement for H_0 and H_1) would never be triggered for H_1.

For every $\textsc{Dec}(c)$ query, the probability that it was first query where c satisfies $f^{-1}((n, ek), c) = (m, k)$ is at most $2^n / (2^{n+\tau} - q_{\textsc{Enc}} - q_{\textsc{Dec}}) \leqslant 2/2^{\tau}$. This follows because there are 2^n values in the domain of f^{-1} that map to (m, k) for some m and because it is the first query where c satisfies $f^{-1}((n, ek), c) = (m, k)$, the view of the adversary is dependent only on points of f^{-1} which are mapped to $\bot$ or (m', k') where $k' \neq k$. Taking a union bound over all $\textsc{Dec}$ queries, this event happens with probability at most $2q_{\textsc{Dec}}/2^{\tau}$ (where we use the fact that $q_{\textsc{Dec}} + q_{\textsc{Enc}} \leqslant 0.5 \cdot \tau$). This gives us, $\mathsf{Pr}[\mathsf{H}_0] \leqslant \mathsf{Pr}[\mathsf{H}_1] + (q_{\textsc{Enc}} + 2q_{\textsc{Dec}})/2^{\tau}$.

TRANSITION FROM H_1 TO H_2. The only change in H_2 is that in ENC for $i = k$, c_0 sampled uniformly at random instead of evaluating f. Note that this changes behavior in ENC of H_2 only if the c_0 sampled is the same as c_0 for some previous ENC query – since the size of $\mathsf{PKE.C}$ is always at least 2^τ, this happens with probability at most $q_{\text{ENC}}/2^\tau$. For DEC queries on $c = c^*$, the behavior is identical in H_1 and H_2 because of the check at the beginning. For a DEC query on $c \neq c^*$, the answer of the query in no way would depend on c_0 sampled for $i = k$. So the behavior of DEC is unchanged. It follows that $\mathsf{Pr}[\mathsf{H}_1] \leqslant \mathsf{Pr}[\mathsf{H}_2] + q_{\text{ENC}}/2^\tau$.

ADVERSARY $\mathcal{A}_1$. Finally we can see that our adversary $\mathcal{A}_1$ (defined in Fig. 4.22) perfectly simulates the view of $\mathcal{A}_{\mathsf{m}}$ in H_3. It was obtained by copying the code of H_3 and then modifying it to query its ENC and DEC oracle as appropriate. It follows that $\mathsf{Pr}[\mathsf{H}_2] = 0.5\mathsf{Pr}[\mathsf{G}^{\$\text{cca-}\mathtt{m}}_{\mathsf{PKE},1}] + 0.5(1 - \mathsf{Pr}[\mathsf{G}^{\$\text{cca-}\mathtt{m}}_{\mathsf{PKE},0}])$. Hence, $2\mathsf{Pr}[\mathsf{H}_2] - 1 = \mathsf{Adv}^{\$\text{cca-}\mathtt{m}}_{\mathsf{PKE}}(\mathcal{A}_1)$.

Adversary $\mathcal{A}_1$'s extra running time comes from using $\mathsf{PKE.E}$ in SIMENC. Its extra memory is that required for running $\mathsf{PKE.E}$, for storing i, and for storing (c^*, m^*). ■

4.6 Extended Discussion of Oracle Adversaries

Following our convention from Section 4.2, the reduction adversaries we provide in a variety of our results are $\mathcal{F}$-oracle adversaries for different choices of $\mathcal{F}$. This is summarized by the table in Fig. 4.1. One justification for why this is acceptable is that this oracle can always be pseudorandomly instantiated, if needed (as captured by Theorem 4.2.1). In this section we will give examples to show that appropriate pseudorandom objects exist.

Result	Security Notion	Required Oracle
Thm. 4.3.1	1UFCMA	$\mathsf{Inj}^{\pm}(\mathsf{DS.M}, [q_{\text{SIGN}}], \mathsf{D.R})$
Thm. 4.3.3	1UFCMA	$\mathsf{Fcs}(\mathsf{DS.M}, [q_{\text{SIGN}}], \mathsf{D.R})$
Thm. 4.3.4	OW-RSA	$\mathsf{Fcs}(\{0,1\}^{\mathsf{R.k}}, \{0,1\}^*, \mathbb{Z}^*_{(\cdot)}) \times \mathsf{Inj}(\{0,1\}^*, [q_{\text{SIGN}}], \{0,1\}^{\mathsf{rl}})$
Thm. 4.4.1	INDR	$\mathsf{Fcs}(\mathsf{NE.N}, \mathsf{NE.C}, \{0,1\}^\tau)$
Thm. 4.5.1	1CCA-$\diamond$	$\mathsf{Fcs}(\mathbb{N}, D, \mathsf{PKE.R})$
Thm. 4.5.2	1\$CCA-$\mathtt{m}$	$\mathsf{Inj}^{\pm}(\mathbb{N} \times \mathsf{PKE.Ek}, \{0,1\}^n \times [q_{\text{ENC}}], \mathsf{PKE.C}(ek, n))$

Table 4.1: Summary of the $\mathcal{F}$-oracle adversaries obtained by the results in this chapter.

However we emphasize that our perspective is that, in general, fixing specific concrete choices for these instantiations is a secondary concern. Suppose, for example, that an efficient, low-memory adversary $\mathcal{A}$ is shown to exist against the security of a digital signature scheme DS which was proven secure with an $\mathcal{F}$-oracle reduction $\mathcal{R}[]$ to cryptographic assumption Π. A motivated cryptographer could easily put together a list

$\mathsf{F}_1, \mathsf{F}_2, \ldots$ or many candidate $\mathcal{F}$-pseudorandom functions. Either instantiating $\mathcal{R}[\mathcal{A}]$'s oracle with one of the functions on this list would give an efficient, low-memory attack against Π or $\mathcal{R}[\mathcal{A}]$ would be successfully attacking the pseudorandomness of *every* F_i on our list (which was chosen after $\mathcal{R}$ and $\mathcal{A}$ were fixed).

Consequently, our priority in this section is to highlight the existence of appropriate pseudorandom objects, not in making optimal choices for them. We optimize our choices for ease of explanation. For example, all the required tweakable random functions (Theorems 4.3.3, 4.3.4, 4.4.1, and 4.5.1) can implemented using a hash function (say SHA2 or SHA3) which is commonly modeled as a random oracle. For $\mathsf{Fcs}(T, D, R)$ given a hash function $H : \{0,1\}^* \to R$ we simply define $\mathsf{F}_K(t, d) = H(K \,\|\, \langle t, d\rangle)$ where is $\langle\cdot,\cdot\rangle$ encoding of tuples from $T \times D$. If $R = \{0,1\}^\tau$, finding such an H is straightforward. For other R of interest we expect to be able to use standard techniques to create such a H from H' with range $\{0,1\}^\tau$ (e.g. using rejection sampling, so $H(x) = H'(i, x)$ for the first $i \in [n]$ such that $H'(i, x) \in R$).

Tweakable injections, $\mathsf{Inj}^{\pm}(T, D, R)$, are needed for Theorems 4.3.1, 4.3.4, and 4.5.2. We first note that a tweakable pseudorandom injection with efficient inversion can be instantiated from blockciphers using the CMC enciphering scheme [HR03]. Since blockciphers can in turn be constructed from PRFs [LR88], the assumption of existence of tweakable pseudorandom injection with efficient inversions is essentially same as the assumption of existence of PRFs. A large, structured tweak set T can always be handled by using a collision-resistant hash function to map it to a more standard tweak set. For examples of interest, the choices of R for Theorems 4.3.1 and 4.3.4 are likely to be $\{0,1\}^{128}$ or $\{0,1\}^{256}$. Thus we can implement them by taking tweakable blockciphers obtained from standard blockciphers and restricting their domain to match D. Should Theorem 4.3.1 require a larger R we can use a large-block blockcipher such as those designed by Hoang, Krovetz, and Rogaway [HKR15].

For Theorem 4.5.2, given tweak (n, ek) we need an injection from elements of $\{0,1\}^n \times [q_{\text{ENC}}]$ to $\mathsf{PKE.C}(ek, n)$ the ciphertext space of PKE using key ek for messages of length n. How this is achieved, will of course depend on the choice of PKE. One common possibility is $\mathsf{PKE.C}(ek, n) = \mathcal{C}(ek) \times \{0,1\}^{n+\tau}$ where $\mathcal{C}$ is some structured set dependent on ek (e.g. $\mathbb{Z}_N$ for RSA) and τ is a constant (e.g. 128). If $\lceil \log q_{\text{ENC}} \rceil$ bits of a random element of $c \in \mathcal{C}(ek)$ look uniformly random (when not conditioned on the rest of c) then we can use a misuse-resistant authenticated encryption scheme with ciphertexts τ bits longer than the input message to encrypt the message $(m, i) \in \{0,1\}^n \times [q_{\text{ENC}}]$ using (n, ek) as a nonce to obtain a ciphertext of length $\lceil \log q_{\text{ENC}} \rceil + n + \tau$. Treating the first $\lceil \log q_{\text{ENC}} \rceil$ bits of the ciphertext as part of some $c \in \mathcal{C}(ek)$ we can use deterministic rejection sampling (using some PRF) to sample the rest of the bits of c.

4.7 KEM/DEM (Application Requiring AE/$CCA-$\mathtt{m}$)

In this section we exhibit reductions which are memory-tight when we use the $w = \mathtt{m}$ variant of definitions, but for which this memory-tightness does not appear possible if we use $w = \diamond$ or $\perp$. The reductions are for proving the security encryption schemes based on the KEM/DEM paradigm. In the paradigm we construct an encryption scheme KD given two encryption schemes KEM and DEM. To encrypt a message m we first sample a random key K for DEM which we encrypt using KEM. Then using DEM with key K we encrypt m. Decryption proceeds by using KEM to recover K and then using it with DEM to recover m.

We consider the cases when KEM (and hence KD) is a secret-key or public-key encryption scheme.[7] In either case we use a secret-key encryption scheme for DEM. For now let us consider the secret-key case; the public-key case is similar. Our goal is to show that if KEM is AE-w secure and DEM is AE-w secure (against multi-user attacks making one encryption query per user), then KD is AE-w secure. Our goal is for these reductions to be memory-tight.

In particular, for motivating the usefulness of -m style definition, we are most interested in the reduction to the security of KEM. In fact, the standard reduction to the security of KEM works and is memory-tight when $w = \mathtt{m}$. (The reduction to the security of DEM will require non-standard steps making use of our efficient tagging technique.) It seems unlikely the reduction to KEM's security can be made memory-tight if $w \in \{\perp, \diamond\}$ instead.

To understand the issue at hand, let us discuss how the proof works at a high level. Broadly, we use a reduction to the security of KEM to switch ciphertexts encrypting K to be random so that K itself is random from the perspective of the adversary. With these random K we can then apply the security of DEM. Our focus is on the memory-tightness of the first step. Given adversary $\mathcal{A}_{\mathsf{kd}}$ against KD we build $\mathcal{A}_{\mathsf{k}}$ against security of KEM as following. On an encryption query for m, adversary $\mathcal{A}_{\mathsf{k}}$ will sample a random K and then ask its own encryption oracle for an encryption c_{k} of K and locally use K to encrypt m into the ciphertext c_{d}. It returns $(c_{\mathsf{k}}, c_{\mathsf{d}})$ to $\mathcal{A}_{\mathsf{kd}}$. The interesting challenge is in simulating decryption queries for some $(c_{\mathsf{k}}, c_{\mathsf{d}})$. When $w = \mathtt{m}$, adversary $\mathcal{A}_{\mathsf{k}}$ can query c_{k} to its own decryption oracle to receive some K and use that to decrypt c_{d}. For $w \in \{\perp, \diamond\}$, this works fine unless c_{k} was previously returned by an encryption query in which case $\mathcal{A}_{\mathsf{k}}$'s oracle returns $\perp$ or $\diamond$ (as appropriate) which does not enable $\mathcal{A}_{\mathsf{k}}$ to simulate decryption properly if c_{d} was not also returned by that encryption query. As such, it is not clear how $\mathcal{A}_{\mathsf{k}}$ could respond correctly without storing all prior encryption queries.

The step described above is main motivation of this section, showing how $w = \mathtt{m}$ can be useful for security proofs. The second step of the proof also required some care to be memory-tight. The natural reduction to

[7] The public-key case is a standard "textbook" construction of an encryption scheme. The secret-key case also arises in practice where a master key is used to encryption subkeys which are used to encryption the message. See, for example, some key-rotation schemes [EPRS17].

$\underline{\mathsf{KD}[\mathsf{NE}_\mathsf{k}, \mathsf{NE}_\mathsf{d}].\mathsf{K}}$	$\underline{\mathsf{KD}[\mathsf{NE}_\mathsf{k}, \mathsf{NE}_\mathsf{d}].\mathsf{E}(K_\mathsf{k}, n, m)}$	$\underline{\mathsf{KD}[\mathsf{NE}_\mathsf{k}, \mathsf{NE}_\mathsf{d}].\mathsf{D}(K_\mathsf{k}, n, c)}$
$K_\mathsf{k} \xleftarrow{\$} \mathsf{NE}_\mathsf{k}.\mathsf{K}$	$K \xleftarrow{\$} \mathsf{NE}_\mathsf{d}.\mathsf{K}$	$(c_\mathsf{k}, c_\mathsf{d}) \leftarrow c$
Return K	$c_\mathsf{k} \leftarrow \mathsf{NE}_\mathsf{k}.\mathsf{E}(K_\mathsf{k}, n, K)$	$K \leftarrow \mathsf{NE}_\mathsf{k}.\mathsf{D}(K_\mathsf{k}, n, c_\mathsf{k})$
	$c_\mathsf{d} \leftarrow \mathsf{NE}_\mathsf{d}.\mathsf{E}(K, 0, m)$	If $K \neq \bot$:
	Return $(c_\mathsf{k}, c_\mathsf{d})$	Return $\mathsf{NE}_\mathsf{d}.\mathsf{D}(K, 0, c_\mathsf{d})$
		Return $\bot$

Figure 4.26: Nonce-based AE scheme $\mathsf{KD}[\mathsf{NE}_\mathsf{k}, \mathsf{NE}_\mathsf{d}]$ constructed from nonce-based AE schemes NE_k and NE_d via the KEM/DEM paradigm.

the multi-user security of DEM works as follows. For an encryption query it samples a random ciphertext for the KEM, samples key K at random, then queries its oracles to create a new user for DEM and have that user encrypt K. On a decryption query $(c_\mathsf{k}, c_\mathsf{d})$, if c_k was a ciphertext it picked randomly for a prior encryption query, it wants to query its decryption oracle to get c_d decrypted. This requires remembering the name of the new user created during the encryption query that returned c_k and would thus not be memory-tight. We use our message encoding technique to make this memory-tight. In particular, to respond to encryption queries we create c_k as the output of an injection applied to the name of the new user being created which allows us to later recover this when responding to decryption queries.

In Section 4.7.1 we formalize the proof when KEM is a secret-key, nonce-based scheme and in Section 4.7.2 we do the same for when it is a public-key scheme.

4.7.1 Secret-key KEM/DEM

KEM/DEM Scheme. Let NE_k and NE_d be nonce-based encryption schemes. Then we define the KEM/DEM nonce-based encryption scheme $\mathsf{KD}[\mathsf{NE}_\mathsf{k}, \mathsf{NE}_\mathsf{d}]$ as shown in Fig. 4.26. Technically KD does not meet our syntax for nonce-based encryption from Section 4.4, due to being randomized. Requiring nonce-based encryption be deterministic is not actually important for our purposes, so we ignore this technicality.

Multi-user AE security. For our proof we will require that NE_d provide multi-user security against attacks making one encryption query. So we require an extension of the notion of AE security to the multi-user setting. The multi-user setting allows the adversary to make encryption and decryption queries for multiple keys. We use the games in Fig. 4.27 to define the multi-user AE (muAE) security. For $w \in \{\mathtt{m}, \diamond, \bot\}$, the advantage against multi-user AE security of a scheme NE is defined as $\mathsf{Adv}^{\mathsf{mu\text{-}ae\text{-}}w}_{\mathsf{NE}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{\mathsf{mu\text{-}ae\text{-}}w}_{\mathsf{NE},1}(\mathcal{A})] - \mathsf{Pr}[\mathsf{G}^{\mathsf{mu\text{-}ae\text{-}}w}_{\mathsf{NE},0}(\mathcal{A})]$.

$\underline{\text{Game } \mathsf{G}^{\mathsf{mu\text{-}ae\text{-}}w}_{\mathsf{NE},b}(\mathcal{A})}$	$\underline{\text{Enc}_b(i,n,m)}$	$\underline{\text{Dec}^w_b(i,n,c)}$
$u \leftarrow 0$	$c_1 \leftarrow \mathsf{NE.E}(K_i,n,m)$	If $M[i,n,c] \neq \bot$
$b' \leftarrow \mathcal{A}^{\text{New},\text{Enc}_b,\text{Dec}^w_b}$	$c_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{NE.cl}(\lvert m\rvert)}$	Return $M[i,n,c]$ if $w = \mathtt{m}$
Return $b' = 1$	$M[i,n,c_b] \leftarrow m$	Return $\diamond$ if $w = \diamond$
$\underline{\text{New}()}$	Return c_b	Return $\bot$ if $w = \bot$
$u \leftarrow u + 1$		$m_1 \leftarrow \mathsf{NE.D}(K_i,n,c)$
$K_u \leftarrow\!\!\$\ \mathsf{NE.K}$		$m_0 \leftarrow \bot$
		Return m_b

Figure 4.27: Game defining multi-user AE security

Security result. The following theorem captures our result, that $\mathsf{KD}[\mathsf{NE_k},\mathsf{NE_d}]$ can be proven AE-$\mathtt{m}$ secure from the AE-$\mathtt{m}$ security of $\mathsf{NE_k}$ and the muAE-$\mathtt{m}$ security of $\mathsf{NE_d}$, where both reductions are memory-tight.

Theorem 4.7.1. *Let* $\mathsf{NE_k}$, $\mathsf{NE_d}$ *be nonce-based encryption schemes and let* $\tau = \mathsf{NE_k.cl}(\mathsf{NE_d.kl}) > 0$. *Let* $\mathcal{A}_a$ *be an* AE-$\mathtt{m}$ *adversary with* $\mathbf{Query}(\mathcal{A}_a) = (q_{\text{Enc}}, q_{\text{Dec}})$. *Define* $(T,D,R) = (\mathsf{NE_k.N}, [q_{\text{Enc}}], \{0,1\}^\tau)$. *Let* $\mathcal{B}_a$ *be as defined in Fig. 4.29 and* $\mathcal{C}_a$ *be the* $\mathsf{Inj}^{\pm}(T,D,R)$*-oracle adversary defined in Fig. 4.31. Then,*

$$\mathsf{Adv}^{\mathsf{ae\text{-}}\mathtt{m}}_{\mathsf{KD}[\mathsf{NE_k},\mathsf{NE_d}]}(\mathcal{A}_a) \leqslant \mathsf{Adv}^{\mathsf{ae\text{-}}\mathtt{m}}_{\mathsf{NE_k}}(\mathcal{B}_a) + \mathsf{Adv}^{\mathsf{mu\text{-}ae\text{-}}\mathtt{m}}_{\mathsf{NE_d}}(\mathcal{C}_a) + q_{\text{Enc}}(q_{\text{Enc}} + 4q_{\text{Dec}})/2^\tau \ .$$

$$\mathbf{Query}(\mathcal{B}_a) = (q_{\text{Enc}}, q_{\text{Dec}}) \qquad \mathbf{Query}(\mathcal{C}_a) = (q_{\text{Enc}}, q_{\text{Dec}})$$

$$\mathbf{Time}(\mathcal{B}_a) = \mathbf{Time}(\mathcal{A}_a) + (q_{\text{Enc}} + q_{\text{Dec}})\mathbf{Time}(\mathsf{NE_d}) \qquad \mathbf{Time}^*(\mathcal{C}_a) = \mathbf{Time}(\mathcal{A}_a)$$

$$\mathbf{Mem}(\mathcal{B}_a) = \mathbf{Mem}(\mathcal{A}_a) + \mathbf{Mem}(\mathsf{NE_d}). \qquad \mathbf{Mem}^*(\mathcal{C}_a) = \mathbf{Mem}(\mathcal{A}_a).$$

Proof. We consider the hybrids H_0 through H_3 and L_0 through L_4 defined in Figs. 4.28 and 4.30. Of these hybrids we will make the following claims, which establish the claimed upper bound on the advantage of $\mathcal{A}_a$.

1. $\mathsf{Pr}[\mathsf{G}^{\mathsf{ae\text{-}}\mathtt{m}}_{\mathsf{KD}[\mathsf{NE_k},\mathsf{NE_d}],1}(\mathcal{A}_a)] = \mathsf{Pr}[\mathsf{H}_0]$
2. $\mathsf{Pr}[\mathsf{H}_0] \leqslant \mathsf{Pr}[\mathsf{H}_1] + \mathsf{Adv}^{\mathsf{ae\text{-}}\mathtt{m}}_{\mathsf{NE_k}}(\mathcal{B}_a)$
3. $\mathsf{Pr}[\mathsf{H}_1] \leqslant \mathsf{Pr}[\mathsf{H}_2] + 0.5 \cdot q^2_{\text{Enc}}/2^\tau$
4. $\mathsf{Pr}[\mathsf{H}_2] = \mathsf{Pr}[\mathsf{H}_3]$
5. $\mathsf{Pr}[\mathsf{H}_3] \leqslant \mathsf{Pr}[\mathsf{L}_0] + 2q_{\text{Enc}}q_{\text{Dec}}/2^\tau$
6. $\mathsf{Pr}[\mathsf{L}_0] \leqslant \mathsf{Pr}[\mathsf{L}_1] + \mathsf{Adv}^{\mathsf{mu\text{-}ae\text{-}}\mathtt{m}}_{\mathsf{NE_d}}(\mathcal{C}_a)$
7. $\mathsf{Pr}[\mathsf{L}_1] \leqslant \mathsf{Pr}[\mathsf{L}_2] + 2q_{\text{Enc}}q_{\text{Dec}}/2^\tau$
8. $\mathsf{Pr}[\mathsf{L}_2] \leqslant \mathsf{Pr}[\mathsf{L}_3] + 0.5 \cdot q^2_{\text{Enc}}/2^\tau$
9. $\mathsf{Pr}[\mathsf{L}_3] = \mathsf{Pr}[\mathsf{G}^{\mathsf{ae\text{-}}\mathtt{m}}_{\mathsf{KD}[\mathsf{NE_k},\mathsf{NE_d}],0}(\mathcal{A}_a)]$

Transition to H_0. We claim that H_0 is identical to $\mathsf{G}^{\mathsf{ae\text{-}}\mathtt{m}}_{\mathsf{KD}[\mathsf{NE_k},\mathsf{NE_d}],1}$. Note that, in the latter, Enc produces honest encryptions using KD and Dec produces honest decryptions. It is immediately clear that the same holds for Enc in H_0 and the else branch of Dec in H_0. Consider the first branch in Dec. We have used

Hybrids H_h for $0 \leqslant h \leqslant 3$	$\textsc{Enc}(n, m)$	$\textsc{Dec}(n, c)$
$f \leftarrow\!\!\$\ \mathsf{Inj}(T, D, R)$ //$\mathsf{H}_{[2,\infty)}$	$u \leftarrow u + 1$ //$\mathsf{H}_{[2,\infty)}$	$(c_\mathsf{k}, c_\mathsf{d}) \leftarrow c$
$l \leftarrow \mathbb{I}$	$K \leftarrow\!\!\$\ \mathsf{NE_d.K}$	If $Y[l(n, c_\mathsf{k})] \neq \bot$:
$u \leftarrow 0$ //$\mathsf{H}_{[2,\infty)}$	$c_\mathsf{k} \leftarrow \mathsf{NE_k.E}(K_\mathsf{k}, n, K)$ //$\mathsf{H}_{[0,1)}$	If $M[l(n, c_\mathsf{k}), c_\mathsf{d}] \neq \bot$: //$\mathsf{H}_{[3,\infty)}$
$K_\mathsf{k} \leftarrow\!\!\$\ \mathsf{NE_k.K}$	$c_\mathsf{k} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{NE_k.cl}(\lvert K\rvert)}$ //$\mathsf{H}_{[1,2)}$	Return $M[l(n, c_\mathsf{k}), c_\mathsf{d}]$ //$\mathsf{H}_{[3,\infty)}$
$b' \leftarrow \mathcal{A}_a^{\textsc{Enc},\textsc{Dec}}$	$c_\mathsf{k} \leftarrow f(n, u)$ //$\mathsf{H}_{[2,\infty)}$	$K \leftarrow Y[l(n, c_\mathsf{k})]$
Return $b' = 1$	$c_\mathsf{d} \leftarrow \mathsf{NE_d.E}(K, 0, m)$	Return $\mathsf{NE_d.D}(K, 0, c_\mathsf{d})$
	$Y[l(n, c_\mathsf{k})] \leftarrow K$	Else:
	$M[l(n, c_\mathsf{k}), c_\mathsf{d}] \leftarrow m$ //$\mathsf{H}_{[3,\infty)}$	$K \leftarrow \mathsf{NE_k.D}(K_\mathsf{k}, n, c_\mathsf{k})$ //$\mathsf{H}_{[0,1)}$
	Return $(c_\mathsf{k}, c_\mathsf{d})$	$K \leftarrow \bot$ //$\mathsf{H}_{[1,\infty)}$
		If $K \neq \bot$:
		Return $\mathsf{NE_d.D}(K, 0, c_\mathsf{d})$
		Return $\bot$

Figure 4.28: First set of hybrids used for proof of Theorem 4.7.1. $\mathbb{I}$ denotes the identity function.

grey highlighting to indicate the relevant code. Note that l is the identity function (its presence will be notationally convenient for future game transitions). Y is a table indexed by n, c_k pairs which stores the key in encrypted c_k. The use of Y in $\textsc{Dec}$ simply recovers this K without decrypting c_k. By the correctness of $\mathsf{NE_k}$ this is identical to having done the decryption, so $\mathsf{Pr}[\mathsf{G}^{\mathsf{ae\text{-}m}}_{\mathsf{KD}[\mathsf{NE_k},\mathsf{NE_d}],1}(\mathcal{A}_a)] = \mathsf{Pr}[\mathsf{H}_0]$ follows.

Transition H_0 to H_1. In hybrid H_1, the ciphertext c_k is now sampled uniformly at random. Also in $\textsc{Dec}$, the key K is assigned the value $\bot$ if the $Y[n, c_\mathsf{k}] = \bot$. These difference correspond to what we expect from the security of $\mathsf{NE_k}$. We define $\mathcal{B}_a$ in Fig. 4.29 from hybrid H_1 by replacing encryption and decryption of K with appropriate queries to its oracle. This adversary is nonce-respecting because $\mathcal{A}_a$ is. Note that the use of table Y in the hybrids matches the table from $\mathsf{G}^{\mathsf{ae\text{-}m}}$. We can see that when interacting with $\mathsf{G}^{\mathsf{ae\text{-}m}}_{\mathsf{NE_k},1}$, $\mathcal{B}_a$ simulates H_0 to $\mathcal{A}_a$ and when interacting with $\mathsf{G}^{\mathsf{ae\text{-}m}}_{\mathsf{NE_k},0}$, $\mathcal{B}_a$ simulates H_1 to $\mathcal{A}_a$. Hence $\mathsf{Pr}[\mathsf{H}_0] = \mathsf{Pr}[\mathsf{H}_1] + \mathsf{Adv}^{\mathsf{ae\text{-}m}}_{\mathsf{NE_k}}(\mathcal{B}_a)$.

After this transition the else branch in $\textsc{Dec}$ is dead code. We remove it later when transitioning to L_0.

Transition H_1 to H_2. In hybrid H_2, instead of sampling c_k at random, we assign it the output of a random injective function f applied to nonce n and a counter u (we will later make u correspond to users in $\mathsf{G}^{\mathsf{mu\text{-}ae\text{-}}w}$). The switching lemma tells us that $\mathsf{Pr}[\mathsf{H}_2] \leqslant \mathsf{Pr}[\mathsf{H}_1] + 0.5 \cdot q_{\textsc{Enc}}^2/2^\tau$.

Transition H_2 to H_3. In hybrid H_3, we introduce a table M which is indexed by n, c_k, c_d (the first two via l) and stores the value of m whose encryption under $K = Y[l(n, c_\mathsf{k})]$ is c_d. This table is used in $\textsc{Dec}$ to skip

Adversary $\mathcal{B}_a^{\text{Enc},\text{Dec}}$	$\text{SimEnc}(n, m)$	$\text{SimDec}(n, c)$
$b' \leftarrow \mathcal{A}_a^{\text{SimEnc},\text{SimDec}}$	$K \leftarrow\!\!\$\ \mathsf{NE_d.K}$	$(c_\mathsf{k}, c_\mathsf{d}) \leftarrow c$
Return b'	$c_\mathsf{k} \leftarrow \text{Enc}(n, K)$	$K \leftarrow \text{Dec}(n, c_\mathsf{k})$
	$c_\mathsf{d} \leftarrow \mathsf{NE_d.E}(K, 0, m)$	If $K \neq \bot$:
	Return $(c_\mathsf{k}, c_\mathsf{d})$	Return $\mathsf{NE_d.D}(K, 0, c_\mathsf{d})$
		Return $\bot$

Figure 4.29: Adversary $\mathcal{B}_a$ for Theorem 4.7.1.

Hybrids L_ℓ for $0 \leqslant \ell \leqslant 3$	$\text{Enc}(n, m)$	$\text{Dec}(n, c)$
$f \leftarrow\!\!\$\ \mathsf{Inj}^{\pm}(T, D, R)$ //$\mathsf{L}_{[0,3)}$	$u \leftarrow u + 1$ //$\mathsf{L}_{[0,3)}$	$(c_\mathsf{k}, c_\mathsf{d}) \leftarrow c$
$l \leftarrow f^{-1}$ //$\mathsf{L}_{[0,2)}$	$K \leftarrow\!\!\$\ \mathsf{NE_d.K}$	If $1 \leqslant f^{-1}(n, c_\mathsf{k}) \leqslant u$: //$\mathsf{L}_{[0,2)}$
$l \leftarrow \mathbb{I}$ //$\mathsf{L}_{[2,\infty)}$	$c_\mathsf{k} \leftarrow f(n, u)$ //$\mathsf{L}_{[0,3)}$	If $Y[l(n, c_\mathsf{k})] \neq \bot$: //$\mathsf{L}_{[2,\infty)}$
$u \leftarrow 0$ //$\mathsf{L}_{[0,3)}$	$c_\mathsf{k} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{NE_k.cl}(\lvert K\rvert)}$ //$\mathsf{L}_{[3,\infty)}$	If $M[l(n, c_\mathsf{k}), c_\mathsf{d}] \neq \bot$:
$K_\mathsf{k} \leftarrow\!\!\$\ \mathsf{NE_k.K}$	$c_\mathsf{d} \leftarrow \mathsf{NE_d.E}(K, 0, m)$ //$\mathsf{L}_{[0,1)}$	Return $M[l(n, c_\mathsf{k}), c_\mathsf{d}]$
$b' \leftarrow \mathcal{A}_a^{\text{Enc},\text{Dec}}$	$c_\mathsf{d} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{NE_d.cl}(\lvert m\rvert)}$ //$\mathsf{L}_{[1,\infty)}$	$K \leftarrow Y[l(n, c_\mathsf{k})]$ //$\mathsf{L}_{[0,1)}$
Return $b' = 1$	$Y[l(n, c_\mathsf{k})] \leftarrow K$	Return $\mathsf{NE_d.D}(K, 0, c_\mathsf{d})$ //$\mathsf{L}_{[0,1)}$
	$M[l(n, c_\mathsf{k}), c_\mathsf{d}] \leftarrow m$	Return $\bot$
	Return $(c_\mathsf{k}, c_\mathsf{d})$	

Figure 4.30: Second set of hybrids used for proof of Theorem 4.7.1.

the step of decrypting c_d. By the correctness of $\mathsf{NE_d}$ this does not change functionality, so $\mathsf{Pr}[\mathsf{H}_2] = \mathsf{Pr}[\mathsf{H}_3]$.

Transition H_3 to L_0. Next we transition to hybrid L_0 shown in Fig. 4.30. We have highlighted all ways that this differs from hybrid H_3 (other than the elimination of the aforementioned dead code in the else branch of Dec). Our changes were twofold, consisting of switching the function l indexing into our tables to f^{-1} and switching the if condition in Dec. Considering the latter first, note that the checks $Y[l(n, c_\mathsf{k})] \neq \bot$ in H_3 and $1 \leqslant f^{-1}(n, c_\mathsf{k}) \leqslant u$ in L_0 can only differ if the second is true and the first is false. This requires the adversary to guess something in the image of $f(n, \cdot)$ other than the (at most) one example it can obtain from Enc. Switching to using $l = f^{-1}$ similarly can only change behavior if a $\text{Dec}(n, (c_\mathsf{k}, c_\mathsf{d}))$ query is made where c_k is in the image of $f(n, \cdot)$, but c_k was not returned in a prior Enc query with n. Note that the domain (and hence the image) of $f(n, \cdot)$ has size q_{Enc} and its range has size 2^τ. So for a given decryption

Adversary $\mathcal{C}_a^{\text{New},\text{Enc},\text{Dec}}$	$\text{SimEnc}(n, m)$	$\text{SimDec}(n, c)$
$f \leftarrow\!\!\$\ \mathsf{Inj}(T, D, R)$	$u \leftarrow u + 1$	$(c_{\mathsf{k}}, c_{\mathsf{d}}) \leftarrow c$
$u \leftarrow 0$	$\text{New}()$	$i \leftarrow f^{-1}(n, c_{\mathsf{k}})$
$b' \leftarrow \mathcal{A}_a^{\text{SimEnc},\text{SimDec}}$	$c_{\mathsf{k}} \leftarrow f(n, u)$	If $1 \leqslant i \leqslant u$:
Return b'	$c_{\mathsf{d}} \leftarrow \text{Enc}(u, 0, m)$	Return $\text{Dec}(i, 0, c_{\mathsf{d}})$
	Return $(c_{\mathsf{k}}, c_{\mathsf{d}})$	Return $\perp$

Figure 4.31: Adversary $\mathcal{C}_a$ for Theorem 4.7.1.

query this bad event happens in H_3 with probability at most $q_{\text{Enc}}/(2^\tau - 1) \leqslant 2q_{\text{Enc}}/2^\tau$. Taking a union bound over all Dec queries gives that $\mathsf{Pr}[\mathsf{H}_3] \leqslant \mathsf{Pr}[\mathsf{L}_0] + 2q_{\text{Enc}}q_{\text{Dec}}/2^\tau$.

Transition L_0 to L_1. In hybrid L_1, the ciphertext c_{d} is now sampled uniformly at random. Also in Dec, if the $M[l(n, c_{\mathsf{k}}), c_{\mathsf{d}}] = \perp$ then the oracle always returns $\perp$. These difference correspond to what we expect from the multi-user security of NE_{d}. We define $\mathcal{C}_a$ in Fig. 4.31 from hybrid L_1 by replacing encryption and decryption of c_{d} with appropriate queries to its oracle. We claim that when interacting with $\mathsf{G}^{\mathtt{mu\text{-}ae\text{-}m}}_{\mathsf{NE},1}$, $\mathcal{C}_a$ simulates L_0 to $\mathcal{A}_a$ and when interacting with $\mathsf{G}^{\mathtt{mu\text{-}ae\text{-}m}}_{\mathsf{NE},0}$, $\mathcal{C}_a$ simulates L_1 to $\mathcal{A}_a$. To see this, note that the variable u in these hybrids matches the same variable in $\mathsf{G}^{\mathtt{mu\text{-}ae\text{-}m}}$. Additionally, if $i = l(n, c_{\mathsf{k}})$ then the values $Y[i]$ and $M[i, c_{\mathsf{d}}]$ in these hybrids always match K_i and $M[i, 0, c_{\mathsf{d}}]$ in $\mathsf{G}^{\mathtt{mu\text{-}ae\text{-}m}}$. (The claims can be rigorously verified by plugging the code of $\mathsf{G}^{\mathtt{mu\text{-}ae\text{-}m}}_{\mathsf{NE},b}$ into $\mathcal{C}_a$ and comparing side-by-side with L_{1-b}.) This gives $\mathsf{Pr}[\mathsf{L}_1] \leqslant \mathsf{Pr}[\mathsf{L}_0] + \mathsf{Adv}^{\mathtt{mu\text{-}ae\text{-}m}}_{\mathsf{NE}}(\mathcal{C}_a)$.

Transition L_1 to L_2. In hybrid L_2, we undo the code changes used to transition from H_3 to L_0. Namely, l is set back to $\mathbb{I}$ and the if statement in Dec is reverted. By the same logic as that prior transition, $\mathsf{Pr}[\mathsf{L}_2] \leqslant \mathsf{Pr}[\mathsf{L}_1] + 2q_{\text{Enc}}q_{\text{Dec}}/2^\tau$.

Transition L_2 to L_3. In hybrid L_3, we undo the code changes used to transition from H_1 to H_2. Namely, we get rid of the injection f and sample c_{k} uniformly. By the same logic as that prior transition, $\mathsf{Pr}[\mathsf{L}_3] \leqslant \mathsf{Pr}[\mathsf{L}_2] + 0.5 \cdot q_{\text{Enc}}^2/2^\tau$.

Final Transition. Finally, we claim that L_3 and $\mathsf{G}^{\mathtt{ae\text{-}m}}_{\mathsf{KD}[\mathsf{NE}_{\mathsf{k}},\mathsf{NE}_{\mathsf{d}}],0}$ are identical. Recall that in the latter, Enc always returns random ciphertexts. The same holds in L_3. Similarly, the two games have the same behavior in only responding with non-$\perp$ values to Dec queries for ciphertexts that were trivially forwarded from earlier Enc queries. In particular, we can see this by noting that $M[l(n, c_{\mathsf{k}}), c_{\mathsf{d}}]$ in L_3 always has the same value as $M[n, (c_{\mathsf{k}}, c_{\mathsf{d}})]$ in $\mathsf{G}^{\mathtt{ae\text{-}m}}_{\mathsf{KD}[\mathsf{NE}_{\mathsf{k}},\mathsf{NE}_{\mathsf{d}}],0}$ and that $M[l(n, c_{\mathsf{k}}), c_{\mathsf{d}}] \neq \perp$ implies $Y[l(n, c_{\mathsf{k}})] \neq \perp$.

$\underline{\text{Game } \mathsf{G}^{\mathsf{mu\text{-}\$cca\text{-}}w}_{\mathsf{NE},b}(\mathcal{A})}$	$\underline{\textsc{Enc}_b(i,n,m)}$	$\underline{\textsc{Dec}^w_b(i,n,c)}$
$u \leftarrow 0$	$c_1 \leftarrow \mathsf{NE.E}(K_i, n, m)$	If $M[i,n,c] \neq \bot$
$b' \leftarrow \mathcal{A}^{\textsc{New},\textsc{Enc}_b,\textsc{Dec}^w_b}$	$c_0 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{NE.cl}(\lvert m\rvert)}$	Return $M[i,n,c]$ if $w = \mathtt{m}$
Return $b' = 1$	$M[i,n,c_b] \leftarrow m$	Return $\diamond$ if $w = \diamond$
$\underline{\textsc{New}()}$	Return c_b	Return $\bot$ if $w = \bot$
$u \leftarrow u + 1$		$m \leftarrow \mathsf{NE.D}(K_i, n, c)$
$K_u \leftarrow\!\!\$\ \mathsf{NE.K}$		Return m

Figure 4.32: Game defining multi-user $CCA security of nonce-based encryption.

■

4.7.2 Public-key KEM/DEM

KEM/DEM SCHEME. Let NE be a nonce-based encryption scheme. Let PKE be a public-key encryption scheme. Then we define the KEM/DEM public-key encryption scheme KD[PKE, NE] as shown in Fig. 4.33.

MULTI-USER $CCA SECURITY OF NE. For our proof we will again require that NE provide multi-user security against attacks making one encryption query. For this proof we will use a $CCA-style definition rather than AE. We use the games in Fig. 4.32 to define the multi-user $CCA (mu$CCA) security of NE. If differs from muAE only in that the decryption oracle always returns honest decryptions of ciphertexts that were not trivially forwarded. For $w \in \{\mathtt{m}, \diamond, \bot\}$, the advantage against multi-user AE security of a scheme NE is defined as $\mathsf{Adv}^{\mathsf{mu\text{-}\$cca\text{-}}w}_{\mathsf{NE}}(\mathcal{A}) = \mathsf{Pr}[\mathsf{G}^{\mathsf{mu\text{-}\$cca\text{-}}w}_{\mathsf{NE},1}(\mathcal{A})] - \mathsf{Pr}[\mathsf{G}^{\mathsf{mu\text{-}ae\text{-}}w}_{\mathsf{NE},0}(\mathcal{A})]$.

UNIFORMITY OF PKE. For our proof we will additionally need to make the statistical assumption that the distribution of sampling ciphertexts randomly from $\mathsf{PKE.C}(ek, \mathsf{NE.kl})$ is close (in statistical distance) to the distribution obtained by encrypting a random K. Formally, we say that PKE is (κ, ε)-*uniform* if for all $(ek, dk) \in \mathsf{PKE.K}$ and all (not necessarily efficient) $\mathcal{A}$ it holds that

$$\mathsf{Pr}[\mathcal{A}(c) = 1 : c \leftarrow\!\!\$\ \mathsf{PKE.C}(ek, \kappa)] - \mathsf{Pr}[\mathcal{A}(c) = 1 : K \leftarrow\!\!\$\ \{0,1\}^\kappa; c \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, K)] \leqslant \varepsilon.$$

As one example, this will hold with $\varepsilon = 0$ for schemes in which $\mathsf{PKE.E}(ek, K; \cdot)$ is injective for all $K \in \{0,1\}^\kappa$ and $\mathsf{PKE.C}(ek, \kappa) = \{\mathsf{PKE.E}(ek, K; r)\}_{K,r}$.

SECURITY RESULT. The following theorem captures that KD[PKE, NE] can be proven $CCA-m secure from the security of PKE and NE, where all reductions are memory-tight.

$\underline{\mathsf{KD}[\mathsf{PKE},\mathsf{NE}].\mathsf{K}}$	$\underline{\mathsf{KD}[\mathsf{PKE},\mathsf{NE}].\mathsf{E}(ek,m)}$	$\underline{\mathsf{KD}[\mathsf{PKE},\mathsf{NE}].\mathsf{D}(dk,c)}$
$(ek,dk) \xleftarrow{\$} \mathsf{PKE.K}$	$K \xleftarrow{\$} \mathsf{NE.K}$	$(c_{\mathsf{k}}, c_{\mathsf{d}}) \leftarrow c$
Return (ek,dk)	$c_{\mathsf{k}} \xleftarrow{\$} \mathsf{PKE.E}(ek,K)$	$K \leftarrow \mathsf{PKE.D}(dk, c_{\mathsf{k}})$
	$c_{\mathsf{d}} \leftarrow \mathsf{NE.E}(K,0,m)$	If $K \neq \bot$:
	Return $(c_{\mathsf{k}}, c_{\mathsf{d}})$	Return $\mathsf{NE.D}(K,0,c_{\mathsf{d}})$
		Return $\bot$

Figure 4.33: A public-key scheme $\mathsf{KD}[\mathsf{PKE},\mathsf{NE}]$ constructed from a public key encryption scheme PKE and a nonce-based encryption scheme NE.

Theorem 4.7.2. *Let* NE *be a nonce-based encryption scheme with* $\mathsf{NE.cl}(n) \geqslant n + \mathsf{NE.xl}$ *for all* n *and let* PKE *be a* $(\mathsf{NE.kl}, \varepsilon)$*-uniform public key encryption scheme. Let* τ *satisfy* $|\mathsf{PKE.C}(ek, \mathsf{NE.kl})| \geqslant 2^{\tau}$ *for all* ek *and* $\mathsf{NE.xl} \geqslant \tau$*. Let* $\mathcal{A}_c$ *be an* \$CCA-m *adversary with* $\mathbf{Query}(\mathcal{A}_c) = (q_{\mathrm{ENC}}, q_{\mathrm{DEC}})$ *and assume* $q_{\mathrm{ENC}} \leqslant 0.5 \cdot 2^{\tau}$*. Define* $T = \mathsf{PKE.Ek}$ *and* $(D_{ek}, R_{ek}) = ([q_{\mathrm{ENC}}], \mathsf{PKE.C}(ek, \mathsf{NE.kl}))$ *for* $ek \in T$*. Define* $T' = \{0,1\}^* \times \mathbb{N}$*,* $D'_{(\mathsf{c_k},l)} = \{0,1\}^l$*, and* $R'_{(\mathsf{c_k},l)} = \{0,1\}^{\mathsf{NE.cl}(l)}$*. Let* $\mathcal{B}_c$ *be as defined in Fig. 4.35, let* $\mathcal{C}_a$ *be the* $\mathsf{Inj}^{\pm}(T,D,R)$*-oracle adversary defined in Fig. 4.37, and let* $\mathcal{E}_c$ *be the* $\mathsf{Inj}^{\pm}(T',D',R')$*-oracle adversary defined in Fig. 4.39. Then,*

$$\mathsf{Adv}^{\$\mathsf{cca}\text{-}m}_{\mathsf{KD}[\mathsf{PKE},\mathsf{NE}]}(\mathcal{A}_c) \leqslant \mathsf{Adv}^{\$\mathsf{cca}\text{-}m}_{\mathsf{PKE}}(\mathcal{B}_c) + \mathsf{Adv}^{\mathsf{mu}\text{-}\$\mathsf{cca}\text{-}m}_{\mathsf{NE}}(\mathcal{C}_a) + \mathsf{Adv}^{\mathsf{cca}\text{-}m}_{\mathsf{PKE}}(\mathcal{E}_c) + 2q_{\mathrm{ENC}} \cdot \varepsilon + (2.5 \cdot q_{\mathrm{ENC}}^2 + 4q_{\mathrm{DEC}})/2^{\tau}$$

$$\mathbf{Query}(\mathcal{B}_c) = (q_{\mathrm{ENC}}, q_{\mathrm{DEC}}) \qquad \mathbf{Query}(\mathcal{C}_a) = (q_{\mathrm{ENC}}, q_{\mathrm{ENC}}, q_{\mathrm{DEC}})$$
$$\mathbf{Time}(\mathcal{B}_c) = \mathbf{Time}(\mathcal{A}_c) + (q_{\mathrm{ENC}} + q_{\mathrm{DEC}})\mathbf{Time}(\mathsf{NE}) \qquad \mathbf{Time}^*(\mathcal{C}_a) = \mathbf{Time}(\mathcal{A}_c)$$
$$\mathbf{Mem}(\mathcal{B}_c) = \mathbf{Mem}(\mathcal{A}_c) + \mathbf{Mem}(\mathsf{NE}). \qquad \mathbf{Mem}^*(\mathcal{C}_a) = \mathbf{Mem}(\mathcal{A}_c).$$

$$\mathbf{Query}(\mathcal{E}_c) = q_{\mathrm{ENC}} + q_{\mathrm{DEC}}$$
$$\mathbf{Time}(\mathcal{E}_c) = \mathbf{Time}(\mathcal{A}_c) + (q_{\mathrm{ENC}} + q_{\mathrm{DEC}})\mathbf{Time}(\mathsf{NE})$$
$$\mathbf{Mem}(\mathcal{E}_c) = \mathbf{Mem}(\mathcal{A}_c) + \mathbf{Mem}(\mathsf{NE}).$$

Proof. We consider the hybrids H_0 through H_3 and L_0 through L_3 defined in Figs. 4.34 and 4.36. Of these hybrids we the following claims establish the upper bound on the advantage of $\mathcal{A}_c$ claimed in the proof.

Hybrids H_h for $0 \leqslant h \leqslant 3$	$\textsc{Enc}(m)$	$\textsc{Dec}(c)$
$f \leftarrow\!\!\$\ \mathsf{Inj}(T, D, R)$ //$\mathsf{H}_{[2,\infty)}$	$u \leftarrow u + 1$ //$\mathsf{H}_{[2,\infty)}$	$(c_\mathsf{k}, c_\mathsf{d}) \leftarrow c$
$l \leftarrow \mathbb{I}$	$K \leftarrow\!\!\$\ \mathsf{NE.K}$	If $Y[l(ek, c_\mathsf{k})] \neq \perp$:
$u \leftarrow 0$ //$\mathsf{H}_{[2,\infty)}$	$c_\mathsf{k} \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, K)$ //$\mathsf{H}_{[0,1)}$	If $M[l(ek, c_\mathsf{k}), c_\mathsf{d}] \neq \perp$: //$\mathsf{H}_{[3,\infty)}$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$	$c_\mathsf{k} \leftarrow\!\!\$\ \mathsf{PKE.C}(ek, \lvert K\rvert)$ //$\mathsf{H}_{[1,2)}$	Return $M[l(ek, c_\mathsf{k}), c_\mathsf{d}]$ //$\mathsf{H}_{[3,\infty)}$
$b' \leftarrow \mathcal{A}_c^{\textsc{Enc},\textsc{Dec}}(ek)$	$c_\mathsf{k} \leftarrow f(ek, u)$ //$\mathsf{H}_{[2,\infty)}$	$K \leftarrow Y[l(ek, c_\mathsf{k})]$
Return $b' = 1$	$c_\mathsf{d} \leftarrow \mathsf{NE.E}(K, 0, m)$	Return $\mathsf{NE.D}(K, 0, c_\mathsf{d})$
	$Y[l(ek, c_\mathsf{k})] \leftarrow K$	Else:
	$M[l(ek, c_\mathsf{k}), c_\mathsf{d}] \leftarrow m$ //$\mathsf{H}_{[3,\infty)}$	$K \leftarrow \mathsf{PKE.D}(dk, c_\mathsf{k})$
	Return $(c_\mathsf{k}, c_\mathsf{d})$	If $K \neq \perp$:
		Return $\mathsf{NE.D}(K, 0, c_\mathsf{d})$
		Return $\perp$

Figure 4.34: First set of hybrids used for proof of Theorem 4.7.2. $\mathbb{I}$ denotes the identity function.

1. $\mathsf{Pr}[\mathsf{G}^{\$\mathtt{cca}\text{-}\mathtt{m}}_{\mathsf{KD}[\mathsf{PKE},\mathsf{NE}],1}(\mathcal{A}_c)] = \mathsf{Pr}[\mathsf{H}_0]$
2. $\mathsf{Pr}[\mathsf{H}_0] \leqslant \mathsf{Pr}[\mathsf{H}_1] + \mathsf{Adv}^{\$\mathtt{cca}\text{-}\mathtt{m}}_{\mathsf{PKE}}(\mathcal{B}_c)$
3. $\mathsf{Pr}[\mathsf{H}_1] \leqslant \mathsf{Pr}[\mathsf{H}_2] + 0.5 \cdot q_{\textsc{Enc}}^2/2^\tau$
4. $\mathsf{Pr}[\mathsf{H}_2] = \mathsf{Pr}[\mathsf{H}_3] = \mathsf{Pr}[\mathsf{L}_0]$
5. $\mathsf{Pr}[\mathsf{L}_0] \leqslant \mathsf{Pr}[\mathsf{L}_1] + \mathsf{Adv}^{\mathtt{mu}\text{-}\$\mathtt{cca}\text{-}\mathtt{m}}_{\mathsf{NE}}(\mathcal{C}_a)$
6. $\mathsf{Pr}[\mathsf{L}_1] = \mathsf{Pr}[\mathsf{L}_2] = \mathsf{Pr}[\mathsf{L}_3]$
7. $\mathsf{Pr}[\mathsf{L}_3] \leqslant \mathsf{Pr}[\mathsf{L}_4] + 0.5 \cdot q_{\textsc{Enc}}^2/2^\tau$
8. $\mathsf{Pr}[\mathsf{L}_4] = \mathsf{Pr}[\mathsf{M}_0]$
9. $\mathsf{Pr}[\mathsf{M}_0] \leqslant \mathsf{Pr}[\mathsf{M}_1] + 0.5 \cdot q_{\textsc{Enc}}^2/2^\tau$
10. $\mathsf{Pr}[\mathsf{M}_1] \leqslant \mathsf{Pr}[\mathsf{M}_2] + 2q_{\textsc{Dec}}/2^\tau$
11. $\mathsf{Pr}[\mathsf{M}_2] \leqslant \mathsf{Pr}[\mathsf{M}_3] + q_{\textsc{Enc}} \cdot \varepsilon$
12. $\mathsf{Pr}[\mathsf{M}_3] \leqslant \mathsf{Pr}[\mathsf{M}_4] + \mathsf{Adv}^{\mathtt{cca}\text{-}\mathtt{m}}_{\mathsf{PKE}}(\mathcal{E}_c)$
13. $\mathsf{Pr}[\mathsf{M}_4] \leqslant \mathsf{Pr}[\mathsf{M}_5] + q_{\textsc{Enc}} \cdot \varepsilon$
14. $\mathsf{Pr}[\mathsf{M}_5] \leqslant \mathsf{Pr}[\mathsf{M}_6] + 2q_{\textsc{Dec}}/2^\tau$
15. $\mathsf{Pr}[\mathsf{M}_6] \leqslant \mathsf{Pr}[\mathsf{M}_7] + 0.5 \cdot q_{\textsc{Enc}}^2/2^\tau$
16. $\mathsf{Pr}[\mathsf{M}_7] \leqslant \mathsf{Pr}[\mathsf{M}_8] + 0.5 \cdot q_{\textsc{Enc}}^2/2^\tau$
17. $\mathsf{Pr}[\mathsf{M}_8] = \mathsf{Pr}[\mathsf{G}^{\$\mathtt{cca}\text{-}\mathtt{m}}_{\mathsf{KD}[\mathsf{PKE},\mathsf{NE}],0}(\mathcal{A}_c)]$

Transition to H_0. We claim that H_0 is identical to $\mathsf{G}^{\$\mathtt{cca}\text{-}\mathtt{m}}_{\mathsf{KD}[\mathsf{PKE},\mathsf{NE}],1}$. Note that, in the latter, Enc produces honest encryptions using KD and Dec produces honest decryptions. It is immediately clear that the same holds for Enc in H_0 and the else branch of Dec in H_0. Consider the first branch in Dec. We have used grey highlighting to indicate the relevant code. Note that l is the identity function $\mathbb{I}$ (its presence will be notationally convenient for future game transitions). Y is a table indexed by ek, c_k pairs which stores the key encrypted in c_k. The use of Y in Dec simply recovers this K without decrypting c_k. By the correctness of PKE this is identical to having done the decryption, so $\mathsf{Pr}[\mathsf{G}^{\$\mathtt{cca}\text{-}\mathtt{m}}_{\mathsf{KD}[\mathsf{PKE},\mathsf{NE}],1}(\mathcal{A}_c)] = \mathsf{Pr}[\mathsf{H}_0]$ follows.

Adversary $\mathcal{B}_c^{\text{Enc},\text{Dec}}(ek)$	SimEnc(m)	SimDec(c)
$b' \leftarrow \mathcal{A}_c^{\text{SimEnc},\text{SimDec}}(ek)$	$K \leftarrow\!\!\$\ \mathsf{NE.K}$	$(c_\mathsf{k}, c_\mathsf{d}) \leftarrow c$
Return b'	$c_\mathsf{k} \leftarrow \text{Enc}(K)$	$K \leftarrow \text{Dec}(c_\mathsf{k})$
	$c_\mathsf{d} \leftarrow \mathsf{NE.E}(K, 0, m)$	If $K \neq \bot$:
	Return $(c_\mathsf{k}, c_\mathsf{d})$	Return $\mathsf{NE.D}(K, 0, c_\mathsf{d})$
		Return $\bot$

Figure 4.35: Adversary $\mathcal{B}_c$ for Theorem 4.7.2.

Transition H_0 to H_1. In hybrid H_1, the ciphertext c_k is now sampled uniformly at random. This difference corresponds to what we expect from the \$CCA-m security of PKE. We define $\mathcal{B}_c$ in Fig. 4.35 from hybrid H_1 by replacing encryption and decryption of K with appropriate queries to its oracle. Note that the use of table Y in the hybrids is similar to the table M from $\mathsf{G}^{\$\text{cca-m}}$ (in Y all entries are additionally indexed by the same ek). We can see that when interacting with $\mathsf{G}^{\$\text{cca-m}}_{\mathsf{PKE},1}$, $\mathcal{B}_c$ simulates H_0 to $\mathcal{A}_c$ and when interacting with $\mathsf{G}^{\$\text{cca-m}}_{\mathsf{PKE},0}$, $\mathcal{B}_c$ simulates H_1 to $\mathcal{A}_c$. Hence $\mathsf{Pr}[\mathsf{H}_0] \leqslant \mathsf{Pr}[\mathsf{H}_1] + \mathsf{Adv}^{\$\text{cca-m}}_{\mathsf{PKE}}(\mathcal{B}_c)$.

Transition H_1 to H_2. In hybrid H_2, instead of sampling c_k at random, we assign it the output of a random injective function f applied to the encryption key ek as the tweak and a counter u (we will later make u correspond to users in $\mathsf{G}^{\text{mu-ae-m}}$). The switching lemma tells us that $\mathsf{Pr}[\mathsf{H}_1] \leqslant \mathsf{Pr}[\mathsf{H}_2] + 0.5 \cdot q_{\text{Enc}}^2/2^\tau$.

Transition H_2 to H_3. In hybrid H_3, we introduce a table M which is indexed by ek, c_k, c_d (the first two via l) and stores the value of m whose encryption under $K = Y[l(ek, c_\mathsf{k})]$ is c_d. This table is used in Dec to skip the step of decrypting c_d. By the correctness of NE this does not change functionality, so $\mathsf{Pr}[\mathsf{H}_2] = \mathsf{Pr}[\mathsf{H}_3]$.

Transition H_3 to L_0. Next we transition to hybrid L_0 shown in Fig. 4.36. We have highlighted all ways that this differs from hybrid H_3. Our changes were twofold, consisting of switching the function l indexing into our tables to f^{-1} and switching the if condition in Dec. Considering the latter first, note that the checks $Y[l(ek, c_\mathsf{k})] \neq \bot$ in H_3 and $1 \leqslant f^{-1}(ek, c_\mathsf{k}) \leqslant u$ in L_0 are identical. Note that $Y[l(ek, c_\mathsf{k})] \neq \bot$ iff c_k was returned by the u'-th Enc query for some $1 \leqslant u' \leqslant u$ which holds iff $f^{-1}(ek, c_\mathsf{k}) = u'$. Switching to using $l = f^{-1}$ cannot change behavior since $(ek, c_\mathsf{k}) = (ek, c'_\mathsf{k})$ iff $f^{-1}(ek, c_\mathsf{k}) = f^{-1}(ek, c'_\mathsf{k}) \neq \bot$. Thus $\mathsf{Pr}[\mathsf{H}_3] = \mathsf{Pr}[\mathsf{L}_0]$.

Transition L_0 to L_1. In hybrid L_1, the ciphertext c_d is now sampled uniformly at random. We define $\mathcal{C}_a$ in Fig. 4.37 from hybrid L_1 by replacing encryption and decryption of c_d with appropriate queries to its oracle. We claim that when interacting with $\mathsf{G}^{\text{mu-\$cca-m}}_{\mathsf{NE},1}$, $\mathcal{C}_a$ simulates L_0 to $\mathcal{A}_c$ and when interacting with $\mathsf{G}^{\text{mu-\$cca-m}}_{\mathsf{NE},0}$, $\mathcal{C}_a$ simulates L_1 to $\mathcal{A}_c$. To see this, note that the variable u in these hybrids matches the same variable in

Hybrids L_ℓ for $0 \leqslant \ell \leqslant 4$	$\textsc{Enc}(m)$	$\textsc{Dec}(c)$
$f \leftarrow\!\!\$\ \mathsf{Inj}^{\pm}(T, D, R)$ //$\mathsf{L}_{[0,4)}$	$u \leftarrow u + 1$ //$\mathsf{L}_{[0,4)}$	$(c_{\mathsf{k}}, c_{\mathsf{d}}) \leftarrow c$
$g \leftarrow\!\!\$\ \mathsf{Fcs}(T', D', R')$ //$\mathsf{L}_{[2,\infty)}$	$K \leftarrow\!\!\$\ \mathsf{NE.K}$	If $1 \leqslant f^{-1}(ek, c_{\mathsf{k}}) \leqslant u$://$\mathsf{L}_{[0,3)}$
$l \leftarrow f^{-1}$ //$\mathsf{L}_{[0,3)}$	$c_{\mathsf{k}} \leftarrow f(ek, u)$ //$\mathsf{L}_{[0,4)}$	If $Y[l(ek, c_{\mathsf{k}})] \neq \bot$://$\mathsf{L}_{[3,\infty)}$
$l \leftarrow \mathbb{I}$ //$\mathsf{L}_{[3,\infty)}$	$c_{\mathsf{k}} \leftarrow\!\!\$\ \mathsf{PKE.C}(ek, \lvert K\rvert)$ //$\mathsf{L}_{[4,\infty)}$	If $M[l(ek, c_{\mathsf{k}}), c_{\mathsf{d}}] \neq \bot$:
$u \leftarrow 0$ //$\mathsf{L}_{[0,4)}$	$c_{\mathsf{d}} \leftarrow \mathsf{NE.E}(K, 0, m)$ //$\mathsf{L}_{[0,1)}$	Return $M[l(ek, c_{\mathsf{k}}), c_{\mathsf{d}}]$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$	$c_{\mathsf{d}} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{NE.cl}(\lvert m\rvert)}$ //$\mathsf{L}_{[1,2)}$	$K \leftarrow Y[l(ek, c_{\mathsf{k}})]$
$b' \leftarrow \mathcal{A}_a^{\textsc{Enc},\textsc{Dec}}(ek)$	$c_{\mathsf{d}} \leftarrow g((c_{\mathsf{k}}, \lvert m\rvert), m)$ //$\mathsf{L}_{[2,\infty)}$	Return $\mathsf{NE.D}(K, 0, c_{\mathsf{d}})$
Return $b' = 1$	$Y[l(ek, c_{\mathsf{k}})] \leftarrow K$	Else:
	$M[l(ek, c_{\mathsf{k}}), c_{\mathsf{d}}] \leftarrow m$	$K \leftarrow \mathsf{PKE.D}(dk, c_{\mathsf{k}})$
	Return $(c_{\mathsf{k}}, c_{\mathsf{d}})$	If $K \neq \bot$:
		Return $\mathsf{NE.D}(K, 0, c_{\mathsf{d}})$
		Return $\bot$

Figure 4.36: Second set of hybrids used for proof of Theorem 4.7.2.

$\mathsf{G}^{\text{mu-\$cca-m}}$. Additionally, if $i = l(ek, c_{\mathsf{k}})$ then the values $Y[i]$ and $M[i, c_{\mathsf{d}}]$ in these hybrids always match K_i and $M[i, 0, c_{\mathsf{d}}]$ in $\mathsf{G}^{\text{mu-\$cca-m}}$. (The claims can be rigorously verified by plugging the code of $\mathsf{G}^{\text{mu-\$cca-m}}_{\mathsf{NE},b}$ into $\mathcal{C}_a$ and comparing side-by-side with L_{1-b}.) This gives $\mathsf{Pr}[\mathsf{L}_0] \leqslant \mathsf{Pr}[\mathsf{L}_1] + \mathsf{Adv}^{\text{mu-\$cca-m}}_{\mathsf{NE}}(\mathcal{C}_a)$.

Transition L_1 to L_2. In hybrid L_2, the ciphertext c_{d} is now the output of a random function g from $\mathsf{Fcs}(T', D', R')$, rather than being sampled uniformly. (Recall $T' = \{0,1\}^* \times \mathbb{N}$, $D'_{(c_{\mathsf{k}},l)} = \{0,1\}^l$, and $R'_{(c_{\mathsf{k}},l)} = \{0,1\}^{\mathsf{NE.cl}(l)}$.) The inputs to g never repeat (because c_{k} never repeats), so this does not modify the behavior of the game, giving $\mathsf{Pr}[\mathsf{L}_1] = \mathsf{Pr}[\mathsf{L}_2]$.

Transition L_2 to L_3. In hybrid L_3, we undo the code changes used to transition from H_3 to L_0. Namely, l is set back to $\mathbb{I}$ and the if statement in $\textsc{Dec}$ is reverted. By the same logic as that prior transition, $\mathsf{Pr}[\mathsf{L}_2] = \mathsf{Pr}[\mathsf{L}_3]$.

Transition L_3 to L_4. In hybrid L_4, we undo the code changes used to transition from H_1 to H_2. Namely, we get rid of the injection f and sample c_{k} uniformly. By the same logic as that prior transition, $\mathsf{Pr}[\mathsf{L}_3] \leqslant \mathsf{Pr}[\mathsf{L}_4] + 0.5 \cdot q_{\textsc{Enc}}^2 / 2^\tau$.

Transition L_4 to M_0. Next we transition to the game M_0 shown in Fig 4.38. In this game we made simplifications to L_4 to prepare us for our final transitions. In particular, we removed the labelling function l and instead index into tables Y and M directly. We additionally drop the use of ek in this indexing. Then

Adversary $\mathcal{C}_a^{\text{New},\text{Enc},\text{Dec}}$	$\text{SimEnc}(m)$	$\text{SimDec}(c)$
$f \leftarrow\!\!\$\ \mathsf{Inj}^{\pm}(T, D, R)$	$u \leftarrow u + 1$	$(c_{\mathsf{k}}, c_{\mathsf{d}}) \leftarrow c$
$u \leftarrow 0$	$\text{New}()$	$i \leftarrow f^{-1}(ek, c_{\mathsf{k}})$
$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$	$c_{\mathsf{k}} \leftarrow f(ek, u)$	If $1 \leqslant i \leqslant u$:
$b' \leftarrow \mathcal{A}_c^{\text{SimEnc},\text{SimDec}}(ek)$	$c_{\mathsf{d}} \leftarrow \text{Enc}(u, 0, m)$	Return $\text{Dec}(i, 0, c_{\mathsf{d}})$
Return b'	Return $(c_{\mathsf{k}}, c_{\mathsf{d}})$	Else:
		$K \leftarrow \mathsf{PKE.D}(dk, c_{\mathsf{k}})$
		If $K \neq \perp$:
		Return $\mathsf{NE.D}(K, 0, c_{\mathsf{d}})$
		Return $\perp$

Figure 4.37: Adversary $\mathcal{C}_a$ for Theorem 4.7.2.

the code in Dec was rewritten for ease of comparison to the final game we are trying to reach (M_8, which we will show is equivalent to $\mathsf{G}^{\$\text{cca-m}}_{\mathsf{KD}[\mathsf{PKE},\mathsf{NE}],0}(\mathcal{A}_c)$). Because $M[ek, c_{\mathsf{k}}, c_{\mathsf{d}}] \neq \perp$ implies $Y[ek, c_{\mathsf{k}}] \neq \perp$, we separated out the if statement which checks M, rather than leaving it nested inside of the check for Y. Rather than repeating the code which runs $\mathsf{NE.D}$ in two separate branches we consolidated to be run at the end of the oracle. None of these changes modify the behavior of the game, so $\mathsf{Pr}[\mathsf{L}_4] = \mathsf{Pr}[\mathsf{M}_0]$.

Note that the final game M_8 we are moving toward differs from this game primarily in that, for decryption queries when $M[ek, c_{\mathsf{k}}, c_{\mathsf{d}}] = \perp$ but $Y[ek, c_{\mathsf{k}}] \neq \perp$ game M_0 uses the key stored in Y (which was chosen at random) to decrypt c_{d} while game M_8 will used whatever key is encrypted in c_{k}. These extra transition were not needed in our secret-key KEM/DEM proof because the final game we were transitioning to in that proof returned $\perp$ for *any* decryption query not directly forwarded from encryption.

Transition M_0 to M_1. To transition to M_1 we sample g as an injection, rather than a function. The switching lemma tells us that $\mathsf{Pr}[\mathsf{M}_0] \leqslant \mathsf{Pr}[\mathsf{M}_1] + 0.5 \cdot q_{\text{Enc}}^2/2^{\tau}$.

Transition M_1 to M_2. In M_2 we replace the use of the table M with g^{-1} in Dec. These games will differ only if the adversary makes a decryption query for $(c_{\mathsf{k}}, c_{\mathsf{d}})$ where $M[c_{\mathsf{k}}, c_{\mathsf{d}}] = \perp$ and $g^{-1}((c_{\mathsf{k}}, |c_{\mathsf{d}}| - \mathsf{NE.xl}), c_{\mathsf{d}}) \neq \perp$. We can bound the probability of such a query in M_1. To make such a query, the adversary must be "guessing" a new point in the image of $g^{-1}((c_{\mathsf{k}}, |c_{\mathsf{d}}| - \mathsf{NE.xl}), \cdot)$ other than the at most q_{Enc} examples it may have been given from Enc. Note that in M_1, these examples from Enc are the only way that g affects the behavior of the game. Note that the range of $g^{-1}((c_{\mathsf{k}}, |c_{\mathsf{d}}| - \mathsf{NE.xl}), \cdot)$ has size $2^{|c_{\mathsf{d}}|}$ and its image has size $2^{|c_{\mathsf{d}}| - \mathsf{NE.xl}}$. Thus (using that $q_{\text{Enc}} \leqslant 0.5 \cdot 2^{\tau}$) any particular decryption query has probability at most

Hybrids M_ℓ for $0 \leqslant \ell \leqslant 8$

$(ek, dk) \leftarrow\!\!\$\ \mathsf{PKE.K}$

$g \leftarrow\!\!\$\ \mathsf{Fcs}(T', D', R')$ //$\mathsf{M}_{[0,1)}, \mathsf{M}_{[7,8)}$

$g \leftarrow\!\!\$\ \mathsf{Inj}(T', D', R')$ //$\mathsf{M}_{[1,7)}$

$b' \leftarrow \mathcal{A}_c^{\text{ENC},\text{DEC}}(ek)$

Return $b' = 1$

ENC(m)

$K \leftarrow\!\!\$\ \mathsf{NE.K}$

$c_\mathsf{k} \leftarrow\!\!\$\ \mathsf{PKE.C}(ek, |K|)$ //$\mathsf{M}_{[0,3)}, \mathsf{M}_{[5,\infty)}$

$K' \leftarrow\!\!\$\ \mathsf{NE.K}$ //$\mathsf{M}_{[3,5)}$

$c_\mathsf{k} \leftarrow\!\!\$\ \mathsf{PKE.E}(ek, K')$ //$\mathsf{M}_{[3,5)}$

$c_\mathsf{d} \leftarrow g((c_\mathsf{k}, |m|), m)$ //$\mathsf{M}_{[0,8)}$

$c_\mathsf{d} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{NE.cl}(|m|)}$ //$\mathsf{M}_{[8,\infty)}$

$Y[c_\mathsf{k}] \leftarrow K$

$M[c_\mathsf{k}, c_\mathsf{d}] \leftarrow m$

Return $(c_\mathsf{k}, c_\mathsf{d})$

DEC(c)

$(c_\mathsf{k}, c_\mathsf{d}) \leftarrow c$

If $M[c_\mathsf{k}, c_\mathsf{d}] \neq \bot$: //$\mathsf{M}_{[0,2)}, \mathsf{M}_{[6,\infty)}$

  Return $M[c_\mathsf{k}, c_\mathsf{d}]$ //$\mathsf{M}_{[0,2)}, \mathsf{M}_{[6,\infty)}$

$m \leftarrow g^{-1}((c_\mathsf{k}, |c_\mathsf{d}| - \mathsf{NE.xl}), c_\mathsf{d})$//$\mathsf{M}_{[2,6)}$

If $m \neq \bot$: Return m//$\mathsf{M}_{[2,6)}$

If $Y[c_\mathsf{k}] \neq \bot$:

  $K \leftarrow Y[c_\mathsf{k}]$//$\mathsf{M}_{[0,4)}$

  $K \leftarrow \mathsf{PKE.D}(dk, c_\mathsf{k})$//$\mathsf{M}_{[4,\infty)}$

Else:

  $K \leftarrow \mathsf{PKE.D}(dk, c_\mathsf{k})$

If $K \neq \bot$:

  Return $\mathsf{NE.D}(K, 0, c_\mathsf{d})$

Return $\bot$

Figure 4.38: Third set of hybrids used for proof of Theorem 4.7.2. Hybrids after M_4 are mostly undoing prior transitions.

$2^{|c_\mathsf{d}|-\mathsf{NE.xl}} / \left(2^{|c_\mathsf{d}|} - q_{\text{ENC}}\right) \leqslant 2/2^\tau$. Applying a union bound gives $\mathsf{Pr}[\mathsf{M}_1] \leqslant \mathsf{Pr}[\mathsf{M}_2] + 2q_{\text{DEC}}/2^\tau$.

TRANSITION M_2 TO M_3. Now in M_3 we sample c_k as the encryption of a random key K' rather than choosing it uniformly at random. The key K' will thus be the key obtained if c_k is decrypted in DEC. A standard hybrid argument using the $(\mathsf{NE.kl}, \varepsilon)$-uniformity of PKE gives that $\mathsf{Pr}[\mathsf{M}_2] \leqslant \mathsf{Pr}[\mathsf{M}_3] + q_{\text{ENC}} \cdot \varepsilon$.

TRANSITION M_3 TO M_4. Next we change the behavior of the decryption oracle when $Y[c_\mathsf{k}] \neq \bot$ and $g^{-1}((c_\mathsf{k}, |c_\mathsf{d}| - \mathsf{NE.xl}), c_\mathsf{d}) = \bot$. In M_3, we used the key $Y[c_\mathsf{k}]$ to decrypt c_d. In M_4, we use the decryption of c_k (which was called K' when originally sampled in ENC). This difference actually corresponds to the CCA security of PKE.

Consider the adversary $\mathcal{E}_c$ shown in Fig. 4.39. It was obtained by modifying the code of hybrids M_3 and M_4 to query its own encryption oracle to obtain c_k when responding to encryption queries and to use its own decryption oracle to obtain K when responding to and query for which it cannot obtain its response using

Adversary $\mathcal{E}_c^{\text{Enc},\text{Dec}}(ek)$	$\text{SimEnc}(m)$	$\text{SimDec}(c)$
$g \leftarrow\!\!\$\ \mathsf{Inj}(T', D', R')$	$K_0 \leftarrow\!\!\$\ \mathsf{NE.K}$	$(c_\mathsf{k}, c_\mathsf{d}) \leftarrow c$
$b' \leftarrow \mathcal{A}_c^{\text{SimEnc},\text{SimDec}}(ek)$	$K_1 \leftarrow\!\!\$\ \mathsf{NE.K}$	$m \leftarrow g^{-1}((c_\mathsf{k}, \lvert c_\mathsf{d}\rvert - \mathsf{NE.xl}), c_\mathsf{d})$
Return $1 - b'$	$c_\mathsf{k} \leftarrow \text{Enc}(K_0, K_1)$	If $m \neq \bot$: Return m
	$c_\mathsf{d} \leftarrow g((c_\mathsf{k}, \lvert m\rvert), m)$	$K \leftarrow \text{Dec}(c_\mathsf{k})$
	Return $(c_\mathsf{k}, c_\mathsf{d})$	If $K \neq \bot$:
		Return $\mathsf{NE.D}(K, 0, c_\mathsf{d})$
		Return $\bot$

Figure 4.39: Adversary $\mathcal{E}_c$ for Theorem 4.7.2.

g^{-1}. We claim that the view of $\mathcal{A}_c$ in M_{3+b} perfectly matches its view when simulated by $\mathcal{E}_c$ in $\mathsf{G}^{\mathsf{cca\text{-}m}}_{\mathsf{PKE},b}$. In particular, K sampled during encryption in M_{3+b} "matches" the K_{1-b} sampled for encryption in $\mathsf{G}^{\mathsf{cca\text{-}m}}_{\mathsf{PKE},b}(\mathcal{E}_c)$ and K' matches K_b. Note that K_b/K' is always the key encrypted in c_k. When $b = 0$, a later decryption query using c_k will use the key stored in $Y[c_\mathsf{k}]$ which is K_1/K. When $b = 1$, a later decryption query using c_k will use the key actually encrypted by c_k which is K_1/K' (in $\mathsf{G}_{\mathsf{PKE},1}$ this value is stored in Y, in M_4 this is obtained by encryption).

Now, noting that $\mathcal{E}_c$ returns the opposite bit of what $\mathcal{A}_c$ returned we have that $\mathsf{Pr}[\mathsf{M}_{3+b}] = 1 - \mathsf{Pr}[\mathsf{G}_{\mathsf{PKE},b}(\mathcal{E}_c)]$ and so $\mathsf{Pr}[\mathsf{M}_3] \leqslant \mathsf{Pr}[\mathsf{M}_4] + \mathsf{Adv}^{\mathsf{cca\text{-}m}}_{\mathsf{PKE}}(\mathcal{E}_c)$.

Transitions M_4 through M_8. The next couple transition all serve to undo prior transitions. In M_5, we switch back to c_k being sampled uniformly. By the same logic as our transition to M_3 we have $\mathsf{Pr}[\mathsf{M}_4] \leqslant \mathsf{Pr}[\mathsf{M}_5] + q_{\text{Enc}} \cdot \varepsilon$. In M_6, we switch back to using the table M rather than g^{-1}. By analogous reasoning to our transition to M_2 we have $\mathsf{Pr}[\mathsf{M}_5] \leqslant \mathsf{Pr}[\mathsf{M}_6] + 2q_{\text{Dec}}/2^\tau$. In M_7, we switch from g being a random injection to a random function. By the switching lemma, $\mathsf{Pr}[\mathsf{M}_6] \leqslant \mathsf{Pr}[\mathsf{M}_7] + 0.5 \cdot q_{\text{Enc}}^2/2^\tau$. In M_8, we switch back to c_d being sampled uniformly rather than using g. This give differing behavior only if c_k ever repeats, giving $\mathsf{Pr}[\mathsf{M}_7] \leqslant \mathsf{Pr}[\mathsf{M}_8] + 0.5 \cdot q_{\text{Enc}}^2/2^\tau$.

Final Transition. Finally, we conclude by observing that M_8 is identical to $\mathsf{G}^{\mathsf{\$cca\text{-}m}}_{\mathsf{KD}[\mathsf{PKE},\mathsf{NE}],0}(\mathcal{A}_c)$. In both Enc returns a uniformly random ciphertext. For a decryption query $\text{Dec}(c)$, if c was returned by an earlier encryption query, then the table M is used to return the message from that query. Otherwise, c is honestly decrypted as specified by $\mathsf{KD}[\mathsf{PKE},\mathsf{NE}]$. Hence, $\mathsf{Pr}[\mathsf{M}_8] = \mathsf{Pr}[\mathsf{G}^{\mathsf{\$cca\text{-}m}}_{\mathsf{KD}[\mathsf{PKE},\mathsf{NE}],0}(\mathcal{A}_c)]$. ■

Chapter 5

MEMORY-TIGHTNESS OF HASHED ELGAMAL

In this chapter we prove the impossibility result for the memory-tight reduction of Hashed ElGamal that we introduced in Section 2.2.1. This chapter is based on our work in [GT20]. We first give a high-level overview of our techniques.

THE SHUFFLING GAME. Our adversary against Hashed ElGamal $\mathcal{A}$ first attempts to detect whether the reduction is using a sufficient amount of memory. The adversary $\mathcal{A}$ is given as input the public key g^v, as well as g^u, as well as a string $C^* \in \{0,1\}^\ell$, which is either a real encapsulation or a random string. It first samples k values $i_1, \ldots, i_k$ from $\mathbb{Z}_p$. It then:

(1) Asks for decapsulation queries for $C_j \leftarrow g^{i_j}$, obtaining values K_j, for $j \in [k]$

(1) Picks a random permutation $\pi : [k] \to [k]$.

(1) Asks for RO queries for $H_j \leftarrow \mathsf{H}(V_j)$ for $j \in [k]$, where $V_j \leftarrow g^{v \cdot i_{\pi(j)}}$.

After this, the adversary checks whether $K_j = H_{\pi(j)}$ for all $j \in [k]$, and if so, it continues its execution, breaking the ODH assumption (inefficiently). If not, it just outputs a random guess.

The intuition here is that no reduction using substantially less than $k \cdot \log p$ bits succeeds in passing the above test – in particular, because the inputs C_j and V_j are (pseudo-)random, and thus incompressible. If the test does not pass, the adversary $\mathcal{A}$ is rendered useless, and thus not helpful to break SDH.

Remark 5.0.1. *The adversary here is described in a way that requires secret randomness, not known to the reduction, and it is easier to think of $\mathcal{A}$ in this way. We will address later in the chapter how to generically make the adversary deterministic.*

Remark 5.0.2. *We stress that this adversary requires memory – it needs to remember the answers $C_1, \ldots, C_k$. However, recall that we adopt a black-box approach to memory-tightness, where our requirement is that the reduction itself uses little memory, regardless of the memory used by the adversary. We also argue this is somewhat necessary – it is not clear how to design a reduction which adapts its memory usage to the adversary, even if given this information in a non-black-box manner. Also, we conjecture different (and much harder to analyze) memory-less adversaries exist enabling a separation. An example is multi-round variant,*

where each round omits **(2)**, *and* **(3)** *only asks a single query* $\mathsf{H}(V_j)$ *for a random* $j \leftarrow\!\!\$\, [k]$, *and checks consistency. Intuitively, the chance of passing each round is roughly* $k \log p/s$, *but we do not know how to make this formal.*

Introducing the GGM. Our intuition is however *false* for an arbitrary group. For instance, if the discrete logarithm (DL) problem is easy in the group, then the reduction can simply simulate the random oracle via a PRF, as suggested in [ACFK17]. Ideally, we could prove that if the DL problem is hard in G, then any PPT reduction given access to $\mathcal{A}$ and with less than $k \cdot \log p$ bits of memory fails to break SDH.[1] Unfortunately, it will be hard to capture a single hardness property of G sufficient for our proof to go through. Instead, we will model the group via the *generic group model* (GGM) [Sho97, Mau05]: We model a group of prime order p defined via a random injection $\sigma : \mathbb{Z}_p \to \mathcal{L}$. An algorithm in the model typically has access to $\sigma(1)$ (in lieu of g) and an evaluation oracle which on input $\mathbf{a}, \mathbf{b} \in \mathcal{L}$ returns $\sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$.

The permutation game. In order to fool $\mathcal{A}$, the reduction can learn information about π via the O_v oracle. For example, it can try to input $C_j = g^{i_j}$ and $V_{j'} = g^{v i_{\pi(j')}}$ (both obtained from $\mathcal{A}$'s queries), and $\mathsf{O}_\mathsf{v}(C_j, V_{j'}) = \mathsf{true}$ if and only if $\pi(j') = j$. More generally, the reduction can compute, for any $\vec{a} = (a_1, \ldots, a_k)$ and $\vec{b} = (b_1, \ldots, b_k)$,

$$C^* = g^{\sum_{j=1}^k a_j i_j} = \prod_{j=1}^k C_j^{a_j} \ , \quad V^* = g^{\sum_{j=1}^k b_j v \cdot i_{\pi(j)}} = \prod_{j=1}^k V_j^{b_j} \ ,$$

and the query $\mathsf{O}_\mathsf{v}(C^*, V^*)$ returns true iff $b_j = a_{\pi(j)}$ for all $j \in [k]$, which we write as $\vec{b} = \pi(\vec{a})$. We abstract this specific strategy in terms of an information-theoretic game – which we refer to as the *permutation game* – which gives the adversary access to an oracle O which takes as inputs pairs of vectors $(\vec{a}, \vec{b})$ from $\mathbb{Z}_p^k$, and returns true iff $\vec{b} = \pi(\vec{a})$ for a secret permutation π. The goal of the adversary is to recover π.

Clearly, a strategy can win with $O(k^2)$ oracle queries $(\vec{e}_i, \vec{e}_j)$ for all i, j, where $\vec{e}_i \in \mathbb{Z}_p^k$ is the unit vector with a 1 in the i-th coordinate, and 0 elsewhere. This strategy requires in particular querying, in its first component, vectors which have rank k. Our first result will prove that this is necessary – namely, assume that an adversary makes a sequence of q queries $(\vec{x}_1, \vec{y}_1), \ldots, (\vec{x}_q, \vec{y}_q)$ such that the rank of $\vec{x}_1, \ldots, \vec{x}_q$ is at most ℓ, then the probability to win the permutation game is of the order $O(q^\ell/k!)$. We will prove this via a compression argument.

Note that roughly, this bound tells us that to win with probability ε and q queries to the oracle, the attacker needs

$$\ell = \Omega\left(\frac{k \log k - \log(1/\varepsilon)}{\log(q)}\right) \ .$$

[1] This statement is somewhat confusing, so note that in general, the existence of a reduction is *not* a contradiction with the hardness of DL, as the reduction is meant to break SDH only given access to an adversary breaking the scheme, and this does not imply the ability to break SDH *without* access to the adversary.

A REDUCTION TO THE PERMUTATION GAME. We will think of the execution of the reduction against our adversary as consisting of two stages – we refer to them as $\mathcal{R}_1$ and $\mathcal{R}_2$. The former learns the decapsulation queries $g^{i_1}, \ldots, g^{i_k}$, whereas the latter learns the RO queries $g^{i_{\pi(1)} v}, \ldots, g^{i_{\pi(k)} v}$, and (without loss of generality) attempts to guess the permutation π. We will lower bound the size of the state ϕ that $\mathcal{R}_1$ passes on to $\mathcal{R}_2$. Both stages can issue $\mathsf{O_v}$ and Eval queries.

Note that non-trivial $\mathsf{O_v}$ queries (i.e., those revealing some information about the permutation), are (except with very small probability) issued by $\mathcal{R}_2$, since no information about π is ever revealed to $\mathcal{R}_1$. As one of our two key steps, we will provide a reduction from the execution of $\mathcal{R}_1, \mathcal{R}_2$ against $\mathcal{A}$ in the GGM to the permutation game – i.e., we build an adversary $\mathcal{D}$ for the latter game simulating the interaction between $\mathcal{R}_1, \mathcal{R}_2$ and $\mathcal{A}$, and such that $\mathcal{R}_1, \mathcal{R}_2$ "fooling" $\mathcal{A}$ results in $\mathcal{D}$ guessing the permutation.

MEMORY VS. RANK. The main question, however, is to understand the complexity of $\mathcal{D}$ in the permutation game, and in particular, the *rank* ℓ of the first component of its queries – as we have seen above, this affects its chance of winning the game.

To do this, we will take a slight detour, and specifically consider a set $\mathcal{Z} \subseteq \mathcal{L}$ of labels (i.e., outputs of σ) that the reduction $\mathcal{R}_2$ comes up with (as inputs to either of Eval or $\mathsf{O_v}$) on its own (in the original execution), i.e., no earlier Eval query of $\mathcal{R}_2$ returned them, and that have been previously learnt by $\mathcal{R}_1$ as an output of its Eval queries. (The actual definition of $\mathcal{Z}$ is more subtle, and this is due to the ability of the adversary to come up with labels *without* knowing the corresponding pre-image.)

Then, we will show two statements about $\mathcal{Z}$:

(i) On the one hand, we show that the rank ℓ of the oracle queries of the adversary $\mathcal{D}$ is upper bound by $|\mathcal{Z}|$ in its own simulation of the execution of $\mathcal{R}_1, \mathcal{R}_2$ with $\mathcal{A}$.

(i) On the other hand, via a compression argument, we prove that the size of $\mathcal{Z}$ is related to the length of ϕ, and this will give us our final upper bound.

This latter statement is by itself not very surprising – one can look at the execution of $\mathcal{R}_2$, and clearly every label in $\mathcal{Z}$ that appears "magically" in the execution must be the result of storing them into the state ϕ. What makes this different from more standard compression arguments is the handling of the generic group model oracle (which admits non-trivial operations). In particular, our compression argument will compress the underlying map σ, and we will need to be able to figure out the pre-images of these labels in $\mathcal{Z}$.

MEMORY-TIGHT AGM REDUCTION. The Algebraic Group Model (AGM) was introduced in [FKL18]. AGM reductions make strong extractability assumptions, and the question of their memory-tightness is an interesting one. In [GT20], we construct a reduction to the discrete logarithm problem that runs an adversary

against the KEM-CCA security of Hashed ElGamal in the AGM such that the reduction is memory-tight but not tight with respect to advantage. We note that the model of our reduction is different than a (full-fledged) GGM reduction which is not black-box, in that it can observe the GGM queries made by the adversary. Our result does not imply any impossibility for these. In turn, AGM reductions are weaker, but our results do not imply anything about them, either.

In the rest of the chapter we provide all the details of this result.

5.1 Preliminaries

In this section, we review the formal definition of the generic group model. We also state ODH and SDH as introduced in [ABR01] in the generic group model.

NOTATION. Let $\mathbb{N} = \{0, 1, 2, \cdots\}$ and, for $k \in \mathbb{N}$, let $[k] = \{1, 2, \cdots, k\}$. We denote by $\mathsf{InjFunc}(S_1, S_2)$ the set of all injective function from S_1 to S_2.

We also let $*$ denote a wildcard element. For example $\exists t : (t, *) \in T$ is true if the set T contains an ordered pair whose first element is t (the type of the wildcard element shall be clear from the context). Let $\mathcal{S}_k$ denote the set of all permutations on $[k]$. We use $f : \mathsf{D} \to \mathsf{R} \cup \{\perp\}$ to denote a partial function, where $f(x) = \perp$ indicates the value of $f(x)$ is undefined. Define in particular $D(f) = \{d \in \mathsf{D} \ : \ f(d) \neq \perp\}$ and $R(f) = \{r \in \mathsf{R} \ : \ \exists d \in \mathsf{D} \ : \ \sigma(d) = r\}$. Moreover, we let $\overline{D(f)} = \mathsf{D} \backslash D(f)$ and $\overline{R(f)} = \mathsf{R} \backslash R(f)$.

We shall use pseudocode descriptions of games inspired by the code-based framework of [BR06]. We make use of pairs of games being identical-until-bad (i.e., the games are identical until a bad flag is set), and apply the so-called "Fundamental Lemma" to upper bound the difference in the probability of the game outputs in terms of the probability of bad being set to true in either of the games. The output of a game is denoted using the symbol $\Rightarrow$. In all games we assume the flag bad is set to false initially. In pseudocode, we denote random sampling using $\leftarrow\!\!\$$, assignment using $\leftarrow$ and equality check using $=$. In games that output boolean values, we use the term"winning" the game to mean that the output of the game is true.

We also introduce some linear-algebra notation. Let S be a set vectors with equal number of coordinates. We denote the rank and the linear span of the vectors by $\mathsf{rank}(S)$ and $\mathsf{span}(S)$ respectively. Let $\vec{x}, \vec{y}$ be vectors of dimension k. We denote $\vec{z}$ of dimension $2k$ which is the concatenation of $\vec{x}, \vec{y}$ as $\vec{z} = (\vec{x}, \vec{y})$. We denote the element at index i of a vector $\vec{x}$ as $\vec{x}[i]$.

5.1.1 Generic Group Model

The *generic group model* [Sho97] captures algorithms that do not use any special property of the encoding of the group elements, other than assuming every element of the group has a unique representation, and that the basic group operations are allowed. This model is useful in proving lower bounds for some problems, but

we use it here to capture reductions that are not specific to the underlying group.

More formally, let the order of the group be a large prime p. Let $\mathbb{Z}_p = \{0, 1, 2, \cdots, p-1\}$. Let $\mathcal{L} \subset \{0,1\}^*$ be a set of size p, called the set of *labels*. Let σ be a random injective mapping from $\mathbb{Z}_p$ to $\mathcal{L}$. The idea is that now every group element in $\mathbb{Z}_p$ is represented by a label in $\mathcal{L}$. An algorithm in this model takes as input $\sigma(1), \sigma(x_1), \sigma(x_2), \cdots, \sigma(x_n)$ for some $x_1, \cdots, x_n \in \mathbb{Z}_p$ (and possibly other inputs which are not group elements). The algorithm also has access to an oracle named Eval which takes as input two labels $\mathbf{a}, \mathbf{b} \in \mathcal{L}$ and returns $\mathbf{c} = \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$. Note that for any d, given $\sigma(x_i)$, $\sigma(d \cdot x_i)$ can be computed using $O(\log d)$ queries to Eval. We denote this operation as $\mathsf{Exp}(\sigma(x_i), d)$. We assume that all labels queried by algorithms in the generic group model are valid i.e. all labels queried by algorithms in the generic group model are in $\mathcal{L}$.[2]

Oracle Diffie-Hellman assumption (ODH). We first formalize the Oracle Diffie-Hellman Assumption (ODH) [ABR01], which we are going to use in lieu of the CCA security of Hashed ElGamal. Suppose, a group has generator g and order p. The domain of hash function H is all finite strings and range is $\{0,1\}^{\mathsf{hLen}}$. The assumption roughly states for $u, v \leftarrow\!\!\$\ \mathbb{Z}_p, W \leftarrow\!\!\$\ \{0,1\}^{\mathsf{hLen}}$, the distributions $(g^u, g^v, \mathsf{H}(g^{uv}))$ and (g^u, g^v, W) are indistinguishable to an adversary who has access to the oracle H_v where $\mathsf{H}_\mathsf{v}(g^x)$ returns $\mathsf{H}(g^{xv})$ with the restriction that it is not queried on g^u.

We give a formalization of this assumption in the random-oracle and generic group models. For a fixed $\mathsf{hLen} \in \mathbb{N}$, let Ω_{hLen} be the set of hash functions mapping $\{0,1\}^*$ to $\{0,1\}^{\mathsf{hLen}}$. In Figure 5.1, we formally define the Games $\mathsf{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}$, $\mathsf{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}$. The advantage of violating ODH is defined as

$$\mathsf{Adv}^{\mathsf{ODH\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A}) = \left|\Pr\left[\mathsf{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A}) \Rightarrow 1\right] - \Pr\left[\mathsf{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A}) \Rightarrow 1\right]\right| .$$

Strong Diffie-Hellman Assumption (SDH). This is a stronger version of the classical CDH assumption. This assumption roughly states that CDH is hard even in the presence of a DDH-oracle O_v where $\mathsf{O}_\mathsf{v}(g^x, g^y)$ is true if and only if $x \cdot v = y$.

We formally define the game $\mathsf{G}^{\mathsf{SDH\text{-}GG}}$ in the generic group model in Figure 5.1. The advantage of violating SDH is defined as

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A}) = \left|\Pr\left[\mathsf{G}^{\mathsf{SDH\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A}) \Rightarrow \mathsf{true}\right]\right| .$$

Note in particular that one *can* upper bound this advantage unconditionally.

We shall drop the $\mathcal{L}$ from the subscript of advantages and games henceforth since the set of labels $\mathcal{L}$ remains the same throughout our chapter.

Black Box reductions in the GGM. We consider black-box reductions in the generic group model. We will limit ourselves to an informal description, but this can easily be formalized within existing formal

[2] We stress that we assume a strong version of the model where the adversary *knows* $\mathcal{L}$.

Game $\mathsf{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A})$:

$\sigma \leftarrow\!\!\$\ \mathsf{InjFunc}(\mathbb{Z}_p \to \mathcal{L})$

$u \leftarrow\!\!\$\ \mathbb{Z}_p; U \leftarrow \sigma(u)$

$v \leftarrow\!\!\$\ \mathbb{Z}_p; V \leftarrow \sigma(v)$

$\mathsf{H} \leftarrow\!\!\$\ \Omega_{\mathsf{hLen}}$

$W \leftarrow \mathsf{H}(\sigma(uv))$

$b \leftarrow \mathcal{A}^{\mathsf{H_v}(.),\mathsf{H}(.),\mathsf{Eval}(.,.)}(U, V, W, \sigma(1))$

Return b

Game $\mathsf{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A})$:

$\sigma \leftarrow\!\!\$\ \mathsf{InjFunc}(\mathbb{Z}_p \to \mathcal{L})$

$u \leftarrow\!\!\$\ \mathbb{Z}_p; U \leftarrow \sigma(u)$

$v \leftarrow\!\!\$\ \mathbb{Z}_p; V \leftarrow \sigma(v)$

$\mathsf{H} \leftarrow\!\!\$\ \Omega_{\mathsf{hLen}}$

$W \leftarrow \{0,1\}^{\mathsf{hLen}}$

$b \leftarrow \mathcal{A}^{\mathsf{H_v}(.),\mathsf{H}(.),\mathsf{Eval}(.,.)}(U, V, W, \sigma(1))$

Return b

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b})$:

Return $\sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$

Oracle $\mathsf{H_v}(\mathbf{a})$:

If $\mathbf{a} = U$ then return $\perp$

Else return $\mathsf{H}(\sigma(\sigma^{-1}(\mathbf{a}) \cdot v))$

Game $\mathsf{G}^{\mathsf{SDH\text{-}GG}}_{\mathcal{L},p,\mathsf{hLen}}(\mathcal{A})$:

$\sigma \leftarrow\!\!\$\ \mathsf{InjFunc}(\mathbb{Z}_p \to \mathcal{L})$

$u \leftarrow\!\!\$\ \mathbb{Z}_p; U \leftarrow \sigma(u)$

$v \leftarrow\!\!\$\ \mathbb{Z}_p; V \leftarrow \sigma(v)$

$z \leftarrow \mathcal{A}^{\mathsf{Eval}(.,.),\mathsf{O_v}(.,.)}(U, V, \sigma(1))$

Return $(z = \sigma(uv))$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b})$:

Return $(\sigma^{-1}(\mathbf{a}) \cdot v = \sigma^{-1}(\mathbf{b}))$

Figure 5.1: Games for ODH and SDH assumptions

frameworks for reductions (see e.g. [RTV04]). We let the reduction $\mathcal{R}$ access an adversary $\mathcal{A}$, and denote by $\mathcal{R}^{\mathcal{A}}$ the resulting algorithm – understood here is that $\mathcal{R}$ supplies inputs, answers queries, etc. In addition, we let $\mathcal{R}$ and $\mathcal{A}$ access the Eval oracle available in the GGM. We stress that the GGM oracle is not under the reduction's control here – typically, the reduction itself will break a (hard) problem in the GGM with help of $\mathcal{A}$. We will allow (for simplicity) $\mathcal{A}$ to be run depending on some secret private coins[3] not accessible

[3]If we want to allow the reduction to control random bits, we model them explicitly as an additional input.

by $\mathcal{R}$. Reductions can run $\mathcal{A}$ several times (with fresh private coins). We call a reduction *straigthline* if it only runs $\mathcal{A}$ *once*.

In our case, the reduction $\mathcal{R}$ will be playing $\mathsf{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}$. It receives as inputs $\sigma(1)$, $U = \sigma(u)$, $V = \sigma(v)$, and has access to the Eval, $\mathsf{O_v}$ oracles, as well as an adversary $\mathcal{A}$ for $\mathsf{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{p,\mathsf{hLen}}$ or $\mathsf{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{p,\mathsf{hLen}}$. The reduction needs therefore to supply inputs $(\sigma(1), U', V', W)$ to $\mathcal{A}$, and to answer its queries to the oracles $\mathsf{H_v}$, as well as queries to H. We will call such a reduction *restricted* if it is straightline *and* $V' = V$.

5.1.2 Compression Lemma

In our lower bound proof we use the compression lemma that was formalized in [DTT10] which roughly means that it is impossible to compress every element in a set with cardinality c to a string less than $\log c$ bits long, even relative to a random string. We state the compression lemma here as a proposition.

Proposition 5.1.1. *Suppose, there is a (not necessarily efficient) procedure* $\mathsf{Encode} : \mathcal{X} \times \mathcal{R} \to \mathcal{Y}$ *and a (not necessarily efficient) decoding procedure* $\mathsf{Decode} : \mathcal{Y} \times \mathcal{R} \to \mathcal{X}$ *such that*

$$\mathsf{Pr}_{x \in \mathcal{X}, r \in \mathcal{R}} \left[\mathsf{Decode}(\mathsf{Encode}(x, r), r) = x\right] \geqslant \varepsilon \ ,$$

then $\log |\mathcal{Y}| \geqslant \log |\mathcal{X}| - \log(1/\varepsilon)$.

5.1.3 Polynomials and Schwartz-Zippel Lemma

Let $\mathsf{p}(X_1, \cdots, X_n)$ be a n variate polynomial. We denote by $\mathsf{p}(x_1, \cdots, x_n)$ the evaluation of p at the point $(x_1, \cdots, x_n)$ throughout the chapter. The polynomial ring in variables $X_1, \cdots, X_n$ over the field $\mathbb{Z}_p$ is denoted by $\mathbb{Z}_p[X_1, \cdots, X_n]$. We state the Schwartz-Zippel Lemma in $\mathbb{Z}_p$ which will be useful later.

Proposition 5.1.2. *Let* p *be a non-zero* n *variate polynomial with degree* d. *Then*

$$\mathsf{Pr}_{x_1, \cdots, x_n \leftarrow\$ \mathbb{Z}_p} \left[Q(x_1, \cdots, x_n) = 0 \pmod p\right] \leqslant \frac{d}{p} \ .$$

5.1.4 Key Encapsulation Mechanism (KEM)

A key-encapsulation mechanism (KEM) consists of three probabilistic polynomial time (PPT) algorithms $\mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap}$. The key generation algorithm Gen is probabilistic and outputs a key-pair (pk, sk). The encapsulation algorithm Encap is a probabilistic algorithm that takes pk as input and outputs a ciphertext c and a key K where $K \in \mathcal{K}$ for some non-empty set $\mathcal{K}$. The decapsulation algorithm Decap is a deterministic algorithm that takes as input the secret key sk and a ciphertext c outputs a key $K \in \mathcal{K}$ if (sk, c) is a valid secret key-ciphertext pair and $\perp$ otherwise. For correctness, it is required that for all pairs (pk, sk) output by Gen, if (K, c) is output by $\mathsf{Encap}(\mathsf{pk})$ then K is the output of $\mathsf{Decap}(sk, c)$.

Single challenge KEM-CCA security. The single challenge CCA security of a KEM is defined by a pair of games called $\mathsf{G}^{\mathsf{KEM\text{-}CCA\text{-}REAL}}, \mathsf{G}^{\mathsf{KEM\text{-}CCA\text{-}RAND}}$. In both games a (pk, sk) pair is generated by Gen, and (c, K) is output by the encapsulation algorithm Encap on input pk. The adversary is provided with (pk, c, K) in $\mathsf{G}^{\mathsf{KEM\text{-}CCA\text{-}REAL}}$ and with (pk, c, K') in $\mathsf{G}^{\mathsf{KEM\text{-}CCA\text{-}RAND}}$ where K' is a randomly sampled element of $\mathcal{K}$. The adversary has access to the decapsulation oracle with sk as the secret key and it can make decapsulation queries on any ciphertext except the ciphertext c and has to output a bit. We define the advantage of violating single challenge KEM-CCA security is defined as the absolute value of the difference of probabilities of the adversary outputting 1 in the two games. A KEM is single challenge CCA-secure if for all non-uniform PPT adversaries the advantage of violating single challenge KEM-CCA security is negligible.

Single challenge KEM-CCA of Hashed ElGamal. We describe the KEM for Hashed ElGamal in a group with order p and generator g and a hash function H. The function Gen samples v at random from $\mathbb{Z}_p$, and returns (g^v, v) as the (pk, sk) pair. The function Encap on input v, samples u at random from $\mathbb{Z}_p$ and returns g^u as the ciphertext and $\mathsf{H}(g^{uv})$ as the key K. The function Decap on input c, returns $\mathsf{H}(c^v)$. Note that Decap in KEM of Hashed ElGamal is identical to the H_v function as defined in the ODH assumption. It follows that the single challenge KEM-CCA security of Hashed ElGamal is equivalent to the ODH assumption. In particular, in the generic group model when H is modeled as a random oracle, the single challenge KEM-CCA security of Hashed ElGamal is equivalent to the ODH assumption in the random oracle and generic group model.

5.2 Result and Proof Outline

In this section, we prove a memory lower bound for restricted black-box reductions from ODH to SDH. We stress that the restricted reduction has access only to the $\mathsf{H}, \mathsf{H}_\mathsf{v}$ queries of the adversary. As discussed above, the ODH assumption is equivalent to the single-challenge KEM-CCA security of Hashed ElGamal, this proves a memory lower-bound for (restricted) black-box reductions of single challenge KEM-CCA security of Hashed ElGamal to the SDH assumption.

Theorem 5.2.1 (Main Theorem)**.** *In the generic group model, with group order p, there exists an ODH adversary $\mathcal{A}$ that makes k H queries and k H_v queries (where k is a polynomial in $\log p$), a function $\varepsilon_1(p, \mathsf{hLen})$ which is negligible in $\log p, \mathsf{hLen}$, and a function $\varepsilon_2(p)$ which is negligible in $\log p$, such that,*

1. $\mathsf{Adv}^{\mathsf{ODH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A}) = 1 - \varepsilon_1(p, \mathsf{hLen})$.

2. *For all restricted black-box reductions $\mathcal{R}$, with s bits of memory and making a total of q (assuming*

$q \geqslant k$ and $6q \leqslant p - 4k - 4$) queries to $\mathsf{O_v}$, Eval,

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{48q^3}{p}\right)^{\frac{k}{8c}} \left(1 + \frac{6q}{p}\right)^{q} + \frac{4q^2 \log p + 13q^2 + 5q}{p} + \varepsilon_2(p) \,,$$

where $c = 4\lceil \frac{\log q}{\log k} \rceil$.

This result implies that if $\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}})$ is non-negligible for a reduction $\mathcal{R}$ making q queries where q is a polynomial in $\log p$, then $s = \Omega(k \log p)$ i.e. the memory required by any restricted black-box reduction grows with the number of queries by $\mathcal{A}$. Hence, there does not exist any efficient restricted black-box reduction from ODH to SDH that is memory-tight.

In Section 5.6, we discuss how rewinding can slightly improve the memory complexity to (roughly) $O(k \log k)$, with heavy computational cost (essentially, one rewinding per oracle query of the adversary). We conjecture this to be optimal, but a proof seems to evade current techniques.

De-randomization. Before we turn to the proof – which also connects several technical lemmas presented across the next sections, let us discuss some aspects of the results. As explained above, our model allows for the adversary $\mathcal{A}$ to be run with randomness unknown to $\mathcal{R}$. This aspect *may* be controversial, but we note that there is a generic way for $\mathcal{A}$ to be made deterministic. Recall that $\mathcal{A}$ must be inefficient for the separation to even hold true. For example, $\mathcal{A}$ can use the injection σ from the generic group model to generate its random coin – say, using $\sigma^{-1}(\mathbf{a}_i)$ as coins a priori fixed labels $\mathbf{a}_1, \mathbf{a}_2, \ldots$. It is a standard – albeit tedious and omitted – argument to show that unless the reduction ends up querying the pre-images (which happens with negligible probability only), the $\sigma^{-1}(\mathbf{a}_i)$'s are good random coins.

Strengthening beyond SDH. We would like to note that our result can be strengthened without much effort to a reduction between ODH and a more general version of SDH. Informally, we can extend our result to every problem which is hard in the generic group model in presence of an $\mathsf{O_v}$ oracle. For example, this could be a problem where given g, g^u, and g^v, the attacker needs to output $g^{f(u,v)}$, where f is (a fixed) two-variate polynomial with degree at least 2. We do not include the proof for the strengthened version for simplicity. However, it appears much harder to extend our result to different types of oracles than $\mathsf{O_v}$, as our proof is tailored at this oracle.

Proof. Here, we give the overall structure, the key lemmas, and how they are combined – quantitatively – to obtain the final result.

First off, Lemma 5.2.2 establishes that there exists an adversary $\mathcal{A}$ such that $\mathsf{Adv}^{\mathsf{ODH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A})$ is close to 1, which we will fix (i.e., when we refer to $\mathcal{A}$, we refer to the one guaranteed to exist by the lemma). The proof of Lemma 5.2.2 is in Section 5.3.1 and the proof of Lemma 5.2.3 is in Section 5.3.2.

Lemma 5.2.2. *There exists an adversary $\mathcal{A}$ and a function $\varepsilon_1(p, \mathsf{hLen})$ such that is negligible in $\log p, \mathsf{hLen}$, and*

$$\mathsf{Adv}^{\mathsf{ODH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A}) = 1 - \varepsilon_1(p, \mathsf{hLen}) \ .$$

After that, we introduce a game, called G_1 and described in Figure 5.3 in Section 5.3.2. Very informally, this is a game played by a two-stage adversary $\mathcal{R}_1, \mathcal{R}_2$ which can pass a state to each other of size s bits and have access to the $\mathsf{Eval}, \mathsf{O}_\mathsf{v}$ oracles. The game captures the essence of the reduction $\mathcal{R}$ the adversary $\mathcal{A}$ of having a sufficient amount of memory. This is made formal in Lemma 5.2.3, where we show that the probability of the reduction $\mathcal{R}$ winning $\mathsf{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}$ while running $\mathcal{A}$ is bounded by the probability of winning G_1.

Lemma 5.2.3. *For every restricted black box reduction $\mathcal{R}$ that runs $\mathcal{A}$ while playing $\mathsf{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}$, there exist adversaries $\mathcal{R}_1, \mathcal{R}_2$ playing G_1, such that the number of queries made by $\mathcal{R}_1, \mathcal{R}_2$ to $\mathsf{Eval}, \mathsf{O}_\mathsf{v}$ is same as the number of queries made by $\mathcal{R}$ to $\mathsf{Eval}, \mathsf{O}_\mathsf{v}$, the state passed from $\mathcal{R}_1$ to $\mathcal{R}_2$ is upper bounded by the memory used by $\mathcal{R}$ and,*

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant \Pr\left[\mathsf{G}_1 \Rightarrow \mathsf{true}\right] + \frac{4k^2(\log p)^2}{p} + \frac{4qk\log p + q^2}{p} \ .$$

We introduce Games $\mathsf{G}_2, \mathsf{G}_3$ in Figure 5.5 in Section 5.3.2. These games are identical to G_1 except for the condition to output true. The condition to output true in these games are disjoint and the disjunction of the two conditions is equivalent to the condition to output true in G_1. A little more specifically, both games depend on a parameter l, which can be set arbitrarily, and in G_3 and G_2 the winning condition of G_1 is strengthened by additional ensuring that a certain set defined during the execution of the game is smaller or larger than l, respectively. Therefore, tautologically,

$$\Pr\left[\mathsf{G}_1 \Rightarrow \mathsf{true}\right] = \Pr\left[\mathsf{G}_2 \Rightarrow \mathsf{true}\right] + \Pr\left[\mathsf{G}_3 \Rightarrow \mathsf{true}\right] \ . \tag{5.1}$$

We now prove the following two lemmas below, in Sections 5.3.3 and 5.3.4,

Lemma 5.2.4. *If $(\mathcal{R}_1, \mathcal{R}_2)$ make q queries to their oracles in total in the game G_2 as defined in Figure 5.5, then*

$$\Pr\left[\mathsf{G}_2 \Rightarrow \mathsf{true}\right] \leqslant \frac{q^l}{k!} + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{k^2+k+2}{p} \ .$$

Lemma 5.2.5. *If the size of the state ϕ output by $\mathcal{R}_1$ is s bits and $(\mathcal{R}_1, \mathcal{R}_2)$ make q queries in total in the game G_3 as defined in Figure 5.5, then*

$$\Pr\left[\mathsf{G}_3 \Rightarrow \mathsf{true}\right] \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{8q^2(2k+2+3q)}{p}\right)^{\frac{l}{2}} \left(1 + \frac{6q}{p}\right)^{\frac{2q-l}{2}} + \frac{k^2+k+2}{p} \ .$$

Combining (5.1) and the result of Lemmas 5.2.4 and 5.2.5 we get,

$$\mathsf{Pr}\left[\mathsf{G}_1 \Rightarrow \mathsf{true}\right] \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{8q^2(2k+2+3q)}{p}\right)^{\frac{l}{2}} \left(1+\frac{6q}{p}\right)^{\frac{2q-l}{2}} + \frac{2(k^2+k+2)}{p} + \frac{q^l}{k!} + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} \,. \quad (5.2)$$

Since $\left(1+\frac{6q}{p}\right)^{\frac{2q-l}{2}} \leqslant \left(1+\frac{6q}{p}\right)^{q}$, combining Lemma 5.2.3, (5.2) we get,

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{8q^2(2k+2+3q)}{p}\right)^{\frac{l}{2}} \left(1+\frac{6q}{p}\right)^{q} + \frac{2(k^2+k+2)}{p} + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{4k^2(\log p)^2}{p} + \frac{4qk\log p + q^2}{p} + \frac{q^l}{k!} \,.$$

We let,

$$\varepsilon_2(p) = \frac{q^l}{k!} + \frac{2(k^2+k+2)}{p} + \frac{4k^2(\log p)^2}{p} \,.$$

Setting $c = \lceil \frac{\log q}{\log k} \rceil$ and $l = \frac{k}{4c}$, $\frac{q^l}{k!} \leqslant \frac{k^{k/4}}{k!}$. By Sterling's approximation $k! \geqslant k^{k+1/2}e^{-k}$. Therefore,

$$\frac{k^{k/4}}{k!} = \frac{k^{k/4}}{k^{k/4}} \frac{e^k}{k^{k/4}} \frac{1}{k^{k/2+1/2}} \,.$$

For $k > e^4$ (we can set $k > e^4$), $\frac{q^l}{k!} \leqslant \frac{1}{k^{k/2+1/2}}$ i.e. $\frac{q^l}{k!}$ is negligible in $\log p$ for k polynomial in $\log p$. Also, $\frac{2(k^2+k+2)}{p} + \frac{4k^2(\log p)^2}{p}$ is negligible in $\log p$ (since k is a polynomial in $\log p$). So, $\varepsilon_2(p)$ is negligible in $\log p$. We have that,

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{8q^2(2k+2+3q)}{p}\right)^{\frac{k}{8c}} \left(1+\frac{6q}{p}\right)^{q} + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{4qk\log p + q^2}{p} + \varepsilon_2(p) \,.$$

where $c = 4\lceil \frac{\log q}{\log k} \rceil$. Assuming $q \geqslant k$ (and thus $q > e^4 > 2$), we get,

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{48q^3}{p}\right)^{\frac{k}{8c}} \left(1+\frac{6q}{p}\right)^{q} + \frac{4q^2\log p + 13q^2 + 5q}{p} + \varepsilon_2(p) \,.$$

■

5.3 *Proof of Theorem*

5.3.1 Adversary $\mathcal{A}$ against ODH

In this section, we construct the ODH adversary $\mathcal{A}$ needed for the proof.

Lemma 5.3.1. *There exists an adversary $\mathcal{A}$ and a function $\varepsilon_1(p, \mathsf{hLen})$ such that is negligible in $\log p, \mathsf{hLen}$, and*

$$\mathsf{Adv}^{\mathsf{ODH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A}) = 1 - \varepsilon_1(p, \mathsf{hLen}) \,.$$

Proof. The adversary $\mathcal{A}$ is formally defined in Figure 5.2. Adversary $\mathcal{A}$ samples $i_1, \cdots, i_k$ from $\mathbb{Z}_p$, and computes $\sigma(i_j), \sigma(i_j \cdot v)$ for all j in $[k]$. It then makes H_v queries on $\sigma(i_j)$'s for all j in $[k]$. Adversary $\mathcal{A}$ then samples a permutation π on $[k] \to [k]$, and then makes H queries on $\sigma(i_{\pi(j)} \cdot v)$'s for all j in $[k]$. If answers of all the H queries are distinct and the answers of all the H_v queries are distinct and for all j in $[k]$, $\mathsf{H}_\mathsf{v}(\sigma(i_j)) = \mathsf{H}(\sigma(i_j \cdot v))$, $\mathcal{A}$ computes the discrete logarithm of V outputs the correct answer. Otherwise it returns a bit sampled uniformly at random. Note that $\mathcal{A}$ is inefficient, but only if it is satisfied from the responses it gets from the reduction using it.

First off, we note that adversary $\mathcal{A}$ does not use its input W until after the flag honest is set. So, W does not affect the setting of honest in any way. Since W is the only difference among the inputs of $\mathsf{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A})$ and $\mathsf{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A})$ to $\mathcal{A}$, $\mathsf{Pr}\left[\mathsf{honest} = 0\right]$ and $\mathsf{Pr}\left[\mathsf{honest} = 1\right]$ are equal in both games. The flag honest is set to 0 in the following two cases.

1. for some distinct $j, l \in [k]$, $\mathsf{ans}_1[j] = \mathsf{ans}_1[l]$ or $\mathsf{ans}_2[j] = \mathsf{ans}_2[l]$). i.e.

$$\exists j, l \in [k], j \neq l : \mathsf{H}(\sigma(i_j)) = \mathsf{H}(\sigma(i_l)) \vee \mathsf{H}(\sigma(i_j \cdot v)) = \mathsf{H}(\sigma(i_l \cdot v)) \ .$$

   We name this event E_1. Note that if for some distinct $j, l \in [k]$, $i_j = i_l$, then the event E_1 happens with probability 1 since $\mathsf{H}(i_j) = \mathsf{H}(i_l)$. We compute the probability of that there exists some distinct $j, l \in [k]$, such that $i_j = i_l$.

$$\mathsf{Pr}\left[\exists j, l \in [k], j \neq l : i_j = i_l\right] = \sum_{j=1}^{k} \frac{j-1}{p} = \frac{k(k-1)}{p} \ .$$

   Also, note that if $v = 0$, E_1 happens with probability 1 since $\mathsf{H}(\sigma(i_j \cdot 0)) = \mathsf{H}(\sigma(0)) = \mathsf{H}(\sigma(i_l \cdot 0))$. Since v is sampled uniformly at random from $\mathbb{Z}_p$, $\mathsf{Pr}\left[v = 0\right] = \frac{1}{p}$.

   We define the event E_2 as follows.

$$(\nexists j, l \in [k], j \neq l : i_j = i_l) \wedge (v \neq 0) \ .$$

   Using the union bound,

$$\mathsf{Pr}\left[\neg E_2\right] \leqslant \frac{k(k-1)}{p} + \frac{1}{p} \ .$$

   Since, σ is an injective function, E_2 implies

$$\nexists j, l \in [k], j \neq l : \sigma(i_j) = \sigma(i_l) \vee \sigma(i_j \cdot v) = \sigma(i_l \cdot v) \ .$$

   Since H is a random oracle, the collision probability is,

$$\mathsf{Pr}\left[\mathsf{H}(r) = \mathsf{H}(s) \,\middle|\, r \neq s\right] = \frac{1}{2^{\mathsf{hLen}}} \ .$$

Therefore, using the union bound we have,

$$\mathsf{Pr}\left[E_1 \,\middle|\, E_2\right] \leqslant \frac{k(k-1)}{2^{\mathsf{hLen}}} \ .$$

Hence,

$$\mathsf{Pr}\left[E_1\right] \leqslant \mathsf{Pr}\left[\neg E_2\right] + \mathsf{Pr}\left[E_1 \,\middle|\, E_2\right] \leqslant \frac{k(k-1)+1}{p} + \frac{k(k-1)}{2^{\mathsf{hLen}}} \ .$$

2. for some $j \in [k]$, $\mathsf{ans}_1[j] \neq \mathsf{ans}_2[j]$ i.e.

$$\exists j \in [k] : \mathsf{H}_{\mathsf{v}}(\sigma(i_j)) \neq \mathsf{H}(\sigma(i_j \cdot v)) \ .$$

This is an impossible event in $\mathsf{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A})$ and $\mathsf{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A})$ by the definition of H_{v}.

Therefore,

$$\mathsf{Pr}\left[\mathsf{honest} = 0\right] = \mathsf{Pr}\left[E_1\right] \leqslant \frac{k(k-1)+1}{p} + \frac{k(k-1)}{2^{\mathsf{hLen}}} \ .$$

In $\mathsf{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A})$, if $\mathsf{honest} = 1$, the output of $\mathsf{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A})$ will always be 1 because $W = \mathsf{H}(\sigma(uv)) = \mathsf{H}(\{I\}) = W'$. Therefore,

$$\mathsf{Pr}\left[\mathsf{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A}) \Rightarrow 1\right] \geqslant \mathsf{Pr}\left[\mathsf{honest} = 1\right] \geqslant 1 - \left(\frac{k(k-1)+1}{p} + \frac{k(k-1)}{2^{\mathsf{hLen}}}\right) \ .$$

In $\mathsf{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A})$, if $\mathsf{honest} = 1$, the probability that $W = W'$ is at most $\frac{1}{2^{\mathsf{hLen}}}$ because W is sampled uniformly at random from the range of H in $\mathsf{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{p,\mathsf{hLen}}$. So we have,

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A}) \Rightarrow 1\right] &\leqslant \mathsf{Pr}\left[\mathsf{honest} = 0\right] \cdot \frac{1}{2} + \mathsf{Pr}\left[\mathsf{honest} = 1\right] \cdot \frac{1}{2^{\mathsf{hLen}}} \\ &\leqslant \frac{k(k-1)+1}{2p} + \frac{k(k-1)}{2^{\mathsf{hLen}+1}} + \frac{1}{2^{\mathsf{hLen}}}\left(1 - \frac{k(k-1)+1}{p} - \frac{k(k-1)}{2^{\mathsf{hLen}}}\right) \ .\end{aligned}$$

Hence,

$$\begin{aligned}&\mathsf{Pr}\left[\mathsf{G}^{\mathsf{ODH\text{-}REAL\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A}) \Rightarrow 1\right] - \mathsf{Pr}\left[\mathsf{G}^{\mathsf{ODH\text{-}RAND\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A}) \Rightarrow 1\right] \\ &\qquad \geqslant 1 - \left[\left(\frac{3k(k-1)+1}{2p} + \frac{3k(k-1)}{2^{\mathsf{hLen}+1}}\right) + \frac{1}{2^{\mathsf{hLen}}}\left(1 - \frac{k(k-1)+1}{p} - \frac{k(k-1)}{2^{\mathsf{hLen}}}\right)\right] \ .\end{aligned}$$

We let, $\varepsilon_1(p, \mathsf{hLen}) = \left(\frac{3k(k-1)+1}{2p} + \frac{3k(k-1)}{2^{\mathsf{hLen}+1}}\right) + \frac{1}{2^{\mathsf{hLen}}}\left(1 - \frac{k(k-1)+1}{p} - \frac{k(k-1)}{2^{\mathsf{hLen}}}\right)$. Since k is polynomial in $\log p$, $\varepsilon_1(p, \mathsf{hLen})$ is negligible in $\log p, \mathsf{hLen}$. Therefore,

$$\mathsf{Adv}^{\mathsf{ODH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{A}) \geqslant 1 - \varepsilon_1(p, \mathsf{hLen}) \ .$$

■

Adversary $\mathcal{A}^{\mathsf{H_v}(.),\mathsf{H}(.),\mathsf{Eval}(.,.)}(U,V,W,\sigma(1))$:

$i_1,\cdots,i_k \xleftarrow{\$} \mathbb{Z}_p$

For each $j \in [k]$:

  $Q_1[j] \leftarrow \mathsf{Exp}(\sigma(1), i_j); Q_2[j] \leftarrow \mathsf{Exp}(V, i_j)$

$\mathsf{honest} \leftarrow 1$

For each $j \in [k]$:

  $\mathsf{ans}_1[j] \leftarrow \mathsf{H_v}(Q_1[j])$

$\pi \xleftarrow{\$} \mathcal{S}_k$

For each $j \in [k]$:

  $\mathsf{ans}_2[\pi(j)] \leftarrow \mathsf{H}(Q_2[\pi(j)])$

If $\exists j, l \in [k], j \neq l : (\mathsf{ans}_1[j] = \mathsf{ans}_1[l] \vee \mathsf{ans}_2[j] = \mathsf{ans}_2[l])$ then $\mathsf{honest} \leftarrow 0$

If $\exists j \in [k] : \mathsf{ans}_1[j] \neq \mathsf{ans}_2[j]$ then $\mathsf{honest} \leftarrow 0$

If $\mathsf{honest} = 1$ then

  $\mathsf{temp} \leftarrow \sigma(1); v \leftarrow 1$

  While $(\mathsf{temp} \neq V)$

    $\mathsf{temp} \leftarrow \mathsf{Eval}(\mathsf{temp}, \sigma(1)); v \leftarrow v + 1$

  $\{I\} \leftarrow \mathsf{Exp}(U, v); W' \leftarrow \mathsf{H}(\{I\}); b \leftarrow (W' = W)$

Else $b \xleftarrow{\$} \{0,1\}$

Return b

Figure 5.2: The adversary $\mathcal{A}$

5.3.2 The Shuffling Games

The Game G_1. We first introduce the two-stage game G_1 played by a pair of adversaries $\mathcal{R}_1$ and $\mathcal{R}_2$. (With some foresight, these are going to be two stages of the reduction.) It is formally described in Figure 5.3. Game G_1 involves sampling $\sigma, i_1, \cdots, i_k, v$ from $\mathbb{Z}_p$, then running $\mathcal{R}_1$, followed by sampling permutation π from $\mathcal{S}_k$ and then running $\mathcal{R}_2$. The first stage $\mathcal{R}_1$ has inputs $\sigma(i_1), \cdots, \sigma(i_k)$ and it outputs a state ϕ of s bits along with k strings in $\{0,1\}^{\mathsf{hLen}}$. The second stage $\mathcal{R}_2$ has inputs $\phi, \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v)$ and it outputs k strings in $\{0,1\}^{\mathsf{hLen}}$. Both the stages $\mathcal{R}_1, \mathcal{R}_2$ have access to oracles $\mathsf{Eval}, \mathsf{O_v}$. Game G_1 outputs true if all the k strings output by $\mathcal{R}_1$ are distinct, and if all the k strings output by $\mathcal{R}_2$ are distinct, and if for all $j \in [k]$, the j^{th} string output by $\mathcal{R}_2$ is identical to the $\pi(j)^{\text{th}}$ string output by $\mathcal{R}_1$. Additionally, G_1 involves some bookkeeping. The $\mathsf{Eval}, \mathsf{O_v}$ oracles in G_1 take an extra parameter named from as input which indicates whether the query was from $\mathcal{R}_1$ or $\mathcal{R}_2$.

We introduce the phrase "seen by" before describing the bookkeeping. A label has been "seen by" $\mathcal{R}_1$ if it was an input to $\mathcal{R}_1$, queried by $\mathcal{R}_1$ or an answer to a previously made $\mathsf{Eval}(.,.,1)$ query. A label has been "seen by" $\mathcal{R}_2$ if it was an input to $\mathcal{R}_2$, queried by $\mathcal{R}_2$ or an answer to a previously made $\mathsf{Eval}(.,.,2)$ query. We describe the sets $\mathcal{X}, \mathcal{Y}_1, \mathcal{Y}_2, \mathcal{Z}$ which are used for bookkeeping in G_1.

- The labels in $\mathcal{X}$ are answers to $\mathsf{Eval}(.,.,1)$ queries such that it has not yet been "seen by" $\mathcal{R}_1$ before the query.

- $\mathcal{Y}_1$ contains all the labels that are input to $\mathcal{R}_1$, queried by $\mathcal{R}_1$ or answers to $\mathsf{Eval}(.,.,1)$ queries i.e. it is the set of labels "seen by" $\mathcal{R}_1$.

- $\mathcal{Y}_2$ contains all the labels that are input to $\mathcal{R}_2$, queried by $\mathcal{R}_1$ or answers to $\mathsf{Eval}(.,.,2)$ queries i.e. it is the set of labels "seen by" $\mathcal{R}_2$.

- All labels in $\mathcal{Z}$ are queried by $\mathcal{R}_2$ and have not been "seen by" $\mathcal{R}_2$ before the query and are in $\mathcal{X}$

The following lemma tells us that we can (somewhat straightforwardly) take a reduction as in the theorem statement, and transform it into an equivalent pair $\mathcal{R}_1, \mathcal{R}_2$ of adversaries for G_1. The point here is that the reduction is very unlikely to succeed in breaking the SDH assumption without doing an effort equivalent to winning G_1 to get $\mathcal{A}$'s help – otherwise, it is left with breaking SDH directly in the generic group model, which is hard.

Proof. We introduce the adversary $\mathcal{B}$ which is identical to $\mathcal{A}$ till setting of honest and then returns a bit uniformly at random. It is formally defined in Figure 5.4. The value of honest will be identical in $\mathcal{A}, \mathcal{B}$. Also, observe that if honest is set to 0 in $\mathcal{A}$, $\mathcal{A}$'s output is identical to that of $\mathcal{B}$.

Consider any restricted black-box reduction $\mathcal{R}$ from ODH to SDH in the random oracle and generic group model that has oracle access to $\mathcal{A}$. During the execution of $\mathcal{R}^{\mathcal{A}}$ if $\mathsf{honest} = 0$ in $\mathcal{A}$, then it has the same output as $\mathcal{B}$. So, Therefore,

$$\mathsf{Pr}\left[\mathsf{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \Rightarrow \mathsf{true}\right] \leqslant \mathsf{Pr}\left[\mathsf{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{B}}) \Rightarrow \mathsf{true}\right] + \mathsf{Pr}\left[\mathsf{honest} = 1 \text{ in } \mathcal{A} \text{ during the execution of } \mathcal{R}^{\mathcal{A}}\right] . \tag{5.3}$$

The adversary $\mathcal{R}^{\mathcal{B}}$ makes at most $2k \log p + q$ queries to its oracles. From the hardness of SDH in the generic group model [ABR01], it follows

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{B}}) \leqslant \frac{4k^2(\log p)^2}{p} + \frac{4qk \log p + q^2}{p} . \tag{5.4}$$

We show that if $\mathsf{honest} = 1$ in $\mathcal{A}$ when $\mathcal{R}$ runs $\mathcal{A}$, $\mathcal{R} = (\mathcal{R}_1, \mathcal{R}_2)$ can run $\mathcal{R}$ and win G_1 . We next describe

Game G_1 :

$\sigma \leftarrow\!\!\$\ \mathsf{InjFunc}(\mathbb{Z}_p, \mathcal{L}); i_1, \cdots, i_k, v \leftarrow\!\!\$\ \mathbb{Z}_p$

$\mathcal{X} \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}; \mathcal{Y}_1 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}$

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k))$

$\pi \leftarrow\!\!\$\ \mathcal{S}_k; \mathcal{Y}_2 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)\}; \mathcal{Z} \leftarrow \varnothing$

$s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \sigma(1), \sigma(v), \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v))$

$\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s'_j) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$

Return win

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

$\mathbf{c} \leftarrow \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$

If $\mathsf{from} = 1$ then

  If $\mathbf{c} \notin \mathcal{Y}_1$ then $\mathcal{X} \xleftarrow{\cup} \{\mathbf{c}\}$

  $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$

If $\mathsf{from} = 2$ then

  If $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$

  If $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$

  $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$

Return $\mathbf{c}$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\mathsf{from} = 1$ then $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$

If $\mathsf{from} = 2$ then

  If $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$

  If $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$

  $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$

return $(v \cdot \sigma^{-1}(\mathbf{a}) = \sigma^{-1}(\mathbf{b}))$

Figure 5.3: Game G_1. We use the phrase $\mathcal{R}_1, \mathcal{R}_2$ win G_1 to mean $\mathsf{G}_1 \Rightarrow \mathsf{true}$. We shall use this convention for all games in the chapter that output boolean values.

how $\mathcal{R}_1, \mathcal{R}_2$ run and simulate $\mathcal{A}$, $\mathsf{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}$ to $\mathcal{R}$.

Recall that $\mathcal{R}_1$ has inputs $\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)$. First off, $\mathcal{R}_1$ samples u uniformly at random from $\mathbb{Z}_p$ and computes $\sigma(u) = \mathsf{Exp}(\sigma(1), u)$. It then starts running $\mathcal{R}$ with inputs $\sigma(1), \sigma(u), \sigma(v)$ and simulates

Adversary $\mathcal{B}^{\mathsf{H_v}(.),\mathsf{H}(.),\mathsf{Eval}(.,.)}(U,V,W,\sigma(1))$:

$i_1,\cdots,i_k \leftarrow\!\!\$\ \mathbb{Z}_p$

For each $j \in [k]$:

  $Q_1[j] \leftarrow \mathsf{Exp}(\sigma(1), i_j); Q_2[j] \leftarrow \mathsf{Exp}(V, i_j)$

$\mathsf{honest} \leftarrow 1$

For each $j \in [k]$:

  $\mathsf{ans}_1[j] \leftarrow \mathsf{H_v}(Q_1[j])$

$\pi \leftarrow\!\!\$\ \mathcal{S}_k$

For each $j \in [k]$:

  $\mathsf{ans}_2[\pi(j)] \leftarrow \mathsf{H}(Q_2[\pi(j)])$

If $\exists j, l \in [k], j \neq l : (\mathsf{ans}_1[j] = \mathsf{ans}_1[l] \vee \mathsf{ans}_2[j] = \mathsf{ans}_2[l])$ then $\mathsf{honest} \leftarrow 0$

If $\exists j \in [k], j \neq l : \mathsf{ans}_1[j] \neq \mathsf{ans}_2[j]$ then $\mathsf{honest} \leftarrow 0$

$b \leftarrow\!\!\$\ \{0,1\}$

Return b

Figure 5.4: Adversary $\mathcal{B}$

the oracles $\mathsf{Eval}, \mathsf{O_v}$ to $\mathcal{R}$. For every $\mathsf{Eval}(\mathbf{a},\mathbf{b})$ query made by $\mathcal{R}$, $\mathcal{R}_1$ makes an $\mathsf{Eval}(\mathbf{a},\mathbf{b},1)$ query to its own oracle and forwards the answer to $\mathcal{R}$. For every $\mathsf{O_v}(\mathbf{a},\mathbf{b})$ query made by $\mathcal{R}$, $\mathcal{R}_1$ makes an $\mathsf{O_v}(\mathbf{a},\mathbf{b},1)$ query to its own oracle and forwards the answer to $\mathcal{R}$. $\mathcal{R}_1$ makes $\mathsf{H_v}$ queries to $\mathcal{R}$ on $\sigma(i_1),\cdots,\sigma(i_k)$ and receives responses $s_1,\cdots,s_k$. $\mathcal{R}_1$ stops the execution of $\mathcal{R}$ after making the $\mathsf{H_v}$ queries, records the local state st of $\mathcal{R}$. Finally, $\mathcal{R}_1$ outputs st along with $s_1,\cdots,s_k$. Recall that $\mathcal{R}_2$ has inputs $\mathsf{st}, \sigma(i_{\pi(1)} \cdot v),\cdots,\sigma(i_{\pi(k)} \cdot v)$. First, $\mathcal{R}_2$ restarts $\mathcal{R}$ from state st. For every $\mathsf{Eval}(\mathbf{a},\mathbf{b})$ query made by $\mathcal{R}$, $\mathcal{R}_2$ makes an $\mathsf{Eval}(\mathbf{a},\mathbf{b},2)$ query to its own oracle and forwards the answer to $\mathcal{R}$. For every $\mathsf{O_v}(\mathbf{a},\mathbf{b})$ query made by $\mathcal{R}$, $\mathcal{R}_2$ makes an $\mathsf{O_v}(\mathbf{a},\mathbf{b},2)$ query to its own oracle and forwards the answer to $\mathcal{R}$. $\mathcal{R}_2$ makes H queries to $\mathcal{R}$ on $\sigma(i_{\pi(1)} \cdot v),\cdots,\sigma(i_{\pi(k)} \cdot v)$ and receives responses $s'_1,\cdots,s'_k$. Finally, $\mathcal{R}_2$ outputs $s'_1,\cdots,s'_k$.

Observe that σ, u, v are sampled identically in $\mathsf{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}), \mathsf{G}_1$, $i_1,\cdots,i_k,\pi$ are sampled identically in $\mathcal{A}, \mathsf{G}_1$ and $\mathcal{R}_1, \mathcal{R}_2$ make the same queries to $\mathcal{R}$ as $\mathcal{A}$ does in the same order. It follows that $\mathcal{R}_1, \mathcal{R}_2$ perfectly simulate $\mathcal{A}$, $\mathsf{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}$ to $\mathcal{R}$. Also note that the number of queries made by $\mathcal{R}_1, \mathcal{R}_2$ to $\mathsf{Eval}, \mathsf{O_v}$ is same as the number of queries made by $\mathcal{R}$ to $\mathsf{Eval}, \mathsf{O_v}$ and the state passed from $\mathcal{R}_1$ to $\mathcal{R}_2$ is upper bounded by the memory used by $\mathcal{R}$ because ϕ is the local state st of $\mathcal{R}$ at a certain point in its execution.

Next we relate the probabilities of $\mathsf{honest} = 1$ in $\mathcal{A}$ and $\mathsf{G}_1 \Rightarrow \mathsf{true}$. If $\mathsf{honest} = 1$ in $\mathcal{A}$ we have the following.

1. There does not exist distinct $j, l \in [k]$ such that $\mathsf{ans}_1[j] = \mathsf{ans}_1[l]$. The equivalent condition in G_1 is $\forall j, l \in [k] : j \neq l \implies s_j \neq s_l$.

2. There does not exist distinct $j, l \in [k]$ such that $\mathsf{ans}_2[j] = \mathsf{ans}_2[l]$. The equivalent condition in G_1 is $\forall j, l \in [k] : j \neq l \implies s'_j \neq s'_l$.

3. For all $j \in [k]$, $\mathsf{ans}_1[j] = \mathsf{ans}_2[j]$. The equivalent condition in G_1 is $\forall j \in [k] : s_{\pi(j)} = s'_j$.

It follows that whenever $\mathsf{honest} = 1$ in $\mathcal{A}$, $\mathsf{G}_1 \Rightarrow \mathsf{true}$. Hence,

$$\mathsf{Pr}\left[\mathsf{honest} = 1 \text{ in } \mathcal{A} \text{ during the execution of } \mathcal{R}^{\mathcal{A}}\right] \leqslant \mathsf{Pr}\left[\mathsf{G}_1 \Rightarrow \mathsf{true}\right] . \tag{5.5}$$

From (5.3) and (5.5), it follows that,

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant \mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{B}}) + \mathsf{Pr}\left[\mathsf{G}_1 \Rightarrow \mathsf{true}\right] . \tag{5.6}$$

Combining (5.4) and (5.6), we have,

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant \mathsf{Pr}\left[\mathsf{G}_1 \Rightarrow \mathsf{true}\right] + \frac{4k^2(\log p)^2}{p} + \frac{4qk \log p + q^2}{p} . \tag{5.7}$$

■

Lemma 5.3.2. *For every restricted black box reduction $\mathcal{R}$ that runs $\mathcal{A}$ while playing $\mathsf{G}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}$, there exist adversaries $\mathcal{R}_1, \mathcal{R}_2$ playing G_1, such that the number of queries made by $\mathcal{R}_1, \mathcal{R}_2$ to $\mathsf{Eval}, \mathsf{O}_{\mathsf{v}}$ is same as the number of queries made by $\mathcal{R}$ to $\mathsf{Eval}, \mathsf{O}_{\mathsf{v}}$, the state passed from $\mathcal{R}_1$ to $\mathcal{R}_2$ is upper bounded by the memory used by $\mathcal{R}$ and,*

$$\mathsf{Adv}^{\mathsf{SDH\text{-}GG}}_{p,\mathsf{hLen}}(\mathcal{R}^{\mathcal{A}}) \leqslant \mathsf{Pr}\left[\mathsf{G}_1 \Rightarrow \mathsf{true}\right] + \frac{4k^2(\log p)^2}{p} + \frac{4qk \log p + q^2}{p} .$$

THE GAMES G_2 AND G_3. In Figure 5.5 we define $\mathsf{G}_2, \mathsf{G}_3$ which have an added check on the cardinality of $\mathcal{Z}$ to output true. Everything else remains unchanged (in particular $\mathsf{Eval}, \mathsf{O}_{\mathsf{v}}$ are unchanged from G_1 and we do not specify them again here). The statement within the thinner box is present only in G_3 and statement within the thicker box is present only in G_2. The changes from G_1 have been highlighted. We shall follow these conventions of using boxes and highlighting throughout the chapter.

The games $\mathsf{G}_2, \mathsf{G}_3$ are identical to G_1 except for the condition to output true. Since this disjunction of the conditions to output true in $\mathsf{G}_2, \mathsf{G}_3$ is equivalent to the condition to output true in G_1, and the conditions to output true in $\mathsf{G}_2, \mathsf{G}_3$ are disjoint, we have,

$$\mathsf{Pr}\left[\mathsf{G}_1 \Rightarrow \mathsf{true}\right] = \mathsf{Pr}\left[\mathsf{G}_2 \Rightarrow \mathsf{true}\right] + \mathsf{Pr}\left[\mathsf{G}_3 \Rightarrow \mathsf{true}\right] .$$

Game $\boxed{\mathsf{G}_2}, \boxed{\mathsf{G}_3}$:

$\sigma \leftarrow\!\!\$\ \mathsf{InjFunc}(\mathbb{Z}_p, \mathcal{L}); i_1, \cdots, i_k, v \leftarrow\!\!\$\ \mathbb{Z}_p$

$\mathcal{X} \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}; \mathcal{Y}_1 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}$

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k))$

$\pi \leftarrow\!\!\$\ \mathcal{S}_k; \mathcal{Y}_2 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)\}; \mathcal{Z} \leftarrow \varnothing$

$s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \sigma(1), \sigma(v), \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v))$

$\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s'_j) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$

$\boxed{\textbf{return } (\mathsf{win} \wedge |\mathcal{Z}| < l)}$ $\boxed{\textbf{return } (\mathsf{win} \wedge |\mathcal{Z}| \geqslant l)}$

Figure 5.5: Games $\mathsf{G}_2, \mathsf{G}_3$. The $\mathsf{Eval}, \mathsf{O_v}$ oracles in $\mathsf{G}_2, \mathsf{G}_3$ are identical to those in G_1 and hence we do not rewrite it here. The newly introduced changes compared to G_1 are highlighted. The statement within the thinner box is present only in G_3 and the statement within the thicker box is present only in G_2.

5.3.3 Proof of Lemma 5.2.4

Recall we are going to prove the following lemma.

Lemma 5.3.3. *If $(\mathcal{R}_1, \mathcal{R}_2)$ make q queries to their oracles in total in the game G_2 as defined in Figure 5.5, then*

$$\Pr[\mathsf{G}_2 \Rightarrow \mathsf{true}] \leqslant \frac{q^l}{k!} + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{k^2+k+2}{p}\ .$$

Proof. We introduce a new game – called the *permutation game* and denoted $\mathbb{PG}$ – in order to upper bound $\Pr[\mathsf{G}_2 \Rightarrow \mathsf{true}]$. In the rest of this proof, we are going to first define the game, and upper bound the winning probability of an adversary. Then, we are going to reduce an adversary for G_2 to one for $\mathbb{PG}$.

The Permutation Game. In Game $\mathbb{PG}$, an adversary has to guess a randomly sampled permutation π over $[k]$. The adversary has access to an oracle that takes as input two vectors of length k and returns true if the elements of the first vector, when permuted using π, results in the second vector and false otherwise. Figure 5.6 formally describes the game $\mathbb{PG}$.

In the following, we say an adversary playing $\mathbb{PG}$ is a (q, l)-query adversary if it makes at most q queries to O, and the rank of the vectors that were the first argument to the O queries returning true is at most l.

The following lemma – which we prove via a compression argument – yields an upper bound on the probability of winning the game for a (q, l)-query adversary.

Lemma 5.3.4. *For a (q, l)-query adversary $\mathcal{A}$ playing $\mathbb{PG}$ the following is true.*

$$\Pr[\mathbb{PG}(\mathcal{A}) \Rightarrow \mathsf{true}] \leqslant \frac{q^l}{k!}\ .$$

Game $\mathbb{PG}(\mathcal{A})$:	**Oracle** $O(\vec{x},\vec{y})$://$\vec{x} \in \mathbb{Z}_p^k, \vec{y} \in \mathbb{Z}_p^k$
$\pi \leftarrow\!\!\$\ \mathcal{S}_k$	Return $(\forall i \in [k] : \vec{x}[\pi(i)] = \vec{y}[i])$
$\pi' \leftarrow \mathcal{A}^{O(.,.)}$	
Return $(\pi = \pi')$	

Figure 5.6: The permutation game $\mathbb{PG}$ being played by adversary $\mathcal{A}$ is denoted by $\mathbb{PG}(\mathcal{A})$

Proof. We construct an encoding of π by running adversary $\mathcal{A}$. In order to run $\mathcal{A}$, all the O queries need to be correctly answered. This can be naively done by storing the sequence number of queries whose answers are true. In fact, of all such queries, we need to just store the sequence number of just those whose first argument is not in the linear span of vectors which were the first argument of previous such queries i.e. we store the sequence number of only those O queries returning true whose first argument form a basis of the first argument of all O queries returning true. This approach works because for every vector $\vec{x}$, there is only a unique vector $\vec{y}$ such that $O(\vec{x},\vec{y}) = 1$. The random tape of the adversary can be derived using the common randomness of $\mathsf{Encode}, \mathsf{Decode}$ and hence the adversary produces identical queries and output. For simplicity, we do not specify this explicitly in the algorithms and treat $\mathcal{A}$ as deterministic. The formal description of the algorithms $\mathsf{Encode}, \mathsf{Decode}$ are in Figure 5.7.

Observe that S is a basis of vectors $\vec{x}$ such that $O(\vec{x},\vec{y}) = \mathsf{true}$. Note that for an $O(\vec{x},\vec{y})$ query returning true, if $\vec{x} \in S$ then the sequence number of the query is stored in enc. Therefore, $(\vec{x},\vec{y}) \in S'$ in Decode. Again, for an $O(\vec{x},\vec{y})$ query returning true, if $\vec{x} \notin S$ then the sequence number of the query is not stored in enc and therefore $(\vec{x},\vec{y}) \notin S'$. So, for an $O(\vec{x},\vec{y})$ query returning true, $(\vec{x},\vec{y}) \in S'$ iff $\vec{x} \in S$. Since, for all $(\vec{x},\vec{y})$ such that $O(\vec{x},\vec{y}) = \mathsf{true}$ we have that for all $i \in [k]$, $\vec{y}[i] = \vec{x}[\pi^{-1}(i)]$, it follows that S' forms a basis of vectors $(\vec{x},\vec{y})$ such that $O(\vec{x},\vec{y}) = \mathsf{true}$.

In $\mathsf{Decode}(\mathsf{enc})$, the simulation of $O(\vec{x},\vec{y})$ is perfect because

- If c is in enc, then $\vec{x} \in S$ in Encode. From the definition of S in Encode, it follows that $O(\vec{x},\vec{y})$ should return true.

- Otherwise we check if $(\vec{x},\vec{y}) \in \mathsf{span}(S')$ and return true if the check succeeds, false otherwise. This is correct since in S' is a basis of vectors $(\vec{x},\vec{y})$ such that $O(\vec{x},\vec{y}) = \mathsf{true}$.

The encoding is a set of $|S|$ query sequence numbers. Since there are at most q queries, the encoding space is at most $\binom{q}{|S|}$. Using $\mathcal{X}$ to be the set S_k, $\mathcal{Y}$ to be the set of all possible encodings, $\mathcal{R}$ to be the set

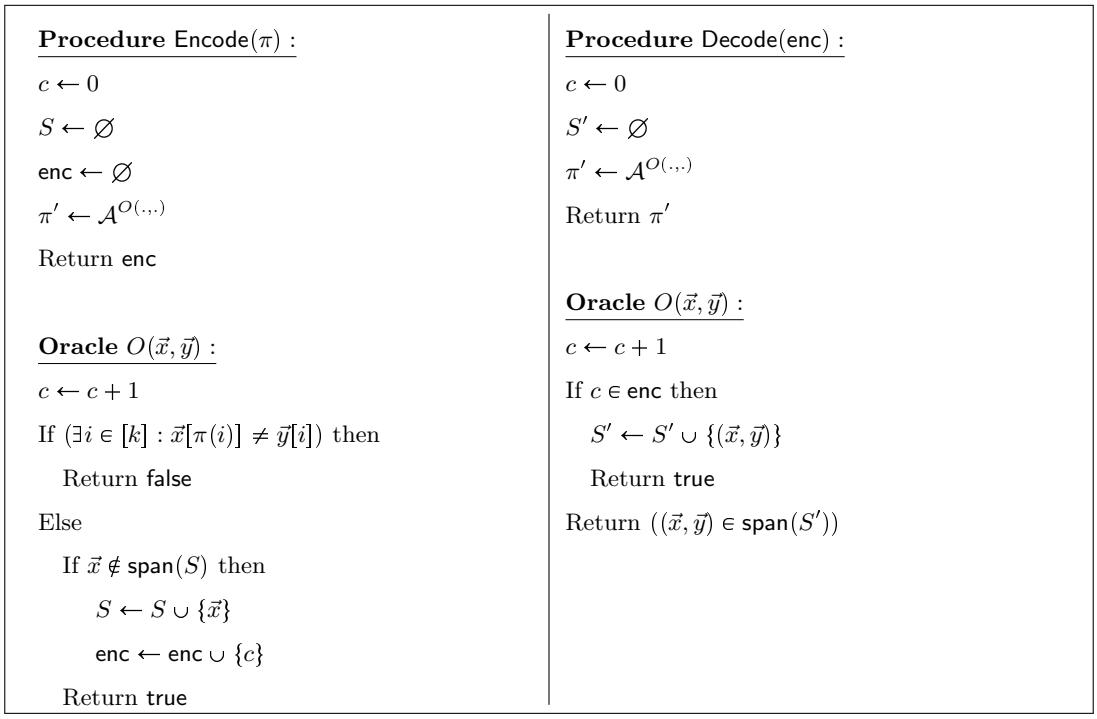


Figure 5.7: Encoding and decoding π using $\mathcal{A}$

of random tapes of $\mathcal{A}$, it follows from Proposition 11.1.1 that,

$$\mathsf{Pr}\left[\text{Decoding is sucessful}\right] \leqslant \frac{\binom{q}{|S|}}{k!} .$$

Since the simulation of $O(\vec{x}, \vec{y})$ is perfect in Decode, decoding is successful if $\mathbb{PG}(\mathcal{A}) \Rightarrow \mathsf{true}$. Therefore,

$$\mathsf{Pr}\left[\mathbb{PG}(\mathcal{A}) \Rightarrow \mathsf{true}\right] \leqslant \frac{\binom{q}{|S|}}{k!} \leqslant \frac{q^{|S|}}{k!} .$$

Since $\mathcal{A}$ is a (q, l)-query adversary, $|S| \leqslant l$. Thus, we have,

$$\mathsf{Pr}\left[\mathbb{PG}(\mathcal{A}) \Rightarrow \mathsf{true}\right] \leqslant \frac{q^l}{k!} \tag{5.8}$$

■

Reduction to $\mathbb{PG}$. We next show that the $\mathsf{Pr}\left[\mathsf{G}_2 \Rightarrow \mathsf{true}\right]$ is upper bounded in terms of the probability of a (q, l)-query adversary winning the game $\mathbb{PG}$.

Lemma 5.3.5. *There exists a (q, l)-query adversary $\mathcal{D}$ against the permutation game $\mathbb{PG}$ such that*

$$\mathsf{Pr}\left[\mathsf{G}_2 \Rightarrow \mathsf{true}\right] \leqslant \mathsf{Pr}\left[\mathbb{PG}(\mathcal{D}) \Rightarrow \mathsf{true}\right] + \frac{2q(2k + 3q + 2)}{p} + \frac{5q}{p} + \frac{k^2 + k + 2}{p} .$$

Procedure $\mathsf{PopulateSetsEval}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathsf{from})$:

If $\mathsf{from} = 1$ then

  If $\mathbf{c} \notin \mathcal{Y}_1$ then $\mathcal{X} \xleftarrow{\cup} \{\mathbf{c}\}$

  $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$

If $\mathsf{from} = 2$ then

  If $\mathbf{a} \in \mathcal{X} \backslash\ \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$

  If $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$

  $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$

Procedure $\mathsf{PopulateSetsO_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\mathsf{from} = 1$ then $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$

If $\mathsf{from} = 2$ then

  If $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$

  If $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$

  $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$

Figure 5.8: Subroutines $\mathsf{PopulateSetsEval}, \mathsf{PopulateSetsO_v}$

Proof. We transform $\mathcal{R}_1, \mathcal{R}_2$ playing G_2 to an adversary $\mathcal{D}$ playing the game $\mathbb{PG}$ through a sequence of intermediate games and use the upper bound on the probability of winning the game $\mathbb{PG}$ established previously to prove an upper bound on $\mathsf{Pr}[\mathsf{G}_2 \Rightarrow \mathsf{true}]$. In order to make the pseudocode for subsequent games compact we define the two subroutines $\mathsf{PopulateSetsEval}, \mathsf{PopulateSetsO_v}$ and invoke them from $\mathsf{Eval}, \mathsf{O_v}$. The subroutines $\mathsf{PopulateSetsEval}, \mathsf{PopulateSetsO_v}$ are formally described in Figure 5.8.

THE GAME G_4. We next describe game G_4 where we introduce some additional bookkeeping. In G_4, every valid label that is an input to $\mathcal{R}_1, \mathcal{R}_2$ or queried by $\mathcal{R}_1, \mathcal{R}_2$ or an answer to a query of $\mathcal{R}_1, \mathcal{R}_2$, is mapped to a polynomial in $\mathbb{Z}_p[I_1, \cdots, I_k, V, T_1, \cdots, T_{2q}]$ where q is the total number of $\mathsf{Eval}, \mathsf{O_v}$ queries made by $\mathcal{R}_1, \mathcal{R}_2$. The polynomial associated with label $\mathbf{a}$ is denoted by $\mathsf{p}_{\mathbf{a}}$. Similarly, we define Λ to be a mapping from polynomials to labels. For all labels $\mathbf{a} \in \mathcal{L}$, $\Lambda(\mathsf{p}_{\mathbf{a}}) = \mathbf{a}$. The mapping from labels to polynomials is done such that for every label $\mathbf{a}$ mapped to $\mathsf{p}_{\mathbf{a}}$,

$$\sigma^{-1}(\mathbf{a}) = \mathsf{p}_{\mathbf{a}}(i_1, \cdots, i_k, v, t_1, \cdots, t_{2q}) \ .$$

For compactness, let us denote $(i_1, \cdots, i_k, v, t_1, \cdots, t_{2q})$ by $\vec{i}$. Prior to running $\mathcal{R}_1$, polynomials $1, V, I_1, \cdots, I_k$, $I_1 V, \cdots, I_k V$ are assigned to $\mathsf{p}_{\sigma(1)}, \mathsf{p}_{\sigma(v)}, \mathsf{p}_{\sigma(i_1)}, \cdots, \mathsf{p}_{\sigma(i_k)}, \mathsf{p}_{\sigma(i_1 \cdot v)}, \cdots, \mathsf{p}_{\sigma(i_k \cdot v)}$ respectively and for all other labels $\mathbf{a} \in \mathcal{L}$, $\mathsf{p}_{\mathbf{a}} = \perp$. The function Λ is defined accordingly. For labels $\mathbf{a}$ queried by $\mathcal{R}_1, \mathcal{R}_2$ that have not been previously mapped to any polynomial (i.e. $\mathsf{p}_{\mathbf{a}} = \perp$), $\mathsf{p}_{\mathbf{a}}$ is assigned T_{new} (new starting from 1 and being incremented for every such label queried), the variable t_{new} is assigned the pre-image of the label and $\Lambda(T_{\mathsf{new}})$ is assigned $\mathbf{a}$. Since there are q queries (each with two inputs), there can be at most $2q$ labels that had not previously been mapped to any polynomial. Hence, the polynomials have variables $I_1, \cdots, I_k, V, T_1, \cdots, T_{2q}$.

For an $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, .)$ query where $\mathbf{c} = \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$, let $\mathsf{p}' = \mathsf{p}_{\mathbf{a}} + \mathsf{p}_{\mathbf{b}}$. From the definition of p, we

Game G_4 :

$\sigma \leftarrow\!\!\$\ \mathsf{InjFunc}(\mathbb{Z}_p, \mathcal{L})$;For each $\mathbf{a} \in \mathcal{L} : \mathsf{p}_{\mathbf{a}} \leftarrow \perp$; For each $\mathsf{p}' \in \mathbb{Z}_p[I_1, \cdots, I_k, V, T_1, \cdots, T_{2q}] : \Lambda(\mathsf{p}') \leftarrow \perp$

$i_1, \cdots, i_k, v \leftarrow\!\!\$\ \mathbb{Z}_p$;$\mathsf{p}_{\sigma(1)} \leftarrow 1; \Lambda(1) \leftarrow \sigma(1)$; If $\mathsf{p}_{\sigma(v)} = \perp$ then $\mathsf{p}_{\sigma(v)} \leftarrow V; \Lambda(V) \leftarrow \sigma(v)$

For each $j \in [k]$:

  If $\mathsf{p}_{\sigma(i_j)} = \perp$ then $\mathsf{p}_{\sigma(i_j)} \leftarrow I_j; \Lambda(I_j) \leftarrow \sigma(i_j)$

  If $\mathsf{p}_{\sigma(v \cdot i_j)} = \perp$ then $\mathsf{p}_{\sigma(v \cdot i_j)} \leftarrow VI_j; \Lambda(VI_j) \leftarrow \sigma(v \cdot i_j)$

$\mathsf{new} \leftarrow 0$; $\mathcal{X} \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}; \mathcal{Y}_1 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}; \mathcal{Z} \leftarrow \varnothing$

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)); \mathcal{Y}_2 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)\}$

$\pi \leftarrow\!\!\$\ \mathcal{S}_k; s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \sigma(1), \sigma(v), \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v))$

$\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s'_j) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$

Return $(\mathsf{win} \wedge |\mathcal{Z}| < l)$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\mathsf{p}_{\mathbf{a}} = \perp$ then $\mathsf{AssignPoly}(\mathbf{a})$

If $\mathsf{p}_{\mathbf{b}} = \perp$ then $\mathsf{AssignPoly}(\mathbf{b})$

$\mathsf{p}' \leftarrow \mathsf{p}_{\mathbf{a}} + \mathsf{p}_{\mathbf{b}}$

If $\Lambda(\mathsf{p}') = \perp$ then

  If $\exists \mathbf{c}' \in \mathcal{L} : \mathsf{p}_{\mathbf{c}'}(\vec{i}) = \mathsf{p}'(\vec{i})$ then $\Lambda(\mathsf{p}') \leftarrow \mathbf{c}'$

  Else

    $\Lambda(\mathsf{p}') \leftarrow \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$;

    $\mathsf{p}_{\Lambda(\mathsf{p}')} \leftarrow \mathsf{p}'$

$\mathsf{PopulateSetsEval}(\mathbf{a}, \mathbf{b}, \Lambda(\mathsf{p}'), \mathsf{from})$

Return $\Lambda(\mathsf{p}')$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\mathsf{p}_{\mathbf{a}} = \perp$ then $\mathsf{AssignPoly}(\mathbf{a})$

If $\mathsf{p}_{\mathbf{b}} = \perp$ then $\mathsf{AssignPoly}(\mathbf{b})$

$\mathsf{ans} \leftarrow (V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}})$

If $(v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i})) \neq \mathsf{ans}$ then

  $\mathsf{ans} \leftarrow (v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i}))$

$\mathsf{PopulateSetsO_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$

Return ans

Procedure $\mathsf{AssignPoly}(\mathbf{l})$:

$\mathsf{new} \leftarrow \mathsf{new} + 1$

$t_{\mathsf{new}} \leftarrow \sigma^{-1}(\mathbf{l})$

$\mathsf{p}_{\mathbf{l}} \leftarrow T_{\mathsf{new}}$

$\Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{l}$

Figure 5.9: G_4 introduces additional bookkeeping. The newly introduced changes compared to G_2 are highlighted.

have that $\mathsf{p}'(\vec{i}) = \sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b})$. If $\Lambda(\mathsf{p}') \neq \perp$, then by definition of Λ, we have $\Lambda(\mathsf{p}') = \mathbf{c}$. If $\Lambda(\mathsf{p}') = \perp$, then exactly one of the following two must be true.

1. The label $\mathbf{c}$ has been mapped to a polynomial which is different from p'. In this case $\mathsf{p}_{\mathbf{c}}(\vec{i}) = \mathsf{p}'(\vec{i})$ and $\Lambda(\mathsf{p}')$ is assigned $\mathbf{c}$.

2. The label $\mathbf{c}$ has not been mapped to any polynomial. In this case, $\mathsf{p}_{\mathbf{c}}$ is assigned p' and $\Lambda(\mathsf{p}')$ is assigned $\mathbf{c}$.

The label $\Lambda(\mathsf{p}')$ is returned as the answer of the Eval query. Note that the output of Eval is $\mathbf{c} = \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$ in all cases, i.e. it is the same as the output of Eval in G_2.

For an $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, .)$ query, we first assign the boolean value $V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}}$ to ans. Note that if ans is true, then $v \cdot \sigma^{-1}(\mathbf{a}) = \sigma^{-1}(\mathbf{b})$. However, we might have that $v \cdot \sigma^{-1}(\mathbf{a}) = \sigma^{-1}(\mathbf{b})$ and $V\mathsf{p}_{\mathbf{a}} \neq \mathsf{p}_{\mathbf{b}}$. When this happens, the boolean value $v(\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i}))$ is assigned to ans. Oracle $\mathsf{O_v}$ returns ans. From the definition of p, it follows that the value returned by $\mathsf{O_v}$ in G_4 is $(v \cdot \sigma^{-1}(\mathbf{a}) = \sigma^{-1}(\mathbf{b}))$ i.e. it is the same as the output of $\mathsf{O_v}$ in G_2.

Figure 5.9 formally describes G_4. The changes in G_4 compared to G_2 have been highlighted. We have already pointed out that the outputs of $\mathsf{O_v}, \mathsf{Eval}$ in G_4 are identical to those in G_2. Since the other changes involve only additional bookkeeping, the outputs of $\mathsf{G}_2, \mathsf{G}_4$ are identical. Therefore

$$\Pr[\mathsf{G}_4 \Rightarrow \mathsf{true}] = \Pr[\mathsf{G}_2 \Rightarrow \mathsf{true}] \ . \tag{5.9}$$

The game G_5. We define G_5 in Figure 5.10 where σ is lazily sampled. Since σ is lazily sampled, it is a partial function and we use previously defined notations $D(\sigma), R(\sigma), \overline{D(\sigma)}, \overline{R(\sigma)}$. There are no other changes in G_5 compared to G_4.

In G_4, σ is sampled uniformly at random from $\mathsf{InjFunc}(\mathbb{Z}_p, \mathcal{L})$. In G_5, if $\sigma(i)$ is previously not defined, it is sampled uniformly from the values in the range that do not have a pre-image and similarly if $\sigma^{-1}(\mathbf{x})$ is previously not defined, it is sampled uniformly from the values in the domain that do not have a image. Therefore, the distribution of σ on the points in $D(\sigma)$ in G_5 is identical to the distribution of σ on the points in $D(\sigma)$ if σ was sampled from $\mathsf{InjFunc}(\mathbb{Z}_p, \mathcal{L})$. So, in G_5, the distribution of σ on the points in $D(\sigma)$ is identical to that of G_4. Since σ is never evaluated on points outside $D(\sigma)$ in either games and there are no other changes in G_5, the outputs of $\mathsf{G}_4, \mathsf{G}_5$ are identical.

$$\Pr[\mathsf{G}_4 \Rightarrow \mathsf{true}] = \Pr[\mathsf{G}_5 \Rightarrow \mathsf{true}] \ . \tag{5.10}$$

The Games G_6 and G_7. Next, we define Games $\mathsf{G}_6, \mathsf{G}_7$. The changes introduced in G_6 from G_5 have been highlighted (as before). In G_7 we want to sample t_j's from $\mathbb{Z}_p$ instead of $\overline{D(\sigma)}$. So in G_6 we first sample them from $\mathbb{Z}_p$ and define a bad event if $\sigma(t_j) \neq \perp$ i.e. $t_j \in D(\sigma)$. If this bad event happens, then we resample t_j from $\overline{D(\sigma)}$. We then remove this resampling in G_7. The main procedure of $\mathsf{G}_6, \mathsf{G}_7$ are identical to G_5 and the only changes in the oracles are the $\mathsf{AssignPoly}$ procedure, hence we rewrite only the $\mathsf{AssignPoly}$ procedure in Figure 5.11.

Observe from the pseudocode that outputs of $\mathsf{G}_5, \mathsf{G}_6$ are identical. Therefore,

$$\Pr[\mathsf{G}_6 \Rightarrow \mathsf{true}] = \Pr[\mathsf{G}_5 \Rightarrow \mathsf{true}] \ . \tag{5.11}$$

Game G_5 :

For each $i \in \mathbb{Z}_p$: $\sigma(i) \leftarrow \perp$;For each $\mathbf{a} \in \mathcal{L}$: $\mathsf{p}_{\mathbf{a}} \leftarrow \perp$

For each $\mathsf{p}' \in \mathbb{Z}_p[I_1, \cdots, I_k, V, T_1, \cdots, T_{2q}]$: $\Lambda(\mathsf{p}') \leftarrow \perp$

$i_1, \cdots, i_k, v \leftarrow\!\!\$\ \mathbb{Z}_p$;$\sigma(1) \leftarrow\!\!\$\ \mathcal{L}$;$\mathsf{p}_{\sigma(1)} \leftarrow 1$; $\Lambda(1) \leftarrow \sigma(1)$

If $\mathsf{p}_{\sigma(v)} = \perp$ then $\sigma(v) \leftarrow\!\!\$\ \overline{R(\sigma)}$; $\mathsf{p}_{\sigma(v)} \leftarrow V$

$\Lambda(V) \leftarrow \sigma(v)$

For each $j \in [k]$:

  If $\mathsf{p}_{\sigma(i_j)} = \perp$ then $\sigma(i_j) \leftarrow\!\!\$\ \overline{R(\sigma)}$;$\mathsf{p}_{\sigma(i_j)} \leftarrow I_j$;$\Lambda(I_j) \leftarrow \sigma(i_j)$

  If $\mathsf{p}_{\sigma(v \cdot i_j)} = \perp$ then $\sigma(v \cdot i_j) \leftarrow\!\!\$\ \overline{R(\sigma)}$;$\mathsf{p}_{\sigma(v \cdot i_j)} \leftarrow VI_j$; $\Lambda(VI_j) \leftarrow \sigma(v \cdot i_j)$

$\mathsf{new} \leftarrow 0; \mathcal{X} \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}$; $\mathcal{Y}_1 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}$

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k))$

$\pi \leftarrow\!\!\$\ \mathcal{S}_k; \mathcal{Y}_2 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)\}; \mathcal{Z} \leftarrow \varnothing$

$s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \sigma(1), \sigma(v), \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v))$

$\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s'_j) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$

Return $(\mathsf{win} \wedge |\mathcal{Z}| < l)$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\mathsf{p}_{\mathbf{a}} = \perp$ then

  $\mathsf{AssignPoly}(\mathbf{a})$

If $\mathsf{p}_{\mathbf{b}} = \perp$ then

  $\mathsf{AssignPoly}(\mathbf{b})$

$\mathsf{p}' \leftarrow \mathsf{p}_{\mathbf{a}} + \mathsf{p}_{\mathbf{b}}$

If $\Lambda(\mathsf{p}') = \perp$ then

  If $\exists \mathbf{c}' \in \mathcal{L} : \mathsf{p}_{\mathbf{c}'}(\vec{i}) = \mathsf{p}'(\vec{i})$ then

    $\Lambda(\mathsf{p}') \leftarrow \mathbf{c}'$

  Else

    $\sigma(\mathsf{p}'(\vec{i})) \leftarrow\!\!\$\ \overline{R(\sigma)}; \Lambda(\mathsf{p}') \leftarrow \sigma(\mathsf{p}'(\vec{i}))$

    $\mathsf{p}_{\Lambda(\mathsf{p}')} \leftarrow \mathsf{p}'$

$\mathsf{PopulateSetsEval}(\mathbf{a}, \mathbf{b}, \Lambda(\mathsf{p}'), \mathsf{from})$

Return $\Lambda(\mathsf{p}')$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\mathsf{p}_{\mathbf{a}} = \perp$ then

  $\mathsf{AssignPoly}(\mathbf{a})$

If $\mathsf{p}_{\mathbf{b}} = \perp$ then

  $\mathsf{AssignPoly}(\mathbf{b})$

$\mathsf{ans} \leftarrow (V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}})$

If $(v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i})) \neq \mathsf{ans}$ then

  $\mathsf{ans} \leftarrow (v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i}))$

$\mathsf{PopulateSetsO_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$

Return ans

Procedure $\mathsf{AssignPoly}(\mathbf{l})$:

$\mathsf{new} \leftarrow \mathsf{new} + 1$

$t_{\mathsf{new}} \leftarrow\!\!\$\ \overline{D(\sigma)}; \sigma(t_{\mathsf{new}}) \leftarrow \mathbf{l}$

$\mathsf{p_l} \leftarrow T_{\mathsf{new}}$

$\Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{l}$

Figure 5.10: G_5 lazily samples σ. The newly introduced changes compared to G_4 are highlighted.

Game $\boxed{\mathsf{G}_6}, \mathsf{G}_7$:

Procedure $\mathsf{AssignPoly}(\mathbf{l})$:

$\mathsf{new} \leftarrow \mathsf{new} + 1$; $t_{\mathsf{new}} \leftarrow\!\!\$\ \mathbb{Z}_p$

If $\sigma(t_{\mathsf{new}}) \neq \perp$ then $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{t_{\mathsf{new}} \leftarrow\!\!\$\ \overline{D(\sigma)}}$

$\sigma(t_{\mathsf{new}}) \leftarrow \mathbf{l}$; $\mathsf{p}_{\mathbf{l}} \leftarrow T_{\mathsf{new}}$; $\Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{l}$

Figure 5.11: In G_7 we remove the restriction on t_j's. The main procedure of $\mathsf{G}_6, \mathsf{G}_7$ are identical to G_5 and the only changes in the oracles are the AssignPoly procedure, hence we write only the AssignPoly procedure. The changes in the procedure from that in G_5 have been highlighted. The boxed statement is present only in G_6.

After $\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k), \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)$ are sampled, the size of $D(\sigma)$ is at most $2k+2$. For every query $D(\sigma)$ can grow by size at most 3 (in case of Eval queries). Therefore, the size of $D(\sigma)$ is at most $2k+3q+2$. Since bad is set only when an element from $D(\sigma)$ is sampled,

$$\mathsf{Pr}\left[\mathsf{bad} = \mathsf{true} \text{ in } \mathsf{G}_7\right] \leqslant \frac{2q(2k+3q+2)}{p} .$$

It is evident from their pseudocodes that $\mathsf{G}_6, \mathsf{G}_7$ are identical-until-bad. From the Fundamental Lemma Game Playing we get,

$$\mathsf{Pr}\left[\mathsf{G}_6 \Rightarrow \mathsf{true}\right] \leqslant \mathsf{Pr}\left[\mathsf{G}_7 \Rightarrow \mathsf{true}\right] + \frac{2q(2k+3q+2)}{p} . \tag{5.12}$$

In G_7, note that $R(\sigma) = R(\Lambda)$ and $D(\sigma)$ is not used anywhere. Also the condition $\mathsf{p}_{\sigma(j)} \neq \perp$ is equivalent to the condition that there exists some p such that $\Lambda(\mathsf{p}) \neq \perp$ and $\mathsf{p}(\vec{i}) = j$ and the condition there exists $\mathbf{c}' \in \mathcal{L}$ such that $\mathsf{p}_{\mathbf{c}'}(\vec{i}) = \mathsf{p}'(\vec{i})$ is equivalent to the condition there exists some p such that $\Lambda(\mathsf{p}) \neq \perp$ and $\mathsf{p}(\vec{i}) = \mathsf{p}'(\vec{i})$. Hence, G_7 can be rewritten without σ.

The Games G_8 and G_9. Next, we introduce games $\mathsf{G}_8, \mathsf{G}_9$. In G_8 we remove σ (we had previously pointed out G_7 can be rewritten without σ), remove some redundant code and the bad event in G_7 and add new bad events when there is some p such that $\Lambda(\mathsf{p}) \neq \perp$ and $\mathsf{p}(\vec{i}) = \mathsf{p}'(\vec{i})$ in Eval and $(v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i})) \neq \mathsf{ans}$ in O_{v}. Hence outputs of $\mathsf{G}_7, \mathsf{G}_8$ are identical.

$$\mathsf{Pr}\left[\mathsf{G}_7 \Rightarrow \mathsf{true}\right] = \mathsf{Pr}\left[\mathsf{G}_8 \Rightarrow \mathsf{true}\right] . \tag{5.13}$$

Since $\vec{i}$ affects only the setting of bad in $\mathsf{Eval}, \mathsf{O}_{\mathsf{v}}$ we see that $\vec{i}$ may well have been sampled at the very end of G_9 and it can be checked if any of the $\mathsf{Eval}, \mathsf{O}_{\mathsf{v}}$ queries would have set bad to true. The degree of any polynomial is at most 2 since every polynomial has monomials only of the form I_j, VI_j, T_j, V and constants. We compute the probability of $\mathsf{bad} \leftarrow \mathsf{true}$ in G_9 in $\mathsf{Eval}, \mathsf{O}_{\mathsf{v}}$.

Game $\boxed{\mathsf{G}_8}, \mathsf{G}_9$:

For each $\mathbf{a} \in \mathcal{L}$: $\mathsf{p}_{\mathbf{a}} \leftarrow \perp$; For each $\mathsf{p}' \in \mathbb{Z}_p[I_1, \cdots, I_k, V, T_1, \cdots, T_{2q}]$: $\Lambda(\mathsf{p}') \leftarrow \perp$

$i_1, \cdots, i_k, v \leftarrow\!\!\$\ \mathbb{Z}_p$; $\Lambda(1) \leftarrow\!\!\$\ \mathcal{L}$; $\mathsf{p}_{\Lambda(1)} \leftarrow 1$

If $\exists \mathsf{p} : \Lambda(\mathsf{p}) \neq \perp \wedge \mathsf{p}(\vec{i}) = v$ then $\Lambda(V) \leftarrow \Lambda(\mathsf{p})$

else $\Lambda(V) \leftarrow\!\!\$\ \overline{R(\Lambda)}$; $\mathsf{p}_{\Lambda(\mathsf{p})} \leftarrow V$

For each $j \in [k]$:

  If $\exists \mathsf{p} : \Lambda(\mathsf{p}) \neq \perp \wedge \mathsf{p}(\vec{i}) = i_j$ then $\Lambda(I_j) \leftarrow \Lambda(\mathsf{p})$

  else $\Lambda(I_j) \leftarrow\!\!\$\ \overline{R(\Lambda)}$; $\mathsf{p}_{\Lambda(I_j)} \leftarrow I_j$

  If $\exists \mathsf{p} : \Lambda(\mathsf{p}) \neq \perp \wedge \mathsf{p}(\vec{i}) = i_j \cdot v$ then $\Lambda(VI_j) \leftarrow \Lambda(\mathsf{p})$

  else $\Lambda(VI_j) \leftarrow\!\!\$\ \overline{R(\Lambda)}$; $\mathsf{p}_{\Lambda(VI_j)} \leftarrow VI_j$

$\mathsf{new} \leftarrow 0$; $\mathcal{X} \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}$; $\mathcal{Y}_1 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}$

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)}(\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k))$

$\pi \leftarrow\!\!\$\ \mathcal{S}_k$; $\mathcal{Y}_2 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(VI_1), \cdots, \Lambda(VI_k)\}$; $\mathcal{Z} \leftarrow \varnothing$

$s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \Lambda(1), \Lambda(V), \Lambda(I_{\pi(1)} \cdot V), \cdots, \Lambda(I_{\pi(k)} \cdot V))$

$\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s'_j) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$

Return $(\mathsf{win} \wedge |\mathcal{Z}| < l)$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\mathsf{p}_{\mathbf{a}} = \perp$ then $\mathsf{AssignPoly}(\mathbf{a})$

If $\mathsf{p}_{\mathbf{b}} = \perp$ then $\mathsf{AssignPoly}(\mathbf{b})$

$\mathsf{p}' \leftarrow \mathsf{p}_{\mathbf{a}} + \mathsf{p}_{\mathbf{b}}$

If $\Lambda(\mathsf{p}') = \perp$ then

  If $\exists \mathsf{p} : \Lambda(\mathsf{p}) \neq \perp \wedge \mathsf{p}(\vec{i}) = \mathsf{p}'(\vec{i})$ then

    $\mathsf{bad} \leftarrow \mathsf{true}$

    $\boxed{\Lambda(\mathsf{p}') \leftarrow \Lambda(\mathsf{p})}$

  $\boxed{else}$ $\Lambda(\mathsf{p}') \leftarrow\!\!\$\ \overline{R(\Lambda)}$; $\mathsf{p}_{\Lambda(\mathsf{p}')} \leftarrow \mathsf{p}'$

$\mathsf{PopulateSetsEval}(\mathbf{a}, \mathbf{b}, \Lambda(\mathsf{p}'), \mathsf{from})$

Return $\Lambda(\mathsf{p}')$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\mathsf{p}_{\mathbf{a}} = \perp$ then $\mathsf{AssignPoly}(\mathbf{a})$

If $\mathsf{p}_{\mathbf{b}} = \perp$ then $\mathsf{AssignPoly}(\mathbf{b})$

$\mathsf{ans} \leftarrow (V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}})$

If $(v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i})) \neq \mathsf{ans}$ then

  $\mathsf{bad} \leftarrow \mathsf{true}$

  $\boxed{\mathsf{ans} \leftarrow (v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i}))}$

$\mathsf{PopulateSetsO_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$

Return ans

Procedure $\mathsf{AssignPoly}(\mathbf{l})$:

$\mathsf{new} \leftarrow \mathsf{new} + 1$

$t_{\mathsf{new}} \leftarrow\!\!\$\ \mathbb{Z}_p$

$\mathsf{p}_{\mathbf{l}} \leftarrow T_{\mathsf{new}}$

$\Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{l}$

Figure 5.12: Games $\mathsf{G}_8, \mathsf{G}_9$ introduce new bad events. We remove σ from these games. The changes from G_7 have been highlighted. The boxed statements are present only in G_8.

In Eval, bad is set to true if $\Lambda(\mathsf{p}') = \perp$ and for some label $\mathbf{c}'$ $\mathsf{p}_{\mathbf{c}'}(\vec{i}) = \mathsf{p}'(\vec{i})$. From the definition of Λ, it follows that $\Lambda(\mathsf{p}') = \perp$ implies $\mathsf{p}_{\mathbf{c}'} \neq \mathsf{p}'$. We therefore have $\mathsf{p}'(\vec{i}) - \mathsf{p}_{\mathbf{c}'}(\vec{i}) = 0$. Note that $\mathsf{p}' - \mathsf{p}_{\mathbf{c}'}$ is not

identically zero since $\Lambda(\mathsf{p}') = \perp$ and the degree of $\mathsf{p}' - \mathsf{p}_{\mathbf{c}'}$ is at most 2. Observe that the elements of $\vec{i}$ are sampled from $\mathbb{Z}_p$ and could have been sampled at the very end of G_9 because $\vec{i}$ does not affect anything other than setting bad. So, the probability $\mathsf{p}'(\vec{i}) - \mathsf{p}_{\mathbf{c}'}(\vec{i}) = 0$ is bounded by $\frac{2}{p}$ by Proposition 5.1.2. Since there are at most q Eval queries, the probability that bad is set to true due to this condition in any of the Eval queries in G_9 is bounded by $\frac{2q}{p}$ using the union bound.

In $\mathsf{O_v}$, bad is set to true is when $(v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i})), (V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}})$ are different boolean values. Note that when $(V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}}) = \mathsf{true}$ then $(v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i}))$. The only case when the two boolean values differ is when $(V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}}) = \mathsf{false}$ and $v\mathsf{p}_{\mathbf{a}}(\vec{i}) = \mathsf{p}_{\mathbf{b}}(\vec{i})$ i.e. $(V\mathsf{p}_{\mathbf{a}} - \mathsf{p}_{\mathbf{b}})$ is not identically zero but $v\mathsf{p}_{\mathbf{a}}(\vec{i}) - \mathsf{p}_{\mathbf{b}}(\vec{i}) = 0$. Note that the degree of $(V\mathsf{p}_{\mathbf{a}} - \mathsf{p}_{\mathbf{b}})$ is at most 3 (since $V\mathsf{p}_a$ can have degree at most 3) . Like the previous analysis since the elements of $\vec{i}$ are sampled from $\mathbb{Z}_p$ and could have been sampled at the very end of G_9, the probability $v\mathsf{p}_{\mathbf{a}}(\vec{i}) - \mathsf{p}_{\mathbf{b}}(\vec{i}) = 0$ is bounded by $\frac{3}{p}$ by Proposition 5.1.2. Since there are at most q $\mathsf{O_v}$ queries, the probability that bad is set to true due to this condition in any of the $\mathsf{O_v}$ queries in G_9 is bounded by $\frac{3q}{p}$ using the union bound. Therefore,

$$\mathsf{Pr}\left[\mathsf{bad} = \mathsf{true} \text{ in } \mathsf{G}_9\right] \leqslant \frac{5q}{p} .$$

Note that G_8 and G_9 are identical-until-bad. Therefore, using the Fundamental Lemma of Game Playing we get,

$$\mathsf{Pr}\left[\mathsf{G}_8 \Rightarrow \mathsf{true}\right] \leqslant \mathsf{Pr}\left[\mathsf{G}_9 \Rightarrow \mathsf{true}\right] + \frac{5q}{p} . \tag{5.14}$$

We note that in G_9, Λ becomes an injective function since for $\mathsf{p} \neq \mathsf{p}'$, $\Lambda(\mathsf{p}) \neq \Lambda(\mathsf{p}')$. Therefore, Λ^{-1} is well defined. This means that for all labels $\mathbf{l}$, $\mathsf{p}_{\mathbf{l}}$ can be replaced by $\Lambda^{-1}(\mathbf{l})$ throughout G_9. Also,$(V\mathsf{p}_{\mathbf{a}} = \mathsf{p}_{\mathbf{b}})$ is always returned from $\mathsf{O_v}$ and hence the check on ans can be omitted. The bad events can be removed from G_9 because it does not affect its execution in any way. That in turn implies t_j's do not affect the execution of G_9 and can also be removed. Since G_{10} only introduces a new bad event compared to G_9 and the bad event does not affect the output,

$$\mathsf{Pr}\left[\mathsf{G}_9 \Rightarrow \mathsf{true}\right] = \mathsf{Pr}\left[\mathsf{G}_{10} \Rightarrow \mathsf{true}\right] . \tag{5.15}$$

We need to compute the probability that bad is set to true in procedure $\mathsf{RestrictedSample}$ of G_{10}. First we compute the probability that $v \in \{0,1\}$,

$$\mathsf{Pr}_{v \leftarrow\$ \mathbb{Z}_p}\left[v \in \{0,1\}\right] = \frac{2}{p} .$$

Note that just before i_j sampled in G_{10} the size of $\mathcal{S} \cup \mathcal{S}'$ is at most $2j$. We compute the probability that $i_j \in \mathcal{S} \cup \mathcal{S}'$ for some $j \in [k]$,

$$\mathsf{Pr}_{i_j \leftarrow\$ \mathbb{Z}_p}\left[i_j \in \mathcal{S} \cup \mathcal{S}'\right] \leqslant \frac{2j}{p} .$$

Game $\mathsf{G}_{10}, \boxed{\mathsf{G}_{11}}$:

For each $\mathsf{p} \in \mathbb{Z}_p[I_1, \cdots, I_k, V, T_1, \cdots, T_{2q}]$: $\Lambda(\mathsf{p}) \leftarrow \perp$

$i_1, \cdots, i_k, v \leftarrow \mathsf{RestrictedSample}()$;$\Lambda(1) \leftarrow\!\!\$\ \mathcal{L}$

If $\exists \mathsf{p} : \Lambda(\mathsf{p}) \neq \perp \wedge \mathsf{p}(\vec{i}) = v$ then $\Lambda(V) \leftarrow \Lambda(\mathsf{p})$ else $\Lambda(V) \leftarrow\!\!\$\ \overline{R(\Lambda)}$

For each $j \in [k]$:

  If $\exists \mathsf{p} : \Lambda(\mathsf{p}) \neq \perp \wedge \mathsf{p}(\vec{i}) = i_j$ then $\Lambda(I_j) \leftarrow \Lambda(\mathsf{p})$ else $\Lambda(I_j) \leftarrow\!\!\$\ \overline{R(\Lambda)}$

  If $\exists \mathsf{p} : \Lambda(\mathsf{p}) \neq \perp \wedge \mathsf{p}(\vec{i}) = i_j \cdot v$ then $\Lambda(VI_j) \leftarrow \Lambda(\mathsf{p})$ else $\Lambda(VI_j) \leftarrow\!\!\$\ \overline{R(\Lambda)}$

$\mathsf{new} \leftarrow 0; \mathcal{X} \leftarrow \Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}; \mathcal{Y}_1 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}$

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,1),\mathsf{O_v}(.,.,1)}(\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k))$

$\pi \leftarrow\!\!\$\ \mathcal{S}_k; \mathcal{Y}_2 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(VI_1), \cdots, \Lambda(VI_k)\}; \mathcal{Z} \leftarrow \varnothing$

$s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2),\mathsf{O_v}(.,.,2)}(\phi, \Lambda(1), \Lambda(V), \Lambda(I_{\pi(1)} \cdot V), \cdots, \Lambda(I_{\pi(k)} \cdot V))$

$\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s'_j) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$

Return $(\mathsf{win} \wedge |\mathcal{Z}| < l)$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\Lambda^{-1}(\mathbf{a}) = \perp$ then

  $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{a}$

If $\Lambda^{-1}(\mathbf{b}) = \perp$ then

  $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{b}$

$\mathsf{p} \leftarrow \Lambda^{-1}(\mathbf{a}) + \Lambda^{-1}(\mathbf{b})$

If $\Lambda(\mathsf{p}) = \perp$ then

  $\Lambda(\mathsf{p}) \leftarrow\!\!\$\ \overline{R(\Lambda)}$

$\mathsf{PopulateSetsEval}(\mathbf{a}, \mathbf{b}, \Lambda(\mathsf{p}), \mathsf{from})$

Return $\Lambda(\mathsf{p})$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\Lambda^{-1}(\mathbf{a}) = \perp$ then

  $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{a}$

If $\Lambda^{-1}(\mathbf{b}) = \perp$ then

  $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{b}$

$\mathsf{PopulateSetsO_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$

Return $(V\Lambda^{-1}(\mathbf{a}) = \Lambda^{-1}(\mathbf{b}))$

Procedure $\mathsf{RestrictedSample}()$:

$v \leftarrow\!\!\$\ \mathbb{Z}_p$;If $v \in \{0, 1\}$ then $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{v \leftarrow\!\!\$\ \mathbb{Z}_p \backslash \{0, 1\}}$

$\mathcal{S} \leftarrow \{1\}; \mathcal{S}' \leftarrow \{v^{-1}\}$

For each $j \in [k]$:

  $i_j \leftarrow\!\!\$\ \mathbb{Z}_p$; If $i_j \in \mathcal{S} \cup \mathcal{S}'$ then $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{i_j \leftarrow\!\!\$\ \mathbb{Z}_p \backslash (\mathcal{S} \cup \mathcal{S}')}$

  $\mathcal{S} \xleftarrow{\cup} \{i_j\}; \mathcal{S}' \xleftarrow{\cup} \{v^{-1} \cdot i_j\}$

Return $i_1, \cdots, i_k, v$

Figure 5.13: In games $\mathsf{G}_{10}, \mathsf{G}_{11}$ $\mathsf{p}_{\mathbf{a}}$ has been replaced by $\Lambda^{-1}(\mathbf{a})$ for labels $\mathbf{a}$ and t_j's are removed. The procedure $\mathsf{AssignPoly}$ is no longer written separately, its code is written inline instead. In G_{11}, i_j's, v are sampled with the restriction that the set $\{1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdot v\}$ has cardinality $2k + 2$. The changes from G_9 have been highlighted. The boxed statements are present only in G_{11}.

Game G_{11} :

For each $\mathsf{p} \in \mathbb{Z}_p[I_1, \cdots, I_k, V, T_1, \cdots, T_{2q}]$: $\Lambda(\mathsf{p}) \leftarrow \perp; \Lambda(1) \leftarrow\!\!\$\ \mathcal{L}$

$\Lambda(V) \leftarrow\!\!\$\ \overline{R(\Lambda)}$

For each $j \in [k]$:

$\Lambda(I_j) \leftarrow\!\!\$\ \overline{R(\Lambda)}; \Lambda(VI_j) \leftarrow\!\!\$\ \overline{R(\Lambda)}$

$\mathsf{new} \leftarrow 0; \mathcal{X} \leftarrow \{CorrespondingString(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}; \mathcal{Y}_1 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}$

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)}(\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k))$

$\pi \leftarrow\!\!\$\ \mathcal{S}_k; \mathcal{Y}_2 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(VI_1), \cdots, \Lambda(VI_k)\}; \mathcal{Z} \leftarrow \varnothing$

$s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \Lambda(1), \Lambda(V), \Lambda(I_{\pi(1)} \cdot V), \cdots, \Lambda(I_{\pi(k)} \cdot V))$

$\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s'_j) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$

Return $(\mathsf{win} \wedge |\mathcal{Z}| < l)$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\Lambda^{-1}(\mathbf{a}) = \perp$ then

  $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{a}$

If $\Lambda^{-1}(\mathbf{b}) = \perp$ then

  $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{b}$

$\mathsf{p} \leftarrow \Lambda^{-1}(\mathbf{a}) + \Lambda^{-1}(\mathbf{b})$

If $\Lambda(\mathsf{p}) = \perp$ then

  $\Lambda(\mathsf{p}) \leftarrow\!\!\$\ \overline{R(\Lambda)}$

$\mathsf{PopulateSetsEval}(\mathbf{a}, \mathbf{b}, \Lambda(\mathsf{p}), \mathsf{from})$

Return $\Lambda(\mathsf{p})$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\Lambda^{-1}(\mathbf{a}) = \perp$ then

  $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{a}$

If $\Lambda^{-1}(\mathbf{b}) = \perp$ then

  $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{b}$

$\mathsf{PopulateSetsO_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$

Return $(V\Lambda^{-1}(\mathbf{a}) = \Lambda^{-1}(\mathbf{b}))$

Figure 5.14: Game G_{11} has been written more compactly than the one in Figure 5.13 by removing redundant code.

In G_{10}, since bad can be set to true only in $\mathsf{RestrictedSample}$, the probability that bad is set to true. Therefore, using the union bound,

$$\Pr\left[\mathsf{bad} = \mathsf{true} \text{ in } \mathsf{G}_{10}\right] \leqslant \frac{2}{p} + \sum_{j=1}^{k} \frac{2j}{p} \leqslant \frac{k^2 + k + 2}{p} .$$

Note that G_{10} and G_{11} are identical-until-bad. Therefore, using the Fundamental Lemma of Game Playing we get,

$$\Pr\left[\mathsf{G}_{10} \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathsf{G}_{11} \Rightarrow \mathsf{true}\right] + \frac{k^2 + k + 2}{p} . \tag{5.16}$$

Procedure $\mathsf{PolyMultCheck}(\mathsf{p_a}, \mathsf{p_b})$:

If $\exists j : (\mathsf{coefficient}(\mathsf{p_a}, T_j) \neq 0 \vee \mathsf{coefficient}(\mathsf{p_a}, VI_j) \neq 0)$ then return false

If $\exists j : (\mathsf{coefficient}(\mathsf{p_b}, T_j) \neq 0 \vee \mathsf{coefficient}(\mathsf{p_b}, I_j) \neq 0)$ then return false

If $\mathsf{coefficient}(\mathsf{p_b}, V) \neq \mathsf{coefficient}(\mathsf{p_a}, 1)$ then return false

For each $j \in [k]$: $\vec{x}[j] \leftarrow \mathsf{coefficient}(\mathsf{p_a}, I_j)$; $\vec{y}[j] \leftarrow \mathsf{coefficient}(\mathsf{p_b}, VI_j)$

If $O(\vec{x}, \vec{y}) = \mathsf{true}$ then

  If $\vec{x} \notin \mathsf{span}(S)$ then $S \xleftarrow{\cup} \{\vec{x}\}; \mathcal{Z}' \xleftarrow{\cup} \{\mathbf{a}\}$

  If $|S| = l$ then ABORT

  Return true

Else return false

Figure 5.15: **Subroutine** $\mathsf{PolyMultCheck}$ **for simulating** $\mathsf{O_\vee}$**.** In particular, $\mathsf{coefficient}(\mathsf{p}, M)$ returns the coefficient of the monomial M in the polynomial p. The sets S and $\mathcal{Z}'$ have no effect on the behavior, and are only used in the analysis of $\mathcal{D}$. The symbol ABORT indicates that $\mathcal{D}$ aborts and outputs $\perp$.

THE GAMES G_{10} AND G_{11}. We introduce a procedure named $\mathsf{RestrictedSample}$ in order to sample $i_1, \cdots, i_k, v$ instead of sampling them uniformly at random, replace p by Λ^{-1}, remove t_j's and the bad events in G_9. In G_{11}, $\mathsf{RestrictedSample}$ sets bad to true if $v \in \{0, 1\}$ or if the cardinality of the set $\{1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdot v\}$ is less than $2k+2$. In G_{11}, $\mathsf{RestrictedSample}$ samples these values with the restriction that $|\{1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdot v\}| = 2k + 2$.

We note that since $i_1, \cdots, i_k, v$ sampled in G_{11} satisfy $|1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdot v| = 2k + 2$, the three set of if statements after sampling v will always evaluate to true. Since $i_1, \cdots, i_k, v$ do not affect the execution of G_{11} at any other point, the sampling of $i_1, \cdots, i_k, v$ and the three set of if statements after sampling v can be omitted from G_{11}. Therefore, G_{11} can be rewritten as shown in Figure 5.14. We next upper bound $\mathsf{Pr}[\mathsf{G}_4 \Rightarrow \mathsf{true}]$ in terms of $\mathsf{Pr}[\mathsf{G}_{11} \Rightarrow \mathsf{true}]$ in Lemma 5.3.6.

Lemma 5.3.6. *For the games* $\mathsf{G}_4, \mathsf{G}_{11}$*, we have,*

$$\mathsf{Pr}[\mathsf{G}_4 \Rightarrow \mathsf{true}] \leqslant \mathsf{Pr}[\mathsf{G}_{11} \Rightarrow \mathsf{true}] + \frac{2q(2k + 3q + 2)}{p} + \frac{5q}{p} + \frac{k^2 + k + 2}{p} .$$

Proof. Combining (5.10) to (5.16) we get,

$$\mathsf{Pr}[\mathsf{G}_4 \Rightarrow \mathsf{true}] \leqslant \mathsf{Pr}[\mathsf{G}_{11} \Rightarrow \mathsf{true}] + \frac{2q(2k + 3q + 2)}{p} + \frac{5q}{p} + \frac{k^2 + k + 2}{p} .$$

■

Adversary $\mathcal{D}$:

For each $\mathsf{p} \in \mathbb{Z}_p[I_1, \cdots, I_k, V, K_1, \cdots, K_k, T_1, \cdots, T_{2q}]$: $\Lambda(\mathsf{p}) \leftarrow \perp$

$\Lambda(1) \leftarrow\!\!\$\ \mathcal{L}; \Lambda(V) \leftarrow\!\!\$\ \overline{R(\Lambda)}$

For each $j \in [k]$:

  $\Lambda(I_j) \leftarrow\!\!\$\ \overline{R(\Lambda)}$; $\Lambda(K_j) \leftarrow\!\!\$\ \overline{R(\Lambda)}$

$\mathsf{new} \leftarrow 0; \mathcal{X} \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}; \mathcal{Y}_1 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k)\}$

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)}(\Lambda(1), \Lambda(V), \Lambda(I_1), \cdots, \Lambda(I_k))$

$\mathcal{Y}_2 \leftarrow \{\Lambda(1), \Lambda(V), \Lambda(K_1), \cdots, \Lambda(K_k)\}; \mathcal{Z} \leftarrow \emptyset; \mathcal{Z}' \leftarrow \emptyset; S \leftarrow \emptyset$

$s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \Lambda(1), \Lambda(V), \Lambda(K_1), \cdots, \Lambda(K_k))$

$\mathsf{win}' \leftarrow (\{s_1, \cdots, s_l\} = \{s'_1, \cdots, s'_l\}) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$

If $\mathsf{win}' = \mathsf{true}$ then

  For each $i, j \in [k]$: If $s_i = s'_j$ then $\pi(i) = j$

  Return π

Else return $\perp$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\Lambda^{-1}(\mathbf{a}) = \perp$ then

  $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{a}$

If $\Lambda^{-1}(\mathbf{b}) = \perp$ then

  $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{b}$

$\mathsf{p} \leftarrow \mathsf{p_a} + \mathsf{p_b}$

If $\Lambda(\mathsf{p}) = \perp$ then $\Lambda(\mathsf{p}) \leftarrow\!\!\$\ \overline{R(\Lambda)}$

$\mathsf{PopulateSetsEval}(\mathbf{a}, \mathbf{b}, \Lambda(\mathsf{p}), \mathsf{from})$

Return $\Lambda(\mathsf{p})$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\Lambda^{-1}(\mathbf{a}) = \perp$ then

  $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{a}$

If $\Lambda^{-1}(\mathbf{b}) = \perp$ then

  $\mathsf{new} \leftarrow \mathsf{new} + 1; \Lambda(T_{\mathsf{new}}) \leftarrow \mathbf{b}$

$\mathsf{PopulateSetsO_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$

$\mathsf{PolyMultCheck}(\mathsf{p_a}, \mathsf{p_b})$

Procedure $\mathsf{PopulateSetsEval}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathsf{from})$:

If $\mathsf{from} = 1$ then

  If $\mathbf{c} \notin \mathcal{Y}_1$ then $\mathcal{X} \xleftarrow{\cup} \{\mathbf{c}\}$

  $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$

If $\mathsf{from} = 2$ then

  If $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$

    If $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$

  $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$

Procedure $\mathsf{PopulateSetsO_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\mathsf{from} = 1$ then $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$

If $\mathsf{from} = 2$ then

  If $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$

  If $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$

  $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$

Figure 5.16: Adversary $\mathcal{D}$ which plays the permutation game $\mathbb{PG}$. The changes in $\mathcal{D}$ compared to G_{11} have been highlighted.

THE ADVERSARY $\mathcal{D}$. Next, we construct the adversary $\mathcal{D}$ that plays $\mathbb{PG}$ by simulating G_{11} to $\mathcal{R}_1, \mathcal{R}_2$, where the permutation π is the secret permutation from $\mathbb{PG}$. As we will discuss below, the core of the adversary $\mathcal{D}$ will boil down to properly simulating the O_v oracle using the O oracle from $\mathbb{PG}$ *and* simulating the labels $\sigma(i_{\pi(j)})$ (and the associated polynomials) correctly without knowing π. After a correct simulation, $\mathcal{D}$ will simply extract the permutation π.

To see how this can be done, let us first have a closer look at G_{11}. Let us introduce the shorthand $K_j = VI_{\pi(j)}$ for $j \in [k]$. With this notation, every polynomial input to or output from Eval is a linear combination of the monomials $1, I_1, \ldots, I_k, V, K_1, \ldots, K_k, T_1, T_2, \ldots$. Now, it is convenient to slightly rethink the check of whether $V\mathsf{p_a} = \mathsf{p_b}$ within O_v with this notation. First off, we observe that if either of the polynomial contains a monomial of the form T_i, the check fails. In fact, it is immediately clear that the check can only possibly succeed is if $\mathsf{p_a}$ is a linear combination of 1 and the I_j's and $\mathsf{p_b}$ is a linear combination of V and the K_j's. Now, assume that

$$\mathsf{p_a}(I_1, \ldots, I_k) = a_0 + \sum_{j=1}^{k} \vec{x}[j] \cdot I_j \ ,$$

$$\mathsf{p_b}(V, K_1, \ldots K_k) = b_0 \cdot V + \sum_{j=1}^{k} \vec{y}[j] \cdot K_j \ .$$

Then, $V \cdot \mathsf{p}_a = \mathsf{p}_b$ if and only if $a_0 = b_0$ *and* $\vec{y}[j] = \vec{x}[\pi(j)]$ for all $j \in [k]$. If we are now in Game $\mathbb{PG}$, and π is the chosen permutation, then this is equivalent to $O(\vec{x}, \vec{y}) = \mathsf{true}$ and $a_0 = b_0$.

This leads naturally to the adversary $\mathcal{D}$, which we formally describe in Figure 5.16. The adversary will simply sample labels $\mathbf{f}_1, \ldots, \mathbf{f}_k$ for $\sigma(v \cdot i_{\pi(1)}), \ldots, \sigma(v \cdot i_{\pi(k)})$, and associate with them polynomials in the variables $K_1, \ldots, K_j$. Other than that, it simulates the game G_{11}, with the exception that the check $V \cdot \mathsf{p_a} = \mathsf{p_b}$ is not implemented using the above approach – summarized in Figure 5.15. Note that $\mathcal{D}$ aborts when $|S| = l$ and makes at most q queries to O. Thus $\mathcal{D}$ is a (q, l)-query adversary against $\mathbb{PG}$. If $\mathcal{D}$ does not abort, then its simulation of G_{11} is perfect. If $\mathsf{G}_{11} \Rightarrow \mathsf{true}$ and $\mathcal{D}$ does not abort, then win' shall be true and $\mathcal{D}$ will output the correct π.

The rest of the proof will now require proving that whenever G_{11} outputs true our adversary $\mathcal{D}$ will never abort due to the check $|S| = l$. Since $\mathsf{G}_{11} \Rightarrow \mathsf{true}$ only if $|\mathcal{Z}| < l$, the following lemma implies that $\mathcal{D}$ does not abort if $\mathsf{G}_{11} \Rightarrow \mathsf{true}$.

Lemma 5.3.7. *Let* $(\vec{x}_1, \vec{y}_1), \cdots, (\vec{x}_u, \vec{y}_u)$ *be the queries made by* $\mathcal{D}$ *to* O *which return* true. *Then,*

$$\mathsf{rank}(\vec{x}_1, \cdots, \vec{x}_u) \leqslant |\mathcal{Z}| \ .$$

Proof. We introduce an operator $\mathsf{proj}(\mathsf{p})$ which maps a polynomial to a vector in $\mathbb{Z}_p^k$ consisting of the coefficients of the monomials $I_1, \ldots, I_k$. We will show below that for all $i \in [u]$, there exists a set $\mathcal{Z}_i \subseteq \mathcal{Z}$

such that $\vec{x}_i \in \mathsf{span}(\mathsf{proj}(\Lambda^{-1}(\mathcal{Z}_i)))$. From this, the claim is immediate, because

$$\begin{aligned}\mathsf{rank}(\vec{x}_1, \cdots, \vec{x}_u) &\leqslant \mathsf{rank}\left(\bigcup_{i=1}^{u} \mathsf{proj}(\Lambda^{-1}(\mathcal{Z}_i))\right) \\ &\leqslant \left|\bigcup_{i=1}^{u} \mathsf{proj}(\Lambda^{-1}(\mathcal{Z}_i))\right| \leqslant \left|\bigcup_{i=1}^{u} \mathcal{Z}_i\right| \leqslant |\mathcal{Z}| \ .\end{aligned}$$

First off, note that every non-zero query $(\vec{x}_i, \vec{y}_i)$ that returns true is made during the simulation of the execution of $\mathcal{R}_2$. This is because such a query requires the polynomial associated with $\vec{y}_i$ to contain a monomial in K_j for at least some $j \in [k]$, and such polynomials can only occur in the execution of $\mathcal{R}_2$. Also note that every vector $\vec{x}_i$ is associated with a polynomial p_i. This polynomial *must* have form $a_0 + \sum_{j=1}^{k} \vec{x}_i[j] \cdot I_j$, as discussed before.

Let us look at an execution of $\mathcal{D}$ after-the-fact, and more specifically, the simulated execution of $\mathcal{R}_1, \mathcal{R}_2$ within $\mathcal{D}$. Let $\mathbf{a}_i$ be the label associated with the polynomial p_i. If it has not been output by an Eval query ever, then it must be that $\mathbf{a}_i = \Lambda(I_j)$ for some $j \in [k]$. Because $\mathbf{a}_i \in \mathcal{X}$, then $\mathbf{a}_i \in \mathcal{Z}$, and thus $\mathcal{Z}_i = \{\mathbf{a}_i\}$. Similarly, assume that $\mathbf{a}$ was output last by an Eval query of $\mathcal{R}_1$. Then, it must be that $\mathbf{a}_i$ has been added to $\mathcal{X}$, because every output of an Eval query by $\mathcal{R}_1$ is aded to $\mathcal{X}$, *unless* the string $\mathbf{a}_i$ was associated earlier with one of the T_z monomials. But this label would not lead the oracle query to output 1. Therefore, $\mathbf{a}_i$ is added to $\mathcal{Z}$ upon $\mathcal{R}_2$'s corresponding $\mathsf{O_v}$ query, and we can set $\mathcal{Z}_i = \{\mathbf{a}_i\}$.

We are left with the harder case that $\mathbf{a}_i$ is the output of an Eval query of $\mathcal{R}_2$. Note that because of this, it is not hard to see that we can write

$$\mathsf{p}_i = \sum_{j=1}^{r} \gamma_j \cdot \Lambda^{-1}(\mathbf{b}_j) \ ,$$

where $\mathbf{b}_1, \ldots, \mathbf{b}_j$ are inputs to Eval queries made by $\mathcal{R}_2$. This in turn implies that

$$\vec{x}_i = \mathsf{proj}(\mathsf{p}_i) = \sum_{j=1}^{r} \gamma_j \cdot \mathsf{proj}(\Lambda^{-1}(\mathbf{b}_j)) \ .$$

We now claim that if $\mathsf{proj}(\Lambda^{-1}(\mathbf{b}_j)) \neq \vec{0}$, then it must be that $\mathbf{b}_j \in \mathcal{X}$, and therefore it is added to $\mathcal{Z}$. (Either within the Eval query, or in an earlier $\mathsf{O_v}$ query using it.)

Note that if $\mathsf{proj}(\Lambda^{-1}(\mathbf{b}_j)) \neq 0$, then $\mathbf{b}_j$ is not a fresh label input to an Eval query by either of $\mathcal{R}_1$ or $\mathcal{R}_2$ (as it would have been associated with T_z, for some z), and it is not one of the labels $\Lambda(1), \Lambda(V), \Lambda(K_1), \ldots, \Lambda(K_k)$ input to $\mathcal{R}_2$. So, $\mathbf{b}_j$ is either $\Lambda(I_1), \ldots, \Lambda(I_k)$, *or* the output of an Eval query by $\mathcal{R}_1$ which is added to $\mathcal{X}$. In both cases, $\mathbf{b}_j \in \mathcal{X}$. Hence, we can now define

$$\mathcal{Z}_i = \left\{\mathbf{b}_j \ : \ j \in [k] \ , \ \ \mathsf{proj}(\Lambda^{-1}(\mathbf{b}_j)) \neq \vec{0}\right\} \ .$$

This concludes the proof. ■

We have established that if G_{11} outputs true, then $\mathcal{D}$ will not abort and hence $\mathcal{D}$ simulates G_{11} to $\mathcal{R}_1, \mathcal{R}_2$ perfectly. If $\mathsf{win} = \mathsf{true}$ in G_{11}, the checks by $\mathcal{D}$ succeed and $\mathcal{D}$ outputs the correct permutation and wins $\mathbb{PG}$. Therefore, $\mathcal{D}$ is a (q, l)-query adversary such that $\mathbb{PG}(\mathcal{D}) \Rightarrow \mathsf{true}$ if $\mathsf{G}_{11} \Rightarrow \mathsf{true}$. Hence,

$$\mathsf{Pr}\left[\mathsf{G}_{11} \Rightarrow \mathsf{true}\right] \leqslant \mathsf{Pr}\left[\mathbb{PG}(\mathcal{D}) \Rightarrow \mathsf{true}\right] . \tag{5.17}$$

Combining Lemma 5.3.6 and (5.9),(5.17) we get,

$$\mathsf{Pr}\left[\mathsf{G}_2 \Rightarrow \mathsf{true}\right] \leqslant \mathsf{Pr}\left[\mathbb{PG}(\mathcal{D}) \Rightarrow \mathsf{true}\right] + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{k^2+k+2}{p} . \tag{5.18}$$

■

Combining (5.8) and (5.18) we get,

$$\mathsf{Pr}\left[\mathsf{G}_2 \Rightarrow \mathsf{true}\right] \leqslant \frac{q^l}{k!} + \frac{2q(2k+3q+2)}{p} + \frac{5q}{p} + \frac{k^2+k+2}{p} .$$

■

5.3.4 Memory Lower Bound when $|\mathcal{Z}| \geqslant l$ (Proof of Lemma 5.2.5)

Recall that we need to prove the following lemma, which we do by using a compression argument.

Lemma 5.3.8. *If the size of the state ϕ output by $\mathcal{R}_1$ is s bits and $(\mathcal{R}_1, \mathcal{R}_2)$ make q queries in total in the game G_3 as defined in Figure 5.5, then*

$$\mathsf{Pr}\left[\mathsf{G}_3 \Rightarrow \mathsf{true}\right] \leqslant 2 \cdot 2^{\frac{s}{2}} \left(\frac{8q^2(2k+2+3q)}{p}\right)^{\frac{l}{2}} \left(1+\frac{6q}{p}\right)^{\frac{2q-l}{2}} + \frac{k^2+k+2}{p} .$$

Proof. Our proof does initial game hopping, with easy transitions. It first introduces a new game, G_{12} whose minor difference from game G_3 is that it samples $i_1, \cdots, i_k, v$ using $\mathsf{RestrictedSample}$ which was previously used in game G_{11}. It adds a bad flag while sampling $i_1, \cdots, i_k, v$ which is set to true if v is in $\{0, 1\}$ or if $|1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdot v| < 2k+2$. The bad event does not affect the output of G_{12} in any way. Observe that even though the sampling of $i_1, \cdots, i_k, v$ is written in a different manner in G_{12}, it is identical to that in G_3. In all other respects these two games are identical.

$$\mathsf{Pr}\left[\mathsf{G}_3 \Rightarrow \mathsf{true}\right] = \mathsf{Pr}\left[\mathsf{G}_{12} \Rightarrow \mathsf{true}\right] . \tag{5.19}$$

Games $\mathsf{G}_{12}, \mathsf{G}_{13}$ differ in the procedure $\mathsf{RestrictedSample}$ and the condition to return true. Note that the conditions of bad being set to true is identical in $\mathsf{G}_{12}, \mathsf{G}_{13}$ and given that bad is not set to true, G_{13} returns true whenever G_{12} returns true. Therefore, using the Fundamental Lemma of Game Playing

$$\mathsf{Pr}\left[\mathsf{G}_{12} \Rightarrow \mathsf{true}\right] \leqslant \mathsf{Pr}\left[\mathsf{G}_{13} \Rightarrow \mathsf{true}\right] + \mathsf{Pr}\left[\mathsf{bad} = \mathsf{true} \text{ in } \mathsf{G}_{13}\right] .$$

Game $\boxed{\mathsf{G}_{12}}, \boxed{\mathsf{G}_{13}}$:

$\sigma \xleftarrow{\$} \mathsf{InjFunc}(\mathbb{Z}_p, \mathcal{L}); i_1, \cdots, i_k, v \leftarrow \mathsf{RestrictedSample}()$

$\mathcal{X} \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}; \mathcal{Y}_1 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}$

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k))$

$\pi \xleftarrow{\$} \mathcal{S}_k; \mathcal{Y}_2 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)\}; \mathcal{Z} \leftarrow \varnothing$

$s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \sigma(1), \sigma(v), \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v))$

$\mathsf{win} \leftarrow (\forall j \in [k] : s_{\pi(j)} = s'_j) \wedge (\forall j, l \in [k] : j \neq l \implies s_j \neq s_l \wedge s'_j \neq s'_l)$

Return $(\boxed{\mathsf{win} \wedge} |\mathcal{Z}| \geqslant l)$

Procedure RestrictedSample() :

$v \xleftarrow{\$} \mathbb{Z}_p$; If $v \in \{0, 1\}$ then $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{v \xleftarrow{\$} \mathbb{Z}_p \backslash \{0, 1\}}$

$\mathcal{S} \leftarrow \{1\}; \mathcal{S}' \leftarrow \{v^{-1}\}$

For each $j \in [k]$

  $i_j \xleftarrow{\$} \mathbb{Z}_p$; If $i_j \in \mathcal{S} \cup \mathcal{S}'$ then $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{i_j \xleftarrow{\$} \mathbb{Z}_p \backslash (\mathcal{S} \cup \mathcal{S}')}$

  $\mathcal{S} \xleftarrow{\cup} \{i_j\}; \mathcal{S}' \xleftarrow{\cup} \{v^{-1} \cdot i_j\}$

Return $i_1, \cdots, i_k, v$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

$\mathbf{c} \leftarrow \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$

If $\mathsf{from} = 1$ then

  If $\mathbf{c} \notin \mathcal{Y}_1$ then $\mathcal{X} \xleftarrow{\cup} \{\mathbf{c}\}$

  $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$

If $\mathsf{from} = 2$ then

  If $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$

  If $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$

  $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$

Return $\mathbf{c}$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, \mathsf{from})$:

If $\mathsf{from} = 1$ then $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$

If $\mathsf{from} = 2$ then

  If $\mathbf{a} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{a}\}$

  If $\mathbf{b} \in \mathcal{X} \backslash \mathcal{Y}_2$ then $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{b}\}$

  $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$

Return $(v \cdot \sigma^{-1}(\mathbf{a}) = \sigma^{-1}(\mathbf{b}))$

Figure 5.17: Games $\mathsf{G}_{12}, \mathsf{G}_{13}$. The statement within the thinner box is present only in G_{12} and the statement within the thicker box is present only in G_{13}. The newly introduced changes compared to G_3 are highlighted.

From our analysis in the proof of Lemma 5.3.6, we have established that the probability of bad being set to true in RestrictedSample is at most $\frac{k^2+k+2}{p}$. Since in G_{13} bad is set only in RestrictedSample, the probability

of bad being set to true is the same. Hence, we get,

$$\Pr\left[\mathsf{G}_{12} \Rightarrow \mathsf{true}\right] \leqslant \Pr\left[\mathsf{G}_{13} \Rightarrow \mathsf{true}\right] + \frac{k^2+k+2}{p} \,. \tag{5.20}$$

The compression argument. We assume $\Pr\left[\mathsf{G}_{13} \Rightarrow \mathsf{true}\right] = 2\varepsilon$. We say a σ is "good" in G_{13} if

$$\Pr\left[\mathsf{G}_{13} \Rightarrow \mathsf{true} \,\middle|\, \sigma \text{ was sampled in } \mathsf{G}_{13}\right] \geqslant \varepsilon \,. \tag{5.21}$$

It follows from Markov's inequality that at least ε fraction of σ's are "good". The following lemma captures the essence of our compression argument.

Lemma 5.3.9. *If the state output by $\mathcal{R}_1$ in G_{13} has size s bits, all the "good" σ's can be encoded such that the size of the encoding space is at most*

$$2^s p! \left(1+\frac{6q}{p}\right)^{(2q-l)} \left(\frac{p}{8q^2(2k+2+3q)}\right)^{-l} ,$$

and decoded correctly with probability at least ε.

We first give some intuition regarding how we achieve compression and show how Lemma 5.3.9 leads to an upper bound on $\Pr\left[\mathsf{G}_3 \Rightarrow 1\right]$. We defer the proof of Lemma 5.3.9 to Section 5.3.5.

Intuition regarding compression. Observe in G_{13}, the labels in $\mathcal{Z}$ were queried by $\mathcal{R}_2$ (these labels were not seen by $\mathcal{R}_2$ before they were queried) and were answers to $\mathcal{R}_1$ and were not seen by $\mathcal{R}_1$ before the query. The core idea is that for all $\mathbf{a} \in \mathcal{L}\backslash\mathcal{Z}$, we store exactly one of $\mathbf{a}$ or its pre-image in the encoding and for all labels in $\mathcal{Z}$, we store neither the label nor its pre-image. Since $\mathcal{R}_2$ queries all the labels in $\mathcal{Z}$, these labels can be found by running $\mathcal{R}_2$ while decoding. Since all the labels in $\mathcal{Z}$ are answers to queries of $\mathcal{R}_1$ and were not seen by $\mathcal{R}_1$ before the query, their pre-images can be figured out while running $\mathcal{R}_1$.

High level outlines of Encode, Decode. In Encode, we simulate the steps of G_{13} to $\mathcal{R}_1, \mathcal{R}_2$, including bookkeeping and then run $\mathcal{R}_1$ again assuming the particular σ we are compressing is sampled in G_{13}. In Decode, we run $\mathcal{R}_2$ and then $\mathcal{R}_1$ to recover σ. We treat the values $i_1, \cdots, i_k, v, \pi$ as part of the common randomness provided to Encode, Decode (we assume they are sampled from the same distribution they are sampled from in G_{13}). The random tapes of $\mathcal{R}_1, \mathcal{R}_2$ can also be derived from the common randomness of Encode, Decode. For simplicity, we do not specify this explicitly in the algorithms and treat $\mathcal{R}_1, \mathcal{R}_2$ as deterministic.

Running $\mathcal{R}_2$. First off, we assume that $\mathcal{R}_1$ queries labels that it has "seen" before and $\mathcal{R}_2$ queries labels that $\mathcal{R}_1$ has "seen" or it has "seen" before. We shall relax this assumption later. Ideally, we would want to just store only ϕ, the inputs labels to $\mathcal{R}_2$ and the labels that are answers to $\mathcal{R}_2$'s queries. We append the input labels of $\mathcal{R}_2$ and labels that are answers to its Eval queries that it has not "seen" before to a list named

Lbls. However, it is easy to see that this information is not enough to answer $\mathsf{O_v}$ queries during decoding, as answering $\mathsf{O_v}$ queries inherently requires knowledge about pre-images of $\mathcal{R}_2$. This naturally leads to the idea of maintaining a mapping of all the labels "seen by" $\mathcal{R}_2$ to their pre-images.

The mapping T of labels to pre-image expressions. The pre-images of input labels and the labels that were results of sequence of Eval queries on its input labels by $\mathcal{R}_2$, are known. However, $\mathcal{R}_2$ might query labels which were neither an input to it nor an answer to one of its Eval queries. Such a label is in $\mathcal{Z}$ since we have assumed that all labels queried by $\mathcal{R}_2$ were "seen by" $\mathcal{R}_1$ or "seen by" $\mathcal{R}_2$ before. We represent the pre-images of labels in $\mathcal{Z}$ using a placeholder variable X_n where n is incremented for every such label. Note that the pre-image of every label seen by $\mathcal{R}_2$ can be expressed as a linear polynomial in the X_n's (these linear polynomials are referred to as pre-image expressions from hereon). Therefore we maintain a mapping of all labels "seen by" and their pre-image expressions in a list of tuples named T. Our approach is inspired by a similar technique used by Corrigan-Gibbs and Kogan in [CK18]. Like in [CK18], we *stress* that the mapping T is not a part of the encoding.

For Eval queries, we can check if there is a tuple in T whose pre-image expression is the sum of the pre-image expressions of the input labels. If that is the case, we return the label of such a tuple. Otherwise, we append the answer label to Lbls. For $\mathsf{O_v}$ queries, we can return true if the pre-image expression of the first input label multiplied by v gives the pre-image expression of the second input label. Otherwise we return false.

Surprises. There is a caveat, however. There might arise a situation that the label which is the answer to the Eval query is present in T but its pre-image expression is not the sum of the pre-image expressions of the input labels. We call such a situation a "surprise" and we call the answer label in that case a "surprise label". For $\mathsf{O_v}$ queries, there might be a surprise when the answer of the $\mathsf{O_v}$ query is true but the pre-image expression of the first input label multiplied by v is different pre-image expression of the second input label. In this case we call the second input label the surprise label. We assign a sequence number to each query made by $\mathcal{R}_2$, starting from 1 and an index to each tuple in T, with the indices being assigned to tuples in the order they were appended to T. To detect the query where the surprise happens, we maintain a set named Srps_1 that contains tuples of query sequence numbers and indices of the surprise label in T. This set Srps_1 is a part of the encoding. Note that whenever there is a surprise, it means that two different pre-image expressions evaluate to the same value. Since these two pre-image expressions are linear polynomials, at least one variable can be eliminated from T by equating the two pre-image expressions.

Running $\mathcal{R}_1$. Now that we have enough information in the encoding to run $\mathcal{R}_2$, we consider the information we need to add to the encoding to run $\mathcal{R}_1$ after $\mathcal{R}_2$ is run. First, we need to provide $\mathcal{R}_1$ its input labels. Our initial attempt would be to append the input labels of $\mathcal{R}_1$ (except $\sigma(1), \sigma(v)$, which are already present)

to Lbls. However, some of these input labels to $\mathcal{R}_1$ might have already been "seen by" $\mathcal{R}_2$. Since all labels "seen by" $\mathcal{R}_2$ are in T, we need a way to figure out which of $\sigma(i_j)$'s are in T. Note that such a label was either queried by $\mathcal{R}_2$ or an answer to a query of $\mathcal{R}_2$ (cannot have been an input to $\mathcal{R}_2$ given the restrictions on $i_1, \cdots, i_k, v$). Suppose q was the sequence number of the query in which $\sigma(i_j)$ was queried or an answer. The tuple (q, b, j) is added to the set Inputs where b can take values $\{1, 2, 3\}$ depending on whether $\sigma(i_j)$ was the first input label, the second input label or the answer label respectively. This set Inputs is a part of the encoding. The rest of the labels $\sigma(i_j)$, which do not appear in T, are added to T with their pre-images and the labels are appended to Lbls. Note that for all queries of $\mathcal{R}_1$, it follows from our assumption that the input labels will be in T. For every surprise, we add a tuple of sequence number and an index in T to the set Srps_2.

Relaxing the assumption. When we allow $\mathcal{R}_2$ to query labels it has not seen before or $\mathcal{R}_1$ has not seen, there are two issues. First, we need to add a tuple for the label in T (since T, by definition contains a tuple for all labels queried by $\mathcal{R}_2$). We solve this issue by adding the tuple made of the label and its pre-image. We have no hope of recovering the pre-image later, hence, we append the pre-image to a list named Vals. This list needs to be a part of the encoding since the pre-image of the label needs to be figured out to be added to T during decoding. For queries of $\mathcal{R}_1$, if the input label is not present in T, we do the same thing. The second issue that comes up when we relax the assumption is that we need to distinguish whether an input label was in $\mathcal{Z}$ or not. We solve this issue by maintaining a set of tuples named Free. For all labels in $\mathcal{Z}$ that are not an input label to $\mathcal{R}_1$, we add the tuple consisting of the sequence number of the query of $\mathcal{R}_2$ and b to Free where b set to 1 indicates it was the first input label and b set to 2 indicates it was the second input label.

The final steps. The labels the are absent in T are appended to a list named RLbls. If $|\mathcal{Z}| < l$, a fixed encoding D (the output of Encode for some fixed σ when $|\mathcal{Z}| \geqslant l$) is returned. Otherwise the encoding of σ consisting of Lbls, RLbls, Vals, Inputs, Srps_1, Srps_2, Free, ϕ is returned.

Wrapping up. The set of all "good" σ's has size at least $\varepsilon p!$ (where we have used that the total number of injective functions from $\mathbb{Z}_p \to \mathcal{L}$ is $p!$). Using $\mathcal{X}$ to be the set of the "good" σ's, $\mathcal{Y}$ to be the set of encodings, $\mathcal{R}$ to be the set of cartesian product of the domains of $i_1, \cdots, i_k, v, \pi$, the set of all random tapes of $\mathcal{R}_1$ the set of all random tapes of $\mathcal{R}_2$ and $\mathcal{L}$, it follows from Lemma 5.3.9 and Proposition 11.1.1 that

$$\begin{aligned}\log\left(\mathsf{Pr}\left[\text{Decoding is correct}\right]\right) \leqslant & s + (2q - l)\log\left(1 + \frac{6q}{p}\right) \\ & - l\log\left(\frac{p}{8q^2(2k + 2 + 3q)}\right) - \log\varepsilon\ .\end{aligned}$$

We have from Lemma 5.3.9 that $\mathsf{Pr}\left[\text{Decoding is correct}\right] \geqslant \varepsilon$. Therefore,

$$2\log\varepsilon \leqslant s + (2q-l)\log\left(1+\frac{6q}{p}\right) - l\log\left(\frac{p}{8q^2(2k+2+3q)}\right) .$$

Since $\mathsf{Pr}\left[\mathsf{G}_{13} \Rightarrow \mathsf{true}\right] = 2\varepsilon$, using (5.19) and (5.20) we have,

$$\mathsf{Pr}\left[\mathsf{G}_3 \Rightarrow \mathsf{true}\right] \leqslant 2\cdot 2^{\frac{s}{2}}\left(\frac{8q^2(2k+2+3q)}{p}\right)^{\frac{l}{2}}\left(1+\frac{6q}{p}\right)^{\frac{2q-l}{2}} + \frac{k^2+k+2}{p} .$$

■

5.3.5 Proof of Lemma 5.3.9

We provided the intuition behind our compression of σ in Section 5.3.4. We formally define the $\mathsf{Encode}, \mathsf{Decode}$ algorithms here and then analyze the size of the encoding space for "good" σ's.

THE Encode PROCEDURE. As we have previously mentioned Encode simulates G_{13} to $\mathcal{R}_1, \mathcal{R}_2$ and then runs $\mathcal{R}_1$ again. The Encode procedure has been formally defined in Figures 5.18 and 5.19. We shall next explain the pseudocode in detail and relate it to the intuition we provided in Section 5.3.4. In order to understand the pseudocode of Encode, initially ignore the code involving $\mathsf{T}, \mathsf{Lbls}, \mathsf{Vals}, \mathsf{RLbls}, \mathsf{Free}, \mathsf{Srps}_1, \mathsf{Srps}_2$.

First off, note that the oracles $\mathsf{Eval}(.,.,1), \mathsf{Eval}(.,.,2), \mathsf{Eval}(.,.,3)$ have identical outputs to the oracle $\mathsf{Eval}(.,.)$ in G_{13} and the oracles $\mathsf{O}_{\mathsf{v}}(.,.,1), \mathsf{O}_{\mathsf{v}}(.,.,2), \mathsf{O}_{\mathsf{v}}(.,.,3)$ have identical outputs to the oracle $\mathsf{O}_{\mathsf{v}}(.,.)$ in G_{13}. As mentioned previously the randomness input to Encode i.e. $i_1, \cdots, i_v, v, \pi$ are distributed according to the distribution of these values in G_{13}. Observe that till $\mathcal{R}_2$ is run, Encode perfectly simulates G_{13} to $\mathcal{R}_1, \mathcal{R}_2$. Then, $\mathcal{R}_1$ is run again. Recall that we treat $\mathcal{R}_1, \mathcal{R}_2$ as deterministic without loss of generality (alternatively we could have specified the random tape of these adversaries as an input to $\mathsf{Encode}, \mathsf{Decode}$). Therefore, $\mathcal{R}_1$ shall make the same queries when it is run the second time as it did in its first run since it receives identical answers to its queries (we have previously argued that the outputs of oracles $\mathsf{Eval}(.,.,1), \mathsf{Eval}(.,.,3)$ are identical and the outputs of oracles $\mathsf{O}_{\mathsf{v}}(.,.,1), \mathsf{O}_{\mathsf{v}}(.,.,3)$ are identical on same inputs).

Now that we have explained the high-level structure of Encode and related it to the intuition provided before, we shall delve into the details of its book-keeping. Recall that we say that a label has been seen by $\mathcal{R}_1$ if it was an input to $\mathcal{R}_1$, queried by $\mathcal{R}_1$ or an answer to a previously made $\mathsf{Eval}(.,.,1)$ query and a label has been seen by $\mathcal{R}_2$ if it was an input to $\mathcal{R}_2$, queried by $\mathcal{R}_2$ or an answer to a previously made $\mathsf{Eval}(.,.,2)$ query. To begin with we note that the sets $\mathcal{Z}, \mathcal{X}, \mathcal{Y}_1, \mathcal{Y}_2$ are never updated after $\mathcal{R}_2$ is run in Encode and hence are identical to what they would have been in G_{13} if the σ input to Encode was sampled.

The ordered first-in first-out (FIFO) list Lbls first contains the input labels of $\mathcal{R}_2$, then the labels that are answers to the Eval queries of $\mathcal{R}_2$ and had not been seen before by $\mathcal{R}_2$ (these labels are ordered in the order of the queries). Next, it contains labels that are input to $\mathcal{R}_1$ that were not queried by $\mathcal{R}_2$ during its

Procedure $\mathsf{Encode}(\sigma, (i_1, \cdots, i_k, v, \pi))$:

$\mathcal{X} \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}; \mathcal{Y}_1 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k)\}$

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k))$

$\mathcal{Y}_2 \leftarrow \{\sigma(1), \sigma(v), \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)\}; \mathcal{Z} \leftarrow \varnothing$

$\mathsf{T} \leftarrow []; \mathsf{T.append}((\sigma(1), 1, 1)); \mathsf{T.append}((\sigma(v), v, 1))$

For each $j \in [k]$: $\mathsf{T.append}(\sigma(i_j \cdot v), i_j \cdot v, 1))$

$\mathsf{Free} \leftarrow \varnothing; \mathsf{Srps}_1 \leftarrow \varnothing; \mathsf{Srps}_2 \leftarrow \varnothing; n \leftarrow 0; \mathsf{Inputs} \leftarrow \varnothing; c_1 \leftarrow 0; c_2 \leftarrow 0$

$s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \sigma(1), \sigma(v), \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v))$

For each $j \in [k]$:

  If $(\sigma(i_j), *, *) \notin \mathsf{T}$ then $\mathsf{T.append}((\sigma(i_j), i_j, 1))$;

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,3), \mathsf{O_v}(.,.,3)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k))$

$\mathsf{Lbls} \leftarrow []; \mathsf{Vals} \leftarrow []; \mathsf{RLbls} \leftarrow []$

For each $j \in [|\mathsf{T}|]$:

  $(\mathbf{s}, m, f) \leftarrow \mathsf{T}[j]$

  If $f = 1$ then $\mathsf{Lbls.append}(\mathbf{s})$

  If $f = 2$ then $\mathsf{Vals.append}(m)$

For each $j \in [p]$:

  If $(\sigma(j), *, *) \notin \mathsf{T}$ then $\mathsf{RLbls.append}(\sigma(j))$

If $|\mathcal{Z}| < l$ then return D

Else return $(\mathsf{Lbls}, \mathsf{RLbls}, \mathsf{Vals}, \mathsf{Srps}_1, \mathsf{Srps}_2, \mathsf{Inputs}, \mathsf{Free}, \phi)$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, 1)$:

$\mathbf{c} \leftarrow \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$

If $\mathbf{c} \notin \mathcal{Y}_1$ then $\mathcal{X} \xleftarrow{\cup} \{\mathbf{c}\}$

$\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$

Return $\mathbf{c}$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, 1)$:

$\mathbf{c} = \sigma(v \cdot \sigma^{-1}(\mathbf{a}))$

  $\mathcal{Y}_1 \xleftarrow{\cup} \{\mathbf{a}, \mathbf{b}\}$

Return $(\mathbf{b} = \mathbf{c})$

Figure 5.18: The first part of the Encode procedure. If $|\mathcal{Z}| < l$, a fixed encoding D (the output of Encode for some fixed σ when $|\mathcal{Z}| \geqslant l$) is returned. The function $\mathsf{T.append}(\mathsf{tuple})$ appends tuple into T. The first inserted tuple has index 1 and the index increases by 1 for every subsequent insertion. $\mathsf{T}[i]$ returns the tuple at index i in T. The function $\mathsf{T}(\mathbf{a})$ returns pre-image expression A such that $(\mathbf{a}, A, *) \in \mathsf{T}$. The function $\mathsf{VarReduce}(\mathsf{T}, C, C')$ equates expressions C and C', expresses the variable with highest subscript in the equation (say X_j) in terms of other variables, and substitutes X_j throughout T.

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, 2)$:

$c_1 \leftarrow c_1 + 1$

$\mathbf{c} \leftarrow \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$

If $\mathbf{a} \notin \mathcal{Y}_2$ then

  $\mathsf{AddToTable}(\mathbf{a}, c_1, 1); \mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}\}$

If $\mathbf{b} \notin \mathcal{Y}_2$ then

  $\mathsf{AddToTable}(\mathbf{b}, c_1, 2); \mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{b}\}$

If $(\mathbf{c}, *, *) \notin \mathsf{T}$ then

  $\mathsf{T.append}(\mathbf{c}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), 1)$

  $\mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{c}\}$

  If $\exists j \in [k] : \mathbf{c} = \sigma(i_j)$ then

    $\mathsf{Inputs} \xleftarrow{\cup} \{(c_1, 3, j)\}$

    $\mathsf{VarReduce}(\mathsf{T}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), i_j)$

Else

  If $\mathsf{T}(\mathbf{c}) \neq \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b})$ then

    $\mathsf{Srps}_1 \xleftarrow{\cup} \{(c_1, \mathsf{T.index}(\mathbf{c}))\}$

    $\mathsf{VarReduce}(\mathsf{T}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), \mathsf{T}(\mathbf{c}))$

Return $\mathbf{c}$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, 2)$:

$c_1 \leftarrow c_1 + 1$

$\mathbf{c} \leftarrow \sigma(v \cdot \sigma^{-1}(\mathbf{a}))$

If $\mathbf{a} \notin \mathcal{Y}_2$ then

  $\mathsf{AddToTable}(\mathbf{a}, c_1, 1); \mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{a}\}$

If $\mathbf{b} \notin \mathcal{Y}_2$ then

  $\mathsf{AddToTable}(\mathbf{b}, c_1, 2); \mathcal{Y}_2 \xleftarrow{\cup} \{\mathbf{b}\}$

If $(\mathbf{c}, *, *) \notin \mathsf{T}$ then

  Return false

Else

  If $\mathsf{T}(\mathbf{c}) \neq v\mathsf{T}(\mathbf{a})$ then

    $\mathsf{Srps}_1 \xleftarrow{\cup} \{(c_1, \mathsf{T.index}(\mathbf{c}))\}$

    $\mathsf{VarReduce}(\mathsf{T}, v\mathsf{T}(\mathbf{a}), \mathsf{T}(\mathbf{c}))$

Return $(\mathbf{b} = \mathbf{c})$

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, 3)$:

$c_2 \leftarrow c_2 + 1$

$\mathbf{c} \leftarrow \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$

If $(\mathbf{a}, *, *) \notin \mathsf{T}$ then $\mathsf{T.append}(\mathbf{l}, \sigma^{-1}(\mathbf{a}), 2)$

If $(\mathbf{b}, *, *) \notin \mathsf{T}$ then $\mathsf{T.append}(\mathbf{l}, \sigma^{-1}(\mathbf{b}), 2)$

If $(\mathbf{c}, *, *) \notin \mathsf{T}$ then $\mathsf{T.append}(\mathbf{c}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), 1)$

Else

  If $\mathsf{T}(\mathbf{c}) \neq \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b})$ then

    $\mathsf{Srps}_2 \xleftarrow{\cup} \{(c_2, \mathsf{T.index}(\mathbf{c}))\}$

    $\mathsf{VarReduce}(\mathsf{T}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), \mathsf{T}(\mathbf{c}))$

Return $\mathbf{c}$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, 3)$:

$c_2 \leftarrow c_2 + 1$

$\mathbf{c} \leftarrow \sigma(v \cdot \sigma^{-1}(\mathbf{a}))$

If $(\mathbf{a}, *, *) \notin \mathsf{T}$ then $\mathsf{T.append}(\mathbf{l}, \sigma^{-1}(\mathbf{a}), 2)$

If $(\mathbf{b}, *, *) \notin \mathsf{T}$ then $\mathsf{T.append}(\mathbf{l}, \sigma^{-1}(\mathbf{b}), 2)$

If $(\mathbf{c}, *, *) \notin \mathsf{T}$ then return false

Else

  If $\mathsf{T}(\mathbf{c}) \neq v\mathsf{T}(\mathbf{a})$

    $\mathsf{Srps}_2 \xleftarrow{\cup} \{(c_2, \mathsf{T.index}(\mathbf{c}))\}$

    $\mathsf{VarReduce}(\mathsf{T}, v\mathsf{T}(\mathbf{a}), \mathsf{T}(\mathbf{c}))$

Return $(\mathbf{b} = \mathbf{c})$

Procedure $\mathsf{AddToTable}(\mathbf{l}, c, b)$:

If $\mathbf{l} \in \mathcal{X}$ then

  $\mathcal{Z} \xleftarrow{\cup} \{\mathbf{l}\}$;

  If $\exists j \in [k] : \mathbf{l} = \sigma(i_j)$ then $\mathsf{Inputs} \xleftarrow{\cup} \{(c, b, j)\}; \mathsf{T.append}(\mathbf{l}, i_j, 0)$

  Else $\mathsf{Free} \xleftarrow{\cup} \{(c, b)\}; n \leftarrow n + 1; \mathsf{T.append}(\mathbf{l}, X_n, 0)$

Else $\mathsf{T.append}(\mathbf{l}, \sigma^{-1}(\mathbf{l}), 2)$

Figure 5.19: Second part of the Encode procedure.

Procedure $\mathsf{Decode}((\mathsf{Lbls}, \mathsf{RLbls}, \mathsf{Vals}, \mathsf{Srps}_1, \mathsf{Srps}_2, \mathsf{Inputs}, \mathsf{Free}, \phi), (i_1, \cdots, i_k, v, \pi))$:

For each $j \in \mathbb{Z}_p$: $\sigma(j) \leftarrow \perp; z_1 \leftarrow 1; z_2 \leftarrow 1; z_3 \leftarrow 1; \mathsf{T} \leftarrow []$

$\sigma(1) \leftarrow \mathsf{Lbls}[z_1]; z_1 \leftarrow z_1 + 1; \mathsf{T}.\mathsf{append}((\sigma(1), 1, 1))$

$\sigma(v) \leftarrow \mathsf{Lbls}[z_1]; z_1 \leftarrow z_1 + 1; \mathsf{T}.\mathsf{append}((\sigma(v), v, 1))$

For each $j \in [k]$: $\sigma(i_j \cdot v) \leftarrow \mathsf{Lbls}[z_1]; z_1 \leftarrow z_1 + 1; \mathsf{T}.\mathsf{append}((\sigma(i_j \cdot v), i_j \cdot v, 1))$

$n \leftarrow 0; c_1 \leftarrow 0; c_2 \leftarrow 0$

$s'_1, s'_2, \cdots, s'_k \leftarrow \mathcal{R}_2^{\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)}(\phi, \sigma(1), \sigma(v), \sigma(i_{\pi(1)} \cdot v), \cdots, \sigma(i_{\pi(k)} \cdot v))$

For each $j \in [k]$: If $\sigma(i_j) = \perp$ then $\sigma(i_j) \leftarrow \mathsf{Lbls}[z_1]; z_1 \leftarrow z_1 + 1$

$\phi, s_1, \cdots, s_k \leftarrow \mathcal{R}_1^{\mathsf{Eval}(.,.,3), \mathsf{O_v}(.,.,3)}(\sigma(1), \sigma(v), \sigma(i_1), \cdots, \sigma(i_k))$

For each $j \in [|\mathsf{T}|]$: $(\mathbf{s}, m, f) \leftarrow \mathsf{T}[j]; \sigma(m) \leftarrow \mathbf{s}$

For each $j \in [p]$: If $\sigma(j) = \perp$ then $\sigma(j) \leftarrow \mathsf{RLbls}[z_3]; z_3 \leftarrow z_3 + 1$

Return σ

Oracle $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, 2)$:

$c_1 \leftarrow c_1 + 1$

If $(\mathbf{a}, *, *) \notin \mathsf{T}$ then $\mathsf{AddToTable}(\mathbf{a}, c_1, 1)$

If $(\mathbf{b}, *, *) \notin \mathsf{T}$ then $\mathsf{AddToTable}(\mathbf{b}, c_1, 2)$

If $\exists \mathbf{c}' : (\mathbf{c}', \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$ then

  $\mathbf{c} \leftarrow \mathbf{c}'$

Elseif $\exists i : (c_1, i) \in \mathsf{Srps}_1$ then

  $(\mathbf{c}, \mathsf{T}(\mathbf{c}), f) \leftarrow \mathsf{T}[i]$

  $\mathsf{VarReduce}(\mathsf{T}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), \mathsf{T}(\mathbf{c}))$

Else

  $\mathbf{c} \leftarrow \mathsf{Lbls}[z_1]; z_1 \leftarrow z_1 + 1$

  $\mathsf{T}.\mathsf{append}(\mathbf{c}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), 1)$

  If $\exists j \in [k] : (c_1, 3, j) \in \mathsf{Inputs}$ then

    $\sigma(i_j) \leftarrow \mathbf{c}$

    $\mathsf{VarReduce}(\mathsf{T}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), i_j)$

  Return $\mathbf{c}$

Oracle $\mathsf{O_v}(\mathbf{a}, \mathbf{b}, 2)$:

$c_1 \leftarrow c_1 + 1$

If $(\mathbf{a}, *, *) \notin \mathsf{T}$ then $\mathsf{AddToTable}(\mathbf{a}, c_1, 1)$

If $(\mathbf{b}, *, *) \notin \mathsf{T}$ then $\mathsf{AddToTable}(\mathbf{b}, c_1, 2)$

If $\exists \mathbf{c}' : (\mathbf{c}', v\mathsf{T}(\mathbf{a}), *) \in \mathsf{T}$ then

  $\mathbf{c} \leftarrow \mathbf{c}'$

Elseif $\exists i : (c_1, i) \in \mathsf{Srps}_1$ then

  $(\mathbf{c}, \mathsf{T}(\mathbf{c}), f) \leftarrow \mathsf{T}[i]$

  $\mathsf{VarReduce}(\mathsf{T}, v\mathsf{T}(\mathbf{a}), \mathsf{T}(\mathbf{c}))$

Else return false

Return $(\mathbf{b} = \mathbf{c})$

Figure 5.20: The first part of the Decode procedure. The function $\mathsf{T}.\mathsf{append}(\mathsf{tuple})$ appends tuple into T. The first inserted tuple has index 1 and the index increases by 1 for every subsequent insertion. $\mathsf{T}[i]$ returns the tuple at index i in T. The function $\mathsf{T}.\mathsf{index}(\mathbf{a})$ returns the index of the tuple of the form $(\mathbf{a}, *, *)$ in T and the function $\mathsf{T}(\mathbf{a})$ returns pre-image expression A such that $(\mathbf{a}, A, *) \in \mathsf{T}$. The function $\mathsf{VarReduce}(\mathsf{T}, C, C')$ equates expressions C and C', expresses the variable with highest subscript in the equation (say X_j) in terms of other variables, and substitutes X_j throughout T.

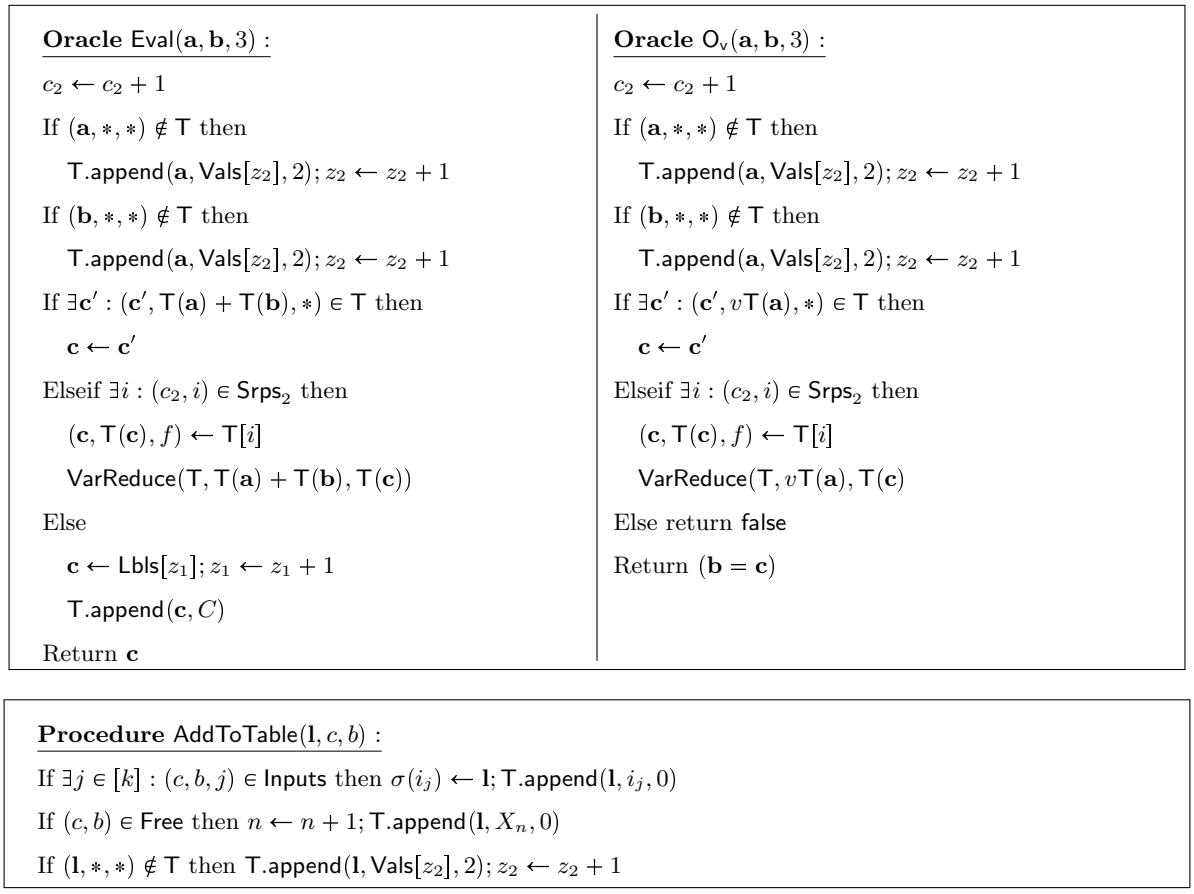


Figure 5.21: The second part of the Decode procedure.

run (ordered in the order of inputs of $\mathcal{R}_1$). Finally, it contains labels that are answers to the Eval queries of $\mathcal{R}_1$ during its second run that had not been seen by it before (during the second run) and are not among labels seen by $\mathcal{R}_2$ during its run (again ordered in the order of the queries).

The ordered (FIFO) list Vals first contains the pre-image of the labels (i.e. values in $\mathbb{Z}_p$) that were queried by $\mathcal{R}_2$ that it had not seen before (these pre-images are ordered in the order of the queries). It also contains the pre-image of the labels that queried by $\mathcal{R}_1$ during its second run that had not been seen by it before (during the second run) and are not among labels seen by $\mathcal{R}_2$ during its run (again ordered in the order of the queries).

The set Inputs is used to keep track of the labels that are inputs to $\mathcal{R}_1$ that $\mathcal{R}_2$ makes a query on or is an answer to $\mathcal{R}_2$'s Eval query. Each element of the set is a tuple that consists of the sequence number of the query, a flag indicating whether the label was the first or the second label of a query or the output of a

query and the j such that the label is $\sigma(i_j)$.

The data structure T is an ordered list of tuples with each tuple consisting of a label, its pre-image expression (as defined in Section 5.3.4) and a flag which takes value $0, 1, 2$. The flag set to 1 indicates that the label in the tuple needs to be written to the list Lbls and the flag set to 2 indicates that the pre-image of the label in the tuple needs to be written to the list Vals. The tuples containing the input labels of $\mathcal{R}_2$ are first added to T with the pre-image of the labels being the pre-image expression. For every unique label seen by $\mathcal{R}_2$ a tuple containing the label is added to T when the label is queried or is an answer to a query. When $\mathcal{R}_2$ queries a label, there might already be a tuple containing the label- in that case, no new tuple containing the label is added to T. In case there was no tuple in T containing the queried label, the label might either have been **(a)** an input to $\mathcal{R}_1$ or **(b)** a label that was an answer to $\mathcal{R}_1$'s Eval query and had not been seen by $\mathcal{R}_1$ before the query or **(c)** neither of **(a)**, **(b)**. For case **(a)** and **(c)** the pre-image expression for the label is its actual pre-image. For case **(b)**, the pre-image expression of the label is a placeholder variable X_n (with n incremented every time for a new placeholder). In this case, a tuple that consists of the sequence number of the query, whether the label was the first or the second label of the query is added to the set Free. Other than the input labels and the labels queried by $\mathcal{R}_2$, the only labels seen by $\mathcal{R}_2$ are the answers to the Eval queries. Consider an Eval query by $\mathcal{R}_2$ on labels $\mathbf{a}, \mathbf{b}$. Let $\mathbf{c} = \sigma(\sigma^{-1}(\mathbf{a}) + \sigma^{-1}(\mathbf{b}))$. Suppose no tuple in T contains $\mathbf{c}$, in that case a tuple containing the label $\mathbf{c}$ and the pre-image expression the sum of the pre-image expressions of labels $\mathbf{a}$ and $\mathbf{b}$ is added to T. On the other hand if T contains $\mathbf{c}$ but its pre-image expression differs from the sum of the pre-image expressions of labels $\mathbf{a}$ and $\mathbf{b}$, then $\mathbf{c}$ is a surprise label as defined in Section 5.3.4. The two pre-image expressions are equated and the variable with highest subscript in the equation (say X_j) is expressed in terms of other variables, and then X_j is substituted throughout T (we name this the $\mathsf{VarReduce}$ procedure). A tuple that consists of the sequence number of the query, the index of the tuple containing the label $\mathbf{c}$ is added to the set Srps_1. As mentioned previously, a surprise label may arise even for a $\mathsf{O_v}$ query by $\mathcal{R}_2$. Suppose an $\mathsf{O_v}$ query was made on labels $\mathbf{a}, \mathbf{b}$. In this case $\mathbf{b}$ is a surprise label if there is a tuple containing $\mathbf{b}$ such that the pre-image expression is different from the pre-image expression of the $\mathbf{a}$ by v. Again, in this case in a similar fashion, one variable is substituted throughout T using the $\mathsf{VarReduce}$ procedure and a tuple that consists of the sequence number of the query, the index of the tuple containing the label $\mathbf{b}$ is added to the set Srps_1.

After $\mathcal{R}_2$ finishes running, tuples containing input labels of $\mathcal{R}_1$ that $\mathcal{R}_2$ did not see are added to T with the pre-image expression being the actual pre-images (this ensures that all labels input to $\mathcal{R}_1$ are in T). Next, when $\mathcal{R}_1$ is run, for every unique label seen by $\mathcal{R}_1$ by that is not contained in a tuple in T, a tuple containing the label is added to T in the way similar to while running $\mathcal{R}_2$ with the only major differences being **(a)** no placeholder variables are introduced in this run i.e. for input labels that were previously not

in T, the pre-image expression is the pre-image of the label itself **(b)** surprises are kept track of in the set Srps_2 in this case.

After $\mathcal{R}_1$ is run for the second time, the sets $\mathsf{Lbls}, \mathsf{Vals}$ are populated based on the entries in T. All the labels that are not in T are inserted into the list RLbls in lexicographical order of the pre-images. Finally if $|\mathcal{Z}| \leqslant l$, the encoding D for some fixed σ for which $|\mathcal{Z}| \leqslant l$ is returned. Otherwise $\mathsf{Lbls}, \mathsf{RLbls}, \mathsf{Vals}, \mathsf{Srps}_1, \mathsf{Srps}_2$, $\mathsf{Inputs}, \mathsf{Free}$ along with ϕ, the state output by $\mathcal{R}_1$ is returned by Encode.

THE Decode PROCEDURE. The Decode procedure, as mentioned previously runs $\mathcal{R}_2$ and then $\mathcal{R}_1$. It starts running $\mathcal{R}_2$ using the state ϕ and the first $k+2$ labels in Lbls. It maintains the data structure T just like Encode. It inserts tuples containing the input labels of $\mathcal{R}_2$ into T with the pre-image expressions being the pre-images (which it gets from the input randomness). On an Eval or $\mathsf{O_v}$ query on labels $\mathbf{a}, \mathbf{b}$, it first checks if a tuple containing $\mathbf{a}$ is in T. If not it checks if a tuple containing the sequence number of the query, 1 and some $j \in [k]$ is present in Inputs- if that is the case, the pre-image expression of $\mathbf{a}$ is i_j. It then checks whether a tuple containing the sequence number of the query and 1 is present in Free- in that case the pre-image expression of $\mathbf{a}$ is a new placeholder variable X_n (n is incremented). Otherwise, it reads the next value of Vals and assigns it to the pre-image expression of $\mathbf{a}$. For the other input label $\mathbf{b}$, the same steps are followed.

In order to find the answer label for an Eval query, it first checks whether there is a tuple in Srps_1 with the first entry being the sequence number of the query and some i- if that is the case the label in the i^{th} tuple of T is returned as answer and the pre-image expression of the i^{th} tuple of T and the sum of the pre-image expressions of the two inputs are equated and one variable is substituted throughout T using the $\mathsf{VarReduce}$ procedure. Otherwise, it returns removes the top label from the list Lbls and returns it. It also adds a tuple containing the label to T. The $\mathsf{O_v}$ queries are answered in an analogous fashion. Also note that for every label queried by $\mathcal{R}_2$ and every label that is an answer to $\mathcal{R}_2$'s Eval query, the Decode procedure checks whether it is an input label to $\mathcal{R}_1$ using the Inputs set.

After running $\mathcal{R}_2$, some of the inputs to $\mathcal{R}_1$ might have been figured out using the set Inputs during $\mathcal{R}_2$'s query. The rest are obtained by removing the elements of the list Lbls and assigning them to unassigned inputs of $\mathcal{R}_1$. For each such label, a tuple containing the label is inserted to T. Then Decode runs $\mathcal{R}_1$ with the state ϕ and its input labels. The queries of $\mathcal{R}_1$ are answered in a similar fashion like the queries of $\mathcal{R}_2$.

After $\mathcal{R}_1$ has finished running, for every tuple in T, Decode assigns the labels to the pre-image expression (we shall argue that if the input to Decode was a valid encoding i.e. not D, then after the execution of $\mathcal{R}_1$ all the pre-image expressions in T are actual pre-images i.e. there are no more placeholders in T). Finally, the values in $\mathbb{Z}_p$ that have not been assigned images are assigned images using the labels in RLbls.

We say that the output of Encode is a valid encoding if the output was produced after the check $|\mathcal{Z}| \geqslant l$ succeeded in Encode. We claim that Decode produces the correct σ whenever it receives a valid encoding as

input. The main observation here is that T is identically populated in $\mathsf{Encode}, \mathsf{Decode}$. From the pseudocode and description above of $\mathsf{Encode}, \mathsf{Decode}$, it is not very difficult to infer correctness. Nonetheless, we provide a proof sketch of correctness of decoding to Section 5.5.

Next we shall relate the probability of Encode returns a valid encoding to the probability of $\mathcal{R}_1, \mathcal{R}_2$ winning G_{13}. We have already argued that the outputs of $\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1), \mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)$ and the sets $\mathcal{Z}, \mathcal{Y}_1, \mathcal{Y}_2$ are identical in G_{13} and Encode and the outputs of $\mathsf{Eval}(.,.,3), \mathsf{O_v}(.,.,3)$ of Encode are identical to the outputs of $\mathsf{Eval}(.,.,1), \mathsf{O_v}(.,.,1)$ of G_{13} respectively.

Therefore, $\mathsf{G}_{13} \Rightarrow \mathsf{true}$ implies that $|\mathcal{Z}| \geqslant l$ in Encode. So, for good σ's, the probability that Encode produces a valid encoding (i.e. does not output D) is ε.

Since, Encode produces a valid encoding whenever $|\mathcal{Z}| \geqslant l$, i.e. whenever G_{13} outputs true given that σ was sampled in G_{13} we have,

$$\mathsf{Pr}\left[\text{Decoding is correct}\right] \geqslant \varepsilon \,. \tag{5.22}$$

At the end of execution of Encode, let $|\mathsf{Lbls}| = b_1, |\mathsf{RLbls}| = b_2, |\mathsf{Vals}| = o, |\mathsf{Free}| = f, |\mathsf{Inputs}| = i, |\mathsf{Srps}_1| = s_1, |\mathsf{Srps}_2| = s_2, |\mathsf{T}| = t$. Let the size of the internal state output by $\mathcal{R}_1$ be s. We make the following observations about the outputs of Encode.

- For every $i \in \mathbb{Z}_p$, exactly one of the following is true.
  - i is present in Vals.
  - $\sigma(i)$ is present in Lbls.
  - $\sigma(i)$ is present in RLbls.
  - $\sigma(i)$ was queried by $\mathcal{R}_2$ and an entry was added to Inputs.
  - $\sigma(i)$ was queried by $\mathcal{R}_2$ and an entry was added to Free.

  Therefore, $|\mathsf{Lbls}| + |\mathsf{RLbls}| + |\mathsf{Vals}| + |\mathsf{Free}| + |\mathsf{Inputs}| = p$ i.e. $b_1 + b_2 + o + f + i = p$. Also note that every label that is not present in T is added to RLbls, so $b_2 = p - t$ and $b_1 = t - o - (i + f)$

- Since a placeholder X_i is introduced only when an entry is added to Free and at least one placeholder is removed from T whenever an entry is added to Srps_1 or Srps_2, we have $|\mathsf{Srps}_1| + |\mathsf{Srps}_2| \leqslant |\mathsf{Free}|$ i.e. $s_1 + s_2 \leqslant f$.

- $|\mathsf{Free}| + |\mathsf{Inputs}| \geqslant |\mathcal{Z}|$ because every entry in $\mathcal{Z}$ is in one of $\mathsf{Free}, \mathsf{Inputs}$. We also have that $|\mathcal{Z}| \geqslant l$ (for any encoding that is output, including D). It follows that $|\mathsf{Free}| + |\mathsf{Inputs}| \geqslant l$ i.e. $f + i \geqslant l$.

Claim 5.3.10. *If σ is "good" then the size of the encoding space is at most*

$$2^s p!\left(1+\frac{6q}{p}\right)^{(2q-l)}\left(\frac{p}{8q^2(2k+2+3q)}\right)^{-l}.$$

Proof. Below, we establish upper bounds on the number of possibilities of various components present in the encoding.

- Each entry in Free is a tuple where the first element is in $[q]$ and the second element is in $[2]$. There are f entries in total. Therefore, the number of possibilities of Free is $(2q)^f$.

- Since each entry in Srps_1 consists of triple whose first element is in $[q]$, second element is in $[t]$ (an index of T). Therefore, the number of possibilities of Srps_1 is $(2tq)^{s_1}$.

- Since each entry in Srps_2 consists of triple whose first element is in $[q]$, second element is in $[t]$. Therefore, the number of possibilities of Srps_2 is $(2tq)^{s_2}$.

- Each entry of Inputs consists of a tuple whose first element is in $[q]$ and the second element is in $[3]$ and the third element is in $[k]$. Since there are i entries in total, there are at most $(3qk)^i$ possibilities.

- The list values has o entries. Since $2k+2$ values are already known to not be in Vals (as $1, v, i_1, \cdots, i_k, i_1 \cdot v, \cdots, i_k \cdots v$ will never be in Vals) and all the entries are distinct and are in $\mathbb{Z}_p$, there are $\binom{p-2k-2}{o} o!$ possibilities.

- The list Lbls has $b_1 = t - o - (i+f)$ distinct labels each of which are in $\mathcal{L}$ (a set of size p). Hence, the total number of possibilities are at most $\binom{p}{t-o-(i+f)}(t-o-(i+f))!$.

- Since the labels being added to the list RLbls are the only ones absent in T, and T is fully populated in Decode before RLbls is used, it suffices to remember their lexicographical ordering. There are $p-t$ known distinct labels in RLbls. The number of possibilities are $(p-t)!$.

- The state output by $\mathcal{R}_1$ is s bits, hence the number of possibilities are 2^s.

The encoding space is at most the product of the number of possibilities of all the components. Hence, the size of the encoding space is upper bounded by

$$2^s (2q)^f (2tq)^{(s_1+s_2)} (3qk)^i \binom{p-2k-2}{o} o! \binom{p}{t-o-(i+f)} (t-o-(i+f))!(p-t)!\,.$$

Since $s_1 + s_2 \leqslant f$, and $4t > 3k$ (since $t \geqslant 2k+2$), the size of the encoding space is at most

$$2^s(4tq^2)^{(f+i)}\binom{p-2k-2}{o}o!\binom{p}{t-o-(i+f)}(t-o-(i+f))!(p-t)!\ .$$

Since $o \leqslant 2q - |\mathcal{Z}| \leqslant 2q - l$ (as $\mathsf{Vals}, \mathcal{Z}$ are populated only for labels on which queries are made), we have

$$\begin{aligned}\binom{p}{t-o-(i+f)}(t-o-(i+f))!\binom{p-2k-2}{o}o!(p-t)! &\leqslant p!\left(\frac{(p-2k-2)!}{(p-2k-2-(i+f))!}\frac{(p-t)!}{(p-t+o+i+f)!}\right)\\ &\leqslant p!\left(\frac{(p-2k-2)^o}{(p-t)^{o+i+f}}\right)\\ &\leqslant p!\left(\frac{p-2k-2}{p-t}\right)^{2q-l}(p-t)^{-(i+f)}\ .\end{aligned}$$

Since $t \leqslant 2k+2+3q$ (size of T can increase by at most 3 on every query), $i+f \geqslant l$, $o \leqslant 2q-l$, and $p-2k-2-3q \geqslant \frac{p}{2}$ (because $6q \leqslant p-4k-4$) we have the size of the encoding space is at most

$$2^s p!\left(1+\frac{6q}{p}\right)^{(2q-l)}\left(\frac{p}{8q^2(2k+2+3q)}\right)^{-l}\ .$$

This concludes the proof. ■

5.4 Conclusions

Despite a clear restriction of our result to straightline reductions, we believe the main contribution of this work is the introduction of novel techniques for proving lower bounds on the memory of reductions that will find wider applicability. In particular, we clearly departed from the framework of prior works [ACFK17, WMHT18] tailored at the usage of lower bounds for streaming algorithms, and provided the first lower bound for "algebraic" proofs in the public-key domain. The idea of a problem-specific proof of memory could be helpful elsewhere.

Of course, there are several open problems. It seems very hard to study the role of rewinding for such reductions. In particular, the natural approach is to resort to techniques from communication complexity (and their incarnation as streaming lower bounds), as they are amenable to the multi-pass case. The simple combinatorial nature of these lower bounds however is at odds with the heavily structured oracles we encounter in the generic group model. Another problem we failed to solve is to give an adversary $\mathcal{A}$ in our proof which uses little memory – we discuss a candidate in the body, but analyzing it seems to give us difficulties similar to those of rewinding.

This latter point makes a clear distinction, not discussed by prior works, between the *way* in which we prove memory-tightness (via reductions using small memory), and its most general interpretation, as defined in [ACFK17], which would allow the reduction to adapt its memory usage to that of $\mathcal{A}$.

5.5 Proof Sketch of Correctness of Decoding in Section 5.3.5

We shall argue that Decode produces the correct σ whenever it receives a valid encoding as input. So we assume throughout that the input to Decode is a valid encoding.

We shall show that T is identically populated in Encode, Decode and the answers to the queries of $\mathcal{R}_2, \mathcal{R}_1$ are identical in Encode, Decode. This would mean that the runs of $\mathcal{R}_2, \mathcal{R}_1$ would be identical in Encode, Decode since answers to every query is identical in Encode, Decode. In Encode, after running $\mathcal{R}_1$, note that T does not have any placeholder variables remaining because all the elements were $\mathcal{Z}$ were answers to $\mathcal{R}_1$ and would have had the placeholder variables for their pre-images removed by the time $\mathcal{R}_1$ finishes. So, all the tuples of T are triples of labels, pre-images and the flag after running $\mathcal{R}_1$. Since Encode returns a valid encoding (i.e. the check $|\mathcal{Z}| \geqslant l$ succeeds), the labels in RLbls are the ones absent in T appended in the lexicographical order of their pre-images. Now we shall use the fact that T is identical in Decode after running $\mathcal{R}_1$ i.e. ll the tuples of T are triples of labels, pre-images and the flag after running $\mathcal{R}_1$. Note that in Decode, the labels in T are correctly assigned to their pre-images and all unassigned pre-images in $\mathbb{Z}_p$ are assigned the labels in RLbls sequentially i.e. for all $x \in \mathbb{Z}_p$, $\sigma(x)$ is recovered correctly. Therefore Decode produces the correct σ. So all we need to show now is that T is identically populated in Encode, Decode and answers to every query is identical in Encode, Decode.

T IS IDENTICALLY POPULATED AND QUERIES ARE ANSWERED IDENTICALLY IN Encode, Decode. In Encode, since T is created after $\mathcal{R}_1$ is run for the first time, T is not affected by the first run of $\mathcal{R}_1$. Now, observe that the first $k+2$ tuples appended to T are $(\sigma(1), 1, 1), (\sigma(v), v, 1), (\sigma(i_1 \cdot v), i_1 \cdot v, 1), \cdots, (\sigma(i_k \cdot v), i_k \cdot v, 1)$ in Encode. It follows that $\sigma(1), \sigma(v), \cdots, \sigma(i_1 \cdot v), \cdots, \sigma(i_k \cdot v)$ are the first $k+2$ labels appended to Lbls. This means that in Decode, $\sigma(1), \sigma(v)$, $\sigma(i_j \cdot v)$'s are correctly assigned using the first $k+2$ labels in Lbls and the first $k+2$ tuples in T are the same as that in Encode. This means the first $k+2$ tuples are identically populated in T in Encode, Decode. Next, we sketch a proof that T is identically populated in Encode, Decode on every query made by $\mathcal{R}_2, \mathcal{R}_1$. We shall also show that the answers to the queries are identical in Encode, Decode. Note that T is modified only by the append and VarReduce operations. We shall be using the phrase "a label $\mathbf{l}$ is not in T" to mean $(\mathbf{l}, *, *) \notin \mathsf{T}$ throughout this proof sketch.

We shall first show that T is identical in Encode, Decode after an $\mathsf{Eval}(.,.,2)$ query if it was identical in Encode, Decode before the query and that the answer to the query is identical in Encode, Decode. We can prove a similar statement about $\mathsf{O_v}(.,.,2)$ queries analogously. We shall then show T is identical in Encode, Decode after before $\mathcal{R}_1$ is run in both of them for the last time. Then we shall that T is identical in Encode, Decode after an $\mathsf{Eval}(.,.,3)$ or $\mathsf{O_v}(.,.,3)$ query if it was identical in Encode, Decode before the query and that the answers to the queries are identical in Encode, Decode. This would conclude our proof that T is identically populated in Encode, Decode and that the queries of $\mathcal{R}_2$ and $\mathcal{R}_1$ are answered identically.

Assuming that T is identical in $\mathsf{Encode}, \mathsf{Decode}$ before an $\mathsf{Eval}(.,.,2)$ query, we show that T is identical after the query in $\mathsf{Encode}, \mathsf{Decode}$ and the answer to the query is identical in $\mathsf{Encode}, \mathsf{Decode}$. In Encode, when an $\mathsf{Eval}(.,.,2)$ query (assume the query sequence number is c_1) is made by $\mathcal{R}_2$, $\mathsf{AddToTable}$ is invoked on the input labels if they are not present in T (the input label not being in $\mathcal{Y}_2$ is equivalent to it not being in T). Suppose $\mathbf{l}$ is such an input label that is not present in T in Encode- this also means that $\mathbf{l}$ is not present in T in Decode from our assumption. Therefore, $\mathsf{AddToTable}(\mathbf{l}, c, b)$ would be invoked in both $\mathsf{Encode}, \mathsf{Decode}$. There are three possibilities.

1. In Encode, $\mathbf{l} \in \mathcal{X}$ and for some $j \in [k]$, $\mathbf{l} = \sigma(i_j)$. In this case, observe that (c, b, j) is added to Inputs. Since, this is the only point where tuples of the form $(*, 1, *), (*, 2, *)$ are added to Inputs in Encode, the equivalent condition in Decode is that there exists some $j \in [k]$ such that $(c, b, j) \in \mathsf{Inputs}$. Note that the same tuple is appended to T in $\mathsf{Encode}, \mathsf{Decode}$ when these equivalent conditions are true.

2. In Encode, $\mathbf{l} \in \mathcal{X}$ and for no $j \in [k]$, $\mathbf{l} = \sigma(i_j)$. In this case, observe that (c, b) is added to Free. Since, this is the only point where tuples are added to Free in Encode, the equivalent condition in Decode is $(c, b) \in \mathsf{Free}$. Again, note that the same tuple is appended to T in $\mathsf{Encode}, \mathsf{Decode}$ when these equivalent conditions are true.

3. In Encode, neither of the previous two conditions are true. In Decode, note that when $\mathsf{AddToTable}$ was invoked, $\mathbf{l}$ was not present in T and if either of the preceding two conditions were true, then a tuple containing $\mathbf{l}$ was appended to T. Therefore, $\mathbf{l}$ is not in T if and only if neither of the preceding conditions are true. Since the two preceding conditions are equivalent in $\mathsf{Encode}, \mathsf{Decode}$, the complement of the disjunction of the two conditions are also equivalent in $\mathsf{Encode}, \mathsf{Decode}$. Since the tuple $(\mathbf{l}, \sigma^{-1}(\mathbf{l}), 2)$ is added to T in Encode, it is easy to see that $\sigma^{-1}(\mathbf{l})$ is the element popped out to the list Vals. Therefore, again the same tuple is appended to T in $\mathsf{Encode}, \mathsf{Decode}$.

Therefore, after the invocation of $\mathsf{AddToTable}$ function, T is identical in $\mathsf{Encode}, \mathsf{Decode}$.

Note that the condition that $\mathbf{c}$ is in T and $\mathsf{T}(\mathbf{c}) \neq \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b})$ in Encode implies that there does not exist $\mathbf{c}'$ such that $(\mathbf{c}', \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$. Also a tuple of the form $(c_1, *)$ is added to Srps_1 in Encode if and only if $\mathbf{c}$ is in T and $\mathsf{T}(\mathbf{c}) \neq \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b})$. Therefore, the condition $\mathbf{c}$ is in T and $\mathsf{T}(\mathbf{c}) \neq \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b})$ is equivalent to the condition that there does not exist $\mathbf{c}'$ such that $(\mathbf{c}', \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$ and there is some i such that $(c_1, i) \in \mathsf{Srps}_1$ in Decode. Since T is identical up until this point, the operation $\mathsf{VarReduce}$ is identical in $\mathsf{Encode}, \mathsf{Decode}$.

The only other point when tuples are appended to T in Encode on an $\mathsf{Eval}(\mathbf{a}, \mathbf{b}, 2)$ query is when a tuple containing $\mathbf{c}$ (the answer of the Eval query) is added. Note that a tuple containing $\mathbf{c}$ is added to T in

Encode if $\mathbf{c}$ is not in T. If $\mathbf{c}$ is not in T, then $(*, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \notin \mathsf{T}$ which in turn implies that no tuple of the form of $(c_1, *)$ was added to Srps_1. Therefore the equivalent condition in Decode is that for no $\mathbf{c}'$, $(\mathbf{c}', \mathsf{T}(\mathbf{a})+\mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$ and for no i, $(c_1, i) \in \mathsf{Srps}_1$. It is easy to see that $\mathbf{c}$ is the label popped from the list Lbls in Decode when the condition is true. Therefore, again, the same tuple is appended to T in Encode, Decode. Also, the same label is the answer to the Eval query in Encode, Decode. Again, the condition $\mathbf{c}$ is not in T and for some $j \in [k]$, $\mathbf{c} = \sigma(i_j)$ in Encode is equivalent to the condition that for no $\mathbf{c}'$, $(\mathbf{c}', \mathsf{T}(\mathbf{a})+\mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$ and for no i, $(c_1, i) \in \mathsf{Srps}_1$ and for some $j \in [k]$, $(c_1, 3, j) \in \mathsf{Inputs}$ because a tuple of the form $(c_1, 3, *)$ is added to Inputs if and only if $\mathbf{c}$ is not in T and for some $j \in [k] : \mathbf{c} = \sigma(i_j)$ in Encode. Therefore, the VarReduce operation is identical in Encode, Decode. Also, in this case the same label $\sigma(i_j)$ is returned in Encode, Decode. In Encode, the condition $\mathbf{c}$ is in T and $\mathsf{T}(\mathbf{c}) = \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b})$ is equivalent to the condition there exists $\mathbf{c}'$ such that $(\mathbf{c}', \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$. Since, T is identical up until this point in Encode, Decode, the condition $\mathbf{c}$ is in T and $\mathsf{T}(\mathbf{c}) = \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b})$ in Encode is equivalent condition $(\mathbf{c}, \mathsf{T}(\mathbf{a}) + \mathsf{T}(\mathbf{b}), *) \in \mathsf{T}$ in Decode. Note that for this equivalent condition, no changes are made to T in Encode, Decode and the same label is returned in both procedures.

Thus, we have argued that the changes to T and the answer is identical in Encode, Decode on an $\mathsf{Eval}(.,.,2)$ query. Therefore, if T was identical in Encode, Decode before an $\mathsf{Eval}(.,.,2)$ query, it remains identical after the query too and the answer of the $\mathsf{Eval}(.,.,2)$ query is identical in Encode, Decode.

Using very similar arguments we can prove that if T was identical in Encode, Decode before an $\mathsf{O_v}(.,.,2)$ query, it remains identical after the query too and the answer of the $\mathsf{O_v}(.,.,2)$ query is identical in Encode, Decode. We omit the proof since it is very similar to the previous one.

Since T was identical before running $\mathcal{R}_2$ in Encode, Decode, it remains identical after running $\mathcal{R}_2$ since the changes made on each $\mathsf{Eval}(.,.,2), \mathsf{O_v}(.,.,2)$ query are identical in Encode, Decode and all the answers to queries of $\mathcal{R}_2$ are identical in Encode, Decode. In Encode, the condition $\sigma(i_j)$ is not in T is equivalent to the condition $\sigma(i_j) = \perp$ in Decode because whenever a tuple of containing $\sigma(i_j)$ was appended to T in Decode, the value of $\sigma(i_j)$ was set and for $\sigma(i_j)$'s not in T, the value of $\sigma(i_j)$ is $\perp$. Therefore, T remains identical in Encode, Decode till before running $\mathcal{R}_1$ in Encode, Decode.

Just like we argued that T remains identical after running $\mathcal{R}_2$ and all the answers to queries of $\mathcal{R}_2$ are identical in Encode, Decode, we can use very similar arguments to prove the T remains identical after running $\mathcal{R}_1$ in Encode, Decode and all the answers to queries of $\mathcal{R}_1$ are identical in Encode, Decode. We omit the proof since it is very similar to the proof before.

There are no other changes to T after running $\mathcal{R}_1$ in Encode, Decode. Therefore, T is identical in Encode, Decode and all the answers to queries of $\mathcal{R}_2, \mathcal{R}_1$ are identical in Encode, Decode.

5.6 Rewinding: Conjecture and Obstacles

We briefly discuss the barriers in extending our result to consider rewinding.

A REDUCTION. First off, we informally describe a reduction $\mathcal{R}$ that *does* rewind the adversary $\mathcal{A}$ and answers all the $\mathsf{H}, \mathsf{H}_{\mathsf{v}}$ queries using $O(k \log k + \log p)$ bits of memory where k is the total number of H and H_{v} query $\mathcal{A}$ makes. To simulate the random oracle in a memory-efficient way, the reduction $\mathcal{R}$ uses a PRF with key k_f whose domain is $\mathcal{L}$ and range is $\{0,1\}^{\mathsf{hLen}}$ and a list L. It keeps a counter on the number of queries by $\mathcal{A}$ and increments it by 1 on each H and H_{v} query. The reduction $\mathcal{R}$ samples a key k_f (assume k_f can be expressed in r bits) for the PRF and initializes L to empty before running $\mathcal{A}$.

For the first query made by $\mathcal{A}$ (H or H_{v}), $\mathcal{R}$ responds with $\mathsf{PRF}(k_f, 1)$ and appends 1 to L. If the i^{th} query by $\mathcal{A}$ is a H_{v} query on $\mathbf{a}$, $\mathcal{R}$ records $\mathbf{a}$, rewinds $\mathcal{A}$ to its beginning, and checks if $\mathcal{A}$ had made a H_{v} query on $\mathbf{a}$ before the i^{th} query or if $\mathcal{A}$ had made a H query on $\mathbf{b}$ such that $\mathsf{O}_{\mathsf{v}}(\mathbf{a}, \mathbf{b}) = 1$ before the i^{th} query. If $\mathcal{A}$ does find either of these to be true on the j^{th} query, $\mathcal{R}$ appends $L[j]$ to L. Similarly, if the i^{th} query by $\mathcal{A}$ is a H query on $\mathbf{a}$, $\mathcal{R}$ records $\mathbf{a}$, rewinds $\mathcal{A}$ to its beginning, and checks if $\mathcal{A}$ had made a H query on $\mathbf{a}$ before the i^{th} query or if $\mathcal{A}$ had made a H_{v} query on $\mathbf{b}$ such that $\mathsf{O}_{\mathsf{v}}(\mathbf{b}, \mathbf{a}) = 1$ before the i^{th} query. If $\mathcal{A}$ does find either of these to be true on the j^{th} query, $\mathcal{R}$ appends $L[j]$ to L. While rewinding, $\mathcal{R}$ answers the t^{th} query ($t < i$) by $\mathsf{PRF}(k_f, L[t])$. Clearly, using this strategy, $\mathcal{R}$ simulates the answers to the queries of $\mathcal{A}$ correctly. Note that k_f can be expressed in r bits, L can be expressed in at most $k \log k$ bits, $\mathbf{a}$ can be expressed in at most $\log p$ bits and the query sequence number can be expressed in at most $\log k$ bits. Therefore, $\mathcal{R}$ uses at most $r + k \log k + \log p + \log k = O(k \log k + \log p)$ bits of memory.

REMARKS. Note that even if $\mathcal{R}$ rewinds $\mathcal{A}$, $\mathcal{R}$ needs to answer the queries of $\mathcal{A}$ from its beginning to the point where $\mathcal{R}$ rewound it, again. Therefore, it seems unavoidable that $\mathcal{R}$ needs to remember some information about each query of $\mathcal{A}$. Since there are k queries, this information would take at least $\log k$ bits for each query (since answer to each query could be unique) and $k \log k$ bits in total. Hence, we conjecture a lower bound of $\Omega(k \log k)$ bits even when $\mathcal{R}$ rewinds $\mathcal{A}$.

We are not aware of any techniques that could help us establish this bound, and this appears a much harder problem than the straightline case.

Chapter 6

A GENERIC MEMORY-TIGHTNESS IMPOSSIBILITY FOR $\mathrm{INDR} + \mathrm{CTXT} \implies \mathrm{AE}$

A meaningful question is whether we can give a memory-tight reduction beyond the setting of channels, and reduce AE security to INDR and CTXT security in the most general sense. In this chapter, we show that this is unlikely by giving impossibility results for black-box reductions. This chapter is based on our work in [GJT20].

We consider reductions to INDR and CTXT which are restricted, but note that all prior impossibility results on memory-tight reductions [ACFK17, WMHT18, GT20] make similar or stronger restrictions. In particular, we require the reductions to simulate their encryption oracles "faithfully" to an AE adversary, i.e., if they answer an encryption query with a ciphertext C, the same query (1) has been asked to the encryption oracle available to the reduction and (2) it has returned C. This restriction is natural, and we are not aware of any reductions evading it.

Straightline reductions. Our first result builds an (inefficient) adversary $\mathcal{A}$ against AE security which no straightline reduction can use to (1) break CTXT security (regardless of the memory available to the reduction) or, more importantly, to (2) break INDR security (unless the reduction uses an amount of memory proportional to the query complexity of the adversary). Moreover, $\mathcal{A}$ uses little memory, and thus our result implies impossibility even for "weakly memory-tight reductions" which adapt their memory usage. This is unlike recent works [WMHT18, GT20], which only rule out reductions with memory independent of that of the adversary.

At a high level, $\mathcal{A}$ forces the reduction to complete a memory-hard task before being useful. If the reduction succeeds, $\mathcal{A}$ executes an (inefficient) procedure to break INDR security. (And importantly, this procedure does not help in breaking CTXT security!) More in detail, the first part of $\mathcal{A}$'s execution consists of *challenge rounds*. In each of these rounds, $\mathcal{A}$ encrypts random plaintexts $M_1, \ldots, M_u$, which result in ciphertexts $C_1, \ldots, C_u$, and also picks a random index $i^* \in [u]$. It then asks for the decryption of C_{i^*}, and checks whether the response equals M_{i^*}. If so, it moves to the next round, if not it aborts by doing something useless. Only if all rounds are successful $\mathcal{A}$ proceeds to break INDR security. We use techniques borrowed from the setting of random oracles with auxiliary input (AI-ROM) [CDGS18] to prove that the probability that all rounds are successful decays exponentially as long as the reduction's memory does not fit all of $M_1, \ldots, M_u$.

FULL REWINDING. The restriction to straightline reductions seem too restrictive: After all, a reduction could (1) wait for a decryption query C_{i*}, then (2) rewind the adversary to re-ask $M_1, M_2, \ldots$ until M_{i*} is asked. The caveat is that our definition of INDR security does not allow for *re-asking* encryption queries (again, as pointed out above, such a notion would prevent us from using the results of [JT19, Din20]). Therefore, if we assume that all the reduction can do is remember (say) S plaintext-ciphertext pairs, the above adversary $\mathcal{A}$ will fail to pass a challenge round with probability at least $1 - S/u$.

Still, this does not mean that rewinding cannot help when allowing more general adversarial strategies. While handling arbitrary rewinding appears to be out of reach, we make partial progress by extending our proof (and our construction of $\mathcal{A}$) to show that "full" rewinding (i.e., re-running $\mathcal{A}$ from the beginning) does not help. This is the same rewinding model considered in prior memory-tightness lower bounds [ACFK17]. However, in those results, one obtains a rewinding-memory tradeoffs (in that reducing memory would require more rewinding). Here, our result is absolute, in the sense that if memory is too small, no amount of rewinding can help.

6.1 Negative Results for Memory-tight AE Reductions

In this section we give impossibility results for giving a memory-tight reduction (for a natural restricted class of black-box reductions) from AE-$\mathtt{m}$ security to nonce-respecting INDR and CTXT-1 security. This establishes that our restriction to the channel setting for Section 3.3 was necessary for our positive results.

BLACK-BOX REDUCTIONS. A reduction $\mathcal{R}$ maps an adversary $\mathcal{A}$ to an adversary $\mathcal{R}[\mathcal{A}]$. We consider reductions that run an AE-$\mathtt{m}$ adversary $\mathcal{A}$ in a black-box manner as shown in Fig. 6.1. It starts with initial state σ output by $\mathcal{R}.\mathsf{Init}$. The parameter $\mathcal{R}.\mathrm{rew}$ determines how many times $\mathcal{R}$ will perform a full rewind of $\mathcal{A}$. Then it runs $\mathcal{A}$ while simulating its encryption and decryption oracles. For every encryption query, $\mathcal{R}$ runs $\mathcal{R}.\mathsf{SimEnc}$ with the query and its state as input to produce the updated state, a flag rf, and a ciphertext. If the flag rf is true, then $\mathcal{R}$ starts running $\mathcal{A}$ from the beginning again. Otherwise, it answers with the query answer $\mathcal{R}.\mathsf{SimEnc}$ returned. Decryption queries are handled analogously. If $\mathcal{R}$ did not rewind $\mathcal{A}$ before $\mathcal{A}$ finished its execution, then it runs $\mathcal{R}.\mathsf{Upd}$ on $\mathcal{A}$'s output to updates its state and starts running $\mathcal{A}$ from the beginning if has not already rewinded $\mathcal{R}.\mathrm{rew}$ times. Finally, $\mathcal{R}$ outputs whatever $\mathcal{R}.\mathsf{F}^{\mathrm{inner}}(\sigma)$ returns. The following definition captures some restrictions we will place on reductions.

Definition 6.1.1. *Let $\mathcal{R}$ be a reduction using the syntax from Fig. 6.1. It is* full-rewinding *if $\mathcal{R}.\mathrm{rew} > 0$ or* straightline *if $\mathcal{R}.\mathrm{rew} = 0$. It is* nonce-respecting *if $\mathcal{R}[\mathcal{A}]$ is nonce-respecting when $\mathcal{A}$ is nonce-respecting. It is* faithful *if $\mathcal{R}[\mathcal{A}]$ answers encryption queries of $\mathcal{A}$ consistent with its own encryption oracle, i.e., $\mathcal{R}$ responds with C on an encryption query made on (N, M), only if it previously queried its own encryption oracle with (N, M) and received C as the answer.*

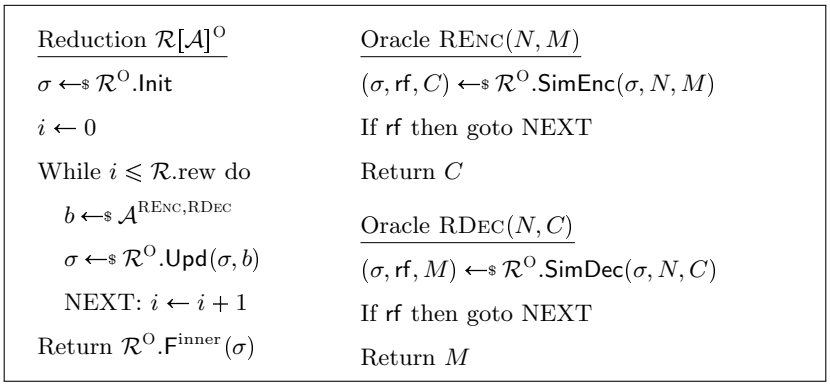


Figure 6.1: Syntax of a black-box reduction $\mathcal{R}$ running AE-1 adversary $\mathcal{A}$. We represent the oracles $\mathcal{R}$ has access to collectively as O.

Additional notation. We fix an understood nonce-based encryption scheme NE for which we assume that $\{0,1\}^{\mathsf{ml}} \subseteq \mathsf{NE.M}$. We also assume $\mathbb{N} \times \mathbb{N} \subseteq \mathsf{NE.N}$ and we use $\mathsf{N} = \mathsf{NE.N}$ as shorthand. We assume that $[\mathsf{NE.K}] = \{0,1\}^{\mathsf{kl}}$. We let $\mathsf{C} = \{0,1\}^{\mathsf{NE.cl}(\mathsf{ml})}$. We also introduce some new notation for the complexity of an algorithm $\mathcal{A}$. First, $\mathbf{Mem}(\mathcal{A})$ is defined as the number of bits of memory that $\mathcal{A}$ uses. The total number of queries to its oracles is $\mathbf{Query}(\mathcal{A})$, and the number of computation steps $\mathbf{Time}(\mathcal{A})$. For a reduction $\mathcal{R}$ we use $\mathbf{Mem}(\mathcal{R})$ to denote the number of bits of memory that $\mathcal{R}$ uses in addition any memory of the adversary it runs.

Information theoretic lemma. We give a lemma that will be a useful sub-component of our proofs. It pertains to game $\mathsf{G}^{\mathsf{it\text{-}chl\text{-}}r}_{u,m}$ in Fig. 6.2. It is an r-round game, played by a two-stage adversary $(\mathcal{D}_1, \mathcal{D}_2)$. In each round, $\mathcal{D}_1$ gets state σ from the prior round, along with u random strings $M_1, \ldots, M_u$ each of length m. Adversary $\mathcal{D}_1$ outputs state σ which is input to $\mathcal{D}_2$ along with a randomly sampled index j^* from $[u]$. Then $\mathcal{D}_2$ outputs a string M and state σ that is passed to $\mathcal{D}_1$ in the next round. If $M = M_{j^*}$, we say that $(\mathcal{D}_1, \mathcal{D}_2)$ has answered the challenge of this round correctly. If $(\mathcal{D}_1, \mathcal{D}_2)$ answers all the r challenges correctly, the game returns true. Otherwise it returns false. We define $\mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}}r}_{u,m}(\mathcal{D}_1, \mathcal{D}_2) = \mathsf{Pr}[\mathsf{G}^{\mathsf{it\text{-}chl\text{-}}r}_{u,m}(\mathcal{D}_1, \mathcal{D}_2)]$. Adversary $(\mathcal{D}_1, \mathcal{D}_2)$ is S-bounded if the state output by $\mathcal{D}_1$ is at most S bits long. We can prove the following.

Lemma 6.1.2. *If $(\mathcal{D}_1, \mathcal{D}_2)$ is S-bounded, then*

$$\mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}}r}_{u,m}(\mathcal{D}_1, \mathcal{D}_2) \leqslant \left(\frac{2(S+m)}{u} + \frac{3}{2^m}\right)^r .$$

The proof reduces to the $r = 1$ case and analyzes this case using techniques from the AI-ROM setting [CDGS18].

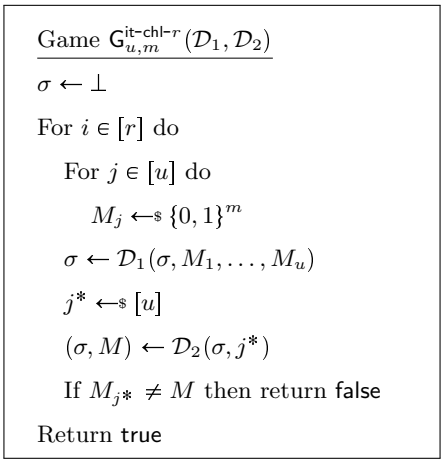


Figure 6.2: Information theoretic game played by adversary $(\mathcal{D}_1, \mathcal{D}_2)$.

Proof. We split this lemma into the following two claims. The first claim bounds advantage of any S-bounded adversary playing the one round version of the game ($r = 1$). The second step bounds the advantage of an adversary playing the full r rounds in using an adversary playing only one round.

Claim 6.1.3. *If $(\mathcal{D}_1, \mathcal{D}_2)$ is S-bounded, then* $\mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}1}}_{u,m}(\mathcal{D}_1, \mathcal{D}_2) \leqslant \frac{2S+2m}{u} + \frac{3}{2^m}$

Claim 6.1.4. *If $\mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}1}}_{u,m}(\mathcal{D}_1, \mathcal{D}_2) \leqslant \varepsilon$ for all S-bounded $(\mathcal{D}_1, \mathcal{D}_2)$, then $\mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}}r}_{u,m}(\mathcal{D}_1^*, \mathcal{D}_2^*) \leqslant \varepsilon^r$ for all r and all S-bounded $(\mathcal{D}_1^*, \mathcal{D}_2^*)$.*

First claim. In order to show the upper bound on $\mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}1}}_{u,m}(\mathcal{D}_1, \mathcal{D}_2)$, we first define the game $\mathsf{G}^{\mathsf{bf\text{-}it\text{-}chl}}_{u,m,P}$ in Fig. 6.3. It is the "Bit-Fixed Random Oracle" [CDGS18] version of $\mathsf{G}^{\mathsf{it\text{-}chl\text{-}1}}_{u,m}$. (The sequence of u random strings that are input to the adversary can be viewed as a random oracle $[u] \to \{0,1\}^m$.) This game is played by a two-stage adversary $(\mathcal{B}_1, \mathcal{B}_2)$. The first stage $\mathcal{B}_1$ outputs two lists I and Q, each of length P. All the entries of I are in $[u]$ and all the entries of Q are in $\{0,1\}^m$. Then u bit-strings $M_1, \cdots, M_u$ are sampled uniformly at random from $\{0,1\}^m$. For each $j \in [P]$, the bit-string M_i is overwritten by $Q[j]$, where $i = I[j]$. The second stage adversary $\mathcal{B}_2$ is given the lists and a randomly sampled j^* as input. It returns $M \in \{0,1\}^m$. The game returns true if $M = M_{j*}$ and false otherwise. We define $\mathsf{Adv}^{\mathsf{bf\text{-}it\text{-}chl}}_{u,m,P}(\mathcal{B}_1, \mathcal{B}_2) = \mathsf{Pr}[\mathsf{G}^{\mathsf{bf\text{-}it\text{-}chl}}_{u,m,P}(\mathcal{B}_1, \mathcal{B}_2)]$.

We claim that $\mathsf{Adv}^{\mathsf{bf\text{-}it\text{-}chl}}_{u,m,P}(\mathcal{B}_1, \mathcal{B}_2) \leqslant \frac{P}{u} + \frac{1}{2^m}$. The first term captures the probability that j^* is in I. The second term captures the probability that $\mathcal{B}_2$ guesses the random string $M_{j*} \in \{0,1\}^m$ when j^* is not in I. Since $(\mathcal{D}_1, \mathcal{D}_2)$ is S-bounded, Theorem 6 of [CDGS18] gives that for any $P \in \mathbb{N}$ and every $\gamma > 0$, if $P \geqslant S + \log \gamma^{-1}$ then there exists an adversary $(\mathcal{B}_1, \mathcal{B}_2)$ such that

$$\mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}1}}_{u,m}(\mathcal{D}_1, \mathcal{D}_2) \leqslant 2 \cdot \mathsf{Adv}^{\mathsf{bf\text{-}it\text{-}chl}}_{u,m,P}(\mathcal{B}_1, \mathcal{B}_2) + \gamma\ .$$

Game $\mathsf{G}^{\mathsf{bf\text{-}it\text{-}chl}}_{u,m,P}(\mathcal{B}_1,\mathcal{B}_2)$	Adversary $\mathcal{D}'_1(\perp, M_1, \cdots, M_u)$
$(I,Q) \leftarrow\!\!\$\ \mathcal{B}_1$ // $\lvert I\rvert = \lvert Q\rvert = P$	While(true)
For $j \in [u]$ do	$\mathsf{win} \leftarrow 1; \phi \leftarrow \perp$
$M_j \leftarrow\!\!\$\ \{0,1\}^m$	For $k \in [i-1]$ do
For $j \in [P]$ do	For $j \in [u]$ do
$M_{I[j]} \leftarrow Q[j]$	$M'_j \leftarrow\!\!\$\ \{0,1\}^m$
$j^* \leftarrow\!\!\$\ [u]$	$\phi \leftarrow \mathcal{D}^*_1(\phi, M'_1, \cdots, M'_u)$
$M \leftarrow\!\!\$\ \mathcal{B}_2(I,Q,j^*)$	$j^* \leftarrow\!\!\$\ [u]$
If $M_{j^*} \neq M$ then return false	$(\phi, M') \leftarrow \mathcal{D}^*_2(\phi, j^*)$
Return true	If $M' \neq M'_{j^*}$ then $\mathsf{win} \leftarrow 0$
	If $\mathsf{win} = 1$ then break
	$\phi \leftarrow \mathcal{D}^*_1(\phi, M_1, \cdots, M_u)$
	Return ϕ

Figure 6.3: Game $\mathsf{G}^{\mathsf{bf\text{-}it\text{-}chl}}_{u,m,P}$ and adversary $\mathcal{D}'_1$ used in the proof of Lemma 6.1.2.

Setting $\gamma = \frac{1}{2^m}$ and $P = S + m$, gives the claimed bound.

Second claim. Let us assume, towards contradiction, that there exists S-bounded $(\mathcal{D}^*_1, \mathcal{D}^*_2)$ such that $\mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}}r}_{u,m}(\mathcal{D}^*_1, \mathcal{D}^*_2) > \varepsilon^r \neq 0$. Then we can construct an adversary $(\mathcal{D}'_1, \mathcal{D}'_2)$ such that $\mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}}1}_{u,m}(\mathcal{D}'_1, \mathcal{D}'_2) > \varepsilon$. We give $\mathcal{D}'_1$ in Fig. 6.3 and let $\mathcal{D}'_2 = \mathcal{D}^*_2$. We say $(\mathcal{D}^*_1, \mathcal{D}^*_2)$ wins round j if $\mathsf{G}^{\mathsf{it\text{-}chl\text{-}}r}_{u,m}$ did not return false before the $(j+1)^{\text{st}}$ iteration. It follows that,

$$\mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}}r}_{u,m}(\mathcal{D}^*_1, \mathcal{D}^*_2) = \prod_{j=1}^{r} \mathsf{Pr}\left[(\mathcal{D}^*_1, \mathcal{D}^*_2) \text{ wins round } j \,\middle|\, (\mathcal{D}^*_1, \mathcal{D}^*_2) \text{ wins round } j-1\right].$$

Since $\mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}}r}_{u,m}(\mathcal{D}^*_1, \mathcal{D}^*_2) > \varepsilon^r$, there must exist some $i \in [r]$, such that,

$$\mathsf{Pr}\left[(\mathcal{D}^*_1, \mathcal{D}^*_2) \text{ wins round } i \,\middle|\, (\mathcal{D}^*_1, \mathcal{D}^*_2) \text{ wins round } i-1\right] > \varepsilon\,.$$

Adversary $\mathcal{D}'_1$ keeps running $(\mathcal{D}^*_1, \mathcal{D}^*_2)$ until it wins round $i-1$ on randomly generated $M_{i,j}$'s. Once this occurs $\mathcal{D}'_1$ runs $\mathcal{D}^*_1$ on its own inputs and returns the state returned by $\mathcal{D}^*_1$. Adversary $\mathcal{D}'_2$ runs $\mathcal{D}^*_2$ on its own inputs and returns whatever $\mathcal{D}^*_2$ returns. Note that $(\mathcal{D}'_1, \mathcal{D}'_2)$ is S-bounded because $(\mathcal{D}^*_1, \mathcal{D}^*_2)$ is. Moreover,

$$\mathsf{Pr}\left[\mathsf{G}^{\mathsf{it\text{-}chl\text{-}}1}_{u,m}(\mathcal{D}'_1, \mathcal{D}'_2)\right] = \mathsf{Pr}\left[(\mathcal{D}^*_1, \mathcal{D}^*_2) \text{ win round } i \,\middle|\, (\mathcal{D}^*_1, \mathcal{D}^*_2) \text{ win round } i-1\right]\,.$$

Thus $\mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}}r}_{u,m}(\mathcal{D}'_1, \mathcal{D}'_2) > \varepsilon$, which contradicts our assumption.

■

6.1.1 Memory Lower Bound for Straightline Reductions

Our first theorem shows that it is not possible to give memory-tight, straightline reductions proving the AE-1 security of an encryption scheme from its INDR and CTXT-1 security. (As the theorem statement is somewhat complicated, we will describe how to interpret it below.)

Theorem 6.1.5 (Impossibility for straightline reductions)**.** *Let* NE *be a nonce-based encryption scheme. Fix* $u, r \in \mathbb{N}$ *and define the nonce-respecting adversary* $\mathcal{A}$ *as shown in Fig. 6.5. Let* $\mathcal{R}$ *be a straightline, nonce-respecting, faithful black-box reduction from* AE-1 *to nonce-respecting* INDR *with* $\mathbf{Mem}(\mathcal{R}) = S$*. Let* $\mathcal{R}'$ *be a straightline, nonce-respecting, faithful reduction from* AE-1 *to* CTXT-1*. Then, we can construct adversaries* $\mathcal{C}$ *and* $\mathcal{W}$ *such that,*

(i) $\mathsf{Adv}^{\mathsf{ae\text{-}1}}_{\mathsf{NE}}(\mathcal{A}) \geqslant 1 - \dfrac{2^{\mathsf{kl}}}{2^{2\mathsf{NE.cl}(\mathsf{ml})}}$,

(i) $\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{A}]) \leqslant 2 \cdot \left(\dfrac{2(S + \log r + \mathsf{kl}) + 2\mathsf{ml}}{u} + \dfrac{3}{2^{\mathsf{ml}}}\right)^r + 4 \cdot \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C})$,

(i) $\mathsf{Adv}^{\mathsf{ctxt\text{-}1}}_{\mathsf{NE}}(\mathcal{R}'[\mathcal{A}]) \leqslant \mathsf{Adv}^{\mathsf{ctxt\text{-}1}}_{\mathsf{NE}}(\mathcal{R}'[\mathcal{W}])$.

Moreover, $\mathcal{A}$ *satisfies* $\mathbf{Query}(\mathcal{A}) = (u+1) \cdot r + 2$ *and* $\mathbf{Mem}(\mathcal{A}) \leqslant 2\mathsf{kl} + 2\mathsf{ml} + 2\mathsf{NE.cl}(\mathsf{ml}) + 2\log|\mathsf{N}| + \log u \cdot r$*. Also* $\mathcal{C}$ *and* $\mathcal{W}$ *satisfy* $\mathbf{Query}(\mathcal{C}) < \mathbf{Query}(\mathcal{R}) + \mathbf{Query}(\mathcal{A})$*,* $\mathbf{Time}(\mathcal{C}) \in O(\mathbf{Query}(\mathcal{A}) + \mathbf{Time}(\mathcal{R}))$*, and* $\mathbf{Query}(\mathcal{W}) = \mathbf{Query}(\mathcal{A})$ *and* $\mathbf{Time}(\mathcal{W}) \in O(\mathbf{Query}(\mathcal{A}))$*.*

To interpret this theorem, assume that the parameters of NE are such that the advantage of $\mathcal{A}$ is essentially one. Hence, a successful pair of reductions $\mathcal{R}$ and $\mathcal{R}'$ would need at least one of $\mathcal{R}[\mathcal{A}]$ or $\mathcal{R}'[\mathcal{A}]$ to have high advantage. For memory-tight $\mathcal{R}$ and $\mathcal{R}'$ we expect there to be linear functions f_1 and f_2 such that their local computation time and memory usage when interacting with an adversary $\mathcal{A}$ would be bounded by $f_1(q_{\mathcal{A}})$ and $f_2(s_{\mathcal{A}})$ where $q_{\mathcal{A}} = \mathbf{Query}(\mathcal{A})$ and $s_{\mathcal{A}} = \mathbf{Mem}(\mathcal{A})$.

Suppose this was the case. Then we can fix upper bounds for $\log(u)$ and $\log(r)$, determining the memory usage of $\mathcal{A}$ and hence $f_1(s_{\mathcal{A}}) = S$. Now we can pick reasonable u and r such that, $2 \cdot \left(\frac{2(S+\log r+\mathsf{kl})+2\mathsf{ml}}{u} + \frac{3}{2^{\mathsf{ml}}}\right)^r$ is very small (by say making the inside of the parenthesis less than $1/2$ and setting $r = 128$). Then, for one of the reductions to have high advantage, one of $\mathcal{C}$ or $\mathcal{R}'[\mathcal{W}]$ would have to have high advantage. But the efficiencies of these are bounded as small functions of the query complexity of $\mathcal{A}$ (rather than its local runtime) so cannot be too large. But then assuming security of NE prevents any of them from having high advantage.

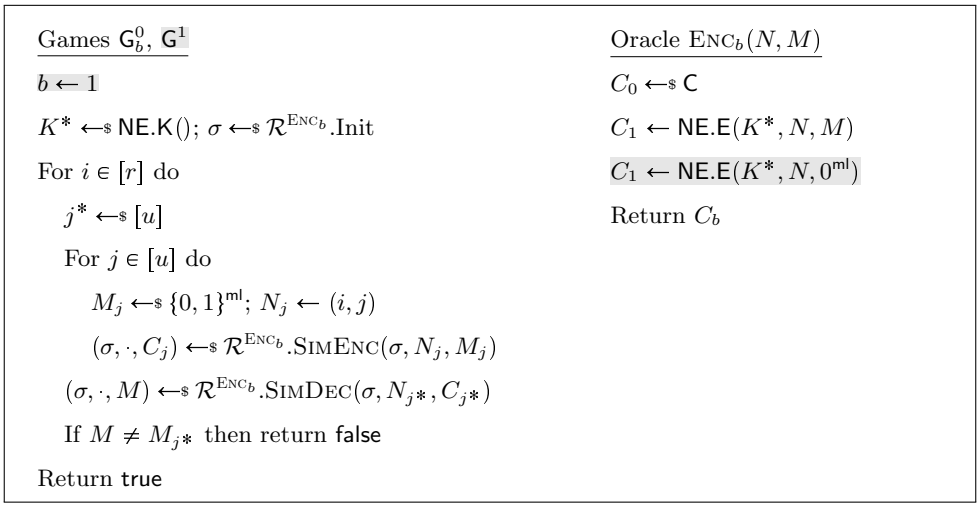


Figure 6.4: Games G^1 and G^0_b for $b \in \{0,1\}$. Highlighted code is only included in G^1.

Proof. Consider the adversary $\mathcal{A}$ in Fig. 6.5 against AE-1 security of NE. Note that it is nonce-respecting. It has a challenge phase followed by an invocation of $\mathcal{B}$. Each iteration of the challenge phase consists of $\mathcal{A}$ making u encryption queries with unique nonces and making one decryption query on one of the u ciphertexts it received as answers chosen uniformly at random with its corresponding nonce. If the answer of the decryption query is not consistent with the prior encryption query, $\mathcal{A}$ returns 1. There are r iterations of the challenge phase. If these are all passed, $\mathcal{A}$ runs adversary $\mathcal{B}$ (shown on the right) with its Enc oracle and outputs whatever $\mathcal{B}$ outputs. From the code of $\mathcal{A}$ we can see that it makes $r \cdot u + 2$ encryption queries, r decryption queries, and satisfies

$$\mathbf{Mem}(\mathcal{A}) \leqslant 2\mathsf{kl} + 2\mathsf{ml} + 2\mathsf{NE.cl}(\mathsf{ml}) + 2\log|\mathsf{N}| + \log u \cdot r\ .$$

To prove the theorem we need to separately establish the three advantage claims (and corresponding statements about the efficiency of various algorithms). For the first claim, note that $\mathsf{Adv}^{\mathsf{ae\text{-}1}}_{\mathsf{NE}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{ae\text{-}1}}_{\mathsf{NE}}(\mathcal{B})$ because M will always equal M_{j*} when $\mathcal{A}$ is playing $\mathsf{G}^{\mathsf{ae\text{-}1}}_{\mathsf{NE},b}$. The simple analysis giving the needed bound on $\mathsf{Adv}^{\mathsf{ae\text{-}1}}_{\mathsf{NE}}(\mathcal{B})$ is deferred to Section 6.3.

For the third claim, consider adversary $\mathcal{W}$ defined as shown in Fig. 6.5. It is identical to $\mathcal{A}$, except that it calls $\mathcal{E}$, which is similar to $\mathcal{B}$ but always returns 1. Because $\mathcal{R}'$ is faithful, $\mathcal{B}$ would never return 0 when run by $\mathcal{R}'[\mathcal{A}]$ playing $\mathsf{G}^{\mathsf{ctxt\text{-}1}}_{\mathsf{NE},b}$ so $\mathsf{Adv}^{\mathsf{ctxt\text{-}1}}_{\mathsf{NE}}(\mathcal{R}'[\mathcal{A}]) = \mathsf{Adv}^{\mathsf{ctxt\text{-}1}}_{\mathsf{NE}}(\mathcal{R}'[\mathcal{W}])$ holds trivially.

We spend the rest of the proof establishing the second claim. Consider the adversary $\mathcal{S}$ in Fig. 6.5. It behaves identically to $\mathcal{A}$ until the flag bad is set. Using the Fundamental Lemma of Game Playing [BR06],

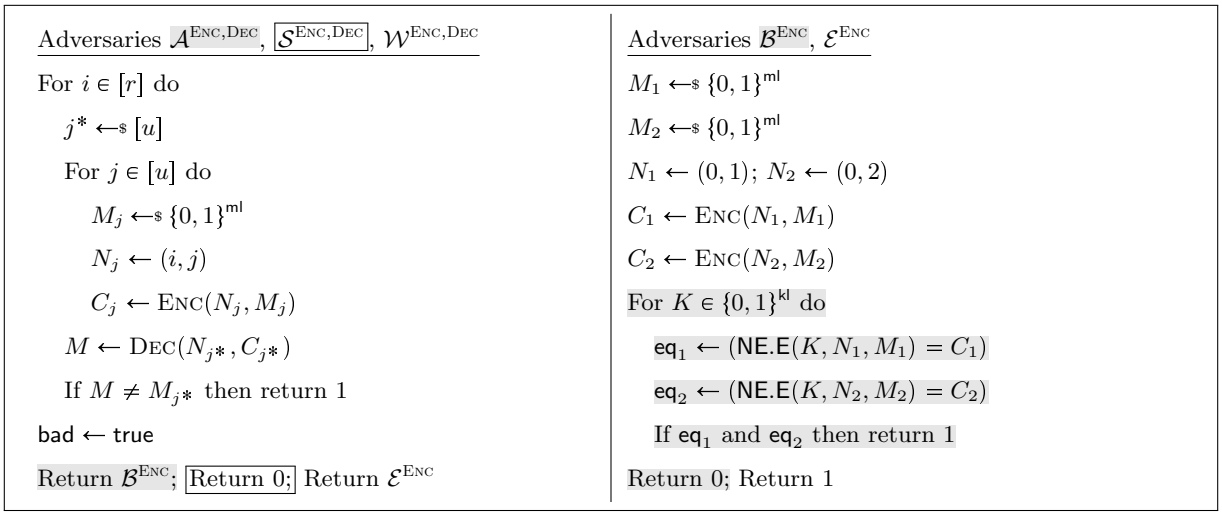


Figure 6.5: Adversaries against the AE-1 security of NE. Boxed code is only included in $\mathcal{S}$. Highlighted code is only included in $\mathcal{A}$ and $\mathcal{B}$.

we can obtain for $b \in \{0,1\}$ that

$$\left|\mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},b}(\mathcal{R}[\mathcal{A}])\right] - \mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},b}(\mathcal{R}[\mathcal{S}])\right]\right| \leqslant \mathsf{Pr}\left[\mathcal{R}[\mathcal{A}] \text{ sets } \mathsf{bad} \text{ in } \mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},b}\right] .$$

Consider the games G^0_b for $b \in \{0,1\}$ in Fig. 6.4. In it, we assume that $\mathcal{R}$ always outputs $\mathsf{rf} = \mathsf{false}$ since it is straightline. Note that G^0_b simulates the challenge phase of $\mathcal{A}$ and the game $\mathsf{G}^{\mathsf{indr}}_b$ to $\mathcal{R}$ perfectly, so it returns true whenever $\mathcal{R}[\mathcal{A}]$ would set bad is set in $\mathsf{G}^{\mathsf{indr}}_b$. From this we can show

$$\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{A}]) \leqslant \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{S}]) + \mathsf{Pr}\left[\mathsf{G}^0_0\right] + \mathsf{Pr}\left[\mathsf{G}^0_1\right] . \tag{6.1}$$

Now consider the game G^1 defined in the same figure. It is identical to either G^0_b except that it answers all encryption queries with the encryption of the message 0^{ml}. We now state two lemmas which give bounds on both $\mathsf{Pr}\left[\mathsf{G}^0_b\right]$'s via G^1. First, in Lemma 6.1.6, we use that the INDR security of NE implies G^1's encryption oracle is indistinguishable from those in either G^0_b to transition to G^1. Next, in Lemma 6.1.7 we give a bound on $\mathsf{Pr}\left[\mathsf{G}^1\right]$ which was obtained by using $\mathcal{R}$ to construct an adversary for $\mathsf{G}^{\mathsf{it\text{-}chl\text{-}}r}_{u,\mathsf{ml}}$ and bounding its advantage with Lemma 6.1.2. The proofs of these lemmas are deferred to Section 6.4.1 and Section 6.5, respectively.

Lemma 6.1.6. *There exist adversaries $\mathcal{C}_1$ and $\mathcal{C}_2$ such that*

$$\mathsf{Pr}\left[\mathsf{G}^0_1\right] \leqslant \mathsf{Pr}\left[\mathsf{G}^0_0\right] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_1),$$
$$\mathsf{Pr}\left[\mathsf{G}^0_0\right] \leqslant \mathsf{Pr}\left[\mathsf{G}^1\right] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_2)$$

where G^0_b *and* G^1 *are defined as in Fig. 6.4. Moreover* $\mathbf{Query}(\mathcal{C}_1) < \mathbf{Query}(\mathcal{R}) + \mathbf{Query}(\mathcal{A})$ *and* $\mathbf{Time}(\mathcal{C}_1) \in O(\mathbf{Query}(\mathcal{A}) + \mathbf{Time}(\mathcal{R}))$*. Adversary* $\mathcal{C}_2$*'s complexity is the same.*

Lemma 6.1.7. *If* $\mathcal{R}$ *is a straightline, nonce-respecting, faithful black-box reduction from* AE-1 *to nonce-respecting* INDR *with* $\mathbf{Mem}(\mathcal{R}) = S$*. Then,*

$$\mathsf{Pr}\left[\mathsf{G}^1\right] \leqslant \left(\frac{2(S + \log r + \mathsf{kl}) + 2\mathsf{ml}}{u} + \frac{3}{2^{\mathsf{ml}}}\right)^r$$

where G^1 *is defined as in Fig. 6.4.*

Applying these lemmas to equation (6.1) gives

$$\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{A}]) \leqslant 2 \cdot \left(\frac{2(S + \log r + \mathsf{kl}) + 2\mathsf{ml}}{u} + \frac{3}{2^{\mathsf{ml}}}\right)^r + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{S}]) + 2 \cdot \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_1) + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_2)\ .$$

To complete the proof, we combine the three INDR adversaries $\mathcal{R}[\mathcal{S}]$, $\mathcal{C}_1$, and $\mathcal{C}_2$. Let $\mathcal{C}$ be the INDR adversary that randomly chooses one of $\mathcal{R}[\mathcal{S}]$, $\mathcal{C}_1$, or $\mathcal{C}_2$ (with probabilities 1/4, 1/2, and 1/4, respectively) then runs the adversary it chose, outputting whatever that adversary does. Simple calculations give

$$4 \cdot \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}) = \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{S}]) + 2 \cdot \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_1) + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_2)\ .$$

The claimed complexity of $\mathcal{C}$ follows from that of $\mathcal{R}[\mathcal{S}]$, $\mathcal{C}_1$, and $\mathcal{C}_2$.

■

6.1.2 Memory Lower Bound for Full-Rewinding Reductions

We can extend our result to cover full-rewinding reductions as captured by the following theorem. Its interpretation works similarly to that of Thm. 6.1.5.

Theorem 6.1.8 (Impossibility for full-rewinding reductions)**.** *Let* NE *be a nonce-based encryption scheme. Fix* $u, r, c \in \mathbb{N}$*. We can construct a nonce-respecting adversary* $\mathcal{A}$ *such that for all full-rewinding, nonce-respecting, restricted reductions* $\mathcal{R}$ *from* AE-1 *to nonce-respecting* INDR *with* $\mathbf{Mem}(\mathcal{R}) = S$ *and all full-rewinding, nonce-respecting, restricted reductions* $\mathcal{R}'$ *from* AE-1 *to* CTXT-1 *there exist adversaries* $\mathcal{C}$ *and* $\mathcal{W}$ *such that,*

(i) $\mathsf{Adv}^{\mathsf{ae\text{-}1}}_{\mathsf{NE}}(\mathcal{A}) \geqslant 1 - \dfrac{2^{\mathsf{kl}}}{2^{2\mathsf{NE.cl}(\mathsf{ml})}}$,

(i) $\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{A}]) \leqslant 2 \cdot \left(\dfrac{2(S + \log r + \mathsf{kl}) + 2\mathsf{ml}}{u} + \dfrac{3}{2^{\mathsf{ml}}}\right)^r + \dfrac{2^{S+1}}{2^{c \cdot \mathsf{NE.cl}(\mathsf{ml})}} + 6 \cdot \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C})$,

(i) $\mathsf{Adv}^{\mathsf{ctxt\text{-}1}}_{\mathsf{NE}}(\mathcal{R}'[\mathcal{A}]) \leqslant \mathsf{Adv}^{\mathsf{ctxt\text{-}1}}_{\mathsf{NE}}(\mathcal{R}'[\mathcal{W}])$.

Moreover, $\mathcal{A}$ *satisfies* $\mathbf{Query}(\mathcal{A}) = c + (u+1)\cdot r + 2$ *and* $\mathbf{Mem}(\mathcal{A}) \leqslant 2\mathsf{kl} + 2\mathsf{ml} + 2\mathsf{NE.cl}(\mathsf{ml}) + 2\log|\mathsf{N}| + \log u \cdot r$. *Also* $\mathcal{C}$ *and* $\mathcal{W}$ *satisfy* $\mathbf{Query}(\mathcal{C}) < \mathbf{Query}(\mathcal{R}) + \mathbf{Query}(\mathcal{A})$, $\mathbf{Time}(\mathcal{C}) \in O(\mathbf{Query}(\mathcal{A}) + \mathbf{Time}(\mathcal{R}))$, $\mathbf{Query}(\mathcal{W}) = \mathbf{Query}(\mathcal{A})$, *and* $\mathbf{Time}(\mathcal{W}) \in O(\mathbf{Query}(\mathcal{A}))$.

The proof of this result has been deferred to Section 6.2. We give a very brief intuition about how this impossibility proof proceeds. We define a new adversary that is similar to $\mathcal{A}$ used for the proof of Thm. 6.1.5, but has an additional "buffer" phase before the challenge phase. In the buffer phase, it makes c encryption queries on a fixed message 0^{ml} using different nonces. The key idea is that if the reduction rewinds the adversary after going past the buffer phase and still manages to pass the challenge phase, it must have remembered the c ciphertexts. Because these c ciphertexts look random (from the INDR security of NE), the memory of the reduction has to grow with c. This rules out low memory reductions that pass the challenge phase after rewinding the adversary after going past the buffer phase. As in the previous section, we can show that if a reduction cannot pass the challenge phase, it cannot have a high advantage of breaking INDR security. If the reduction does not rewind after going past the buffer phase, we can bound its advantage analogously to the straightline reduction case.

6.2 Proof of Theorem 6.1.8 (Impossibility for Full-Rewinding Reductions)

In this section we prove Thm. 6.1.8 which was given in Section 6.1.2.

Consider the adversary $\mathcal{A}$ shown in Fig. 6.6 playing against AE-1 security of NE. It has a buffer phase, followed by a challenge phase, followed by an invocation of adversary $\mathcal{B}$. In the buffer phase, $\mathcal{A}$ makes c encryption queries on a fixed message 0^{ml} using different nonces. The next phase is the challenge phase. Each iteration of the challenge phase consists of $\mathcal{A}$ making u encryption queries to and asking for the decryption of one randomly chosen ciphertext from the ones it received as the answer of the encryption queries. If the answer of the decryption query is not consistent with the corresponding encryption query, then $\mathcal{A}$ halts and returns 1. There are r iterations of the challenge phase. If all of these phases are passed, then $\mathcal{A}$ runs adversary $\mathcal{B}$ which is defined in the same figure (identically to the $\mathcal{B}$ that was defined in Fig. 6.5). From the code of $\mathcal{A}$ we can see that it makes $r \cdot u + c + 2$ encryption queries, r decryption queries, and satisfies $\mathbf{Mem}(\mathcal{A}_2) \leqslant 2\mathsf{kl} + 2\mathsf{ml} + 2\mathsf{NE.cl}(\mathsf{ml}) + 2\log|\mathsf{N}| + \log u \cdot r$.

To prove the theorem we need to separately establish the three advantage claims (and corresponding statements about the efficiency of various algorithms).

First and third claims. For the first claim, note that $\mathsf{Adv}^{\mathsf{ae\text{-}1}}_{\mathsf{NE}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{ae\text{-}1}}_{\mathsf{NE}}(\mathcal{B})$ because M will always equal M_{j*} when $\mathcal{A}$ is playing $\mathsf{G}^{\mathsf{ae\text{-}1}}_{\mathsf{NE},b}$. The simple analysis showingn that $\mathsf{Adv}^{\mathsf{ae\text{-}1}}_{\mathsf{NE}}(\mathcal{B}) \geqslant 1 - 2^{\mathsf{kl}}/2^{2\mathsf{NE.cl}(\mathsf{ml})}$ is given in Section 6.3.

For the third claim, consider adversary $\mathcal{W}$ defined as shown in Fig. 6.6. It is identical to $\mathcal{A}$, except that

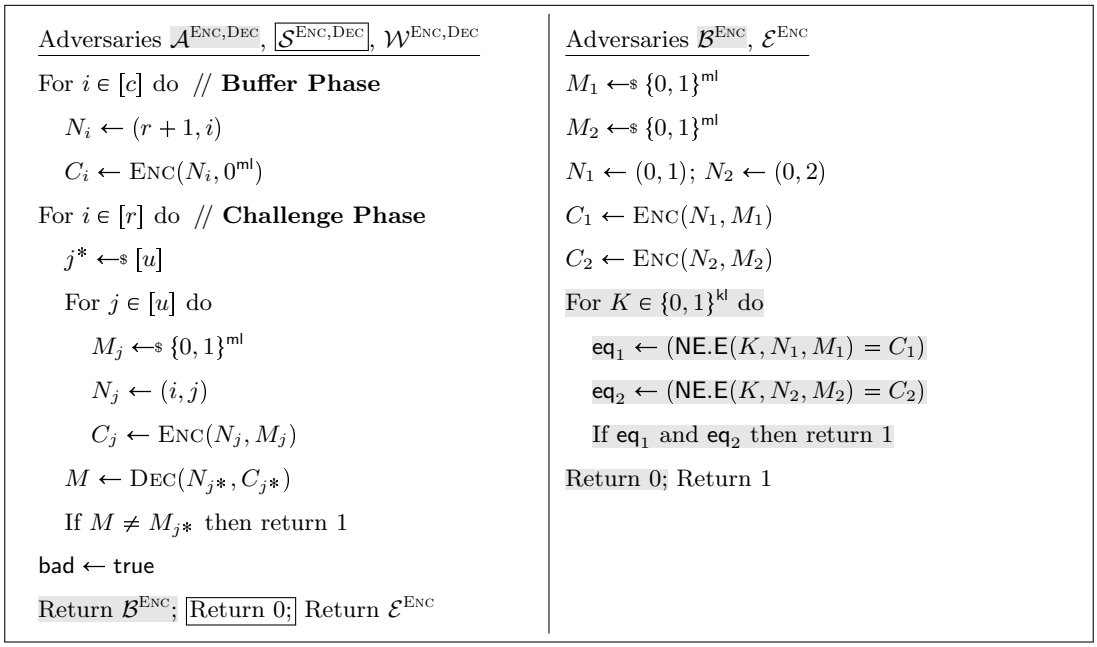


Figure 6.6: Adversaries against the AE-1 security of NE used for proof of Thm. 6.1.8. Highlighted code is only included in $\mathcal{A}$. Boxed code is only included in $\mathcal{S}$. Adversaries $\mathcal{B}$ (used by $\mathcal{A}$) and $\mathcal{E}$ (used by $\mathcal{W}$) are repeated from Fig. 6.5.

it calls $\mathcal{E}$ which is defined in the same figure (identically to the $\mathcal{E}$ that was defined Fig. 6.5), which is similar to $\mathcal{B}$ but always returns 1. Because $\mathcal{R}'$ is faithful, $\mathcal{B}$ would never return 0 when run by $\mathcal{R}'[\mathcal{A}]$ playing $\mathsf{G}^{\mathsf{ctxt}\text{-}1}_{\mathsf{NE},b}$ so $\mathsf{Adv}^{\mathsf{ctxt}\text{-}1}_{\mathsf{NE}}(\mathcal{R}'[\mathcal{A}]) = \mathsf{Adv}^{\mathsf{ctxt}\text{-}1}_{\mathsf{NE}}(\mathcal{R}'[\mathcal{W}])$ holds trivially.

Second claim. We spend the rest of the proof establishing the second claim. Consider the adversary $\mathcal{S}$ in Fig. 6.6. It behaves identically to $\mathcal{A}$ until the flag bad is set. Using the Fundamental Lemma of Game Playing [BR06], we obtain for $b \in \{0,1\}$ that

$$\left|\mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},b}(\mathcal{R}[\mathcal{A}])\right] - \mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},b}(\mathcal{R}[\mathcal{S}])\right]\right| \leqslant \mathsf{Pr}\left[\mathcal{R}[\mathcal{A}] \text{ sets } \mathsf{bad} \text{ in } \mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},b}\right] .$$

We will use the sequence of games shown in Fig. 6.7 to bound these probabilities that $\mathcal{R}[\mathcal{A}]$ sets bad. Start by considering the games G^0_b for $b \in \{0,1\}$. They were obtained by plugging the code of $\mathcal{R}[\mathcal{A}]$ into $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},b}$, then changing the code to output true if $\mathcal{R}$ passes all of $\mathcal{A}$'s challenges and output false otherwise (and adding a flag crw that we will use later). In other words, G^0_b outputs true whenever $\mathcal{R}[\mathcal{A}]$ would set bad in $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},b}$. This gives us that

$$\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{A}]) \leqslant \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{S}]) + \mathsf{Pr}\left[\mathsf{G}^0_0\right] + \mathsf{Pr}\left[\mathsf{G}^0_1\right] . \tag{6.2}$$

Games G_b^0, $\boxed{\mathsf{G}^1}$, $\boxed{\mathsf{h}^2}$

$\boxed{b \leftarrow 1}$

$K^* \leftarrow\!\!\$\ \mathsf{NE.K}(); k \leftarrow 0; \sigma \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}_b}.\mathsf{Init}$

While $k \leqslant \mathcal{R}.\text{rew}$ do

  For $i \in [c]$ do // **Buffer Phase**

    $N_i \leftarrow (r+1, i)$

    $(\sigma, \mathsf{rf}, C_i) \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}_b}.\mathsf{SimEnc}(\sigma, N_i, 0^{\mathsf{ml}})$

    If rf then goto NEXT2

  For $i \in [r]$ do // **Challenge Phase**

    $j^* \leftarrow\!\!\$\ [u]$

    For $j \in [u]$ do

      $M_j \leftarrow\!\!\$\ \{0,1\}^{\mathsf{ml}}$

      $N_j \leftarrow (i, j)$

      $(\sigma, \mathsf{rf}, C_j) \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}_b}.\mathsf{SimEnc}(\sigma, N_j, M_j)$

      If rf then goto NEXT

    $(\sigma, \mathsf{rf}, M) \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}_b}.\mathsf{SimDec}(\sigma, N_{j^*}, C_{j^*})$

    If rf then goto NEXT

    If $M \neq M_{j^*}$ then

      $\sigma \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}_b}.\mathsf{Upd}(\sigma, 1)$

      Goto NEXT

  Return true

  NEXT: $\mathsf{crw} \leftarrow \mathsf{true}$; $k \leftarrow \mathcal{R}.\text{rew} + 1$

  NEXT2: $k \leftarrow k + 1$

Return false

Oracle $\text{ENC}_b(N, M)$

$C_0 \leftarrow\!\!\$\ \mathsf{C}$

$C_1 \leftarrow \mathsf{NE.E}(K^*, N, M)$

$\boxed{C_1 \leftarrow \mathsf{NE.E}(K^*, N, 0^{\mathsf{ml}})}$

Return C_b

Games G^2, $\boxed{\mathsf{G}^3}$

$b \leftarrow 1$

$K^* \leftarrow\!\!\$\ \mathsf{NE.K}(); k \leftarrow 0; \sigma \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}_b}.\mathsf{Init}$

While $k \leqslant \mathcal{R}.\text{rew}$ do

  For $i \in [c]$ do // **Buffer Phase**

    $N_i \leftarrow (r+1, i)$

    $(\sigma, \mathsf{rf}, C_i) \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}_b}.\mathsf{SimEnc}(\sigma, N_i, 0^{\mathsf{ml}})$

    If rf then goto NEXT2

  For $i \in [r]$ do // **Challenge Phase**

    $j^* \leftarrow\!\!\$\ [u]$

    For $j \in [u]$ do

      $M_j \leftarrow\!\!\$\ \{0,1\}^{\mathsf{ml}}$

      $N_j \leftarrow (i, j)$

      $(\sigma, \mathsf{rf}, C_j) \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}_b}.\mathsf{SimEnc}(\sigma, N_j, M_j)$

      If rf then goto NEXT

    $(\sigma, \mathsf{rf}, M) \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}_b}.\mathsf{SimDec}(\sigma, N_{j^*}, C_{j^*})$

    If rf then goto NEXT

    If $M \neq M_{j^*}$ then

      $\sigma \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}_b}.\mathsf{Upd}(\sigma, 1)$

      Goto NEXT

  Return $\mathsf{true} \wedge \mathsf{crw}$

  NEXT: $\mathsf{crw} \leftarrow \mathsf{true}$

  NEXT2: $k \leftarrow k + 1$

Return false

Oracle $\text{ENC}_b(N, M)$

$C_1 \leftarrow \mathsf{NE.E}(K^*, N, 0^{\mathsf{ml}})$

$\boxed{C_1 \leftarrow\!\!\$\ \mathsf{C}}$

Return C_b

Figure 6.7: Sequence of games used in proof Theorem 6.1.8. On the left, boxed code is *not* included in G^1 and highlighted code is only included in h^2. On the right, highlighted code is included in both games (indicating where they differ from G^1) and boxed code is only included in G^3.

Now consider the game G^1 defined in the same figure. It is identical to either G^0_b except that it answers all encryption queries with the encryption of the message 0^{ml}. The following lemma allows us to bound the $\mathsf{Pr}\left[\mathsf{G}^0_b\right]$'s using $\mathsf{Pr}[\mathsf{G}^1]$ and the advantage of two adversaries attacking the INDR security of NE. Its proof is given in Section 6.4.2.

Lemma 6.2.1. *There exist adversaries $\mathcal{C}_1$ and $\mathcal{C}_2$ such that*

$$\mathsf{Pr}\left[\mathsf{G}^0_1\right] \leqslant \mathsf{Pr}\left[\mathsf{G}^0_0\right] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_1) \text{ and } \mathsf{Pr}\left[\mathsf{G}^0_0\right] \leqslant \mathsf{Pr}\left[\mathsf{G}^1\right] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_2)$$

where G^0_b and G^1 are defined as in Fig. 6.7. Moreover $\mathbf{Query}(\mathcal{C}_1) < \mathbf{Query}(\mathcal{R}) + \mathbf{Query}(\mathcal{A})$ *and* $\mathbf{Time}(\mathcal{C}_1) \in O(\mathbf{Query}(\mathcal{A}) + \mathbf{Time}(\mathcal{R}))$. *Adversary $\mathcal{C}_2$'s complexity is the same.*

Next we are going to bound $\mathsf{Pr}\left[\mathsf{G}^1\right] \leqslant \mathsf{Pr}[\mathrm{h}^2] + \mathsf{Pr}[\mathsf{G}^2]$. Game h^2 does not let $\mathcal{R}$ rewind again if it has already passed the buffer, so it captures the possibility that $\mathcal{R}$ passes the challenges on its first attempt past the buffer. Game G^2 only outputs true if $\mathcal{R}$ has passed the buffer, then rewinded, and then passed the buffer again before successfully completing the challenge. To make this formal, let crw denote the event that $\mathsf{crw} = \mathsf{true}$ at the end of the execution of a game and $\overline{\mathsf{crw}}$ to denote the opposite event in the following probability statements. We claim that,

$$\mathsf{Pr}\left[\mathsf{G}^1 \wedge \overline{\mathsf{crw}}\right] \leqslant \mathsf{Pr}\left[\mathrm{h}^2\right] \text{ and } \mathsf{Pr}\left[\mathsf{G}^1 \wedge \mathsf{crw}\right] \leqslant \mathsf{Pr}\left[\mathsf{G}^2\right]. \tag{6.3}$$

The former of these holds because G^1 and h^2 are identical until crw has been set (and hence $\mathsf{Pr}\left[\mathsf{G}^1 \wedge \overline{\mathsf{crw}}\right] = \mathsf{Pr}\left[\mathrm{h}^1 \wedge \overline{\mathsf{crw}}\right]$). The latter holds because G^2 is identical to G^1 except it only outputs true if G^1 would *and* $\mathsf{crw} = \mathsf{true}$.

Because $\mathcal{R}$ only has one attempt to pass the challenge phase in h^2, we can directly bound the probability that it succeeds in this by using $\mathcal{R}$ to construct an adversary for $\mathsf{G}^{\mathsf{it\text{-}chl\text{-}}r}_{u,\mathsf{ml}}$ and bounding its advantage with Lemma 6.1.2. This gives the following lemma.

Lemma 6.2.2. *Let $\mathcal{R}$ be a full-rewinding, nonce-respecting, faithful black-box reduction from* AE-1 *to nonce-respecting* INDR *with* $\mathbf{Mem}(\mathcal{R}) = S$. *Then,*

$$\mathsf{Pr}\left[\mathrm{h}^2\right] \leqslant \left(\frac{2(S + \log r + \mathsf{kl}) + 2\mathsf{ml}}{u} + \frac{3}{2^{\mathsf{ml}}}\right)^r$$

where h^2 is defined as in Fig. 6.7.

Proof. Consider the adversary $(\mathcal{D}_1, \mathcal{D}_2)$ for $\mathsf{G}^{\mathsf{it\text{-}chl\text{-}}r}_{u,\mathsf{ml}}$ defined in Fig. 6.8. It assumes that $\mathcal{R}$ will succeed in h^2 and tries to use this to succeed in its own game. The first time $\mathcal{D}_1$ is executed (i.e. when $\phi = \perp$) it simulates the buffer phase, allowing $\mathcal{R}$ to rewind up to $\mathcal{R}.\mathsf{rew}$ times until $\mathcal{R}$ completes the buffer phase (if $\mathcal{R}$ never

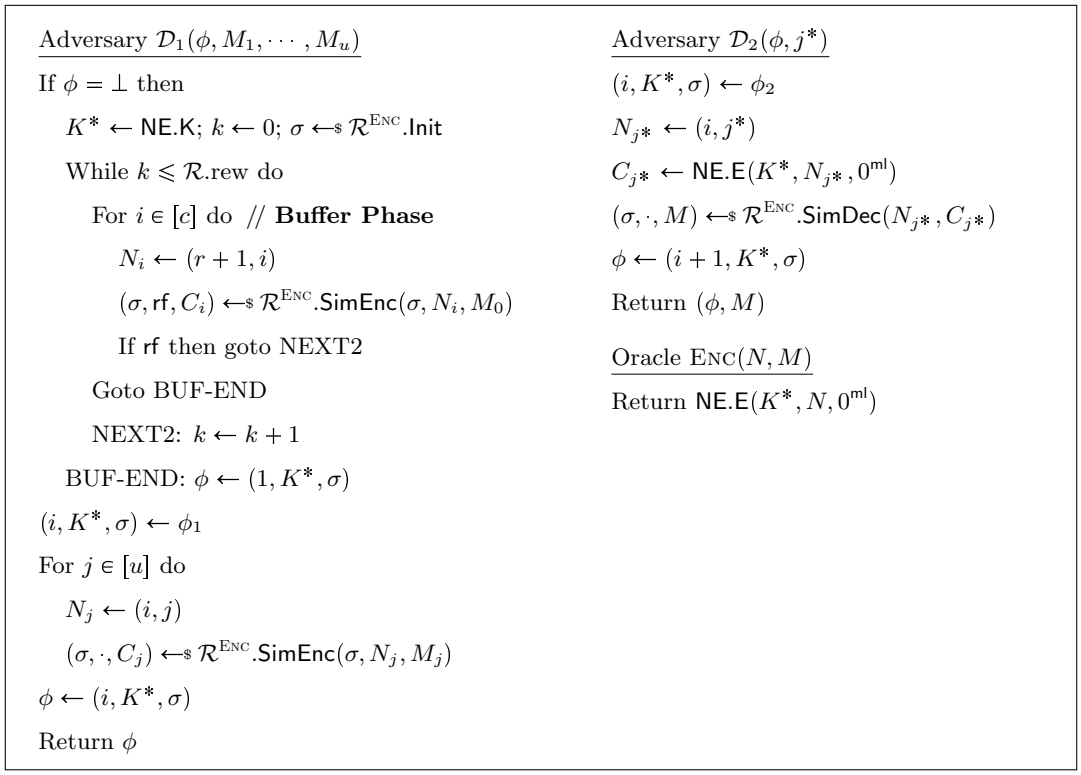


Figure 6.8: Adversary $(\mathcal{D}_1, \mathcal{D}_2)$ used to prove Lemma 6.2.3.

completes the buffer phase, reaching the code "Goto BUF-END", then it couldn't have won in h^2 so we do not need to worry about the behavior of $\mathcal{D}_1$ in this case).

After this, and in all subsequent executions, $\mathcal{D}_1$ uses its input as the messages given to $\mathcal{R}$ in one iteration of the challenge. We let $\mathcal{D}_1$ ignore $\mathcal{R}$'s output rf here because if $\mathcal{R}$ ever output $\mathsf{rf} = \mathsf{true}$ then it would be unable to win that execution of h^2. Then $\mathcal{D}_1$ halts, passing the current state of $\mathcal{R}$ to $\mathcal{D}_2$ which uses $\mathcal{R}$ to attempt to answer its challenge.

The view of $\mathcal{R}$ is perfectly simulated for any execution of h^2 in which it would have succeeded, so $\mathsf{Pr}[\text{h}^2] \leqslant \mathsf{Adv}^{\mathsf{it\text{-}chl\text{-}}r}_{u,\mathsf{ml}}(\mathcal{D}_1, \mathcal{D}_2)$. The state used by $(\mathcal{D}_1, \mathcal{D}_2)$ is a tuple $\phi = (i, K^*, \sigma)$ where $|i| = \log r$, $|K^*| = \mathsf{kl}$, and $|\sigma| = S$ so $(\mathcal{D}_1, \mathcal{D}_2)$ is $S + \log r + \mathsf{kl}$-bounded. Applying Lemma 6.1.2 gives the claimed bound. ■

We now state two lemmas which give us a bound on $\mathsf{Pr}\left[\mathsf{G}^2\right]$ via G^3. First, in Lemma 6.2.3 we use the INDR security of NE to transition from G^2 to G^3 which is identical, except the encryption oracle returns random ciphertexts to $\mathcal{R}$. Its proof is given in Section 6.4.3. Now consider G^3. The reduction $\mathcal{R}$ must pass the buffer phase, then rewind, then pass the buffer phase again to win this game. Because it is faithful,

passing the buffer phase the second time requires $\mathcal{R}$ to remember c random ciphertexts. In Lemma 6.2.4, we use a compression argument (via a lemma from [DTT10]) to give an information theoretic bound on $\Pr\left[\mathsf{G}^3\right]$. Its proof is given in Section 6.6.

Lemma 6.2.3. *There exists an adversary $\mathcal{C}_3$ such that*

$$\Pr\left[\mathsf{G}^2\right] \leqslant \Pr\left[\mathsf{G}^3\right] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_3)$$

where G^2 and G^3 are defined as in Fig. 6.7. Moreover, $\mathbf{Query}(\mathcal{C}_3) < \mathbf{Query}(\mathcal{R}) + \mathbf{Query}(\mathcal{A})$ and $\mathbf{Time}(\mathcal{C}_3) \in O(\mathbf{Query}(\mathcal{A}) + \mathbf{Time}(\mathcal{R}))$.

Lemma 6.2.4. *Let $\mathcal{R}$ be a full-rewinding, nonce-respecting, faithful black-box reduction from* AE-1 *to nonce-respecting* INDR *with $\mathbf{Mem}(\mathcal{R}) = S$. Then,*

$$\Pr\left[\mathsf{G}^3\right] \leqslant \frac{2^S}{|\mathsf{C}|^c} = \frac{2^S}{2^{c \cdot \mathsf{NE.cl(ml)}}}$$

where G^3 is defined as in Fig. 6.7.

We combine (in order) Equation 6.2, Lemma 6.2.1, Equation 6.3, and Lemma 6.2.3 to obtain

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{A}]) &\leqslant \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{S}]) + \Pr\left[\mathsf{G}^0_0\right] + \Pr\left[\mathsf{G}^0_1\right] \\
&\leqslant \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{S}]) + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_1) + 2 \cdot \Pr\left[\mathsf{G}^0_0\right] \\
&\leqslant \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{S}]) + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_1) + 2 \cdot (\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_2) + \Pr\left[\mathsf{G}^1\right]) \\
&\leqslant \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{S}]) + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_1) + 2 \cdot (\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_2) + \Pr[\mathrm{h}^2] + \Pr[\mathsf{G}^2]) \\
&\leqslant \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{S}]) + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_1) + 2 \cdot (\mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_2) + \Pr[\mathrm{h}^2] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_3) + \Pr\left[\mathsf{G}^3\right]).
\end{aligned}$$

To complete the proof, we apply our information theoretic bounds from Lemma 6.2.3 and Lemma 6.2.4, then combine the four INDR adversaries $\mathcal{R}[\mathcal{S}]$, $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$. Let $\mathcal{C}$ be the INDR adversary that randomly chooses one of $\mathcal{R}[\mathcal{S}]$, $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ (with probabilities 1/6, 1/6, 2/6, and 2/6, respectively), then runs the adversary it chose, outputting whatever that adversary does. Simple calculations give

$$6 \cdot \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}) = \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{R}[\mathcal{S}]) + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_1) + 2 \cdot \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_2) + 2 \cdot \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_3)\ .$$

The claimed complexity of $\mathcal{C}$ follows from that of $\mathcal{R}[\mathcal{S}]$, $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$.

6.3 Analysis of Brute-Force Adversary's Advantage

We prove a bound on the advantage of the adversary $\mathcal{B}$ we use in the proofs of Thm. 6.1.5 and Thm. 6.1.8.

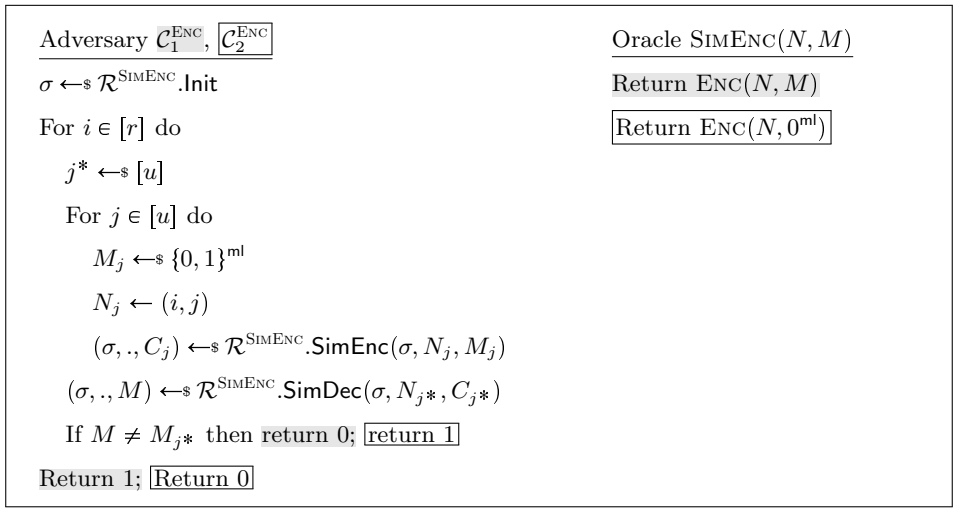


Figure 6.9: Adversaries $\mathcal{C}_1$ and $\mathcal{C}_2$ used for Lemma 6.1.6. The highlighted code is present only in $\mathcal{C}_1$ and the boxed code is present only in $\mathcal{C}_2$.

Lemma 6.3.1. *Fix* $\mathsf{kl}, \mathsf{ml} \in \mathbb{N}$. *Let* NE *be a nonce-based encryption scheme for which* $\{0,1\}^{\mathsf{ml}} \subseteq \mathsf{NE.M}$, $[\mathsf{NE.K}] = \{0,1\}^{\mathsf{kl}}$, *and* $\{(0,1),(0,2)\} \subseteq \mathsf{NE.N}$. *Then,*

$$\mathsf{Adv}^{\mathsf{ae\text{-}1}}_{\mathsf{NE}}(\mathcal{B}) \geqslant 1 - 2^{\mathsf{kl}}/2^{2\mathsf{NE.cl}(\mathsf{ml})}.$$

for $\mathcal{B}$ *as defined in Fig. 6.5.*

Proof. Adversary $\mathcal{B}$ makes two encryption queries then performs a brute-force to check if there is an encryption key consistent with those queries, returning 1 if so, and 0 otherwise. In $\mathsf{G}^{\mathsf{ae\text{-}1}}_1$ there will always be a consistent key so $\mathcal{B}$ will always return 1. For fixed M_1, M_2, and $N_1 \neq N_2$ there are at most 2^{kl} values the ordered pair $(\mathsf{NE.E}(K, N_1, M_1), \mathsf{NE.E}(K, N_2, M_2))$ can take (one for each $K \in \{0,1\}^{\mathsf{kl}}$). Moreover, there are $2^{2\mathsf{NE.cl}(\mathsf{ml})}$ different values of (C_1, C_2) that might be chosen at random by the encryption oracle, so $\mathcal{B}$ will return 1 with probability at most $2^{\mathsf{kl}}/2^{2\mathsf{NE.cl}(\mathsf{ml})}$ when interacting with $\mathsf{G}^{\mathsf{ae\text{-}1}}_{\mathsf{NE},0}$. Combining these observations gives the desired bound.

■

6.4 Proofs of Lemma 6.1.6, Lemma 6.2.1, and Lemma 6.2.3 (INDR Reductions)

6.4.1 Proof of Lemma 6.1.6

Consider the adversary $\mathcal{C}_1$ shown in Fig. 6.9. It was obtained by copying the code of G^0_0 and G^0_1 (from Fig. 6.4), then modifying the code to use its own Enc oracle to respond to encryption queries from $\mathcal{R}$ and,

to return 1 whenever the games would output true and 0 otherwise. Consequently, $\mathcal{C}_1$ playing $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}$ (resp. $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}$) perfectly simulates G^0_0 (resp. G^0_1) to $\mathcal{R}$ and we have

$$\mathsf{Pr}\left[\mathsf{G}^0_0\right] = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}(\mathcal{C}_1)\right] \text{ and } \mathsf{Pr}\left[\mathsf{G}^0_1\right] = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}(\mathcal{C}_1)\right] .$$

It follows that,

$$\mathsf{Pr}\left[\mathsf{G}^0_1\right] \leqslant \mathsf{Pr}\left[\mathsf{G}^0_0\right] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_1) .$$

The claims on $\mathcal{C}_1$'s complexity can be verified from its code.

Now consider the adversary $\mathcal{C}_2$ shown in Fig. 6.9. It was obtained by copying the code of G^0_0 and G^1 (from Fig. 6.4), then modifying the code to use its own Enc oracle to respond to encryption queries from $\mathcal{R}$, and to return 0 whenever the games would outputs true and 1 otherwise. Consequently, $\mathcal{C}_2$ playing $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}$ (resp. $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}$) perfectly simulates G^0_0 (resp. G^1) to $\mathcal{R}$ and we have

$$1 - \mathsf{Pr}\left[\mathsf{G}^0_0\right] = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}(\mathcal{C}_2)\right] \text{ and } 1 - \mathsf{Pr}\left[\mathsf{G}^1\right] = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}(\mathcal{C}_2)\right] .$$

It follows that,

$$\mathsf{Pr}\left[\mathsf{G}^0_0\right] \leqslant \mathsf{Pr}\left[\mathsf{G}^1\right] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_2) .$$

The claims on $\mathcal{C}_2$'s complexity can be verified from its code.

6.4.2 Proof of Lemma 6.2.1

Consider the adversary $\mathcal{C}_1$ shown in Fig. 6.10. It was obtained by copying the code of G^0_0 and G^0_1 (from Fig. 6.7), then modifying the code to use its own Enc oracle to respond to encryption queries from $\mathcal{R}$, and to return 1 whenever the games would output true and 0 otherwise. Consequently, $\mathcal{C}_1$ playing $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}$ (resp. $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}$) perfectly simulates G^0_0 (resp. G^0_1) to $\mathcal{R}$ and we have

$$\mathsf{Pr}\left[\mathsf{G}^0_0\right] = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}(\mathcal{C}_1)\right] \text{ and } \mathsf{Pr}\left[\mathsf{G}^0_1\right] = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}(\mathcal{C}_1)\right] .$$

It follows that,

$$\mathsf{Pr}\left[\mathsf{G}^0_1\right] \leqslant \mathsf{Pr}\left[\mathsf{G}^0_0\right] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_1) .$$

The claims on $\mathcal{C}_1$'s complexity can be verified from its code.

Now consider the adversary $\mathcal{C}_2$ shown in Fig. 6.10. It was obtained by copying the code of G^0_0 and G^1 (from Fig. 6.7), then modifying the code to use its own Enc oracle to respond to encryption queries from $\mathcal{R}$, and to return 0 whenever the games would output true and 1 otherwise. Consequently, $\mathcal{C}_2$ playing $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}$ (resp. $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}$) perfectly simulates G^0_0 (resp. G^1) to $\mathcal{R}$ and we have

$$1 - \mathsf{Pr}\left[\mathsf{G}^0_0\right] = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}(\mathcal{C}_2)\right] \text{ and } 1 - \mathsf{Pr}\left[\mathsf{G}^1\right] = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}(\mathcal{C}_2)\right] .$$

Adversaries $\mathcal{C}_1^{\text{Enc}}$, $\boxed{\mathcal{C}_2^{\text{Enc}}}$

$k \leftarrow 0$; $\sigma \leftarrow\!\!\$\ \mathcal{R}^{\text{SimEnc}}.\mathsf{Init}$

While $k \leqslant \mathcal{R}.\text{rew}$ do

  For $i \in [c]$ do // **Buffer Phase**

    $N_i \leftarrow (r+1, i)$

    $(\sigma, \mathsf{rf}, C_i) \leftarrow\!\!\$\ \mathcal{R}^{\text{SimEnc}}.\mathsf{SimEnc}(\sigma, N_i, 0^{\mathsf{ml}})$

    If rf then goto NEXT2

  For $i \in [r]$ do // **Challenge Phase**

    $j^* \leftarrow\!\!\$\ [u]$

    For $j \in [u]$ do

      $M_j \leftarrow\!\!\$\ \{0,1\}^{\mathsf{ml}}$

      $N_j \leftarrow (i, j)$

      $(\sigma, \mathsf{rf}, C_j) \leftarrow\!\!\$\ \mathcal{R}^{\text{SimEnc}}.\mathsf{SimEnc}(\sigma, N_j, M_j)$

      If rf then goto NEXT

    $(\sigma, \mathsf{rf}, M) \leftarrow\!\!\$\ \mathcal{R}^{\text{SimEnc}}.\mathsf{SimDec}(\sigma, N_{j^*}, C_{j^*})$

    If rf then goto NEXT

    If $M \neq M_{j^*}$ then

      $\sigma \leftarrow\!\!\$\ \mathcal{R}^{\text{Enc}}.\mathsf{Upd}(\sigma, 1)$

      Goto NEXT

  Return 1; $\boxed{\text{Return 0}}$

  NEXT: $\mathsf{crw} \leftarrow \mathsf{true}$

  NEXT2: $k \leftarrow k+1$

Return 0; $\boxed{\text{Return 1}}$

Oracle $\text{SimEnc}(N, M)$

$C \leftarrow \text{Enc}(N, M)$

$\boxed{C \leftarrow \text{Enc}(N, 0^{\mathsf{ml}})}$

Return C

Adversary $\mathcal{C}_3^{\text{Enc}}$

$k \leftarrow 0$; $\sigma \leftarrow\!\!\$\ \mathcal{R}^{\text{SimEnc}}.\mathsf{Init}$

While $k \leqslant \mathcal{R}.\text{rew}$ do

  For $i \in [c]$ do // **Buffer Phase**

    $N_i \leftarrow (r+1, i)$

    $(\sigma, \mathsf{rf}, C_i) \leftarrow\!\!\$\ \mathcal{R}^{\text{SimEnc}}.\mathsf{SimEnc}(\sigma, N_i, 0^{\mathsf{ml}})$

    If rf then goto NEXT2

  For $i \in [r]$ do // **Challenge Phase**

    $j^* \leftarrow\!\!\$\ [u]$

    For $j \in [u]$ do

      $M_j \leftarrow\!\!\$\ \{0,1\}^{\mathsf{ml}}$

      $N_j \leftarrow (i, j)$

      $(\sigma, \mathsf{rf}, C_j) \leftarrow\!\!\$\ \mathcal{R}^{\text{SimEnc}}.\mathsf{SimEnc}(\sigma, N_j, M_j)$

      If rf then goto NEXT

    $(\sigma, \mathsf{rf}, M) \leftarrow\!\!\$\ \mathcal{R}^{\text{SimEnc}}.\mathsf{SimDec}(\sigma, N_{j^*}, C_{j^*})$

    If rf then goto NEXT

    If $M \neq M_{j^*}$ then

      $\sigma \leftarrow\!\!\$\ \mathcal{R}^{\text{SimEnc}}.\mathsf{Upd}(\sigma, 1)$

      Goto NEXT

  If crw then return 1 else return 0

  NEXT: $\mathsf{crw} \leftarrow \mathsf{true}$

  NEXT2: $k \leftarrow k+1$

Return 0

Oracle $\text{SimEnc}(N, M)$

$C \leftarrow \text{Enc}(N, 0^{\mathsf{ml}})$

Return C

Figure 6.10: Adversaries $\mathcal{C}_1$ and $\mathcal{C}_2$ for Lemma 6.2.1 and adversary $\mathcal{C}_3$ for Lemma 6.2.3. Highlighted code is only included in $\mathcal{C}_1$ and boxed code is only included in $\mathcal{C}_2$.

Adversary $\mathcal{D}_1(\phi, M_1, \cdots, M_u)$

If $\phi = \perp$ then

  $i \leftarrow 1$; $K^* \leftarrow \mathsf{NE.K}$; $\sigma \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}}.\mathsf{Init}$

Else

  $(i, K^*, \sigma) \leftarrow \phi$

Foreach $j \in [u]$ do

  $N_j \leftarrow (i, j)$

  $(\sigma, ., C_j) \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}}.\mathsf{SimEnc}(\sigma, N_{i,j}, M_j)$

$\phi \leftarrow (i, K^*, \sigma)$

Return ϕ

Adversary $\mathcal{D}_2(\phi, j^*)$

$(i, K^*, \sigma) \leftarrow \phi$

$N_{j^*} \leftarrow (i, j^*)$

$C_{j^*} \leftarrow \mathsf{NE.E}(K^*, N_{j^*}, 0^{\mathsf{ml}})$

$(\sigma, ., M) \leftarrow\!\!\$\ \mathcal{R}^{\text{ENC}}.\mathsf{SimDec}(N_{j^*}, M_{j^*})$

$\phi \leftarrow (i + 1, K^*, \sigma)$

Return (ϕ, M)

Oracle $\text{ENC}(N, M)$

Return $\mathsf{NE.E}(K^*, N, 0^{\mathsf{ml}})$

Figure 6.11: Adversary $(\mathcal{D}_1, \mathcal{D}_2)$ used in the proof of Lemma 6.1.7.

It follows that,

$$\mathsf{Pr}\left[\mathsf{G}_0^0\right] \leqslant \mathsf{Pr}\left[\mathsf{G}^1\right] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_2)\ .$$

The claims on $\mathcal{C}_2$'s complexity can be verified from its code.

6.4.3 Proof of Lemma 6.2.3

Proof. Consider the adversary $\mathcal{C}_3$ shown in Fig. 6.10. It was obtained by copying the code of G^2 and G^3 (from Fig. 6.7), then modifying the code to use its own ENC oracle to respond to encryption queries from $\mathcal{R}$, and to return 1 whenever the games would output true and 0 otherwise. Consequently, $\mathcal{C}_3$ playing $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}$ (resp. $\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}$) perfectly simulates G^3 (resp. G^2) to $\mathcal{R}$ and we have

$$\mathsf{Pr}\left[\mathsf{G}^3\right] = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},0}(\mathcal{C}_3)\right] \text{ and } \mathsf{Pr}\left[\mathsf{G}^2\right] = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{indr}}_{\mathsf{NE},1}(\mathcal{C}_3)\right]\ .$$

It follows that,

$$\mathsf{Pr}\left[\mathsf{G}^2\right] \leqslant \mathsf{Pr}\left[\mathsf{G}^3\right] + \mathsf{Adv}^{\mathsf{indr}}_{\mathsf{NE}}(\mathcal{C}_3)\ .$$

The claims on $\mathcal{C}_3$'s complexity can be verified from its code. ■

6.5 Proof of Lemma 6.1.7

Consider the adversary $(\mathcal{D}_1, \mathcal{D}_2)$ shown in Fig. 6.11. Adversary $\mathcal{D}_1$ samples a key for NE and starts running $\mathcal{R}$ from the beginning in the first round of $\mathsf{G}^{\mathsf{it\text{-}chl\text{-}}r}_{u,\mathsf{ml}}$ (when $\phi = \perp$) and from the state at which $\mathcal{R}$ was last stopped in later rounds. It responds to the encryption queries of $\mathcal{R}$ as in G^1 (Fig. 6.4). Adversary $\mathcal{D}_1$ makes

u encryption queries which are distributed identically to the queries made to $\mathcal{R}$ in one iteration of G^1 since the inputs to $\mathcal{D}_1$ are sampled uniformly at random in $\mathsf{G}^{\text{it-chl-}r}_{u,\mathsf{ml}}$. It then stops running $\mathcal{R}$ and outputs its state ϕ which consists of the state σ of $\mathcal{R}$ along with the iteration number i and the key K^*. Adversary $\mathcal{D}_2$ resumes running $\mathcal{R}$ from the state it was last stopped and makes a decryption query which is identically distributed to the decryption query made in G^1 because j^* is chosen at random and independently of the M_i's in $\mathsf{G}^{\text{it-chl-}r}_{u,\mathsf{ml}}$. It outputs the message M it received as answer to the decryption query along with a state ϕ which consists of the current state of $\mathcal{R}$, the next iteration number $i+1$, and the key K^*.

It follows from their that during the r iterations of $\mathsf{G}^{\text{it-chl-}r}_{u,\mathsf{ml}}$, the view of $\mathcal{R}$ provided by $(\mathcal{D}_1, \mathcal{D}_2)$ perfectly matches its view in G^1. Note that $\mathsf{G}^{\text{it-chl-}r}_{u,\mathsf{ml}}(\mathcal{D}_1, \mathcal{D}_2)$ will output true whenever G^1 would output true, so $\mathsf{Pr}[\mathsf{G}^1] \leqslant \mathsf{Adv}^{\text{it-chl-}r}_{u,\mathsf{ml}}(\mathcal{D}_1, \mathcal{D}_2)$.

The state of $(\mathcal{D}_1, \mathcal{D}_2)$ is always a tuple (i, K^*, σ) for which $|i| = \log r$, $|K^*| = \mathsf{kl}$, and $|\sigma| = S$ all hold. Hence $(\mathcal{D}_1, \mathcal{D}_2)$ is $(S + \log r + \mathsf{kl})$-bounded and the desired bound follows by applying Lemma 6.1.2.

6.6 *Proof of Lemma 6.2.4 (Compression Argument)*

For this proof we use the compression lemma given in [DTT10], which we state here as a proposition. It quantitatively formalizes that it is impossible to compress a random element in set $\mathcal{X}$ to a string that than $\log|\mathcal{X}|$ bits long, even relative to a random string.

Proposition 6.6.1. *Let* Encode *be a randomized map from* $\mathcal{X}$ *to* $\mathcal{Y}$ *and let* Decode *be a randomized map from* $\mathcal{Y}$ *to* $\mathcal{X}$ *such that*

$$\mathsf{Pr}\left[x \xleftarrow{\$} \mathcal{X} : \mathsf{Decode}(\mathsf{Encode}(x)) = x\right] \geqslant \varepsilon \ ,$$

then $\log|\mathcal{Y}| \geqslant \log|\mathcal{X}| - \log(1/\varepsilon)$.

To apply this proposition we use the reduction $\mathcal{R}$ to construct procedures Encode and Decode which attempt to encode an element of $\mathcal{X} = \mathsf{C}^c$ with a state $\sigma \in \{0,1\}^S = \mathcal{Y}$ and succeed whenever G^3 (Fig. 6.7) would output true. They are shown in Fig. 6.12.

The procedure Encode was obtained by copying the code of G^3, then making two modifications. First, whenever $\mathcal{R}$ makes a ENC oracle query corresponding to the i^{th} buffer query, Encode responds using its input string $T_i \in \mathsf{C}$ (the other queries are answered at random as in G^3). Second, Encode returns the state σ of $\mathcal{R}$ whenever crw would be set in G^3 (i.e. whenever there was a goto NEXT) or whenever G^3 would halt and return an output.

The procedure Decode uses σ to run $\mathcal{R}$ and see if it would ever again pass the buffer phase. If $\mathcal{R}$ does so, Decode does not need to continue simulating its view correctly because it already has the required values. Because Decode does not know how many times $\mathcal{R}$ has already rewinded, it simply allows $\mathcal{R}$ to rewind $\mathcal{R}$.rew

Algorithm $\mathsf{Encode}((T_1, \cdots, T_c))$

$k \leftarrow 0$; $\sigma \leftarrow\!\!\$\ \mathcal{R}^{\text{Enc}}.\mathsf{Init}$

While $k \leqslant \mathcal{R}.\text{rew}$ do

  For $i \in [c]$ do // **Buffer Phase**

    $N_i \leftarrow (r+1, i)$

    $(\sigma, \mathsf{rf}, C_i) \leftarrow\!\!\$\ \mathcal{R}^{\text{Enc}}.\mathsf{SimEnc}(\sigma, N_i, 0^{\mathsf{ml}})$

    If rf then goto NEXT2

  For $i \in [r]$ do // **Challenge Phase**

    $j^* \leftarrow\!\!\$\ [u]$

    For $j \in [u]$ do

      $M_j \leftarrow\!\!\$\ \{0,1\}^{\mathsf{ml}}$

      $N_j \leftarrow (i, j)$

      $(\sigma, \mathsf{rf}, C_j) \leftarrow\!\!\$\ \mathcal{R}^{\text{Enc}}.\mathsf{SimEnc}(\sigma, N_j, M_j)$

      If rf then return σ

    $(\sigma, \mathsf{rf}, M) \leftarrow\!\!\$\ \mathcal{R}^{\text{Enc}}.\mathsf{SimDec}(\sigma, N_{j*}, C_{j*})$

    If rf then return σ

    If $M \neq M_{j*}$ then

      $\sigma \leftarrow\!\!\$\ \mathcal{R}^{\text{Enc}}.\mathsf{Upd}(\sigma, 1)$

      Return σ

  Return σ

  NEXT: $\mathsf{crw} \leftarrow \mathsf{true}$

  NEXT2: $k \leftarrow k+1$

Return σ

Oracle $\text{Enc}(N, M)$

For $i \in [c]$ do

  If $(N, M) = ((r+1, i), 0^{\mathsf{ml}})$ then return T_i

$C \leftarrow\!\!\$\ \mathsf{C}$

Return C

Algorithm $\mathsf{Decode}(\sigma)$

$k \leftarrow 0$

While $k \leqslant \mathcal{R}.\text{rew}$ do

  For $i \in [c]$ do // **Buffer Phase**

    $N_i \leftarrow (r+1, i)$

    $(\sigma, \mathsf{rf}, C_i^*) \leftarrow\!\!\$\ \mathcal{R}^{\text{Enc}}.\mathsf{SimEnc}(\sigma, N_i, 0^{\mathsf{ml}})$

    If rf then goto NEXT2

  NEXT2: $k \leftarrow k+1$

Return $(C_1^*, \cdots, C_c^*)$

Oracle $\text{Enc}(N, M)$

$C \leftarrow\!\!\$\ \mathsf{C}$

Return C

Figure 6.12: Procedures Encode and Decode. Highlighting indicates interesting code changes from G^2.

times. The final output of Decode consists of the strings $C_1^*, \cdots, C_c^*$ that $\mathcal{R}$ responded to buffer queries with.

We argue that Decode answers correctly if G^3 would return true. Since G^3 returns true only if crw is set, we only look at executions of Encode and Decode where crw would have been set in G^3. Observe that Encode simulates G^3 to $\mathcal{R}$ perfectly right up until crw is set in G^3 (the simulation of Enc is perfect because $T_1, \cdots, T_c$ are distributed uniformly at random in C). Then Decode perfectly simulates G^3 to $\mathcal{R}$ from the point crw is set until $\mathcal{R}$ passes the buffer phase (or until G^3 would have halted without $\mathcal{R}$ passing the buffer again). While Encode and Decode may end up running $\mathcal{R}$ for more than $\mathcal{R}.\mathsf{rew}$ iterations, G^3 would have returned false if it has completed $\mathcal{R}.\mathsf{rew}$ iterations and $\mathcal{R}$ had never passed the buffer phase a second time, so we need not worry about incorrectness simulation after the first $\mathcal{R}.\mathsf{rew}$ iterations.

Now, because $\mathcal{R}$ is faithful, it must have queried the buffer queries to Enc before passing the buffer phase in Encode and on every rewind, it answers with the same answers it had given previously. Hence, if $\mathcal{R}$ passes the buffer phase in both Encode and Decode, then it must have responded with $T_1, \cdots, T_c$ in Decode. Consequently, if G^3 would return true, then $T_1, \cdots, T_c$ will be output by Decode. So, we have that

$$\mathsf{Pr}\left[(T_1, \cdots, T_c) \leftarrow\!\!\$\ \mathsf{C}^c : \mathsf{Decode}(\mathsf{Encode}((T_1, \cdots, T_c))) = (T_1, \cdots, T_c)\right] \geqslant \mathsf{Pr}\left[\mathsf{G}^3\right] .$$

Letting $\mathsf{Pr}[\mathsf{G}^3] = \varepsilon$ and applying Proposition 11.1.1 gives $\log 2^S \geqslant \log |\mathsf{C}|^c - \log 1/\varepsilon$. Then rearranging the terms gives $\varepsilon \leqslant 2^S/|\mathsf{C}|^c$, which concludes the proof.

Part II

TIME-SPACE TRADEOFFS: ATTACKS, LIMITATIONS AND FEASIBILITY

Chapter 7

TIME-SPACE TRADEOFFS: INTRODUCTION, PRIOR WORK AND OUR CONTRIBUTIONS

In a seminal work [Hel80], Hellman presented the first example of an adversary exploiting the tradeoffs between various resources. This work studied the hardness of the random function inversion problem, which entails that an adversary invert a function $f : \{0,1\}^n \to \{0,1\}^n$ at $y = f(x)$ where x is a random point in $\{0,1\}^n$. Hellman demonstrated that an adversary might be able to invest a significant amount of effort in preprocessing the particular function f before seeing y, compute some "advice" about it, and thereby significantly reduce the online time required for breaking a particular instance. More precisely, for the specific case when f is a random function, Hellman gave an explicit attack that computes roughly $S = 2^{2n/3}$ bits of advice about f, and needs online time (time after it sees y) roughly $T = 2^{2n/3}$ to invert f on y successfully. Even though the preprocessing cost might be very large, if the adversary needs to invert f on many different values of y, the preprocessing cost gets amortized. For example, if the adversary has to invert on $2^{n/3}$ different y's, the amount of preprocessing per y amortizes to $2^{n/3}$ since the preprocessing is reusable.

Yao [Yao90] showed that Hellman's algorithm for function is optimal when f is a one cycle permutation. De et al. [DTT10] later proved the optimality of Hellman's algorithm for any permutation f. They used a *compression argument* to prove this result. A compression argument is based on a simple information-theoretic idea that random bits cannot be compressed. The results of [Yao90, DTT10] more generally imply that any attack against function inversion with preprocessing advice S and online running time T needs roughly $S \cdot T = 2^n$ to be successful. Till date, this is the best-known limitation for the function inversion problem. In recent work, Corrigan-Gibbs and Kogan [CK19] showed that any improvement to this limitation would imply new lower bounds on depth-two circuits with arbitrary gates.

Fiat and Naor [FN99] later provided a rigorous analysis of Hellman's algorithm and extended it to invert all possible functions f, though with a slightly worse tradeoff of the form roughly $S^3T = 2^{3n}$ for $2^{3n/4} \leqslant S \leqslant 2^n$. De et al. [DTT10] improved this attack for certain ranges of parameters.

Auxiliary-Input Random Oracle Model. Hash functions are a very fundamental primitive in cryptography that are used in many different applications. While it is empirically believed that concrete cryptographic hash functions satisfy various useful security properties, formalizing this seems to be currently out of reach. Thus, in the context of provable security, cryptographic hash functions are usually modeled as random oracles, i.e., completely random functions [BR93]. This is referred to as the *Random Oracle Model (ROM)*.

An implicit assumption in this model is that the adversary is fixed before the random oracle is sampled. Therefore, it does not, in particular, capture adversaries who might be able to invest a large amount of preprocessing on the hash function.

To this end, Unruh formally introduced the Auxiliary-Input Random Oracle Model [Unr07], where an adversary is modelled as a two stage one. The first stage, known as the preprocessing phase gets unbounded access to the random oracle and can compute a limited amount of advice (say S bits, this is known as the auxiliary-input) about it. The second phase, known as the online phase, gets the advice as input, can make a bounded number of queries (say T queries) to the random oracle and has to accomplish some task involving the random oracle. We refer to such an adversary as an (S, T)-AI adversary. Analyzing adversaries in this model is a rather difficult task, since the preprocessing phase is unbounded, and it can compute any arbitrary advice on the random oracle. Unruh introduced a technique known as *presampling* that allows us to upper bound the success probability of an (S, T)-AI adversary with the success probability of an adversary that first *fixes* roughly $S \cdot T$ points of the random oracle, and then tries to accomplish the same task after the rest of the points of the random oracle are re-sampled. In particular, this adversary does not get any advice as input, and is therefore significantly easier to analyze.

Dodis et al. [DGK17] proved quantitative bounds on the security for specific applications of random oracles in the AI-ROM. The applications that they considered included security of the random oracle as a one-way function, collision resistant hash function, pseudorandom generator (PRG), pseudorandom function (PRF), etc. They used compression arguments and were able to prove stronger security guarantees than those that could be proved using the presampling technique developed in [Unr07]. In a subsequent work, Coretti et al. [CDGS18] improved the analysis of the presampling technique and gave simpler proofs of all the results in [DGK17] along with new results. Guo et al. [GLLZ21] extended the presampling technique to prove security against quantum adversaries with auxiliary inputs. They do so by giving an alternative but equivalent view of the presampling technique which allows them to adapt it to the quantum setting.

Two concurrent works [ACDW20, CGLQ20] developed a new technique to analyze security in the AI-ROM, known as the multi-instance framework. Here the probability of any adversary winning in the AI-ROM with S bits of advice is related to the probability of an adversary that has no advice and has to win on roughly S randomly and independently sampled instance. This framework was inspired by the work of Impagliazzo and Kabanets [IK10] which gave constructive proofs of Chernoff bounds. Since the multi-instance adversary does not have any pre-computed advice, analyzing it is easier. In order to analyze this multi-instance adversary [ACDW20] used a compression argument. This compression argument was significantly simpler than the direct compression arguments in [DGK17]. In a follow-up work, Gravin et al. [GGKL21], proved new concentration inequalities on sum of binary random variables which satisfy conditions weaker than K-

wise independence. This helped them re-derive the multi-instance framework in [ACDW20, CGLQ20], and also extend the framework for indistinguishability games.

Other Auxiliary Input Ideal Models. Beyond the Random Oracle Model, there are other ideal models that we consider in cryptography, e.g., the Ideal Cipher Model (ICM), the Random Permutation Model (RPM), the Generic Group Model (GGM) etc. Corretti et al. [CDG18] formalized the auxiliary-input version of these models (AI-ICM, AI-RPM, AI-GGM) and adapted the *presampling* technique to work for AI-ICM, AI-RPM, AI-GGM. Corrigan-Gibbs and Kogan [CK18] studied the hardness of the discrete logarithm problem and variants of the Diffie-Hellman problem in the AI-GGM in [CK18] and proved guarantees on the best possible attacks.

In this thesis, we continue the ongoing line of work exploring time-space tradeoffs in ideal models. Our exploration takes us in three different directions **(1)** studying time-space tradeoffs in finding collisions for standardized hashing mechanisms **(2)** constructing new hashing mechanisms that have optimal security against time-space tradeoff attacks for finding collisions **(3)** exploring what is the actual query complexity of the the preprocessing phase of these attacks. In Sections we describe our concrete contributions in detail.

7.1 Time-Space Tradeoffs for Finding Short Collisions in Merkle-Damgård

The Merkle-Damgård hash function construction [Mer82, Mer88, Mer90a, Dam87] is a popular design for building an arbitrary-size-input compression function from a fixed-size-input compression function. This design is not only extremely fundamental in cryptographic theory, but it also underlies popular hash functions used in practice, most notably MD5, SHA-1, and SHA-2.

The MD construction is defined relative to a compressing function $h\colon [N] \times [M] \to [N]$[1], modeled as a random oracle, as follows. First, for $a \in [N]$ and $\alpha \in [M]$, let $\mathsf{MD}_h(a, \alpha) = h(a, \alpha)$. Then, define recursively

$$\mathsf{MD}_h(a, (\alpha_1, \ldots, \alpha_B)) = h(\mathsf{MD}_h(a, (\alpha_1, \ldots, \alpha_{B-1})), \alpha_B)$$

for $a \in [N]$ and $\alpha_1, \ldots, \alpha_B \in [M]$. The a is referred to as *salt* (sometimes also called IV) and each of the following B elements are referred to as *blocks*.

Due to the ubiquitous influence of this hashing paradigm, both in theory and practice, characterizing the complexity of finding collisions in MD_h (on a random salt) is a fundamental problem. The well-known birthday attack gives a T-query attacker with $\Theta(T^2/N)$ advantage. However, this attack is very generic: it neither takes advantage of the structure of MD_h nor does it utilize the fact that the attacker may have access to some limited amount of "advice" about h due to a long preprocessing phase. But, there is a

[1]We use the notation $[N]$ to denote the set $\{1, 2, \ldots, N\}$ for a natural number N.

good reason that security against non-uniform attackers has become the standard notion of security in the cryptographic literature: it captures the natural idea that an adversary may have been designed to attack specific instances, guaranteeing security against an expensive preprocessing stage, or even unknown future attacks. On the whole, it is widely believed by the theoretical community that non-uniformity is the right cryptographic modeling of attackers, despite being overly conservative and including potentially unrealistic attackers. Therefore, understanding the complexity of finding collisions in MD, allowing preprocessing, is a fundamental problem.

The auxiliary-input random oracle model. The concrete hash functions h used in real-life do not have solid theoretical foundations from the perspective of provable security. Therefore, when analyzing the security of the MD construction, the function h is typically modeled as a completely random one, i.e., a random oracle. We follow the standard approach and model preprocessing adversaries using the influential extension of the random oracle model termed *auxiliary-input random oracle model* (AI-ROM). This model was (implicitly) used, for example, in the classical works of Yao [Yao90] and Fiat and Naor [FN99], and formally defined in the influential work of Unruh [Unr07] which was recently revisited by Dodis, Guo, and Katz [DGK17] and Coretti et al. [CDGS18].

The AI-ROM models preprocessing adversaries as two-stage algorithms $(\mathcal{A}_1, \mathcal{A}_2)$ parametrized by S (for "space") and T (for "time"). The first part $\mathcal{A}_1$ has unbounded access to the random oracle h, and its goal is to compute an S-bit "advice" σ for $\mathcal{A}_2$. The second part $\mathcal{A}_2$ gets the advice σ, can make at most T queries to the random oracle, and attempts to accomplish some task involving h. In our case, $\mathcal{A}_2$ gets a random salt a as a challenge and its goal is to come up with a collision in $\mathsf{MD}_h(a, \cdot)$. Both $\mathcal{A}_1$ and $\mathcal{A}_2$ are unbounded in running time.

Known results. Collision resistance of salted MD hash functions in the AI-ROM was first studied by Coretti, Dodis, Guo, and Steinberger [CDGS18]. Among other results, they showed an attack, loosely based on the idea of rainbow tables [Hel80, Oec03], with advantage $\tilde{\Omega}(ST^2/N)$.[2] [3] They further showed that this attack is optimal. Namely, no attack can have an advantage better than $\tilde{O}(ST^2/N)$. (Notice that this attack beats the naive birthday attack mentioned above for typical values of S.) In a more recent work of Akshima, Cash, Drucker, and Wee [ACDW20] observed that the attack of Coretti et al. [CDGS18] results in very long collisions, of the order of T blocks, which may limit their practical usefulness. While formally, a length T collision does violate collision resistance, it is hard to imagine a natural application where it is useful. Indeed, for reasonable values of T, say $T = 2^{60}$, it is unlikely that such a collision, which is several

[2] Throughout the chapter, the $\tilde{}$ notation suppresses poly-logarithmic terms in N.

[3] By "advantage" we mean the probability of finding a collision.

petabytes long, could damage any widely-used application.

Akshima et al. [ACDW20] therefore raise the very natural question of what is the complexity of finding *short* collisions.

What is the complexity, as a function of S and T (the allowed space and query bounds, respectively), of finding a B-block collision in salted MD?

Although this question is very natural and clean, as mentioned, a complete answer is known only in the extreme cases, either when $B = 2$ or when $B = \tilde{\Omega}(T)$. Indeed, when B is very close to T, the result above of Coretti et al. [CDGS18] implies that the advantage is $\tilde{\Theta}(ST^2/N)$. The case of $B = 2$ was resolved by Akshima et al. [ACDW20] who showed that the advantage is $\tilde{\Theta}(ST/N + T^2/N)$. Even for $B = 3$, a complete answer is not known: The analysis of Akshima et al. [ACDW20] consists of an elaborate case analysis tailored to the $B = 2$ case, and they claim that even for $B = 3$ the proof of their lower bound "*... would be too long and complex to write down*".[4]

The STB conjecture. In terms of upper bounds, Akshima et al. [ACDW20] showed that a variant of Coretti et al.'s (rainbow tables inspired) attack could be generalized to get a B-block collision with advantage $\tilde{\Omega}(STB/N)$. This attack generalized the attack of Coretti et al. [CDGS18] which gives an $O(T)$-block collision with probability $\tilde{\Omega}(ST^2/N)$. With the lack of better bounds on the best possible attack for a wide range of B's (anywhere between $B = 3$ and $B \ll T$), Akshima et al. [ACDW20] put forward the "*STB conjecture*" which posits that the optimal attack for finding length B collisions has advantage $\tilde{\Theta}(STB/N + T^2/N)$ (i.e., the better between their attack and the generic birthday attack).

We believe that our current understanding of the exact security that MD-style constructions could ideally achieve is insufficient. Therefore, given how widespread MD-based hash functions are, progress towards resolving the conjecture is highly important.

7.1.1 Our Results

Our main result confirms the STB conjecture in many new parameter settings. Specifically, we prove two new upper bounds on the advantage of the best attack for finding short collisions in salted MD hash functions in the AI-ROM. The first bound confirms the STB conjecture for all constant values of B. The second result confirms the STB conjecture even for super constant values of B but only for moderately large values of S compared to T.

---

[4] For $B = 1$ a tight bound of $\Theta(S/N + T^2/N)$ is known [DGK17].

STB conjecture is true for all constant B. We show that for any $B \in O(1)$, the advantage of any S-space T-query attacker in finding a length B collision is bounded by $\tilde{O}(ST/N + T^2/N)$, matching the known attack up to poly-logarithmic factors.

Theorem 7.1.1 (Informal; See Theorem 9.3.5)**.** *For every constant B, the STB conjecture is true.*

This theorem is obtained as a special case of a more general bound on the advantage of any S-space T-query attacker in finding a length B collision of the form

$$\tilde{O}\left(\frac{STB^2(\log^2 S)^{B-2}}{N} + \frac{T^2}{N}\right).$$

Note that this bound is meaningful when B is a constant (or slightly bigger) but becomes vacuous when say, $B = \log N$.

STB conjecture is true for all $SB \ll T$. We show that as long as $S, B \ll T$, the conjecture is true again. Specifically, we show that whenever $S^4B^2 \in \tilde{O}(T)$, the maximal advantage of any S-space T-query attacker in finding a length B collision is obtained by the birthday attack, up to poly-logarithmic factors. For example, when $SB \leqslant T^{1/4}$, the maximal advantage is $O(T^2/N)$, and therefore the STB conjecture holds.

Theorem 7.1.2 (Informal; See Theorem 8.4.1)**.** *For every $S^4B^2 \in \tilde{O}(T)$, the STB conjecture is true. For instance, the conjecture holds if either*

- *$B \in \mathsf{poly}\log N$ and $S \in \tilde{O}(T^{1/4})$, or*
- *$B \in \tilde{O}(T^{1/4})$ and $S \in \tilde{O}(T^{1/8})$.*

This theorem is obtained as a special case of a more general bound on the advantage of any S-space T-query attacker in finding a length B collision of the form

$$\tilde{O}\left(\frac{S^4TB^2}{N} + \frac{T^2}{N}\right).$$

A concrete comparison between the results. The two bounds are generally incomparable. While the bound from Theorem 7.1.1 is asymptotically tight whenever B is constant (independent of S, T), it becomes vacuous for say $B = \log N$. On the other hand, the bound from Theorem 7.1.2 is meaningful for all $B \in o(N^{1/2})$, as long as $S^4 \cdot B^2 \ll N$. For instance, assume that $S = N^{1/16}$ and $B \in \Theta(N^{\varepsilon})$ (for $0 < \varepsilon < 1/8$). In this setting, the bound from Theorem 7.1.1 is trivial. On the other hand, the bound from Theorem 7.1.2 gives that any successful attack must satisfy $T \in \tilde{\Omega}(N^{1/2})$ which is strictly better than what the generic $\tilde{O}(ST^2/N)$ bound gives (it only gives $T \in \tilde{\Omega}(N^{15/32})$).

Technical highlight. The main technical component in both of our bounds is a compression argument that uses a "too-good-to-be-true" attacker to non-trivially compress a uniformly random sequence of bits, thereby getting a contradiction. The setup is somewhat similar to the one of Akshima et al. [ACDW20] (although slightly more modular), but our compression argument deviates from theirs significantly. Their argument inherently relied on the fact that there are at most two blocks in the collision, therefore greatly simplifying the possible structures to consider. In contrast, we consider arbitrary length collisions, and thus we have to deal with all possible structures of collisions. Our proof identifies and analyzes a general structure for MD collisions and unveils a natural combinatorial problem that influences the resulting upper bound on the advantage of preprocessing adversaries. Specifically, it turns out that the "dominant extra" terms in both of our bounds ($(\log^2 S)^{B-2}$ in the first bound and S^3 in the second) emerge due to the need to encode a *reverse path* in a general (fan-out 1, but possibly large fan-in) directed graph, where the graph is the one induced by the queries that the adversary makes to the random oracle. Any improvement on this encoding would immediately imply a better upper bound, a fact that we hope will lead to better bounds in the future.

7.1.2 Discussion

As mentioned, the MD paradigm underlies numerous hash function constructions that are central building blocks in many applications. There are several popular variants of the MD paradigm implemented in practice. In this work, we follow previous works and focus on the cleanest variant for concreteness. One prominent variant withstands length extension attacks by padding the input message with its length. (In fact, this is the version suggested by Merkle and Damgård.) We remark that our results directly apply to this padded variant. Specifically, the "STB" attack finds a collision with the same number of blocks, so it readily extends to this padded variant. Our bounds on the best possible attacks also extend to this setting since the argument did not use any specific property on the collision blocks. It is interesting to study other practically used variants and understand if similar results can be obtained. To this end, we hope that the techniques we develop in this work will be helpful.

From a theoretical perspective, no single function can be collision-resistant (in the plain model), as a non-uniform attacker can trivially hardwire a collision. This is why collision resistance is considered with respect to a family of hash functions indexed by a key called *salt*. The salt is chosen after the attacker is fixed (and so is the non-uniform advice about the family of functions). Still, in practice, a single hash function is typically defined by fixing an IV, making it insecure against non-uniform attackers. This contrasts with how we define the collision resistance game, where the IV is chosen randomly after the preprocessing phase. Thus, it may seem that the expensive preprocessing needed for attacks in our model does not represent real-life scenarios. However, often the hash function used in a particular application (relying on collision-resistance) is *salted* by

prepending a random salt value to the input. One such well-known application is *password hashing* [ST79]. Such salting essentially corresponds to the random-IV setting considered here, and, therefore, the attack becomes relevant again.

The primary motivation for our work is to make progress towards the STB conjecture, which we view as a fundamental problem. To this end, we focused on asymptotic bounds as a function of S, T, and B. We hope that the concrete bounds could be improved in future works, affecting design choices of real-life hash functions.

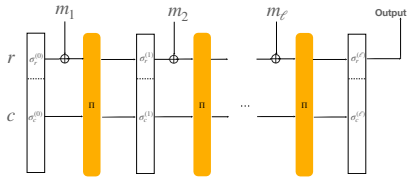
We provide the details of this result in Chapter 8.

7.2 Time-Space Tradeoffs for Short Collisions in Sponge

Due to a series of successful attacks on widely used hash functions such as MD5, SHA-0, and SHA-1, in 2006 the National Institute of Standards and Technology (NIST) organized a competition to create a new hash standard. At that time, the existing hash functions were all based on the well-known Merkle-Damgård hash function construction [Mer82, Mer88, Mer90a, Dam87]. The goal of the competition was to find an alternative, dissimilar cryptographic hashing design. It took almost a decade until the winner, a family of cryptographic functions called Keccak, become a hashing standard called SHA-3. The Keccak family is based on the *sponge construction* [BDPVA07, BDPV08] which was a novel alternative to the popular Merkle-Damgård design. By now, the sponge paradigm is used for building collision resistant hash functions, message authentication codes (MACs), pseudorandom functions (PRFs) [BDPVA11], key derivation functions [GT16], and more.

A sponge function $\mathsf{Sp}\colon \{0,1\}^* \to \{0,1\}^r$ is defined via three parameters: (1) two natural numbers r (for bitrate) and c (for capacity) so that $n = c + r$, (2) an initial state $\sigma^{(0)} = (\sigma_r^{(0)}, \sigma_c^{(0)}) \in \{0,1\}^r \times \{0,1\}^c$, and (3) a function $\Pi : \{0,1\}^n \to \{0,1\}^n$ which is usually thought of as a (public) pseudorandom permutation. The hashing operation (a.k.a. absorbing) is defined by iterating the state by computing a *round function*. Specifically, given a sequence of r-bit blocks $(m_1, m_2, \ldots, m_\ell)$, $\mathsf{Sp}(m_1, m_2, \ldots, m_\ell)$ is defined as:[5]

1. For $i = 1, \ldots, \ell$, do:
   (a) Compute the round function $\Pi((\sigma_r^{(i-1)} \oplus m_i) \parallel \sigma_c^{(i-1)})$ and let $\sigma^{(i)}$ denote the output.
   (b) Parse $\sigma^{(i)}$ as $(\sigma_r^{(i)}, \sigma_c^{(i)}) \in \{0,1\}^r \times \{0,1\}^c$.



2. Output the first r bits of $\sigma^{(\ell)}$, namely, $\sigma_r^{(\ell)}$.

[5] For simplicity, we do not consider padding of the input.

Typically, $\sigma_r^{(0)}$ is initialized to 0 and $\sigma_c^{(0)}$ is a random initialization vector (IV). If one wants to be explicit, we write $\mathsf{Sp}_{r,c,\Pi,\mathsf{IV}}$ for the sponge function. There are several common instances of r and c used in practice, for example in SHA-3-256 $c = 512$ and $r = 1088$, and in SHA-3-512 $c = 1024$ and $r = 576$. These instance are particularly useful since they were designed to be used as drop-in replacements for the corresponding SHA-2 instances, and as such they were intended to have identical (or better) security properties.

Sponge in the random permutation model. The concrete permutations Π that are used in real-life do not have solid theoretical foundations from the perspective of provable security. Therefore, when coming to analyze the security of the sponge construction, we model the permutation Π as a completely random one. That is, the permutation is randomly chosen, and all parties are given (black-box) access to it and its inverse.[6] This is called the *random permutation model* (RPM). Such bounds are used as an approximation to the best possible security level that can be achieved by the corresponding construction in the real-life implementation. This heuristic has been extensively and successfully used in the past several decades, with exceptions (i.e., examples where the real-life implementation and the ideal world construction are separated) being somewhat contrived and artificial. For "natural" applications it is widely believed that the concrete security proven in the RPM is the right bound even in the real-world, assuming the "best possible" instantiation for the idealized permutation is chosen.

As mentioned, the sponge construction was introduced by Bertoni et al. [BDPVA07] and its security was analyzed in a follow-up work [BDPV08] assuming that the underlying hash function is an invertible random permutation. The latter work showed a strong property called *indifferentiability* from a random oracle, which directly implies many other properties such as collision resistance, pseudorandomness, and more.

For instance, the following is known about Sponge's collision resistance (which is perhaps the most widely used property). For fixed c, r, the collision resistance game is defined as follows: a challenger sends a uniformly random IV to the adversary. The adversary "wins" if it is able to come up with distinct $m, m' \in \{0,1\}^*$ for which $\mathsf{Sp}_{r,c,\Pi,\mathsf{IV}}(m) = \mathsf{Sp}_{r,c,\Pi,\mathsf{IV}}(m')$. There is a well-known attack due to the original works of Bertoni et al. [BDPVA07, BDPV08]: the adversary is given an IV and it merely queries the permutation oracle on inputs of the form $(m\|\mathrm{IV})$, where the m's are chosen uniformly at random. If a collision was observed (i.e., the adversary finds distinct m_1, m_2 such that the first r bits of $\Pi(m_1\|\mathrm{IV})$, $\Pi(m_2\|\mathrm{IV})$ are the same), then the adversary wins. By the well-known birthday bound, the success probability of this event is $\Omega(T^2/2^r)$. Alternatively, if two messages m_1, m_2 such that the query returned a state with the same last c bits (i.e., $\Pi(m_1\|\mathrm{IV}) = a_1\|b$ and $\Pi(m_2\|\mathrm{IV}) = a_2\|b$, then $m_1 \,\|\, a_1$ and $m_2 \,\|\, a_2$ form a collision. The success probability

[6] In typical permutation designs, including the permutations underlying the Keccak family, if you have the entire state, you can apply the inverse permutation to go backward to the previous state. This is why we also give free access to the inverse of the permutation as part of the model.

of this event is $\Omega(T^2/2^c)$. Overall, the attacker wins with probability $\Omega(T^2/2^{\min\{c,r\}})$. This is known to be the best possible attack due to the indifferentiability result of [BDPV08].

Non-uniformity / preprocessing attacks. The above discussion assumes that the adversary is uniform in the sense that it starts off with no knowledge about Π, as if it did not exist before it was invoked. However, this does not capture real-life attack scenarios where an attacker can invest a significant amount of preprocessing on the public permutation Π to speed up the actual attack whenever the IV is chosen. This is why most works (at least in theoretical cryptography) model attackers as non-uniform machines, where the attacker could obtain arbitrary but bounded-length advice, before attacking the system. The advice generation phase is called *the offline phase* and the "attack" given the advice and the challenge is called *the online phase*. The output size of the offline phase (i.e., the size of the advice) is denoted S and the number of queries allowed in the online phase is denoted T; computation is free of charge in both phases. This model, being an extension of the RPM where the online adversary may know a bounded-length hint about the permutation, is called the auxiliary-input RPM, or AI-RPM in short. This model was first explicitly put forward by Coretti, Dodis, and Guo [CDG18], naturally extending the influential auxiliary-input random oracle model (AI-ROM) from the seminal work of Unruh [Unr07] (which in turn is an explicit version of the model studied by Hellman [Hel80], Yao [Yao90], and Fiat-Naor [FN99]). Bounds on the power of "auxiliary-input" adversaries are also referred to as "time-space" tradeoffs.

Although the sponge paradigm is becoming widespread, very little is known about its formal security guarantees against such attackers that may have a short preprocessed hint about the permutation computed in an offline phase. In fact, there is an attack that utilizes this extra power to achieve advantage $\Omega(ST^2/2^c + T^2/2^r)$ (notice the extra multiplicative S term).[7] The attack is based on a combination of a birthday-style attack, as above, together with a variant of an attack by Hellman [Hel80] which is nowadays referred to as rainbow tables (due to Oechslin [Oec03]). While this attack uses known techniques, we were not able to find an explicit description of it in the literature and so for completeness, we give the attack and its analysis in Section 9.2.1.[8] [9] Only very recently, in the beautiful work of Coretti et al. [CDG18] (henceforth CDG) it was shown that this attack is optimal; that is, no S-space T-query attackers can find a collision with probability better than $\Omega(ST^2/2^c + T^2/2^r)$.

It turns out that the above attack results in a very long collision. Specifically, for parameters S and T as

[7] Throughout the introduction, for easy of notation, we supress poly-logarithmic (i.e., $\mathsf{poly}(c, r)$ terms inside the big "O/Ω" notation. The formal theorems state the precise bounds.

[8] More precisely, we give a generalization of this attack which finds collisions of length $B \geqslant 2$, and this particular attack follows by setting $B = T$.

[9] A related bound is stated in CDG [CDG18, Table 1] but after communication with an author, they confirmed that the attack was never worked out.

above, the above attack results in a collision of length $\approx T$. While this formally breaks collision resistance, it is hard to imagine a natural application where such a collision would be helpful in an attack. Say we have a system that uses a sponge-based hash with an output of size 256 bits. Running the above attack with $S = T = 2^{60}$ would result in a collision of several petabytes long, which is likely to be practically useless for any natural attack scenario. Therefore, we ask whether there exist attacks that find shorter collisions and what is their success probability. Specifically, we introduce an additional parameter B (for blocks) and require an attacker, on a random IV, to come up with two $\leqslant B$-block messages that collide. The main question studied in this work is:

What is the complexity of a preprocessing attacker in finding a B block collision in a Sponge hash function, assuming the underlying permutation is modeled as random?

7.2.1 Our Results

We initiate the study of time-space tradeoffs for *bounded length collisions in sponge hashing*. First, the known best attack that gives a single-block collision has advantage

$$\Omega\left(\frac{S}{2^c} + \frac{T^2}{2^r}\right). \tag{7.1}$$

In this attack, the preprocessing is used to "remember" a collision for S different IVs. If the challenge IV is in the set of remembered IVs, then the attack succeeds (this happens with probability $S/2^c$); otherwise, we run a birthday-style attack which succeeds with probability $\Omega(T^2/2^r)$. For MD hashing, the analogous bound for $B = 1$ is known to be tight. Second, there is an attack (loosely based on rainbow tables) that has advantage $\Omega(ST^2/2^c + T^2/2^r)$ and results with a $\Omega(T)$-blocks collision [CDG18].

At this point, if one were to speculate that sponge's security guarantees are at least as good as MD's, one would guess that the above attacks should be tight, at least for $B \in \{1, 2\}$. With some luck and labor, we may even be able to prove it. This is where the situation gets interesting. We show that the above speculation is **false** for $B = 1$ and in some natural settings of parameters, **sponge is strictly less secure than MD** for this task. On the other hand, for $B = 2$ we can only prove tightness for a certain range of parameters.

In what follows, we elaborate on our results. We design two new attacks, one designed for any $B \geqslant 2$ and the other specifically for $B = 1$. We complement our attacks with "lower bounds", which are actually upper bounds on the best possible advantage. Specifically, we prove that there is a qualitative jump in the advantage of best possible attacks for finding unbounded-length collisions and those for finding very short collisions (i.e., $B \leqslant 2$).

Attacks

We give two new attacks, one for any $B \geqslant 2$ and the other is specialized for $B = 1$. The generic attack is the first to result with an arbitrary block length collision while the one specialized to $B = 1$ beats the previously known best attack, at least in some range of parameters. By the latter, to the best of our knowledge, we show the first natural application for which sponge hashing is less secure than MD.

A new attack for $B \geqslant 2$. The above-mentioned attack on sponge hashing that has advantage $\Omega(ST^2/2^c + T^2/2^r)$ can be modified to result with a B-block collision for $B \geqslant 2$ and with advantage

$$\Omega\left(\frac{STB}{2^c} + \frac{T^2}{2^{\min\{c,r\}}}\right). \tag{7.2}$$

The attack follows a similar observation of ACDW [ACDW20] regarding MD hashing. Given the upper bound of CDG [CDG18] mentioned earlier, this attack is optimal for $B \in \Omega(T)$. For MD hashing, the analogous bound is known to be tight for $B = 2$ and $B \in \Omega(T)$.

A new attack for $B = 1$. We design a new attack for sponge hashing that results with a a single-block collision. Specifically, we show that if $ST^2 > 2^c$, then there is an attack with advantage

$$\Omega\left(\left(\frac{S^2T}{2^{2c}}\right)^{2/3} + \frac{T^2}{2^r}\right).$$

To see why this attack is superior to the previously known one (Eq. 7.1), we give a setting of parameters where it achieves a significantly higher advantage. Consider $r = c$, $S = 2^{4c/5}$, and $T = 2^{2c/5}$. Indeed, $ST^2 > 2^c$ and therefore we can apply the attack. The previously known best attack (Eq. 7.1) has advantage

$$\Omega\left(\frac{S + T^2}{2^c}\right) = \Omega\left(\frac{1}{2^{c/5}}\right).$$

This attack is the analog of the *provably* best attack for MD. On the other hand, our new attack has strictly better advantage

$$\Omega\left(\left(\frac{S^2T}{2^{2c}}\right)^{2/3} + \frac{T^2}{2^c}\right) = \Omega\left(\left(\frac{2^{8c/5}2^{2c/5}}{2^{2c}}\right)^{2/3} + \frac{1}{2^{c/5}}\right) = \Omega\left(1\right).$$

Thus, at least in this range of parameters, we beat the state-of-the-art attack and show that sponge is *less secure than MD*. In the example above, we chose a setting of parameters where the gap between the attacks is the largest (our attack succeeds with *constant* probability, while the previously known one succeeds with exponentially small probability). However, there are many more concrete settings where our attack is superior, although the gap could be less dramatic. We note that our bounds in this section and the technical overview are simplified for ease of parsing and refer the reader to the technical sections for the exact bounds.

Conceptual novelty: Our attack for $B = 1$ use the famous time-space tradeoffs for function inversion of Hellman [Hel80] and its extension by Fiat-Naor [FN99]. We leverage the possibility of inverse queries to the underlying permutation Π in the random-permutation model. This is in contrast to Merkle-Damgård construction which is analyzed in the random-oracle model that does not permit inverse queries. At a very high level, we use time-space tradeoffs for function inversion to "invert" the function Π^{-1} on a restricted domain. We view this conceptual connection between time-space tradeoffs for collision resistance of sponge hashing and function inversion as novel and hope that it will lead to better designs and additional attacks in the future.

Lower Bounds

We complement the picture by showing "lower bounds", namely impossibility results for better attacks. (In other words, these are upper bounds on the best possible advantage of any attacker.) We prove two such lower bounds, one for the case where $B = 1$ and the other is for $B = 2$, corresponding to our attacks.

On optimal attacks for $B = 2$. We show that any attack for $B = 2$ must have advantage

$$O\left(\frac{ST}{2^{\min(c,r)}} + \frac{S^2T^4}{2^{2c}} + \frac{T^2}{2^{\min\{c,r\}}}\right).$$

We note that this bound is tight with the best known attacks for a large range of parameters, but there still may be a gap otherwise. Specifically, if $ST^3 \leqslant 2^c, r \geqslant c$, then the above bound simplifies to $O(ST/2^c + T^2/2^{\min\{c,r\}})$ which matches the attack from Eq. (7.2). Thus, any improvement on the generic attack from Eq. (7.2) must take advantage of the regime where $ST^3 > 2^c$.

The proof of this result *provably* cannot be obtained via the bit-fixing method. Rather, we obtain the result via a compression argument. In such arguments, an imaginary adversary that is successful too often is used to compress a uniformly random string, a task which is (information-theoretically) impossible.

On optimal attacks for $B = 1$. We show that any attack for $B = 1$ must have an advantage

$$O\left(\frac{ST}{2^c} + \frac{T^2}{2^r}\right).$$

The proof of this result is relatively straightforward by using an optimized version of the remarkable bit-fixing (or presampling) method [Unr07, CDGS18, CDG18]. The main point of distinction of our proof from most previous ones is that we need to apply this technique in the RPM context, so our argument needs to handle inverse queries. (This result might have been known before, but we could not find such a statement, so we give it for completeness.)

We summarize our main results as well as the known best bounds in Fig. 7.1.

We provide the details of this result in Chapter 9.

	Best Attack	Advantage Upper Bound
$B = 1$	$\min\left(\frac{S^2T^2}{2^{2c}}, \left(\frac{S^2T}{2^{2c}}\right)^{2/3}\right) + \frac{S}{2^c} + \frac{T^2}{2^r}$ [Thm 9.2.2]	$\frac{ST}{2^c} + \frac{T^2}{2^r}$ [Thm 9.3.2]
$B = 2$	$\frac{ST}{2^c} + \frac{T^2}{2^{\min\{c,r\}}}$ [Thm 9.2.1]	$\frac{ST}{2^{\min\{c,r\}}} + \frac{S^2T^4}{2^{2c}} + \frac{T^2}{2^{\min\{c,r\}}}$ [Thm 9.3.5]
$B \geqslant 3$	$\frac{STB}{2^c} + \frac{T^2}{2^{\min\{c,r\}}}$ [Thm 9.2.1]	$\frac{ST^2}{2^c} + \frac{T^2}{2^r}$ [CDG18]

Figure 7.1: A summary of the attacks and advantage upper bounds for finding B-block collisions for the Sponge hash function. All bounds are given ignoring $\mathrm{poly}(c, r)$ terms. We note that the attack for $B = T$ is implicitly claimed in [CDG18] based on [CDGS18].

7.3 Optimal security of Keyed Hash Functions: Avoiding Time-Space Tradeoffs for Finding Collisions

Dealing with non-uniform attackers in the context of hashing and collision finding makes it necessary to consider a family of keyed hash functions, rather than a single hash function. Collision finding is then defined via the following two-stage game. First, a (keyed) family H of hash functions is fixed, and the attacker can depend arbitrarily on H. Second, a random key k is sampled, and the adversary needs to find a collision in H relative to k. Intuitively, in order to attack the hash function (e.g., find a collision), a non-uniform attacker must either (a) have some hard-coded information about k, or (b) can essentially be treated as uniform.

For applications, we typically want each member of H to operate on unbounded input lengths. That is, $H\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ should be viewed as a two-input function, operating on (k, m), where $\mathsf{k} \in \{0,1\}^\kappa$ is the key and $m \in \{0,1\}^*$ is an arbitrary length input. Since it is practically infeasible to design a different hash function for every input length, what happens is that a single basic compressing function $h\colon \{0,1\}^a \to \{0,1\}^n$ for some $a > n$ is designed, and then it is iterated in some way to get a hash function that compresses arbitrarily. For instance, the well-known Merkle-Damgård design [Mer90a, Dam89] iterates such a basic compressing function in order to get a variable-input-length hash function that can operate on arbitrary sized data up to some maximum length (e.g., 2^{64} bits).

Building a keyed hash from a single hash function. Observe that for every keyed hash construction, there is an (S, T)-attacker that finds a collision relative to a random k with probability[10] $\Omega(S/2^\kappa + T^2/2^n)$

[10]To simplify notation throughout the introduction, we suppress poly factors in n in the asymptotic $O(\cdot)$ and $\Omega(\cdot)$ notation.

via the following attack. First, the preprocessing adversary outputs $\Omega(S)$ collisions with respect to arbitrary distinct keys. The online adversary receives a random k. If k is in the remembered list from the preprocessing phase, it outputs the corresponding collision. Otherwise, it performs a T-query birthday-style attack. The adversary wins if either the challenge k appears in one of its preprocessed collisions (giving the $S/2^\kappa$ term) or if the birthday attack succeeds (giving the $T^2/2^n$ term). We refer to this attack as the *naive attack*, and say that a construction is *optimally secure* if there is provably no better attack. This brings us to the main question we consider in this work.

Can we build a keyed hash function (i.e., $H\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$)
from non-keyed one (i.e., $h\colon \{0,1\}^a \to \{0,1\}^n$) with optimal non-uniform security?

If we could design an $h_a\colon \{0,1\}^a \to \{0,1\}^n$ for every $a \in \mathbb{N}$, then the above task is easy. We can simply parse the input to the appropriate h_a into two parts, one for the key and the other for the input to $H\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$. That is, define $H(\mathsf{k}, m) = h_{\kappa+|m|}(\mathsf{k}\|m)$, where $\|$ stands for string concatenation and $|\cdot|$ stands for bit length. For this construction, Dodis, Guo and Katz [DGK17] showed that the best attack achieves advantage $O(S/2^\kappa + T^2/2^n)$, matching the advantage of the naive attack.

Unfortunately, it is infeasible to design a different hash function for every input length as discussed above. The design of a new h is a delicate and lengthy process that could take many years to test and standardize. Having a single hash function is therefore more robust security-wise. Thus, the standard procedure is to design a hash function h with fixed input size and then iterate it in some way to get a hash function that supports arbitrary input lengths.

It may seem that standard domain extension techniques for hash functions (like Merkle-Damgård, Sponge, or Merkle trees) provide a solution for this problem. Indeed, their goal is to take a hash function on a small domain and turn it into a hash function with arbitrary-size domain. But, as we point out next, the standard constructions suffer from a significant security loss. A priori, it is not even clear that this security loss is avoidable.

The security of existing constructions. First, consider (a keyed variant of) the Merkle-Damgård (MD) construction [Mer90a, Dam89], perhaps the most widely popular design for getting a hash function on long inputs from one on fixed input sizes. This design is not only extremely fundamental in cryptographic theory, but it also underlies popular hash functions used in practice, most notably MD5, SHA-1, and SHA-2. The $\mathsf{MD}\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ construction iterates the basic hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ by feeding in input blocks of size $s = a - \max\{\kappa, n\}$ one by one. It first pads the message appropriately such that it is a multiple of s bits. For key $\mathsf{k} \in \{0,1\}^\kappa$ and input $m \in \{0,1\}^s$, define $\mathsf{MD}(\mathsf{k}, m) = h(\mathsf{k}\|m)$. Then, for a longer input $m \in (\{0,1\}^s)^\ell$, viewed as ℓ blocks $m_1, \ldots, m_\ell$ each from $\{0,1\}^s$, recursively define

$\mathsf{MD}(\mathsf{k}, (m_1, \ldots, m_\ell)) = h(\mathsf{MD}(\mathsf{k}, (m_1, \ldots, m_{\ell-1})), m_\ell)$. We note that in the standard MD construction (studied, for example, in [CDGS18, ACDW20, GK22, AGL22]), k is only explicitly included once when processing the first message block.

Collision resistance of MD in the AI-ROM was first studied by Coretti, Dodis, Guo, and Steinberger [CDGS18] and more recently by [ACDW20, GK22, AGL22]. It is known that there is an attack, loosely based on the idea of rainbow tables [Hel80, Oec03], which succeeds in finding a collision with probability $\Omega(S/2^\kappa + ST^2/2^n)$. In typical settings of parameters, the $ST^2/2^n$ term dominates the above expression and in this case it is evident that MD suffers from a significant security loss.

Concretely, in the SHA-1 construction, $a = 678$ and $\kappa = n = 160$. If we model the underlying primitive $h\colon \{0,1\}^{678} \to \{0,1\}^{160}$ as a perfectly random function, an (S,T)-attacker with $S = 2^{53}$ and $T = 2^{50}$ will find a collision with probability $\approx 2^{-7}$ (essentially completely breaking the scheme).[11] On the other hand, the best one could hope is a construction with maximal advantage $O(S/2^\kappa + T^2/2^n) \approx 2^{-60}$ (obtained by the naive attack).

Another construction we mention is the Sponge [BDPVA07, BDPV08] construction, an alternative to the Merkle-Damgård design that underlies the modern SHA-3 hashing standard. As opposed to MD, the Sponge construction relies on a random *permutation* $\Pi\colon \{0,1\}^n \to \{0,1\}^n$. Sponge iterates Π by feeding in blocks of size $r < n$ from the input one at a time in a certain way. It results with a keyed hash function $\mathsf{Sp}\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^r$ with $\kappa + r = n$. Coretti et al. [CDGS18] (see also [FGK22]) showed that there is a collision finding (S,T)-attack with advantage $\Omega(ST^2/2^\kappa + T^2/2^r)$ against Sp relative to a random key. Again, we see that there is a non-trivial security loss in this construction.

It is important to note that for every choice of S and T the above attacks on Merkle-Damgård and Sponge beat the naive attack. In particular, there is no non-trivial choice of parameters where MD or Sp achieve the optimal security bound.

Lastly, we mention two other popular (variable-input-length) hash function designs: Merkle trees [Mer88] and the BLAKE family [AHMP08, AMPH14]. The former (Merkle trees) is a popular design that has important features like local opening and can be easily parallelized. Although it is extremely popular both in theory and in practice, we are not aware of a keyed variant that has been studied in the non-uniform setting. The latter (BLAKE) is a runner-up in NIST's competition to create a new hashing standard (where Sponge ended up as the winner). This design is based on the MD design, but they allow the inner hash function h to be keyed at every invocation. We are not aware of a formal study of its security in the non-uniform setting. Looking ahead, two of our main contributions are a proposal and analysis of the non-uniform security of Merkle tree and the MD/ BLAKE design, where the inner hash function h is keyed in

[11] These parameters roughly correspond to an attacker with $\approx$ 1000 terabytes of memory that uses optimized hardware that can compute 3 billion hashes per second for a long weekend.

every invocation. Concretely, we believe that this is the right notion to consider moving forward, in terms of non-uniform security.

A different perspective. Above, we considered the scenario where $h\colon \{0,1\}^a \to \{0,1\}^n$ is given, and we want to build an $H\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ which is as secure as possible for every (S,T)-attacker. A different perspective, slightly more target oriented, is to first fix a desired security level (say 2^{-50}) and the power of adversaries (say $S = T = 2^{60}$) and then understand which h is needed in order to get the desired H. If we use MD (for concreteness) for H, we will need $n \geqslant 230$, but if we had an optimally secure construction of H, we would need only $n \geqslant 170$. The latter could potentially be easier to design and argue about.

7.3.1 Our Results

We provide several constructions of keyed hash functions from non-keyed ones that do not suffer from any security loss (i.e., the naive attack that has advantage $\Theta(S/2^\kappa + T^2/2^n)$ is provably optimal). Our constructions provide various tradeoffs between their efficiency and the range of parameters (S and T) where they achieve optimal security.

Merkle-Damgård and Merkle trees with a keyed inner hash. All of our constructions can be viewed within a framework that builds on the Merkle-Damgård and Merkle tree constructions.

We start by discussing the MD-based approach. We consider an iterative hashing design where a compression phase is performed in every step using an "inner hash" function. The input for the compression phase is the current state and the next input. At the end, the compression phase outputs the next state. Of course, the inner hash function in the compression phase can use h as a subroutine. Abstractly, the compression phase for the MD-based construction is

$$y := \mathsf{compress}(\mathsf{k}, y, m),$$

where importantly the compress function takes k as input. See Figure 7.2 for an illustration. With this notation, the compression function of the standard MD function (at least as studied in numerous recent works including [CDGS18, ACDW20, GK22, AGL22]) is simply $\mathsf{compress}(\mathsf{k}, y, m) = h(y, m)$, and for the first step, y is initialized to k. Notably key is not included in every compression phases.

We next consider a parallelizable hashing design that generalizes the Merkle tree hash function. Here, each input is fed into a "leaf" of the Merkle tree, along with the k value. The compression function is then used to recursively combine outputs in previous levels until a final output is generated. Crucially, we always include k in the compression function. See Figure 7.3 for an illustration. This framework provides an alternative to the generalized MD approach described above. It requires at most a factor of two more calls

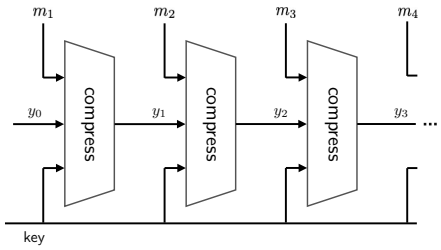


Figure 7.2: Our framework for building keyed hash functions based on the Merkle-Damgård construction with a keyed inner compression function.

to compress, but it is extremely parallelizable. Further, it provides a local opening property, where someone can prove that an individual message block m_i was included in the hash, without providing the full message.

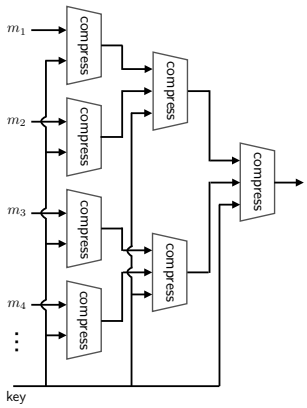


Figure 7.3: Our framework for building keyed hash functions based on the Merkle tree construction.

Our constructions are obtained by different implementations of compress, namely viewing compress as an inner keyed hash function used in the MD and Merkle tree designs. Quantitatively, the MD and Merkle tree approaches give similar results, so we focus our attention on instantiating compress in the case of the MD-based framework. However, all of our main results extend to the setting of the generalized Merkle tree framework, which we provide in Section 10.5 for completeness.

For simplicity of presentation of our results, we slightly simplify notation and assume that κ (the key

length) is equal to n (the output size of the hash function).[12] In our formal theorem statements in the technical sections, κ and n are treated independently when relevant.

Efficiency: We measure efficiency of a given construction by the number of calls to h needed to evaluate H at a single point. For example, in the standard MD construction with an underlying hash that maps $\{0,1\}^a$ to $\{0,1\}^n$, to hash a b-bit input, the query complexity is $b/(a-n)$ (ignoring rounding[13]). Indeed, every application of h takes as input the previous output (n bits) and so it can process $a-n$ bits from the input each time.

Assuming a large inner hash. Our first result shows that optimal security loss is achievable. That is, we show that there is a way to take a random oracle that operates on a fixed input length and get a keyed hash H that operates on arbitrary-length inputs with the following security guarantee: for any S, T, any (S,T)-attacker has minimal possible advantage in finding a collision in H relative to a random key. In words, the new construction is a variant of MD where we also feed k as input in every block. We refer to this construction as the MD construction with a keyed inner hash, in contrast to the standard MD construction where k is only fed in the first block. At a high level, feeding the key into every invocation of h allows us to reduce the probability of finding a long collision in H to that of finding a collision in h, which achieves optimal security $O(S/2^n + T^2/2^n)$ [DGK17]. Refer to Figure 7.4 for an illustration of how the construction works.

Theorem 7.3.1 (Informal; see Theorem 10.2.3)**.** *Assume $h\colon \{0,1\}^a \to \{0,1\}^n$ is modeled as a random oracle with $a > 2n$. Then, there is an $H_1\colon \{0,1\}^n \times \{0,1\}^* \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$ and any (S,T)-attacker, their advantage in finding a collision in $H_1(\mathsf{k},\cdot)$ relative to a random $\mathsf{k} \leftarrow\!\!\$\ \{0,1\}^n$ is $O(S/2^n + T^2/2^n)$.*

2. *One evaluation of H_1 on a given key and a b-bit message requires $b/(a-2n)$ queries to h.*

The above result is optimal in terms of security and is almost as efficient as standard MD if $a \geqslant 2n + \Omega(n)$. For example, if $a = 3n$, processing a b-bit input of H_1 requires querying h as many as $b/(a-2n) = b/n$ times. In the standard MD construction, only $b/(a-n) = b/(2n)$ queries are required, so our construction is less efficient than MD by a small constant factor at most 2 when $a = 3n$.

---

[12] We note that there are constructions that use $\kappa \neq n$ by design (e.g., BLAKE hash [AHMP08, AMPH14] uses $\kappa = n/2$).

[13] To be more precise, MD requires $\lceil (b+n+1)/(a-n) \rceil$ calls to h after padding the input with its length followed by a 1 and a sequence of 0s to fill the remaining current block. However, for ease of presentation, we ignore rounding in the introduction. In the formal theorem statements, we give exact efficiency bounds.

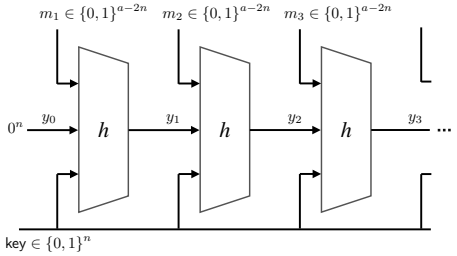


Figure 7.4: The construction H_1 underlying Theorem 7.3.1 given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a > 2n$.

However, H_1 is significantly less efficient than MD if a is roughly $2n$, i.e. h compresses by a factor of 2. For example, if $a = 2n + 1$, then processing a b-bit input of H_1 requires invoking h as many as b times. However, MD requires only $b/(n + 1)$ queries. This is a significant difference. We emphasize that having an efficient construction even when $a \approx 2n$ is not only a technicality but is rather important: concretely, assuming that the basic compressing function shrinks by a factor 2 is extremely common, both in theory and in practice. Thus, our next results are focused on closing this gap.

Instantiating the keyed inner hash with standard MD. To this end, we start by considering a construction H_2 that works for *any* $a > n$ and only incurs a factor of 2 overhead in terms of efficiency relative to MD. While this may seem too good to be true, we pay in terms of the assumptions we need to make to claim optimal security for collision resistance. Namely, the scheme has "optimal security," meaning any (S, T)-attacker can find a collision with probability at most $O(S/2^n + T^2/2^n)$, only whenever $S \leqslant T$ and $ST^2 \leqslant 2^n$.

This main idea behind the construction H_2 is to instantiate the compress function in the MD-based framework of Figure 7.2 with a standard MD hash function. We use k as the key for MD, and we treat $y_{i-1} \| m_i$ as the message. If we use a message block size $|m_i| = n$, this results in only a factor of two overhead relative to MD (essentially, half of the invocations of h incorporate bits of the message m_i, and half of the invocations incorporate bits of the previous output y_{i-1}). This construction is depicted in Figure 7.5 and gives the following result.

Theorem 7.3.2 (Informal; see Theorem 10.3.2)**.** *Assume $h\colon \{0,1\}^a \to \{0,1\}^n$ is modeled as a random oracle with $a > n$. Then, there is an $H_2\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$ such that $S \leqslant T$, $ST^2 \leqslant 2^n$, and any (S,T)-attacker, their advantage in finding a collision in $H_2(\mathsf{k}, \cdot)$ relative to a random $\mathsf{k} \leftarrow\!\!\$\ \{0,1\}^n$ is $O(S/2^n + T^2/2^n)$.*

2. *One evaluation of H_2 on a given key and a b-bit input requires $2 \cdot b/(a-n)$ queries to h.*

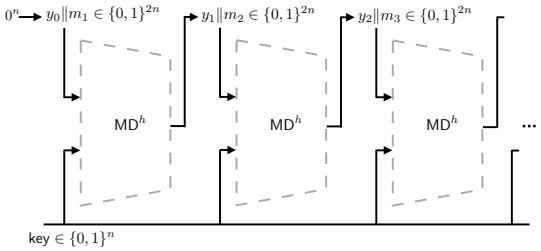


Figure 7.5: The construction H_2 underlying Theorem 7.3.2 given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a > n$. The gray dotted boxes represent the compress function, instantiated with the Merkle-Damgård construction, that uses $\mathsf{k} \in \{0,1\}^n$ as the key and $y_{i-1} \| m_i \in \{0,1\}^{2n}$ as the message.

We note that the assumption that $ST^2 \leqslant 2^n$ in the construction above comes from the fact that best currently known time-space tradeoffs for the collision resistance of standard MD (culminating in [GK22, AGL22] following the works of [CDGS18, ACDW20, CGLQ20]) require this assumption to get optimal bounds when analyzing the ℓ-block MD construction when $\ell \in \omega(1)$. In the special case where we only use a 2-block variant of MD as the underlying compress function, [ACDW20] give tight bounds that do not require that $ST^2 \leqslant 2^n$ (and furthermore, [GK22] gave tight bounds for all constants ℓ). This motivates our next construction, H_3, which uses the 2-block MD construction for compress, but requires that the input size to h satisfies $a > 3n/2$. See Figure 7.6 for an illustration, and the corresponding result is given in the following theorem.

Theorem 7.3.3 (Informal; see Corollary 10.3.3)**.** *Assume $h\colon \{0,1\}^a \to \{0,1\}^n$ is modeled as a random oracle with $a > 3n/2$. Then, there is an $H_3\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$ such that $S \leqslant T$ and any (S,T)-attacker, their advantage in finding a collision in $H_3(\mathsf{k}, \cdot)$ relative to a random $\mathsf{k} \leftarrow\!\!\$\ \{0,1\}^n$ is $O(S/2^n + T^2/2^n)$.*

2. *One evaluation of H_3 on a given key and a b-bit input requires $2 \cdot b/(2a-3n)$ queries to h.*

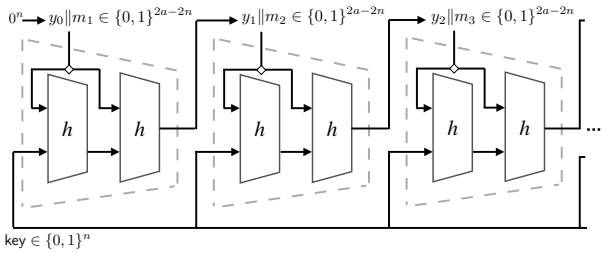


Figure 7.6: The construction H_3 underlying Theorem 7.3.3 given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a > 3n/2$. The gray dotted boxes represent the compress function, instantiated with a two-block Merkle-Damgård construction, that uses $\mathsf{k} \in \{0,1\}^n$ as the key and $y_{i-1}\|m_i \in \{0,1\}^{2a-2n}$ as the message. The $2a-2n$ bit message is split evenly into the first and second call to h, indicated in the figure by a diamond.

Instantiating the keyed inner hash with a 2-level Merkle tree. For our final construction, we seek to build a hash function H_4 that is both efficient and optimally secure whenever $a \approx 2n$ (h is only compressing by a factor of 2), without assuming that $S \leqslant T$. In particular, $S \gg T$ makes sense in many practical scenarios: the pre-processing attacker may have much more than time T to generate its advice string of size S, the online attacker may have easy random access to a structured advice string, or the online time T may be small for applications that enforce a timeout with fixed-time communication session (see [ASS+16] as an example of an attack on a TLS session that requires relatively heavy computation in an offline phase). Lastly, we mention that the bounds we obtain on H_2 and H_3 are tight—there is a non-trivial attack whenever $S > T$ that scales with advantage $\Omega(ST\ell/2^n)$ for standard ℓ-block MD [CDGS18, ACDW20].

For H_4, we instantiate the compress function from the framework of Figure 7.2 using a keyed variant of a Merkle tree construction (this has never been formally defined or analyzed to the best of our knowledge). In our variant, we feed k into all leaves of the Merkle tree. Concretely, in our construction, we use a 2-level Merkle tree and feed k to both of them, corresponding to two distinct invocations of h, and we split the "message" $y_{i-1}\|m_i$ into the remaining input bits for the leaves. The second level of the Merkle tree combines the two outputs from the first level, to produce a n bit output for compress. The full construction of H_4 is illustrated in Figure 7.7. We conjecture that this Merkle tree-based construction is optimally secure for collision resistance (see Remark 7.3.5 for more details), but analyzing its security turns out to be highly non-trivial. In particular, our current analysis is only optimally secure when $ST^2 \leqslant 2^n$, as stated in the following theorem.

Theorem 7.3.4. *Assume $h\colon \{0,1\}^a \to \{0,1\}^n$ is modeled as a random oracle with $a \geqslant 2n$. Then, there is an $H_4\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$ such that $ST^2 \leqslant 2^n$ and any (S,T)-attacker, their advantage in finding a collision in $H_4(\mathsf{k},\cdot)$ relative to a random $\mathsf{k} \leftarrow\!\!\$\ \{0,1\}^\kappa$ is $O(S/2^n + T^2/2^n)$.*

2. *One evaluation of H_4 on a given key and a b-bit input requires $3 \cdot b/(2a-3n)$ queries to h.*

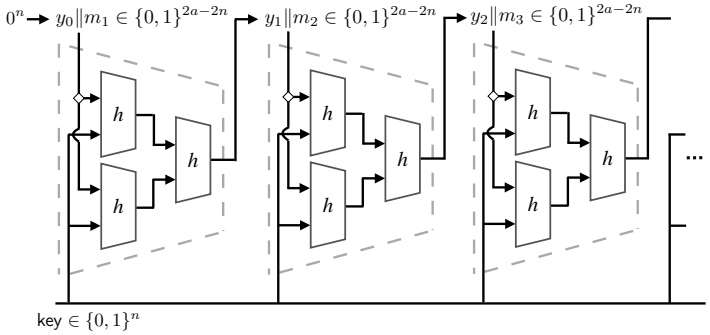


Figure 7.7: The construction H_4 underlying Theorem 7.3.4 given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a \geqslant 2n$. The gray dotted boxes represent the compress function, instantiated with a two-level Merkle tree, that uses $\mathsf{k} \in \{0,1\}^n$ in each leaf and $y_{i-1} \| m_i \in \{0,1\}^{2a-2n}$ as the message. The $2a-2n$ bit message is split evenly between the two leaves of the Merkle tree, indicated by a diamond in the figure. We require $a \geqslant 2n$ so that both outputs from the leaves can be fed into the next layer of the Merkle tree.

We summarize our results in Table 7.1.

Remark 7.3.5 (A conjecture on the security of keyed Merkle trees)**.** *The main building block in our construction of Theorem 7.3.4 is a Merkle tree where the key value k is included only at the leaves. Concretely, we include a key k in each of the leaves of a Merkle tree and then fill in the rest of the leaves with bits of some message $m \in \{0,1\}^*$, and then run the Merkle tree construction (with an unkeyed hash) as normal to get an n-bit output. In this work, we analyze the simplest case where the Merkle tree has depth 2 with only two leaves. However, this naturally generalizes to any number of ℓ leaves resulting in a tree of depth $O(\log \ell)$.*

We conjecture that this approach, where only the leaves are keyed, is as secure as the Merkle tree approach of Figure 7.3 where the inner hash function is keyed at every invocation, including interior nodes. This latter

	Advantage	Efficiency	Input Size	Assumptions
MD	$\Theta(ST^2/2^n)$	$b/(a-n)$	$a > n$	None
H_1 (Thm. 7.3.1)	$\Theta(S/2^n + T^2/2^n)$	$b/(a-2n)$	$a > 2n$	None
H_2 (Thm. 7.3.2)	$\Theta(S/2^n + T^2/2^n)$	$2 \cdot b/(a-n)$	$a > n$	$S \leqslant T, ST^2 \leqslant 2^n$
H_3 (Thm. 7.3.3)	$\Theta(S/2^n + T^2/2^n)$	$2 \cdot b/(2a-3n)$	$a > 1.5n$	$S \leqslant T$
H_4 (Thm. 7.3.4)	$\Theta(S/2^n + T^2/2^n)$	$3 \cdot b/(2a-3n)$	$a \geqslant 2n$	$ST^2 \leqslant 2^n$

Table 7.1: A summary of our results as well as the standard MD construction for reference. The advantage of a construction is given in terms of the probability of (S,T)-attackers to find collisions relative to a random key. The efficiency is measured in terms of the number of calls to h that maps a-bit inputs to n when processing a b-bit input. The assumptions column specifies conditions on various parameters.

approach requires a larger, more complicated, inner hash function, so we would like to avoid this if at all possible.

First, for the simple case of a depth two tree with keyed leaves, we conjecture that the bound we show in this work is not tight (see Theorem 10.5.3 for the exact bound we show). Namely, we believe that we should not need to assume $ST^2 \leqslant 2^n$ (or make any assumptions on S,T) in order to get optimal security. Second, we believe that this intuition should extend to the arbitrary depth Merkle trees that are keyed at the leaves, and we conjecture that it should also achieve optimal (S,T) security without any assumptions on S,T. However, even getting a bound in this case that is optimal in the setting $ST^2 \leqslant 2^n$ we believe would be very interesting. Additionally, handling deeper Merkle trees could potentially allow for constructions that do not necessarily have even a two-to-one structure, meaning that we could build a keyed hash function based on Merkle trees without assuming the hash function h has input size $a \geqslant 2n$.

We provide the details of this result in Chapter 10.

7.4 The Query Complexity of Preprocessing Attacks

While time-space tradeoffs have been extensively studied, question that has received less attention, however, concerns the study of tradeoffs between the *offline* time needed to compute the advice and the online time. To the best of our knowledge, the only such result, due to Corrigan-Gibbs and Kogan [CK18], focuses on

the discrete logarithm problem in the generic-group model.

Initiating a more comprehensive and grounded study of such *offline-online time tradeoffs* is the main goal of this chapter. As in prior works on space-time tradeoffs, we focus on proving lower bounds relying on ideal primitives, thus approximating time with query-complexity. In addition to providing a generic overview of how salting makes preprocessing attacks expensive, we revisit under a new lens the recent works [CDGS18, ACDW20, AGL22, GK22] on preprocessing attacks against *salted* hash functions and the Merkle-Damgård construction.

WHY DOES TIME-COMPLEXITY MATTER? One main reason to study preprocessing attacks is to model non-uniform security. In this case, it is indeed irrelevant *how long* it takes to compute a good advice string since its mere *existence* suffices for an attack, although such an attack may well never be explicitly found. This perspective has emerged from the debate around the use of non-uniformity in security [KM12, BL13], although often the issue can be bypassed entirely via cleverer uniform reductions, as in e.g. [GPR14]. The importance of non-uniform attacks in practice has also been a source of debate [Rog06].

Here, we are concerned with a more practical and less formal perspective where preprocessing is used in practical, explicit attacks for one of two reasons:

1. An attack needs to run very quickly in the online stage (e.g., must succeed before a time-out occurs in an Internet protocol) but can afford to run much slower in an offline stage. For instance, Adrian et al. [ABD+15] use offline computation to break 512-bit finite-field discrete logs in less than a minute of online time, hence compromising legacy versions of the Diffie-Hellman handshake.

2. The advice is computed once and for all and is re-used to attack multiple instances. This is the scenario of *Rainbow Tables* [Oec03], which leverage Hellman-type tradeoffs to speed up attacks against unsalted password hashes.

In both cases, it is imperative that the offline time remains within a feasible range.

WHEN IS PREPROCESSING WORTH IT? Different preprocessing attacks exhibit very different offline-online time tradeoffs. The main goal is to ensure that, thanks to preprocessing, the online complexity of an attack is better than the best preprocessing-free attack. For example, in Rainbow Tables, for a password dictionary of size N, the preprocessing takes time $T_1 \approx N$ to produce advice of size S, for which the online complexity is then $T_2 \approx N/S$. The online complexity bests the optimal online-only attacks, which is $\Omega(N)$; moreover, the preprocessing time is optimal since the *sum* of the offline and online time cannot beat the complexity of the best online only attack.

A more interesting example is finding collisions in the *salted* Merkle-Damgård (MD) construction, as studied in a line of recent works [ACDW20, AGL22, GK22]. Given a (random) compression function h :

$[N] \times [M] \to [N]$, the offline phase of the optimal attack for two-block collisions finds S collisions of the form $h(a_i, m_i) = h(a_i, m_i')$ for $m \neq m'$ and salts $a_1, \ldots, a_S$, for which offline time $T_1 \approx S \cdot \sqrt{N}$ is necessary. Then, the online phase, given the salt a, uses time $T_2 = N/S$ to find m such that $h(a, m) = a_i$ for some i, which yields a collision $m \| m_i$, $m \| m_i'$. This attack achieves the tradeoff $T_1 \times T_2 = N^{3/2}$, and the online time beats the naïve Birthday attack whenever $T_1 = \Omega(N)$. It is not clear, however, whether the tradeoff is optimal, and this is indeed one of the questions we are addressing below.

Our contributions. This chapter initiates an in-depth investigation of the time-complexity of preprocessing attacks, and we focus primarily on salted constructions using hash functions, which is where the most interesting technical questions emerge. In particular:

- **Generic salting.** We propose a generic technique to analyze the common practice of salting to mitigate the effects of preprocessing attacks. We consider a model where every call to the underlying primitive is salted. Qualitatively, our result implies that in most settings, to beat the best online-only attack, one needs to invest an offline effort proportional to compromising the primitive on *every salt*.

- **Concrete bounds for random oracles.** Our generic technique can be combined with a recent work by Jaeger and Tessaro [JT20] to provide concrete quantitative upper bounds on the advantage of an offline-online adversary. These bounds are not always optimal, and we prove more refined bounds. We exemplify this situation by studying the pre-image-resistance and collision-resistance of a salted random oracle.

- **Merkle Damgård construction.** The technical bulk of this chapter studies the salted Merkle-Damgård (MD) construction with a random ideal compression function. Here, salting achieves a more limited effect and still allows for non-trivial tradeoffs between the offline and online complexity of an attack. We deliver quantitative upper bounds on the advantage of breaking pre-image-resistance and collision-resistance of the two-block salted MD for offline-online adversaries.

Salting defeats preprocessing. We start with a result that generically justifies the practice of salting cryptographic primitives to defeat preprocessing attacks. Such results were proved for space-time-complexity in [CDGS18], but we give an analogue result for offline-online query-complexity.

Concretely, we assume that we have a scheme Π^g that relies on a random function $g : [M] \to [N]$, and that the advantage in breaking Π^g, as a function of the number of queries to g, is a well understood quantity. (In particular, here we assume that the security depends only on the number of queries to g.) Now, we replace $g(\cdot)$ with a *salted* hash function $h(a, \cdot)$, where $h : [S] \times [M] \to [N]$. We aim to quantify security for

an attacker $\mathcal{A}$ which, during an offline phase, is allowed to issue T_1 queries to $h(\cdot,\cdot)$. Then, after learning the random salt $a \leftarrow\!\!\$\ [S]$, $\mathcal{A}$ attacks $\Pi^{h(a,\cdot)}$. In this online stage, $\mathcal{A}$ can issue T_2 queries to $h(\cdot,\cdot)$.

We show that if (roughly) T^* queries to g are needed to break Π^g in the *worst case* with very high probability, then for the attacker $\mathcal{A}$ to succeed with high probability as well, $T_1 \geqslant S \cdot T^*/2$ or $T_2 \geqslant T^*/2$ must hold. In other words, the *only* way to beat the best online-only attack is to invest an amount of preprocessing equivalent to that of breaking the scheme *for every choice of the salt*. At the core of this proof is a simple argument that shows how to build an adversary $\mathcal{B}$ against Π^g, achieving the same advantage as $\mathcal{A}$, with *expected* query-complexity $T_1/S + T_2$.

Quantitative bounds for salted random oracles. The above generic result holds for adversaries achieving high advantage. Overall, we would like to go one step further to obtain quantitative precise upper bounds on the advantage of $\mathcal{A}$ as a function of T_1 and T_2 and to characterize the whole advantage curve. As our first result, we combine the above reduction to an adversary with expected query-complexity with the work by Jaeger and Tessaro [JT20]. This allows us to show that any adversary attempting to break pre-image-resistance of a random oracle with a s-bit salt and n-bit outputs succeeds with probability at most

$$\frac{T_1}{S \cdot N} + \frac{T_2}{N}\ ,$$

where $N = 2^n$, $S = 2^s$ and, once again, T_1 and T_2 are the offline- and online-query-complexity. This bound ends up being nearly *exact* in that there are offline- and online-only attacks achieving each of the two terms.

Unfortunately, the same approach via [JT20] yields only suboptimal bounds for other properties, such as the collision resistance of a salted random oracle. Here, we give a direct proof that shows a bound of order

$$\frac{T_1}{S \cdot \sqrt{N}} + \frac{T_2^2}{N}\ .$$

This proof is of independent interest, and uses a compression argument to bound the number of salts for which a collision is found in the preprocessing stage. And indeed, the first term is matched by an attack that finds collisions for $\Omega(\lceil T_1/\sqrt{N} \rceil)$ salts, issuing $\sqrt{N}$ queries for each of these salts.

Tradeoffs for Merkle-Damgård. Our first set of results aims to show that salting prevents offline-online tradeoffs–the best attack is either fully offline or fully online. However, we show that this is not true if we cannot afford to salt *each* call to a primitive. We focus in particular on the salted Merkle-Damgård (MD) construction [Mer90b, Dam90], which has also been central to a recent wave of works in the context of space-time tradeoffs [ACDW20, AGL22, GK22]. Here, we are given a compression function $\mathrm{h} : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$ and a message M that consists of B blocks $M_1, \ldots, M_B \in \{0,1\}^\ell$. To hash M, one sets the initial value y_0 to equal the salt a and computes the final hash y_B by iterating

$$y_i \leftarrow \mathrm{h}(y_{i-1}, M_i)\ .$$

Here, we focus on the case of messages of length at most two, which, as in the case of space-time tradeoffs [ACDW20], already captures many of the challenges. (In fact, we believe that going beyond requires significantly new techniques than those we explore in this chapter.) For *pre-image-resistance*, we prove a bound (which we show to be tight) of the form (when ignoring constant factors and lower-order terms)

$$\frac{T_2}{N} + \frac{T_1 T_2}{N^2} + \frac{T_1^2}{N^3} \ .$$

The most interesting term is the middle one: it is leading, e.g., for $T_1 = N^{6/5}$ and $T_2 = N^{4/5}$, and suggests an inherent tradeoff between offline and online query-complexities. Indeed, this advantage is (roughly) matched by an actual attack that first evaluates $\mathrm{h}(a_i, M)$ on N distinct M's for T_1/N different salts $a_1, \ldots, a_{T_1/N}$. Then, upon learning the value y to invert on, as well as the salt a, the online adversary spends its T_2 queries looking for $M \in \{0,1\}^n$ such that $h(a, M) = a_i$ for some $i \in [T_1/N]$, succeeding with probability $\Omega(T_1 T_2/N)$. Then, given it succeeds, the attacker knows N evaluations of $h(a_i, \cdot)$ and is thus likely able to find $M' \in \{0,1\}^n$ such that $h(a_i, M') = y$. Hence, (M, M') is a pre-image of y.

Collision-resistance of MD. Our most involved result is the analysis of the *collision-resistance* of the two-block MD construction, which in particular relies on a number of sophisticated compression arguments and results in a bound that we know to be only partially tight. Ignoring lower order terms and constant factors, we show a bound of order

$$\frac{T_2^2}{N} + \frac{T_1 T_2}{N^{3/2}} + \frac{T_1}{N^{5/4}} + \frac{T_1^2}{N^{7/3}} \ .$$

Here, we show matching attacks for all terms except the last one. This (potential) lack of tightness of the last term is due to our combinatorial analysis of a special type of offline-only attack. Namely, an offline attacker could repeatedly attempt to find a special type of collision called a *diamond* for a (potential) salt a, namely, four (distinct) queries $\mathrm{h}(a, x_1) = y_1$, $\mathrm{h}(a, x_2) = y_2$, $\mathrm{h}(y_1, x_1') = z_1$, $\mathrm{h}(y_2, x_2') = z_2$ such that $z_1 = z_2$. If the attacker finds a diamond for k salts, then in the online phase it wins with probability k/N (with no further query). Therefore, this boils down to proving a tail inequality on the number of salts for which a diamond is found with T_1 queries. This is challenging since in the regime $T_1 \gg N$ the combinatorics of random functions are not very well understood. The challenge stems from the fact that the "outer" queries $\mathrm{h}(y_1, x_1') = z_1$ and $\mathrm{h}(y_2, x_2') = z_2$ in one diamond could, individually, be part of diamonds for different salts. Our proof uses compression arguments to provide a suitable tail bound, but we leave it as an open problem to improve our analysis (or show it is tight).

Combining space and time. In conclusion, we observe that our approach is entirely dual to that of space-time tradeoffs. The latter completely ignores the issue of *time* to produce advice, whereas we completely ignore the issue of advice size. The obvious question is whether both can be combined, and we currently lack good techniques to combine space and query-complexity.

Relationship with Multi-Instance Security. We note that a remark in [CK18] observed that a lower bound against multiple-discrete-log algorithms also yields lower bounds on the preprocessing time for discrete-log algorithms with preprocessing (observation attributed to Dan Bernstein). We can extend this approach to relate the advantage of an offline-online adversary with the advantage of an adversary playing a multi-instance game. However, we find that this does not give tight bounds for the advantage of offline-online adversaries that succeed with small (sub-constant) probability. In Section 11.5, we illustrate this via an example.

We provide the details of this result in Chapter 11.

Chapter 8

TIME-SPACE TRADEOFFS FOR BOUNDED-LENGTH COLLISIONS IN MERKLE-DAMGåRD HASHING

In this chapter, we study the time-space tradeoffs for short collisions the Merkle-Damgård (MD) hashing. Specifically, we confirm the STB conjecture posed by [ACDW20] in many new parameter settings. For instance, in one result, we show that the conjecture holds for all constant values of B, significantly extending the result of [ACDW20]. This chapter is based on our work in [GK22].

We start off this chapter by providing a high-level overview of our techniques. Both of our results follow a similar high-level rationale, and thus throughout this overview, we mainly focus on the techniques for proving the STB conjecture whenever $B \in O(1)$ (Theorem 7.1.1). Towards the end, we describe the additional ideas needed to obtain the result for $SB \ll T$ (Theorem 7.1.2).

Before explaining the ideas, let us describe the challenge more precisely. We are given a compressing function $h : [N] \times [M] \to [N]$, modeled as a random oracle, and we want to upper bound the probability of a non-uniform attacker in finding a collision in an MD_h instance with a random salt. We model non-uniform attackers by thinking of them as two-stage adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. The offline part $\mathcal{A}_1$ is unbounded in running time, and its only restriction is that it can output only S bits. This output is the non-uniform advice given to the online part $\mathcal{A}_2$ which is then allowed to make up to T queries after which it must output a B-block collision for MD_h and terminate. We assume unbounded running time for both parts $\mathcal{A}_1$ and $\mathcal{A}_2$ and only restrict the output-size for $\mathcal{A}_1$ and the number of queries for $\mathcal{A}_2$. We refer to such a two-stage adversary $(\mathcal{A}_1, \mathcal{A}_2)$ as a (S, T)-adversary. In the context of length-B collisions in MD_h, the game is as follows:

- $\mathcal{A}_1$ has unbounded access to h and it outputs $\sigma \in \{0,1\}^S$.

- $\mathcal{A}_2$ gets σ as input along with a random salt $a \in [N]$.

- $\mathcal{A}_2$ outputs α, α'.

- $\mathcal{A}$ wins if $\alpha \neq \alpha'$, α, α' consist of $\leqslant B$ blocks, and $\mathsf{MD}_h(a, \alpha) = \mathsf{MD}_h(a, \alpha')$.

There are essentially two main generic approaches known in the literature for proving bounds of this sort. The first is the so-called *pre-sampling* technique, originally due to Unruh [Unr07], and the second is

a compression argument. The first technique reduces the problem from considering general hash functions and adversaries $(\mathcal{A}_1, \mathcal{A}_2)$ as above to a simpler model (and associated attacker) called the *bit-fixing* model. The advantage of the latter model is that it is typically easier to analyze and results in clean proofs. The second technique is based on a simple information-theoretic idea that *random bits cannot be compressed.*[1] Thus, an attacker that succeeds in finding collisions is used to compress some random information that is used in the game, and thereby contradiction is reached. This technique, while being extremely influential in many fields and problems in computer science (e.g., "Algorithmic Lovász Local Lemma" [MT10], lower bounds on cryptographic constructions [GT00, DTT10, GT20], analyzing hardness of problems in the non-uniform setting [DGK17, CK18] and time-space tradeoffs for quantum algorithms [CGLQ20]), often results in technical and complex proofs.

It would have been convenient if any non-trivial bound on our problem could be obtained using the bit-fixing technique. Unfortunately, Akshima et al. [ACDW20] observed that finding short collisions is relatively easy in the bit-fixing model. Hence, the only remaining potentially helpful technique is based on compression. Indeed, Akshima et al. [ACDW20], as their primary technical contribution, managed to carry out such an argument for the particular case of $B = 2$, and already then their proof is highly non-trivial and consists of a tedious case analysis. We distill some of the main ideas underlying their general framework approach[2] next—this will be useful for us, as well.

The framework. We reduce the task of handling arbitrary (S, T)-adversaries to the problem of handling (S, T)-adversaries, where the preprocessing part $\mathcal{A}_1$ is degenerate and outputs a fixed string σ, independent of h. Specifically, we define a game, parameterized by $u \in \mathbb{N}_{>0}$, where $\mathcal{A}_2$ has an arbitrary size S string σ hard-coded, and its goal is to find a collision relative to a given salt. $\mathcal{A}_2$ wins the game if it succeeds in finding a collision when executed with *every one* of u uniformly random salts (there is no $\mathcal{A}_1$ in this game). The reduction shows that if $\mathcal{A}_2$ has advantage ε in the modified game, then the best advantage of an (S, T)-adversary in the original game is (roughly) $O(\varepsilon^{1/u})$ for $u \approx S$. This reduction, formalized in Lemma 10.4.1, is adapted from Akshima et al. [ACDW20] and it uses the beautiful "constructive Chernoff bound" of Impagliazzo and Kabanets [IK10].[3]

The advantage of considering the new game is that there is no $\mathcal{A}_1$, so it is easier to handle. But, to obtain a meaningful result for the original (S, T) game, say an upper bound of ε, we need to prove a somewhat stronger upper bound for the new game, that is, roughly ε^u. This means, in other words, that we need to show how to compress about $\log(1/\varepsilon)$ bits *per each one of the u salts.* (Actually, keep in mind that it suffices

[1] Specifically, it is impossible to save w bits of information about a random string, except with probability 2^{-w}.

[2] We note that [CGLQ20] introduced an equivalent framework in independent work.

[3] The use of this reduction is the main (and perhaps only) point of similarity between our proof and [ACDW20]'s.

to achieve this on average!) For our target ε, we therefore have the following goal.

Main challenge: For every one of the u salts, we need to "save/compress" (on average) roughly the following number of bits:

$$\log\left(\min\left\{\frac{N}{uT(\log u)^{2(B-2)}B^2}, \frac{N}{T^2}\right\}\right).$$

By impossibility of non-trivial compression, this would imply that $\mathcal{A}_2$ must succeed with probability at most

$$\varepsilon \leqslant O\left(\left(\frac{uT(\log u)^{2(B-2)}B^2}{N} + \frac{T^2}{N}\right)^u\right),$$

which would give our result when plugged into the framework.

THE COMPRESSION ARGUMENT. The random string that we shall compress consists of the set of salts denoted U, as well as the function h. We give encoding and decoding algorithms that use $\mathcal{A}_2$ first to encode the pair (U, h) and then use the result to fully decode them whenever $\mathcal{A}_2$ wins the game. If $\mathcal{A}_2$ wins with good enough probability, the output of the encoding procedure will be non-trivially short with good probability, which is a contradiction.

Remark 8.0.1. *Akshima et al. [ACDW20] used the same approach for $B = 2$, but their proof does not seem to scale for larger values of B. Specifically, their proof involves an exhaustive case analysis. It seems like a naive attempt to generalize their bound to larger values of B would proliferate the number of cases needed to be analyzed, making it unmanageable. One of our key conceptual insights is a structural characterization of collisions in MD_h that prevents this explosion in the number of cases needed to be handled. While our analysis applies to any B, the number of total cases we consider is roughly the same as Akshima et al.*

A FIRST ATTEMPT AND A GLIMPSE AT THE CHALLENGE. Let us make a strong (and typically false) assumption: *the adversary $\mathcal{A}_2$ never repeats any query to h across all the u runs.*[4] Since we can assume (w.l.o.g) that if $\mathcal{A}_2$ outputs (α, α') when run on salt a, it has queried h at all values needed to compute $\mathsf{MD}_h(a, \alpha)$ and $\mathsf{MD}_h(a, \alpha')$, we are guaranteed that for each of the salts $u \in U$ there are at least two *distinct* queries which have the same answer, i.e., a collision. The indices of these queries reside in $[T]$ as this is the query complexity of $\mathcal{A}_2$ when executed on the particular salt a. Thus, we could avoid encoding the answer of the second query and instead encode these two indices in T and remove the answer of the second query from evaluations of h. This saves us $\log(N/T^2)$ bits for every salt, giving us even more savings than what we are aiming for. Such compression, in turn, would imply that the birthday attack is optimal, no matter what B is (which makes sense given our assumption but is clearly false for general attackers).

[4] While we can assume without loss of generality that $\mathcal{A}_2$ does not repeat queries within a single execution (since it is not memory-bounded), it is not very reasonable to assume that it will never repeat queries across different executions on different salts.

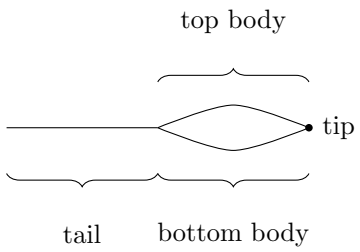


Figure 8.1: A mouse structure. For ease of visual representation we do not draw the nodes and edges of the graph, instead represent it as a continuous structure.

A naive way to get rid of the assumption (that queries never repeat across different u runs) would be to encode the index of the other query among all uT queries made (instead of T). But, this would eventually result in another multiplicative S term in our bound. Namely, we would only be able to save $\log(N/(ST^2))$ bits per salt which is too little for us (as it leads to a trivial upper bound). This motivates us to look more closely at how MD collisions are formed, what kind of queries could be involved, and how we could leverage the fact that collisions are short to get more efficient encoding.

THE MOUSE STRUCTURE AND QUERY TYPES. We consider the graph implicitly formed through the queries made by $\mathcal{A}_2$ when running on salts $a_1, \ldots, a_u \in U$. The nodes of the graph are the possible salts and there is a directed edge from salt a to a' with label α if $h(a, \alpha) = a'$ and this query was made by $\mathcal{A}_2$.

Suppose that $\mathcal{A}_2$, when run on salt a_i, outputs (α, α'). Since $\mathcal{A}_2$ wins on every salt in U, its output on salt a_i, denoted α, α' must satisfy: (1) $\mathsf{MD}_h(a_i, \alpha) = \mathsf{MD}_h(a_i, \alpha')$, (2) $\alpha \neq \alpha'$, and (3) α, α' are at most B blocks long. Without loss of generality we can assume that the adversary $\mathcal{A}_2$ outputs a "minimal" collision, i.e., one that does not contain a prefix which is a collision by itself. For instance, say the collision is $x_1, x_2, x_3, x_4, x_5, x_6$ and $y_1, y_2, y_3, y_4, y_5, y_6$ (wrt salt a) and it happens that x_1, x_2, x_3, x_4 and y_1, y_2, y_3, y_4 already collide (wrt same salt a), we can simply ignore x_5, x_6, y_5, y_6. Considering the *core* sub-graph of the query graph that is induced by queries made by $\mathcal{A}_2$ that are *required* to evaluate $\mathsf{MD}_h(a, \alpha)$ and $\mathsf{MD}_h(a, \alpha')$, we obtain a structure that we call a **mouse structure**. Important parts of the mouse structure are the tail, top and bottom body, and the tip, as depicted in Fig. 8.1. Our entire approach is based on studying these mouse structures to come up with encoding strategies.

Given the concept of a mouse structure and our discussion from above about the adversary, possibly repeating queries motivates us to classify each query into one of three types. The first is called NEW and refers to queries made for the first time. The rest of the queries are called *repeated*, and they are further classified into two types, depending on whether they previously appeared in some mouse structure or not.

Specifically, a repeated query is called REPEATEDUSED if the same query was already made by $\mathcal{A}_2$ when executed on a previous salt, and otherwise, a repeated query is called REPEATEDUNUSED.

Intuitively, we want to save bits when the answer of a NEW query was the input salt of a repeated query. A REPEATEDUSED query facilitates such savings since we can encode the answer to the query by storing its index along with the index of the previous query and the corresponding index within the mouse structure—a total of $\approx \log(uTB)$ bits instead of $\log N$—which eventually turns into an STB/N term in the upper bound, as conjectured. The problem is with encoding REPEATEDUNUSED queries—there seems to be no trivial way to write the index of the previously-made query with less than $\log(uT^2)$ bits, which is too much (since it will eventually turn into a ST^2/N term in the upper bound).

SOME "EASY" MOUSE STRUCTURES. We observe that some cases of mouse structures readily give us a way to have efficient encoding and save sufficiently many bits. We offer three examples to convey intuition on how our analysis is done. Throughout, let us assume that every mouse structure contains at least one NEW query—otherwise, we will ignore this mouse structure altogether.[5]

As a first example, if two NEW queries form the tip of the mouse structure of salt a_j, we can simply encode the index of these queries within the queries made while handling this salt—this uses $2 \log T$ bits instead of $\log N$ bits which is sufficient. As another example, if a self-loop forms the body of the mouse structure and the self loop query is NEW or REPEATEDUNUSED, we can simply encode the index of the self loop query in the list of queries used to handle this salt and avoid encoding the answer of the query—this uses $\log(uT)$ bits instead of $\log N$ which is again sufficient. As the last example, suppose the answer to a NEW query is a salt that appeared in some earlier mouse structure. In this case, we can avoid encoding the answer of the NEW query and instead encode the index of the query and which of the salts in the previous mouse structures is the answer—this uses $\log(2uB)$ bits (because there are at most $2B$ salts in mouse structures and at most u mouse structures) instead of $\log N$ which is also sufficient.

SOME "HARD" MOUSE STRUCTURES. The aforementioned easy cases give rise to a relatively easy encoding that results in an optimal STB/N term. Next, we focus on the more complex cases, which cause our bound to have the extra $\approx B \cdot (\log S)^B$ factor.

Assume that there is a mouse structure where there are two salts a' and a'' such that a' is the input salt to a REPEATEDUSED query, a'' is the answer to a NEW query, and the path from a'' to a' in the mouse structure consists of only REPEATEDUNUSED queries (as shown in Figure 8.2). By definition, the distance

[5] In the technical section, we refer to salts a_j in U that were not the input salt of a query when running $\mathcal{A}_2$ on a_i for $i < j$ as *fresh*. It follows that mouse structures for fresh salts will always have a NEW query. For salts that are not fresh, it is relatively straightforward to achieve some compression by avoiding storing these salts in the encoding of U. For now, the reader can imagine that all salts are fresh for simplicity.

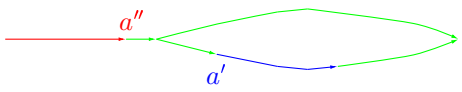


Figure 8.2: An example of a "hard" mouse structure. The NEW queries are drawn in red, REPEATEDUSED queries are drawn in blue and the REPEATEDUNUSED queries are drawn in green.

between a' and a'' is at most B.[6] Potential savings could be achieved by not encoding a'', the answer of the NEW query, as it can essentially be extracted from already-observed queries. We can easily encode the appropriate index of the query in the mouse structure and the salt a' using roughly $\log(uB)$ bits, but how can we encode the information about the path back from a' to a''?

The non-triviality is that for each node on this path, there might be many possible ways to reach it among all the different queries that have been already made, i.e., if each node on this path has fan-in m, namely an m-multi-collision, then the natural encoding of the path back would cost at most $\log(m^B)$ bits, specifying which back edge to take for every node. Here, m could be very large, e.g., as large as S or even larger, making the whole result meaningless for most reasonable parameters settings. (We also need to encode the length of the path, which would lead to the additional multiplicative B factor, but we ignore it in the discussion here.)

We get around this problem of m potentially being very large by observing that one of the following two cases holds:

- *Many small multi-collisions:* either the fan-in of every node along the path (i.e., the number of previously-made queries whose result is a node on this path) is smaller than $\log u$, or

- *One large multi-collision:* there is (at least one) node on the path where the fan-in is at least $\log u$.

In the earlier case, we encode the path back as mentioned above, where we write the index of each back edge. This costs us $B \cdot \log\log u$ bits, which eventually translates to an extra $(\log S)^B$ term,[7] as we have in our main theorem. The question is, what do we do when the second case occurs.

The idea is to leverage the fact that there is a large multi-collision and obtain our savings from a completely different place. Specifically, we remember the index of all the queries involved in the multi-collision and the common answer. A calculation reveals that if the collision consists of $\approx \log u$ edges, we can

[6] By being slightly more careful, we can show that the distance is $B - 2$ but we ignore this fact for the overview.

[7] Remember that the actual term is $(\log S)^{B-2}$ and that is why the proof of Akshima et al. [ACDW20], in which it was assumed that $B = 2$, did not have an extra term that depends on S.

already save enough. One subtlety is that the same multi-collision might repeat throughout many different mouse structures, and we need to make sure not to double count the savings from a single multi-collision more than what we get. The reason why we do not double count is that a single large-enough (i.e., with $\log u$ edges) multi-collision saves us enough bits for about $\log u$ different structures, and at the same time, such a multi-collision can appear in at most $\log u$ mouse structures.

Let us finally remark that while the above description conveys the main idea underlying our compression strategy, it is somewhat simplified and glossed over many technicalities and the complete specification of all possible cases that our full proof covers. We refer to Section 9.3 for full details.

Proving the STB conjecture for $S \cdot B \ll T$. The proof of our second upper bound follows the same overall structure. The only difference is, naturally, in the way we encode reverse paths in mouse structures that have nodes corresponding to repeatedUnused queries between the output a'' of a new query and a node a' corresponding to a repeatedUsed query.

In the earlier proof, when we needed to locate the salt a'' from salt a', we encoded the number of edges in between and all the edges on the path. Here, we prove a purely graph-theoretic lemma saying that if for salt a' there are more than $\approx u^2$ salts a'' such that there are $d > 0$ edges on the shortest path back from a' to a'' in the query graph, then there must be $t \geqslant 1$ multi-collisions in the graph, such that the total number of edges involved in the t multi-collisions is at least $\approx u^2$. While the proof of this lemma is a simple inductive argument, it turns out to be extremely helpful for us. Specifically, if there are $t \geqslant 1$ different multi-collisions such that a total of at least $\approx u^2$ different queries are involved in the t multi-collisions, *we can save enough by only encoding these multi-collisions and nothing else.* To prove this fact, we consider the minimum savings we can get from encoding these t multi-collisions. We show using some elementary calculus that if there are $\approx u^2$ queries involved in t different multi-collisions, the minimum saving is more than the total amount of savings we need.

Equipped with this fact, we split our analysis into the two following scenarios.

1. The first is where for each case where we need to encode the location of a'' from a' at a distance d, there are at most $\approx u^2$ salts – here we simply encode the index of the "right" a'' using $\approx \log u^2$ bits.

2. The other scenario is when for at least one case, there are more than $\approx u^2$ salts. Here we can save enough by encoding the t multi-collisions involving at least $\approx \log u^2$ that the graph-theoretic lemma guarantees us. We get enough savings by only encoding these t multi-collisions.

The u^2 term from the first scenario above turns into an additional factor of the form S^2 in the final bound. Due to additional technicalities that we glossed over during this overview, we suffer another multiplicative factor S in our bound, which amounts to having an S^4 term. Full details appear in Section 8.4.

Chapter Organization. We provide details of these results in the rest of this chapter. In Section 8.1, we introduce our notation, definitions, and well-known propositions that we use in the chapter. We present our modular framework for the analysis (adapted from [ACDW20]) in Section 8.2. In Section 8.3, we prove the STB conjecture for all constant B. Finally, in Section 8.4, we show how to extend this proof to prove the STB conjecture for $SB \ll T$.

8.1 Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ denote the set of all natural numbers and let $\mathbb{N}_{>0} = \mathbb{N} \backslash \{0\}$. For $N \in \mathbb{N}_{>0}$, let $[N] = \{1, 2, \ldots, N\}$ and for $k \in \mathbb{N}$ such that $k \leqslant N$, let $\binom{[N]}{k}$ denote the set of k-sized subsets of $[N]$. For a set X, let $|X|$ be its size and X^+ denote one or more elements of X. We denote $\mathsf{Fcs}(D, R)$ the set of all functions mapping elements in D to the elements of R. We let $x \leftarrow\!\!\$\, \mathcal{D}$ denote sampling x according to the distribution $\mathcal{D}$. We let $*$ denote a wildcard element. For example $(*, z) \in L$ is true if there is an ordered pair in L where z is the second element (the type of the wildcard element shall be clear from the context). If D is a set, we overload notation and let $x \leftarrow\!\!\$\, D$ denote uniformly sampling from the elements of D. For a bit-string s we use $|s|$ to denote the number of bits in s.

When referring to directed graphs in this , we mean directed multi-graphs, i.e., these directed graphs might have parallel edges. All logarithms in this chapter are for base 2 unless otherwise stated.

Merkle-Damgård (MD) hashing. For $N, M \in \mathbb{N}_{>0}$, let $h : [N] \times [M] \to [N]$ be a compression function. We recursively define Merkle-Damgård (MD) hashing $\mathsf{MD}_h : [N] \times [M]^+ \to [N]$ as $\mathsf{MD}_h(a, \alpha) = h(a, \alpha)$ for $a \in [N], \alpha \in [M]$ and

$$\mathsf{MD}_h(a, (\alpha_1, \ldots, \alpha_B)) = h(\mathsf{MD}_h(a, (\alpha_1, \ldots, \alpha_{B-1})), \alpha_B)$$

for $a \in [N]$ and $\alpha_1, \ldots, \alpha_B \in [M]$. The elements of $[M]$ shall be referred to as *blocks* and a, the first input to MD_h is referred to as the *salt*. This is the same abstraction of plain MD hashing as the one used in [ACDW20].

Auxiliary-input Random Oracle Model (AI-ROM). We use the Auxiliary-Input Random Oracle Model (AI-ROM) introduced by Unruh [Unr07] to study non-uniform adversaries in the Random Oracle Model. This model is parameterized by two non-negative integers S and T and an adversary $\mathcal{A}$ is divided into two stages $(\mathcal{A}_1, \mathcal{A}_2)$. Adversary $\mathcal{A}_1$, referred to as the preprocessing phase of $\mathcal{A}$, has unbounded access to the random oracle h and outputs an S-bit auxiliary input σ. Adversary $\mathcal{A}_2$, referred to as the online phase, gets σ as input and can make T queries to h, attempting to accomplish some goal involving the function h. Formally, we say that $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is an (S, T)-AI adversary if $\mathcal{A}_1$ outputs S bits and $\mathcal{A}_2$ issues T queries to its oracle. We next formalize the salted-collision resistance of MD hash functions in AI-ROM.

Game $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}(\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2))$	Subroutine $\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2))$
1. $h \xleftarrow{\$} \mathsf{Fcs}([N]\times[M],[N])$	1. $\sigma \xleftarrow{\$} \mathcal{A}_1(h)$
2. $a \xleftarrow{\$} [N]$	2. $(\alpha,\alpha') \xleftarrow{\$} \mathcal{A}_2^h(\sigma,a)$
3. Return $\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A})$	3. Return true if:
	(a) $\alpha \neq \alpha'$,
	(b) α,α' consist of $\leqslant B$ blocks from $[M]$, and
	(c) $\mathsf{MD}_h(a,\alpha) = \mathsf{MD}_h(a,\alpha')$
	4. Else, return false

Figure 8.3: The bounded-length collision resistance game of salted MD hash in the AI-ROM, denoted $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}$.

Salted short collision resistance of MD in AI-ROM. We formalize the hardness of bounded-length collision resistance of salted MD hash functions in the AI-ROM. The game is parametrized by N, M, and B. The game first samples a function h uniformly at random from $\mathsf{Fcs}([N]\times[M],[N])$ and a salt a uniformly at random from $[N]$. Then, $\mathcal{A}_1$ is given unbounded access to h, and it outputs σ. At this time, $\mathcal{A}_2$ is given the auxiliary input σ, a salt a, as well as oracle access to h, and it needs to find $\alpha \neq \alpha'$ such that (1) $|\alpha|, |\alpha'| \leqslant B \cdot M$, and (2) $\mathsf{MD}_h(a,\alpha) = \mathsf{MD}_h(a,\alpha')$. This game, denoted $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}$, is explicitly written in Fig. 8.3. In Fig. 8.3, we write the adversary's execution in its own subroutine only for syntactical purposes (as we shall use it later in our proof).

Definition 8.1.1 (AI-CR Advantage)**.** *For parameters $N, M, B \in \mathbb{N}$, the advantage of an adversary $\mathcal{A}$ against the bounded-length collision resistance of salted MD in the AI-ROM is*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{MD},N,M,B}(\mathcal{A}) = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}(\mathcal{A}) = \mathsf{true}\right]$$

For parameters $S, T \in \mathbb{N}$, we overload notation and denote

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{MD},N,M,B}(S,T) = \max_{\mathcal{A}} \left\{\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{MD},N,M,B}(\mathcal{A})\right\},$$

where the maximum is over all (S,T)-AI adversaries.

The compression lemma. Our proof uses the well-known technique of finding an "impossible compression". The main idea, formalized in the following proposition, is that it is impossible to compress a random element in set $\mathcal{X}$ to a string shorter than $\log|\mathcal{X}|$ bits long, even relative to a random string.

Proposition 8.1.2 (E.g., [DTT10])**.** *Let* Encode *be a randomized map from* $\mathcal{X}$ *to* $\mathcal{Y}$ *and let* Decode *be a randomized map from* $\mathcal{Y}$ *to* $\mathcal{X}$ *such that*

$$\mathsf{Pr}_{x \leftarrow\!\$\, \mathcal{X}}\left[\mathsf{Decode}(\mathsf{Encode}(x)) = x\right] \geqslant \varepsilon.$$

Then, $\log|\mathcal{Y}| \geqslant \log|\mathcal{X}| - \log(1/\varepsilon)$.

Constructive Chernoff bound. We use the following "constructive" Chernoff bound taken from Akshima et al. [ACDW20] which in turn is a variant of a similar lemma of Impagliazzo and Kabanets [IK10]. We refer to [ACDW20, Section 9] for a proof of this proposition.

Proposition 8.1.3 ([ACDW20, Theorem 1])**.** *Let* $N \in \mathbb{N}_{>0}$, $u \in \mathbb{N}$ *such that* $u \leqslant N$, $0 < \delta < 1$ *and let* $X_1, \ldots, X_N$ *be 0/1 random variables. Let* $X = X_1 + \cdots + X_N$ *and let* U *be an independent random subset of* $[N]$ *of size* u.

$$\textit{If} \quad \mathsf{Pr}\left[\bigwedge_{i \in U} X_i\right] \leqslant \delta^u, \quad \textit{then} \quad \mathsf{Pr}\left[X \geqslant \max\{6\delta N, u\}\right] \leqslant 2^{-u}.$$

8.2 The Framework: Reducing the Problem to a Multi-instance Collision Finder

Our task here is to upper-bound the advantage of an adversary in finding a short collision in a salted MD, according to the game $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}$ described in Figure 9.3. First, without loss of generality, in what follows, we assume that the adversary is deterministic. This follows since we can transform any probabilistic attacker into a deterministic one by hard-wiring the best randomness (see Adleman [Adl78]).

We reduce the task of bounding the advantage of an attacker in finding a short collision in a salted MD, according to the game $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{N,M,B}$, to a "multi-instance" game where the adversary does not have a preprocessing phase but instead only has a non-uniform auxiliary input, chosen *before* the random oracle h. The latter game is easier to analyze. Although the statement and reduction below were implicit in the work of Akshima et al. [ACDW20], we make it formal and hopefully useful for future works.

We define the following "multi-instance" game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{N,M,B,u}(\sigma, \mathcal{A}_2)$, where the preprocessing part of the adversary $\mathcal{A}_1$ is degenerate and outputs the fixed string σ. More precisely, the game has the following steps:

1. $h \leftarrow\!\$\, \mathsf{Fcs}([N] \times [M], [N])$
2. $U \leftarrow\!\$\, \binom{[N]}{u}$
3. Define $\mathcal{A}_1$ to be the algorithm that always outputs the string σ.
4. Return true if $\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)) = \mathsf{true}$ for every $a \in U$. Otherwise, return false.

For a string σ and an adversary $\mathcal{A}_2$, define

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\sigma,\mathcal{A}_2) = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{mi\text{-}cr}}_{N,M,B,u}(\sigma,\mathcal{A}_2)\right].$$

Lemma 8.2.1. *Fix $N, M, B, S, T, u \in \mathbb{N}_{>0}$. Then,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{MD},N,M,B}(S,T) \leqslant 6 \cdot \left(\max_{\sigma,\mathcal{A}_2}\left\{\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\sigma,\mathcal{A}_2)\right\}\right)^{\frac{1}{u}} + 2^{S-u},$$

where the maximum is taken over all $\sigma \in \{0,1\}^S$ and T-query algorithms $\mathcal{A}_2$.

Proof. Define

$$\gamma := \left(\max_{\sigma,\mathcal{A}_2}\left\{\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\sigma,\mathcal{A}_2)\right\}\right)^{\frac{1}{u}}.$$

We need to upper bound $\mathsf{Pr}_{h,a}\left[\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A}) = \mathsf{true}\right]$ for every (S,T)-AI adversary $\mathcal{A}$. We say that h is *easy* for adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ if $\mathcal{A}$ wins the $\mathsf{AI\text{-}CR}_{h,a}$ game on a random $a \xleftarrow{\$} [N]$ with probability 6γ. That is,

$$h \text{ is easy for } \mathcal{A} \iff \mathsf{Pr}_{a \xleftarrow{\$} [N]}\left[\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A}) = \mathsf{true}\right] \geqslant 6\gamma.$$

By the law of total probability,

$$\begin{aligned}
\mathsf{Pr}_{h,a}\left[\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A}) = \mathsf{true}\right] =& \mathsf{Pr}_{h,a}\left[\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A}) = \mathsf{true} \mid h \text{ is easy for } \mathcal{A}\right] \cdot \mathsf{Pr}_h[h \text{ is easy for } \mathcal{A}] + \\
& \mathsf{Pr}_{h,a}\left[\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A}) = \mathsf{true} \mid h \text{ is not easy for } \mathcal{A}\right] \cdot \mathsf{Pr}_h[h \text{ is not easy for } \mathcal{A}] \\
\leqslant& \mathsf{Pr}_h[h \text{ is easy for } \mathcal{A}] + \mathsf{Pr}_{h,a}\left[\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A}) = \mathsf{true} \mid h \text{ is not easy for } \mathcal{A}\right] \\
\leqslant& \mathsf{Pr}_h[h \text{ is easy for } \mathcal{A}] + 6\gamma.
\end{aligned}$$

The lemma readily follows from the following claim where we prove an upper bound on $\mathsf{Pr}_h[h \text{ is easy for } \mathcal{A}]$.

Claim 8.2.2. *For any (S,T)-AI adversary $\mathcal{A}$, it holds that $\mathsf{Pr}_h[h \text{ is easy for } \mathcal{A}] \leqslant 2^{S-u}$.*

Proof. Fix an adversary as in the statement $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. Define $\mathcal{A}_\sigma$ the canonical preprocessing adversary that outputs the string $\sigma \in \{0,1\}^S$, ignoring its input. Denote $\mathcal{A}_\sigma$ the (S,T)-AI adversary $(\mathcal{A}_\sigma, \mathcal{A}_2)$. By definition and by a union bound, it holds that

$$\begin{aligned}
\mathsf{Pr}_h[h \text{ is easy for } \mathcal{A}] &\leqslant \mathsf{Pr}_h\left[\exists \sigma \in \{0,1\}^S \colon h \text{ is easy for } \mathcal{A}_\sigma\right] \\
&\leqslant 2^S \cdot \max_{\sigma \in \{0,1\}^S} \mathsf{Pr}_h\left[h \text{ is easy for } \mathcal{A}_\sigma\right]. \qquad (8.1)
\end{aligned}$$

For a fixed $a \in [N]$, $\sigma \in \{0,1\}^S$, and $h \in \mathsf{Fcs}([N] \times [M], [N])$, define the indicator $X^{\sigma}_{h,a} = 1$ if and only if $\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A}_\sigma) = \mathsf{true}$. By definition and with this notation, for every $\sigma \in \{0,1\}^S$, it holds that

$$\begin{aligned}\mathsf{Pr}_h\left[h \text{ is easy for } \mathcal{A}_\sigma\right] &= \mathsf{Pr}_h\left[\mathsf{Pr}_{a \xleftarrow{\$} [N]}\left[\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A}_\sigma) = \mathsf{true}\right] \geqslant 6\gamma\right] \\ &= \mathsf{Pr}_h\left[\mathsf{Pr}_{a \xleftarrow{\$} [N]}\left[X^{\sigma}_{h,a}\right] \geqslant 6\gamma\right] \\ &= \mathsf{Pr}_h\left[\sum_{a \in [N]} X^{\sigma}_{h,a} \geqslant 6N\gamma\right]. \end{aligned} \tag{8.2}$$

By definition, $\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\sigma, \mathcal{A}_2)$ is the advantage of the T-query attacker $\mathcal{A}_2$ given auxiliary input σ in finding collisions in salted MD relative to h and on u uniformly random salts denoted U. Written differently, for every $\sigma \in \{0,1\}^S$ and T query adversary $\mathcal{A}_2$, it holds that

$$\begin{aligned}\mathsf{Pr}_{h,U}\left[\bigwedge_{a \in U} X^{\sigma}_{h,a}\right] &= \mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\sigma, \mathcal{A}_2) \\ &\leqslant \max_{\sigma^*, \mathcal{A}_2^*}\left\{\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\sigma^*, \mathcal{A}_2^*)\right\} = \gamma^u, \end{aligned}$$

where the last equality follows from the definition of γ. Applying Proposition 10.1.1, we get that for every $\sigma \in \{0,1\}^S$, it holds that

$$\mathsf{Pr}_h\left[\sum_{a \in [N]} X^{\sigma}_{h,a} \geqslant 6N\gamma\right] \leqslant 2^{-u}.$$

Plugging this back into (8.2), we get that $\mathsf{Pr}_h\left[h \text{ is easy for } \mathcal{A}_\sigma\right] \leqslant 2^{-u}$. In turn, plugging the latter back into (8.1), we get that $\mathsf{Pr}_{h,a}\left[\mathsf{AI\text{-}CR}_{h,a}(\mathcal{A}) = \mathsf{true}\right] \leqslant 2^{S-u}$, as needed. ■

■

8.3 Proving the STB Conjecture for $B \in O(1)$

This section proves an upper bound on the advantage of any auxiliary-input adversary in the bounded-length collision resistance game of salted MD hash in the AI-ROM. The main theorem is stated next.

Theorem 8.3.1. *Let $C = 2^{16} \cdot 6 \cdot e^2$. For any $N, M, B, S, T \in \mathbb{N}_{>0}$ and fixing $\hat{S} := S + \log N$, it holds that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{MD},N,M,B}(S,T) \leqslant C \cdot \max\left\{\left(\frac{\hat{S}TB^2\left(\frac{3e\log\hat{S}}{\log\log\hat{S}}\right)^{2(B-2)}}{N}\right), \left(\frac{T^2}{N}\right)\right\} + \frac{1}{N}.$$

Theorem 9.3.5 follows as a direct corollary of Lemma 10.4.1 together with the following lemma and setting $u = S + \log N$.

Lemma 8.3.2 (Hardness for a multi-instance collision finder)**.** *Fix N, M, B, S, T, $u \in \mathbb{N}_{>0}$ and $\sigma \in \{0,1\}^S$. Then, for any $\mathcal{A}_2$ that makes at most T queries to its oracle, it holds that*

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\sigma,\mathcal{A}_2) \leqslant \left(2^{16}e^2 \cdot \max\left\{\left(\frac{uTB^2(3e\log u/\log\log u)^{2(B-2)}}{N}\right),\left(\frac{T^2}{N}\right)\right\}\right)^u.$$

The rest of this section is devoted to the proof of Lemma 9.3.6. Unlike the proof of Lemma 10.4.1, the proof of this lemma is novel and differs completely from that of Akshima et al. [ACDW20]. The key conceptual insight is a structural characterization of collisions in MD_h that prevents the explosion in the number of cases that [ACDW20] faced during the case analysis.

We are interested in bounding the advantage of the best strategy, i.e., a pair $(\sigma,\mathcal{A}_2)$ where $\sigma \in \{0,1\}^S$ is a fixed string and $\mathcal{A}_2$ is a T-query algorithm, of finding bounded-length collisions in a salted MD with respect to the game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{N,M,B,u}(\sigma,\mathcal{A}_2)$. Recall that in this game, $\mathcal{A}_2$ needs to find proper collisions for u randomly chosen salts, denoted U. The main idea in the proof is to use any such adversary $(\sigma,\mathcal{A}_2)$ to represent the function h *as well as* the set of random salts U with as few bits as possible. If the adversary is "too good to be true," we will get an impossible representation, contradicting Proposition 11.1.1.

Non-trivial range. If either

$$\frac{T^2}{N} > 1 \quad \text{or} \quad \frac{uTB^2(3e\log u/\log\log u)^{2(B-2)}}{N} > 1,$$

then Lemma 9.3.6 is trivially true. Hence, from now on we assume that both of the above left hand side terms are upper bounded by 1.

Setup. Denote

$$\zeta^* := \left(2^{16}e^2 \cdot \max\left\{\left(\frac{uTB^2(3e\log u/\log\log u)^{2(B-2)}}{N}\right),\left(\frac{T^2}{N}\right)\right\}\right)^u.$$

Assume the existence of an adversary $\mathcal{A} = (\sigma,\mathcal{A}_2)$, where $\sigma \in \{0,1\}^S$ is a string and $\mathcal{A}_2$ is a T-query adversary, that contradict the inequality stated in the lemma. That is, there is $\zeta > \zeta^*$ such that

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\mathcal{A}) := \zeta > \zeta^*. \tag{8.3}$$

Define $\mathcal{G}$ to be the set of functions-sets of salts pairs for which the attacker succeeds in winning the game for every salt in the set relative to the function, That is,

$$\mathcal{G} = \left\{(U,h) \;\middle|\; \begin{array}{c} U \in \binom{[N]}{u}, \\ h \in \mathsf{Fcs}([N]\times[M],[N]), \end{array} \forall a \in U : \mathsf{AI\text{-}CR}_{h,a}(\mathcal{A}) = \mathsf{true}\right\}.$$

Recall that ζ is defined to be the advantage of $\mathcal{A}$ in the game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{N,M,B,u}(\mathcal{A})$ in which h and U are chosen uniformly, and then $\mathcal{A}$ needs to find a collision with respect to every one of the u salts in U. Therefore,

$$|\mathcal{G}| = \zeta \cdot \binom{N}{u} \cdot N^{MN}.$$

In what follows we define an encoding and a decoding procedure such that the encoding procedure gets as input U, h such that $U \in \binom{[N]}{u}$ and $h \in \mathsf{Fcs}([N] \times [M], [N])$, and it outputs an L bit string, where $L = \log\left(\zeta^* \cdot \binom{N}{u} \cdot N^{MN}\right)$. The decoding procedure takes as input the string L and outputs U^*, h^*. It will hold that $U^* = U$ and $h^* = h$ with probability ζ.[8] Using Proposition 11.1.1, this would give us that

$$\log \zeta \leqslant L - \log\left(\binom{N}{u} \cdot N^{MN}\right) \Longrightarrow \zeta \leqslant \zeta^*$$

which is a contradiction to the assumption (see (9.3)).

Notation and Definitions. Fix $(U, h) \in \mathcal{G}$. Let $U = \{a_1, \ldots, a_u\}$ where the a_i's are ordered lexicographically. Let $\mathsf{Qrs}(a) \in ([N] \times [M])^T$ be the list of queries that $\mathcal{A}_2$ makes to h when executed with input (σ, a). Namely, for $a \in [N]$,

$$\mathsf{Qrs}(a) = \left\{(a', \alpha') \in [N] \times [M] \mid \mathcal{A}_2(\sigma, a) \text{ queries } h \text{ on } (a', \alpha')\right\} .$$

Note that $\mathsf{Qrs}(a)$ is indeed a set as we can assume (without loss of generality) that $\mathcal{A}_2$ never repeats queries in a single execution (since $\mathcal{A}_2$ can just store all of its past queries).

We say that $a' \in \mathsf{Slts}(a)$ if there is some $\alpha' \in [M]$ such that (a', α') is an entry in $\mathsf{Qrs}(a)$. Namely, for $a, a' \in [N]$,

$$a' \in \mathsf{Slts}(a) \Longleftrightarrow \exists \alpha' \in [M] \text{ s.t. } (a', \alpha') \in \mathsf{Qrs}(a).$$

We define the set of *fresh* salts in U. A salt a_i for $i \in [u]$ is called fresh if it was never used as the salt in any query performed by $\mathcal{A}_2$ while being executed on salts a_j for $j \leqslant i-1$ which are fresh. The first salt a_1 is always fresh. A salt a_i for $i \geqslant 2$ is fresh if for any fresh a_j for $j \leqslant i-1$, $a_i \notin \mathsf{Slts}(a_j)$. Namely, denoting the set of fresh salts by U_{fresh}, we have the following inductive (on $i \in [u]$) definition:

$$a_i \in U_{\mathsf{fresh}} \Longleftrightarrow \forall j \leqslant i-1, a_j \in U_{\mathsf{fresh}} \colon a_i \notin \mathsf{Slts}(a_j).$$

Looking ahead, we define U_{fresh} like this because we run $\mathcal{A}_2$ on the salts in U_{fresh} in lexicographical order, and this definition ensures that each salt that $\mathcal{A}_2$ is executed on was not queried by it previously. Denote

$$F := |U_{\mathsf{fresh}}| \quad \text{and} \quad U_{\mathsf{fresh}} = \{a'_1, \ldots, a'_F\} \text{ (ordered lexicographically).}$$

[8] Essentially, we will show that for all $(U, h) \in |\mathcal{G}|$, if the encoding procedure produces output L, then the decoding procedure on input L outputs U^*, h^* such that $U^* = U$ and $h^* = h$.

Denote

$$\forall i \in [F] \colon \mathsf{Q}_i := \mathsf{Qrs}(a'_i) \quad \text{and} \quad \mathsf{Q}_{\mathsf{fresh}} := \mathsf{Q}_1 \parallel \ldots \parallel \mathsf{Q}_F,$$

where $\parallel$ is the concatenation operator. Let $\mathsf{Q}_{\mathsf{fresh}}[r]$ be the rth query in the list $\mathsf{Q}_{\mathsf{fresh}}$. Note that $r \in [F \cdot T]$. For every $a \in U \backslash U_{\mathsf{fresh}}$, let t_a be the minimum value such that $\mathsf{Q}_{\mathsf{fresh}}[t_a]$ is a query with salt a. Define the set of *prediction* queries as

$$\mathsf{P} := \{t_a \mid a \in U \backslash U_{\mathsf{fresh}}\}.$$

The encoding algorithm will output $U_{\mathsf{fresh}}, \mathsf{P}$, which suffices to recover the set U by running $\mathcal{A}_2$.

We let $\tilde{h}$ be the list of $h(a, \alpha)$ values when executed on distinct queries in $\mathsf{Q}_{\mathsf{fresh}}$, in the same order as they appear in $\mathsf{Q}_{\mathsf{fresh}}$, followed by the evaluation of h on the following values in lexicographical order of the inputs.

$$\{(a, \alpha) : a \in [N], \alpha \in [M]\} \backslash \mathsf{Q}_{\mathsf{fresh}} \ .$$

Therefore, $\tilde{h}$ is initialized to contain the evaluation of h at all points in its domain. Looking ahead, in the encoding procedure, we will remove elements from $\tilde{h}$ as needed to compress h.

Function and Query graphs. A notion that will be useful is that of a "function graph".

Definition 8.3.3 (Function graph)**.** *For a function $h : [N] \times [M] \to [N]$, consider the following directed graph: it has N nodes labelled with elements of $[N]$ and each node has exactly M outgoing edges, each labelled with elements of $[M]$. There is an edge from node a_i to a_j labelled α if and only if $h(a_i, \alpha) = a_j$.*

We define the notion of query graph for an adversary as follows.

Definition 8.3.4 (Query graph)**.** *Execution of an adversary $\mathcal{A}_2$ on salts $a'_1, \ldots, a'_F$ defines a query graph as follows. Initially the graph is empty. Whenever $\mathcal{A}_2$ queries (a, α) to h, we add a node with label a if not already present and add an edge $(a, h(a, \alpha))$ with label α if not already present.*

Fact 8.3.5. *The query graph is always a sub-graph of the function graph of h.*

Structure of collisions: The mouse structure. Since adversary $\mathcal{A}_2$ succeeds on all of the salts in U, it holds that for every $j \in [F]$, the output of the adversary is (α_j, α'_j) such that $\alpha_j \neq \alpha'_j$, $\mathsf{MD}_h(a_j, \alpha_j) = \mathsf{MD}_h(a_j, \alpha'_j)$ and both α_j, α'_j are at most B blocks long. We can assume without loss of generality that the colliding messages α_j and α'_j are "minimal" (because otherwise, we can trim α_j, α'_j to obtain a shorter collision). The evaluations of h in order to compute $\mathsf{MD}_h(a'_j, \alpha_j)$ and $\mathsf{MD}_h(a'_j, \alpha'_j)$ induce a structure that we call a *mouse structure* as shown in Fig. 8.4. More explicitly, suppose the output of the $\mathcal{A}_2$ is $(\alpha_j =$

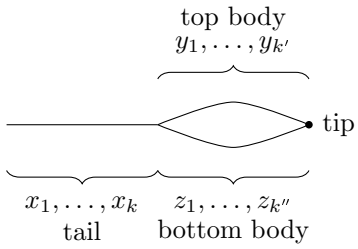


Figure 8.4: A mouse structure. For ease of visual representation we do not draw the nodes and edges of the graph, instead represent it as a continuous structure.

$(\alpha_{j,1}, \ldots, \alpha_{j,B_1}), \alpha'_j = (\alpha'_{j,1}, \ldots, \alpha'_{j,B_2}))$ for $B_1, B_2 \leqslant B$ such that $\alpha_{j,i} = \alpha'_{j,i}$ for all $1 \leqslant i \leqslant k$ where $k \geqslant 0$. Define $(x_1, \ldots, x_k, y_1, \ldots, y_{k'}, z_1, \ldots, z_{k''})$ where $k' = B_1 - k, k'' = B_2 - k$ as follows.

$$x_1 = h(a'_j, \alpha_{j,1}) \ , \ x_i = h(x_{i-1}, \alpha_{j,i}) \text{ for } 1 < i \leqslant k$$

$$y_1 = \begin{cases} h(a'_j, \alpha_{j,1}) & \text{if } k = 0 \\ h(x_k, \alpha_{j,k+1}) & \text{otherwise} \end{cases} \ , \ y_i = h(y_{i-1}, \alpha_{j,i+k}) \text{ for } 1 < i \leqslant B_1 - k$$

$$z_1 = \begin{cases} h(a'_j, \alpha'_{j,1}) & \text{if } k = 0 \\ h(x_k, \alpha'_{j,k+1}) & \text{otherwise} \end{cases} \ , \ z_i = h(z_{i-1}, \alpha'_{j,i+k}) \text{ for } 1 < i \leqslant B_2 - k$$

Then $(x_1, \ldots, x_k, y_1, \ldots, y_{k'}, z_1, \ldots, z_{k''})$ form a mouse structure as specified in Figure 8.4. Without loss of generality, we can assume that the mouse structure is present in the query graph of $\mathcal{A}_2$ before it outputs the answer for salt a'_j. We refer to this structure as the mouse structure for salt a'_j.

We define $\mathsf{MouseQrs}_j$ to be the set of queries in the mouse structure. Similarly, we define $\mathsf{MouseSlts}_j$ as the set of salts that comprise the mouse structure. By definition, $1 \leqslant |\mathsf{MouseQrs}_j|, |\mathsf{MouseSlts}_j| \leqslant 2B$.

Classifying queries. We classify every one of the queries in $\mathsf{Q}_{\mathsf{fresh}}$ into one of 3 types by scanning through them in order. Recall that $\mathsf{Q}_{\mathsf{fresh}}$ consists of F blocks, each consisting of T queries. Each block contains a mouse structure as in Figure 8.4. The first type of query is called new. A new query did not appear in any previous block. Non-new queries are called *repeated* and they are classified further into one of 2 types: repeatedUsed, and repeatedUnused. A query (a, α) would be a repeatedUsed query if it was made as part of a mouse structure during some earlier salt in U_{fresh}. Lastly, a repeatedUnused query is one that was made before but is not part of any mouse structure. That is,

1. NEW: A query with index $r \in [F \cdot T]$ is NEW if there does not exist $r' < r$ such that $\mathsf{Q}_{\mathsf{fresh}}[r'] = \mathsf{Q}_{\mathsf{fresh}}[r]$.

2. A non-NEW queries is called *repeated*. We classify the latter into two subcategories:

   (a) REPEATEDUSED: A query in Q_j with index $s \in [T]$ such that $h(\mathsf{Q}_j[s]) = a$ is REPEATEDUSED if it is not NEW and $\mathsf{Q}_j[s] \in \mathsf{MouseQrs}_i$ for some $i < j$.

   (b) REPEATEDUNUSED: A query in Q_j with index $s \in [T]$ such that $h(\mathsf{Q}_j[s]) = a$ is REPEATEDUNUSED if it is not NEW, $\mathsf{Q}_j[s] \notin \mathsf{MouseQrs}_i$ for all $i < j$.

Note that this classification covers all queries made during execution. The following is a simple observation.

Claim 8.3.6. *Every mouse structure has at least one* NEW *query.*

Proof. For every $j \in [F]$, the queries in Q_j with salt a'_j are necessarily NEW because we defined U_{fresh} to contain a'_j's that were not queried earlier by $\mathcal{A}_2$ when run on a'_i for $i < j$, and also assumed that $\mathcal{A}_2$ does not repeat queries during a single execution. ■

8.3.1 The Compression Argument

As mentioned, our goal is to compress (U, h), and we will achieve this by using our collision finding adversary $\mathcal{A}_2$. The encoding procedure shall output the set U_{fresh}, the set P, the list $\tilde{h}$ with some entries removed and additional lists and sets. We will be describing the details of these lists and sets below and which entries we remove from $\tilde{h}$. It is instructive to think about how decoding would work to understand the intuition behind our encoding strategy. The decoding procedure will run $\mathcal{A}_2$ on the salts in U_{fresh} and answer its queries using the entries in $\tilde{h}$ and the other lists and sets it gets as input. We need to encode enough information in these lists and sets so that the decoding procedure can correctly answer the queries whose answers will be removed from $\tilde{h}$. The salts in $U \backslash U_{\mathsf{fresh}}$ will be recovered using P. Our main goal is to show that we are compressing when we remove entries of $\tilde{h}$ and instead use additional lists and set. Our ways to compress will depend on the induced mouse structure in each Q_j for $j \in [F]$. To this end, we first classify the mouse structures into six broad cases. We classify the jth mouse structure for each $j \in [F]$ into the first of the following six cases it satisfies, e.g., if a mouse structure satisfies both cases 2 and 3, we categorize it into 2.

1. There is a NEW query (a, α) such that $h(a, \alpha) = a$.

2. There are two distinct NEW queries (a_1, α_1), (a_2, α_2) such that $h(a_1, \alpha_1) = h(a_2, \alpha_2)$.

3. There is a NEW query (a, α) such that $h(a, \alpha) = a'$ and $a' \in \mathsf{MouseSIts}_i$ for some $i < j$.

4. There is a REPEATEDUNUSED query (a, α) such that $h(a, \alpha) = a$.

5. There is at least one salt a such that $a \in \mathsf{MouseSlts}_i$ for some $i < j$ and there is a path of at most $B-2$ edges in the mouse structure from a back to a' where a' is an answer to a NEW query.

6. There are no REPEATEDUSED queries in the mouse structure.

Note that these cases cover all mouse structures.

Claim 8.3.7. *Every mouse structure can be categorized into one of the cases 1 to 6.*

Proof. We will show that if a mouse structure does not satisfy case 6, it has to satisfy case 5, which suffices to prove our claim. Since the mouse structure is not in 6, it has a REPEATEDUSED query. Let the input salt of this query be a. Moreover, since the first query of the mouse structure has to be new, let the answer salt of this query be a'. Since the longest path in the mouse structure is of length B, it follows that there are at most $B-2$ edges in the mouse structure between a and a'. Hence, case 5 is satisfied. ■

COMPRESSION BUDGET. Recall that we need to prove that the size of the output of the encoding procedure is

$$L = \log\left(\left(2^{16}e^2 \cdot \max\left\{\left(\frac{uTB^2m_0^{2(B-2)}}{N}\right), \left(\frac{T^2}{N}\right)\right\}\right)^u \cdot \binom{N}{u} \cdot N^{MN}\right)$$

bits, where $m_0 = 3e \log u / \log\log u$. In other words, we need to show that the encoding procedure saves at least

$$u \cdot \log\left(\min\left\{\frac{N}{4T^2}, \frac{N}{4uTB^2m_0^{2(B-2)}}\right\}\right) - 2u\log e - 14u \tag{8.4}$$

bits overall.

Required savings in $\tilde{h}$. As mentioned earlier, the output of the encoding algorithm will consist of $U_{\mathsf{fresh}}, \mathsf{P}, \tilde{h}$, and some additional sets and lists. The lists U_{fresh} and P will suffice to recover the set U. The list $\tilde{h}$ and the additional sets and lists are used to recover h.

Denoting $|U_{\mathsf{fresh}}| = F$ and $|U| = u$, we can describe P using $\binom{FT}{u-F}$ bits. Therefore, U, which is trivially described using $\log\binom{N}{u}$ bits, can be encoded using the following number of bits

$$\log\left(\binom{FT}{u-F}\binom{N}{F}\right).$$

Therefore, the saving in bits in the description of U is at least

$$\begin{aligned}
&\log\binom{N}{u} - \log\left(\binom{FT}{u-F}\binom{N}{F}\right) \geqslant \log\left(\frac{\left(\frac{N}{u}\right)^u}{\left(\frac{eFT}{u-F}\right)^{u-F}\left(\frac{eN}{F}\right)^F}\right) \\
&= \log\left(\frac{N^{u-F}(u-F)^{u-F}F^F}{e^u(FT)^{u-F}u^u}\right) = (u-F)\log\left(\frac{N}{FT}\right) + \log\left(\frac{(u-F)^{u-F}F^F}{e^u u^u}\right) \\
&= (u-F)\log\left(\frac{N}{FT}\right) - \log\left(e^u\left(\frac{u}{F}\right)^F\left(\frac{u}{u-F}\right)^{u-F}\right) \\
&\geqslant (u-F)\log\left(\frac{N}{uT}\right) - u\log 4e\,. \qquad (8.5)
\end{aligned}$$

where the first inequality uses the basic bounds for binomial coefficients $(n/r)^r \leqslant \binom{n}{r} \leqslant (en/r)^r$, and the last inequality follows since $x \leqslant 2^x$ for every $x \geqslant 0$, and $u \geqslant F$.

By subtracting (9.5) (how much we save in U) from (9.4) (how much we need to save in total), it suffices to show that we save at least

$$\begin{aligned}
u \cdot \log\left(\min\left\{\frac{N}{4T^2}, \frac{N}{4uTB^2m_0^{2(B-2)}}\right\}\right) - 2u\log e \\
- 14u - (u-F)\log\left(\frac{N}{uT}\right) + u\log 4e
\end{aligned}$$

bits while encoding h. Since $\log(N/uT) \geqslant \log\left(\min\left\{N/4T^2, N/4uTB^2m_0^{2(B-2)}\right\}\right)$, this is at most the following number of bits.

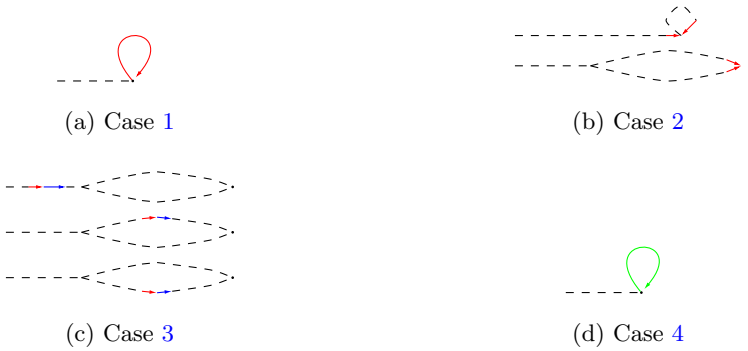
$$F \cdot \log\left(\min\left\{\frac{N}{4T^2}, \frac{N}{4uTB^2m_0^{2(B-2)}}\right\}\right) - u\log e - 12u \qquad (8.6)$$

To show that the compression indeed achieves the savings from (9.7), we will show that for every salt in U_{fresh}, we can save at least the following number of bits, except for a few cases.

$$\log\left(\min\left\{N/4T^2, N/4uTB^2m_0^{2(B-2)}\right\}\right)\,.$$

In the rare cases where we cannot save as much, we will incur a small penalty. We will show that the cumulative penalty we incur is at most $7u + u\log e$ bits. Additionally, we will label each of the salts in F with a few bits that describe its "type" (according to the cases described above – case 5 will have 3 subcategories, and case 6 will have 10 subcategories), and for this 5 bits will suffice. This will cost, in total, another $5u$ bits, and therefore the total size of the encoding will indeed be bounded by the term from (9.7).

We now describe the details of how we handle each case. Assuming that the mouse structure for salt a'_j satisfies a particular case, we describe the encoding procedure, calculate the amount of compression we get, and then explain how decoding would work. In Section 8.3.2 we handle Cases 1 to 4, in Section 8.3.3,



(a) Case 1 (b) Case 2

(c) Case 3 (d) Case 4

Figure 8.5: Cases 1 to 4. The NEW queries are drawn in red, the REPEATEDUSED queries are drawn in blue, and the REPEATEDUNUSED queries are drawn in green. The black dashed lines indicate zero or more queries of any type.

we handle case 5, and lastly in Section 8.3.4, we handle case 6. We note that here we describe the details such that they are locally verifiable, i.e., we describe the encoding, show that we save enough and describe the decoding for each case locally. We made this choice due to the complexity of the encoding and decoding procedures- we felt it was much easier to parse in the locally verifiable form. For sake of completeness, we do provide the full pseudocode of encoding and decoding in Section 8.3.5.

8.3.2 Handling Cases 1 to 4

In each of the four cases below, we will be saving more bits than we need, i.e., more bits than

$$\log\left(\min\left\{N/4T^2, N/4uTB^2m_0^{2(B-2)}\right\}\right) .$$

CASE 1. The jth mouse structure contains a NEW query (a,α) such that $h(a,\alpha)=a$, as depicted in Fig. 8.5a. The encoding procedure stores the index of the query (a,α) in Q_j in a list L_1 and removes the entry $h(a,\alpha)$ from $\tilde{h}$.

In decoding, if the current (jth) salt is categorized as case 1, then it removes the front index in the list L_1, and denote the index by i. It answers the ith h query, denoted (a,α), with a and sets $h(a,\alpha)=a$.

Since the index of the query (a,α) in Q_j is in $[T]$, and we remove one element of $\tilde{h}$, we save $\log(N/T)$ which is more than what we need to save.

CASE 2. The jth mouse structure contains two distinct NEW queries (a_1,α_1), (a_2,α_2) such that $h(a_1,\alpha_1)=h(a_2,\alpha_2)$, and (a_1,α_1) is queried before (a_2,α_2). This is depicted in Fig. 8.5b. The encoding procedure

stores the pair indices of the queries (a_1, α_1), (a_2, α_2) in Q_j in a list L_2 and removes the entry $h(a_2, \alpha_2)$ from $\tilde{h}$.

In decoding, if the current (jth) salt is categorized as case 2, then it removes the front element (i_1, i_2) in the list L_2. Suppose that the i_1th h query while running on salt a'_j is on (a_1, α_1). The decoding procedure gets the answer to this query from $\tilde{h}$. It answers the i_2th h query on (a_2, α_2) with $h(a_1, \alpha_1)$ and sets $h(a_2, \alpha_2) = h(a_1, \alpha_1)$.

Since the pair of indices of the queries $(a_1, \alpha_1), (a_2, \alpha_2)$ in Q_j are in $[T]$, and we remove one element of $\tilde{h}$, we save $\log(N/T^2)$ bits which is more than what we need to save.

Case 3. The jth mouse structure contains a NEW query (a, α) such that $h(a, \alpha) = a'$ and $a' \in \mathsf{MouseSlts}_i$ for some $i < j$. This is depicted in Fig. 8.5c. The encoding procedure stores the tuple consisting of i, the index of the query (a, α) in Q_j, and the lexicographical order of a' in $\mathsf{MouseSlts}_i$ in a list L_3 and removes the entry $h(a, \alpha)$ from $\tilde{h}$.

In decoding, if the current (jth) salt is categorized as case 3, it removes the front element (i_1, i_2, i_3) in the list L_3. Suppose the i_2th h query while running on salt a'_j is on (a, α). It answers the query with a' such that a' is the salt in $\mathsf{MouseSlts}_{i_1}$ whose lexicographical order is i_3. It sets $h(a, \alpha) = a'$.

Since $i \in [F]$, $F \leqslant u$, the index of (a, α) in Q_j is in $[T]$, the lexicographical index of a' in $\mathsf{MouseSlts}_i$ is in $[2B]$, and we remove one element of $\tilde{h}$, we save $\log(N/(2uTB))$ bits which is more than what we need to save.

Case 4. The jth mouse structure contains a REPEATEDUNUSED query (a, α) such that $h(a, \alpha) = a$. This is depicted in Fig. 8.5d. The encoding procedure stores the smallest index of the query (a, α) in $\mathsf{Q}_{\mathsf{fresh}}$ in a set S and removes the entry $h(a, \alpha)$ from $\tilde{h}$. Note that we never add an index corresponding to the same query multiple times to S. Indeed, if an index associated with (a, α) already appears in S, the next query on (a, α) will be a REPEATEDUSED query and not a REPEATEDUNUSED one, meaning that it cannot be added to S again.

In decoding, if the current (jth) salt is categorized as case 4, it checks for every h query on (a, α) whether S contains the index of the query in $\mathsf{Q}_{\mathsf{fresh}}$. If so it answers with a and sets $h(a, \alpha) = a$.

Since the smallest index of the query (a, α) in $\mathsf{Q}_{\mathsf{fresh}}$ is in $[FT]$, and we remove one element of $\tilde{h}$, we save at least $\log(N/(uT))$ which is more than necessary.

8.3.3 Handling Case 5

In this section, we describe our compression strategy in case the jth mouse structure for salt a'_j is categorized as case 5. This is depicted in Fig. 8.6. That is, there is at least one salt d such that $d \in \mathsf{MouseSlts}_i$ for some $i < j$ and there are at most $B - 2$ edges in the mouse structure between d and s where s is an answer to

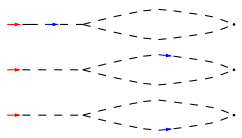


Figure 8.6: Mouse structure for case 5

a NEW query. If there are several possible candidate pairs, we choose one where the number of edges is the smallest between the source and the destination salt.

Intuition. We refer to the salt s in the answer to the NEW query as the *source* salt, and the salt that appears in some earlier mouse structure as the *destination*. We are guaranteed that the path in the mouse structure from the source salt to the destination salt consists of at most $B-2$ edges that are REPEATEDUNUSED queries. (There is at least one intermediate edge between the source and the destination because otherwise, the answer of a new query will be the input of a REPEATEDUSED query, in which case this scenario would have been classified into case 3. Additionally, all the intermediate edges must be REPEATEDUNUSED since we consider the shortest possible path from such source to such destination.)

Let the NEW query, whose answer is s, be (a, α). Suppose the path from the s to the d in the mouse structure consists of REPEATEDUNUSED queries $(a_1, \alpha_1), \ldots, (a_p, \alpha_p)$ where $a_1 = s$, $p \leqslant B-2$. The main idea is to avoid encoding $s = h(a, \alpha)$ and recover it by encoding the lexicographical order of d in $\mathsf{MouseSlts}_i$ and encoding the path required to backtrack from d to s in the query graph at the time (a, α) is queried, i.e., encoding which of the past queries were $(a_p, \alpha_p), \ldots, (a_1, \alpha_1)$ (in this order). The problem is that, in general, the path back from d to s might be too expensive to encode. This depends on the number of "other" edges incident on the nodes on the real path back from t to s. If all nodes on the path have very few edges incident on them, say less than m_0 of them, we encode each back edge using $\log m_0$ bits per node, which requires with at most $\log(B(m_0)^{B-2})$ bits for the whole path back (the term B comes from encoding the length of the path). But, if some node has many adjacent edges, namely a *multi-collision* with more than m_0 edges, we will need to take advantage of this fact to obtain our savings (and not encode the path back from d to s).

Definition 8.3.8 (Large multi-collision)**.** *We say that queries $q_1, \cdots, q_m$ form an m-way multi-collision if all the q_i's are distinct and all the $h(q_i)$ are equal. We say that the multi-collision is* large *if $m \geqslant m_0$, where $m_0 := 3e \log u / \log\log u$.*

If any of the queries in $(a_p, \alpha_p), \ldots, (a_1, \alpha_1)$ are involved in a large multi-collision of REPEATEDUNUSED queries in the query graph so far, we say we have "encountered a large multi-collision". In what follows, we explain how to obtain the required compression if a large multi-collision was *not* encountered.

ENCODING WHEN NO LARGE MULTI-COLLISION. Suppose that the NEW query whose answer is the source salt s is (a, α), the destination salt is d and $d \in \mathsf{MouseSlts}_i$ for some $i < j$. The path back from d to s contains only nodes that have at most m_0 adjacent edges in the corresponding query graph since we have not encountered a large multi-collision. The encoding procedures constructs a tuple consisting of:

- the index i,
- the index of query (a, α) in Q_j,
- the lexicographical index of d in $\mathsf{MouseSlts}_i$, and
- the path back from d to s in the query graph when (a, α) is queried.

It stores the tuple in a list L_5. Finally, it removes the entry $h(a, \alpha)$ from $\tilde{h}$.

In decoding, if the current (jth) salt is categorized as case 5 *without a large multi-collision*, it detects the query (a, α) from its index in Q_j, then finds the salt d using the index i and the lexicographical order of d in $\mathsf{MouseSlts}_i$, and finally finds s using the path back from d to s. It answers the query with s.

Since $i \in [F]$, the index of (a, α) in Q_j is in $[T]$, the lexicographical index of a' in $\mathsf{MouseSlts}_i$ is in $[2B]$, the path back from d to s can be encoded in $\log(B(m_0)^{B-2})$ bits, and we remove one element of $\tilde{h}$, we save at least $\log(N/(2uTB^2(m_0)^{B-2}))$ bits which is more than necessary.

ENCODING WITH LARGE MULTI-COLLISION. Suppose that the NEW query whose answer is the source salt s is (a, α), the destination salt is d and $d \in \mathsf{MouseSlts}_i$ for some $i < j$. Suppose further that the path back from d to s contains at least one node that has $m \geqslant m_0$ adjacent edges in the corresponding query graph (i.e., an m-multi-collision) such that these m edges are REPEATEDUNUSED queries when running $\mathcal{A}_2$ on a_j. First, observe that the multi-collision does not involve a self-loop. Indeed, if any node in the mouse structure has a REPEATEDUNUSED query whose answer is itself, the mouse structure would be classified into case 4, and therefore we will never reach this case.

At this point, we argue that we can record the multi-collision by encoding the indices of all queries associated with the multi-collision and the center, and remove their answers from $\tilde{h}$. To this end, we store

$\log N + \log \binom{FT}{m}$ bits and remove $m \log N$ bits. We have that the saving is at least

$$\begin{aligned} m \log N - \log N - \log \binom{FT}{m} &\geqslant \log \left(N^{m-1} \cdot \left(\frac{m}{eFT} \right)^m \right) \\ &= \log \left(\left(\frac{N}{FT} \right)^{m-2} \cdot \frac{N}{T^2} \cdot \frac{m^m}{e^m F^2} \right) \\ &\geqslant \log \left(\left(\frac{N}{uT} \right)^{m-2} \cdot \frac{N}{T^2} \cdot \frac{m^m}{e^m u^2} \right) \\ &\geqslant \log \left(\left(\frac{N}{uT} \right)^{m-2} \cdot \frac{N}{T^2} \right) \end{aligned} \tag{8.7}$$

bits, where the first inequality follows by using the binomial inequality $\binom{FT}{m} \leqslant (eFT/m)^m$, the second inequality follows since $F \leqslant u$, and the last inequality follows since for $m \geqslant m_0 = 3e \log u / \log \log u$ it holds that $m^m \geqslant e^m u^2$. Therefore, we have

Claim 8.3.9. *The number of bits saved is at least*

$$(m-1) \cdot \log \left(\min \left\{ \frac{N}{T^2}, \frac{N}{uTB^2 m_0^{2(B-2)}} \right\} \right).$$

Proof. The claim follows since the minimum between the two terms is always upper bound by $N/(uT)$ and upper bounded by N/T^2. ■

Thus, every m-multi-collision we record allows us to save the number of bits corresponding to $m-1$ mouse structures. It is left to argue that we do not over-count, namely, that we do not count the removal of the same element from $\tilde{h}$ twice. Indeed, the same multi-collision may be encountered in several mouse structures. To this end, observe that if a salt in a mouse structure is the center of a large multi-collision which we had recorded earlier, we will be in one of the following two cases:

1. There is a query in this mouse structure whose answer is the center of the multi-collision, and this query was in an earlier mouse structure.

2. There is a query in this mouse structure whose answer is the center of the multi-collision, and this query was not in any earlier mouse structure. (Note that by the structure of collisions, i.e., a mouse structure, there could be either one such query or two.)

Case 1 need not be handled. The reason is that if the condition in it holds, then the multi-collision is, in fact, outside of the (shortest) path from the source to the destination. Therefore the scenario will either be classified as a mouse structure *without* a large multi-collision or a different large multi-collision will be encountered for this mouse structure.

In case 2, first note that when we encounter a multi-collision, we add the relevant queries to the multi-collision if they were not already recorded and only then remove the corresponding entry from $\tilde{h}$. Therefore, we never remove anything twice. The last point we need to argue is that the we get enough savings. Above we showed that we have sufficient saving for $m-1$ mouse structures. Recall that we defined that we encounter a large multi-collision if at least m_0 REPEATEDUNUSED queries are involved in a multi-collision on some path we care about. It follows from this that a multi-collision of size m will be relevant in at most $m-m_0$ mouse structures- beyond that, it will no longer be a multi-collision because there will be less than m_0 REPEATEDUNUSED queries among the queries of the multi-collision. Since $m_0 \geqslant 1$, savings for $m-1$ mouse structures is sufficient for us.

Overall, as claimed earlier, case 5 has 3 different further categorizations– no multi-collisions and the two cases for multi-collisions.

8.3.4 Handling Case 6

Handling this case, where there are no REPEATEDUSED edges, is somewhat more complicated compared to the previous ones as we need to further divide the compression strategy into the four following sub-cases.

(a) The body of the mouse structure has at least one NEW query, and there is a NEW query at the tip of the mouse structure

(b) The body of the mouse structure has at least one NEW query, but there are no NEW queries at the tip of the mouse structure

(c) The body of the mouse structure has no NEW queries, and the body of the mouse structure is a directed cycle

(d) The body of the mouse structure has no NEW queries, but the body of the mouse structure is not a directed cycle.

The cases a and b are easier to deal with, and the strategy we will use for them mirrors the strategy used for case 5. The cases c and d are more challenging to deal with. We present the strategies for each of them individually *assuming that we do not encounter any large multi-collisions while encoding paths.* The case for large multi-collisions is handled the same as it was handled in Section 8.3.3.

HANDLING SUB-CASE A. In this case, the mouse structure looks like one of the structures shown in Fig. 8.7a. We make the following claim.

Claim 8.3.10. *A mouse structure that has been categorized in case 6 a must contain three distinct queries q_1, q_2, q_3 such that*

- *q_1, q_2 are* NEW *queries.*
- *q_3 is a* REPEATEDUNUSED *query.*
- *q_2, q_3 form the tip of the mouse structure.*
- *The path back from q_3 to q_1 has only* REPEATEDUNUSED *queries.*

The proof of this claim follows readily from the definition of the case, and it is easiest to see from Fig. 8.7a. Nevertheless, below we provide detailed proof of this claim and remark that we shall omit the detailed proof in similar claims made below.

Proof. [of Claim 8.3.10] Let the NEW query at the tip of the mouse structure be q_2 and let the REPEATEDUNUSED query at the tip of the mouse structure be q_3. (Indeed, we are guaranteed that the other query at the tip of the mouse structure is a REPEATEDUNUSED query because otherwise this mouse structure would have been categorized into either case 2 or case 5). Let q_1 be the first NEW query when backtracking from q_3 along with the mouse structure — we are guaranteed such a NEW query because on backtracking, eventually, we will reach the salt a'_j and by the definition of fresh salts, any query with a'_j as the input salt is NEW. We are guaranteed that q_1 is different from q_2 because q_1 is not at the tip of the mouse structure. Given these definitions of q_1, q_2, q_3 the claim follows. ■

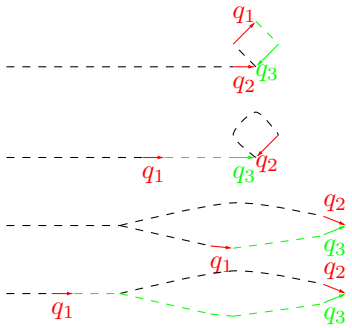
Given this claim, the encoding procedure defines a tuple consisting of the following:

- the indices in Q_j of the q_1, q_2,
- the smallest index of q_3 in $\mathsf{Q}_{\mathsf{fresh}}$, and
- the path back from input salt of q_3 to the answer of q_1 in the query graph when q_1 was queried

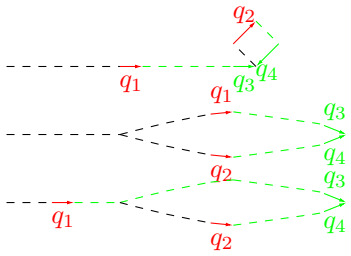
This tuple will be stored in a list. The answers corresponding to queries q_1, q_2 is removed from $\tilde{h}$.

The decoding procedure figures out the answer to query q_2 from the answer of q_3, and the answer to q_1 using the path back from the input salt of q_3 to the answer of q_1 in the query graph.

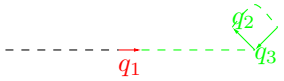
The indices of q_1, q_2 in Q_j can be encoded in $2 \log T$ bits, the index of q_3 in $\mathsf{Q}_{\mathsf{fresh}}$ can be encoded in $\log(FT) \leqslant \log(uT)$ bits, and the path requires at most $\log(Bm_0^{B-2})$ bits (the calculation is the same as was done in Section 8.3.3 for case 5). Therefore, we save a total of $\log N^2/(uT^3Bm_0^{B-2}) = \log N/T^2 + \log N/(uTBm_0^{B-2})$ bits which is sufficient.



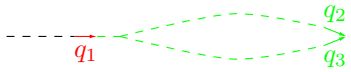
(a) Mouse structures with at least one NEW query in their body and a NEW query at the tip



(b) Mouse structures with at least one NEW query in their body and no NEW query at the tip



(c) Mouse structure with no NEW query in the body, but a NEW query at the tip



(d) Mouse structure with no NEW query in the body, and no NEW query at the tip

Figure 8.7: Mouse structures categorized in case 6. The NEW queries are drawn in red, and the REPEATEDUNUSED queries are drawn in green. The black dashed lines indicate zero or more NEW or REPEATEDUNUSED queries.

HANDLING SUB-CASE B. In this case the mouse structure looks like one of the structures shown in Fig. 8.7b. We claim that a mouse structure that is categorized in case 6 b must contain distinct queries q_1, q_2, q_3, q_4 such that

- q_1, q_2 are NEW queries
- q_3, q_4 are REPEATEDUNUSED queries such that q_3 was made for the first time before q_4
- q_3, q_4 form the tip of the mouse structure
- The paths back from q_3 to q_1 and q_4 to q_2 have only REPEATEDUNUSED queries

The proof of this claim can be verified from Fig 8.7b. Since the details are very similar to the proof of Claim 8.3.10, we omit them. The encoding procedure, in this case, will define a tuple consisting of the following:

- the indices in Q_j of the q_1, q_2
- the smallest indices of q_3, q_4 in $\mathsf{Q}_{\mathsf{fresh}}$
- the path back from the input salt of q_3 to the answer of q_1 in the query graph when q_1 was queried.
- the path back from the input salt of q_4 to the answer of q_2 in the query graph when q_2 was queried.

This tuple is stored in a list. The answer of the queries q_1, q_2, q_4 are removed from $\tilde{h}$.

The decoding procedure figures out the answer to the query q_4 from the answer of query q_3 which came before it and the answer of queries q_2 and q_1 using the paths back from q_4 to q_2 and q_3 back to q_1 in the query graph, respectively.

Therefore, by a calculation similar to the one above, we save at least the following number of bits which is sufficient.

$$\log N^3/(u^2T^4Bm_0^{2(B-2)}) = \log N/T^2 + 2\log N/(uTBm_0^{B-2})\,.$$

HANDLING SUB-CASE C. In this case the mouse structure looks like one of the structures shown in Fig. 8.7c. We claim that a mouse structure that has been categorized in case 6 c must have three queries q_1, q_2, q_3 in it such that

- q_1 is a NEW query

- q_2, q_3 are REPEATEDUNUSED queries

- q_3 is one of the queries at the tip of the mouse structure

- The answer of q_3 is the input salt of q_2

- The body of the mouse structure is a single directed cycle starting with q_2 and ending with q_3

- The queries in the mouse structure between q_1 and the answer of q_3 are REPEATEDUNUSED queries.

The proof of this claim can be verified from Fig 8.7c. Since the details are similar to the proof of Claim 8.3.10, we omit it.

Remark 8.3.11 (Naive encoding fails)**.** *Note that a similar encoding strategy to the one we used in sub-cases a and b, i.e., encoding the indices of* q_2, q_3 *in* $\mathsf{Q}_{\mathsf{fresh}}$*, the index of* q_1 *in* Q_j *and the paths required to backtrack, will not suffice. Indeed, it will save no more than* $\log N^2/(F^2T^3) = \log N/T^2 + \log N/(F^2T)$ *bits which might not suffice since* $N/(F^2T)$ *could be less than* 1*, in general.*

The *key idea* for compression is that we will achieve enough cumulative compression over all the mouse structures that satisfy this sub-case by storing some of the components of the encoding in an unordered set. We provide the details below.

SCENARIO 1. First, let us additionally assume that the body of the mouse structure is formed in sequence, i.e., the query q_3 was made for the first time after all the other queries in the body of the mouse had been made for the first time. In this case, we encode the index of queries q_2 and q_1 in a list, but the index of the query q_3 and the path back from q_3 to q_2 is encoded in an unordered set. In more detail, the encoding procedure stores the following:

- A tuple consisting of the minimum index of the query q_2 in $\mathsf{Q}_{\mathsf{fresh}}$, index of query q_1 in Q_j, and the path back to the answer salt of q_1 from answer salt of q_3 in the query graph when q_1 was queried. This tuple is stored in a list ordered by j.

- A tuple consisting of the minimum index of the query q_3 in $\mathsf{Q}_{\mathsf{fresh}}$ along with the path back from the input salt of q_3 to the input salt of q_2 in the query graph when q_3 was queried. This is stored in an unordered set.

The answers of queries q_1 and q_3 are removed from $\tilde{h}$.

The decoding procedure answers q_3 as follows. It detects which query is q_3 and the path back from the input salt of q_3 to the input salt of q_2 from the unordered set, locates the right q_2, and answers with the input salt of q_2. It answers q_1 by using the path from the answer salt of q_3 back to the answer of q_1.

If there are y mouse structures that are captured by this sub-case, then the total number of bits saved over the y mouse structures is at least

$$\begin{aligned} & y \log N^2 - y \log FT - y \log T - \log \binom{FT}{y} - 2y \log(Bm_0^{B-2}) \\ & \geqslant y \left(\log \frac{N}{T^2} + \log \frac{N}{uTB^2 m_0^{2(B-2)}} \right) - u - y \log e, \end{aligned}$$

where the last inequality follows since $F \leqslant u$ and

$$\binom{FT}{y} \leqslant \left(\frac{eFT}{y} \right)^y \leqslant e^y T^y \left(\frac{F}{y} \right)^y \leqslant e^y T^y (2^{F/y})^y \leqslant e^y T^y 2^F \leqslant e^y T^y 2^u.$$

Ignoring the missing $u + y \log e$ bits in the encoding size, we indeed save as much as we need to save. The missing $u + y \log e$ bits are consumed by the additional terms we introduced in the compression budget (see (9.7)): u is consumed by the penalty budget of $14u$ and $y \log e$ by the penalty budget of $u \log e$.

Scenario 2. Here, the mouse structure is not formed in sequence. Namely, there is a query q_4 in the body such that it was the last query to be made in the body. The encoding procedure stores the following:

- A tuple consisting of the minimum index of the query q_3 in $\mathsf{Q}_{\mathsf{fresh}}$, index of query q_1 in Q_j, and the path back to the answer salt of q_1 from the answer salt of q_3 in the query graph when q_1 was queried. This tuple is stored in a list ordered by j.

- A tuple consisting of the minimum index of the query q_4 in $\mathsf{Q}_{\mathsf{fresh}}$ along with the path back to the answer salt of q_4 from the input salt of q_4 in the query graph when q_4 was queried. This tuple is stored in an unordered set.

The answers corresponding of queries q_1, q_4 are removed from $\tilde{h}$.

The decoding procedure answers q_4 by using the path corresponding to q_4 and finding the answer salt of q_4 in the query graph. It answers q_1 using the path back from answer of q_3.

The compression analysis is the same as in scenario 1, so we avoid repetition. We remark that here we incur a similar penalty of $y \log e + u$ bits, as well.

Handling sub-case d. In this case the mouse structure looks the structure shown in Fig. 8.7d. We claim that a mouse structure that has been categorized in case 6 d must have three queries q_1, q_2, q_3 in it such that

- q_1 is a NEW query

- query q_2 was made for the first time before the query q_3 was made for the first time

- q_2, q_3 are REPEATEDUNUSED queries that form the tip of the mouse structure

- The paths from q_1 to q_2 and q_1 to q_3 in the mouse structure have only REPEATEDUNUSED queries.

The proof of this claim can be verified from Fig 8.7d. The details are similar to the proof of Claim 8.3.10 and are therefore omitted.

The main idea for the compression is similar to that in sub-case c i.e., we achieve enough cumulative compression over all the mouse structures that satisfy this sub-case by storing some of the components of the encoding in an unordered set.

Definition 8.3.12 (Frontal tail queries)**.** *Queries in the mouse structure that occur after the query q_1 but before the body are the* frontal tail queries*.*

SCENARIO 1. First, let us additionally assume that the last query to be made for the first time among the queries in the body of the mouse structure and the frontal tail queries is q_3. In this scenario, the encoding procedure stores the following.

- A tuple consisting of the minimum index of the query q_2 in $\mathsf{Q}_{\mathsf{fresh}}$, index of query q_1 in Q_j, the path back from the input salt of q_2 to the salt that is the answer of q_1 when in the query graph when q_3 is queried. This tuple is stored in a list ordered by j.

- A tuple consisting of the minimum index of the query q_3 in $\mathsf{Q}_{\mathsf{fresh}}$ and the path back from the input salt of q_3 to the salt that is the answer of q_1 when in the query graph when q_3 is queried. This tuple is stored in an unordered set.

The answers of queries q_1, q_3 are removed from $\tilde{h}$.

The decoding procedure answers q_3 by using the path corresponding to q_3 to locate the answer of q_1, and then for every candidate q_2 it checks whether the corresponding path back leads to the answer of q_1. Note that, only for the correct q_2 this would be true. Otherwise, if it were true for some other q_2 as well, then the mouse structure corresponding to the other q_2 would have been categorized in case 5. Once the correct q_2 has been found, it answers with the answer salt of q_2. It has already located the answer of q_1, in this case.

If there are y mouse structures that are captured by this sub-case, then the total number of bits saved over the y mouse structures is at least

$$\begin{aligned} & y \log N^2 - y \log FT - y \log T - \log \binom{FT}{y} - y \log(Bm_0^{B-2}) \\ & \geqslant y \left(\log \frac{N}{T^2} + \log \frac{N}{uTBm_0^{B-2}} \right) - u - y \log e, \end{aligned}$$

where the last inequality follows since $F \leqslant u$ and $\binom{FT}{y} \leqslant e^y T^y 2^u$, as in sub-case c. Note that in this case, as in sub-case c, we incur another (global) $u + y \log e$ penalty which we can tolerate by the total compression budget; ignoring this penalty, we compress sufficiently.

Scenario 2. Suppose there is a query q_4 in the body that is not at the tip of the mouse structure, and it was the last query to be made for the first time among the body queries and the frontal tail queries. Assume without loss of generality that q_2 is the query in the same body curve as q_4. The encoding procedure stores the following

- A tuple consisting of the minimum index of the queries q_2, q_3 in $\mathsf{Q}_{\mathsf{fresh}}$, index of query q_1 in Q_j, the path back to the answer salt of q_4 from the input salt of query q_2 in the query graph when q_4 is queried, and the path back from the input salt of q_3 to the answer salt of query q_1 in the query graph when q_4 is queried. This tuple is stored in a list ordered by j.

- A tuple consisting of the minimum index of the query q_4 in $\mathsf{Q}_{\mathsf{fresh}}$, and the path back from the input salt of q_4 to the answer salt of query q_1 in the query graph when q_4 is queried. This tuple is stored in an unordered set.

The answers of queries q_1, q_3, q_4 are removed from $\tilde{h}$.

The decoding procedure answers q_3 using the answer of the corresponding q_2. It answers q_4 by first locating the right q_3 by using the paths back from input salt of q_4 to the answer salt of q_1 and from input salt q_3 to the answer salt of q_1. This automatically locates the right q_2– it locates the answer of q_4 using the path back from the input salt of q_2 to the answer salt of q_4. It has already found the answer of q_1 in this case.

A similar calculation as in the previous scenario shows that this encoding procedure saves at least this many bits:

$$y \left(\log \frac{N}{T^2} + 2 \log \frac{N}{uTB^2 m_0^{2(B-2)}} \right) - y \log e - u$$

which is again within our budget (incurring with a global $u + y \log e$ penalty).

In this remaining scenario, there is a query q_4 in the frontal tail that is made for the first time after all the body queries and the other frontal tail queries. We split it again into two cases, depending on the order in which the queries were made.

Scenario 3. We will first consider the case when one of the queries q_2, q_3 at the tip of the mouse structure is the last to be made for the first time among the body queries. Recall the first body node of the mouse structure was the node from which the two body curves of the mouse separated (see Fig. 8.4). The encoding procedure here will store the following.

- A tuple consisting of the minimum index of the queries q_2 in $\mathsf{Q}_{\mathsf{fresh}}$, index of query q_1 in Q_j, the minimum index of the query q_4 in $\mathsf{Q}_{\mathsf{fresh}}$, the path back to the answer salt of q_4 from the first body node of the mouse structure in the query graph when q_4 is queried, the path back from the input salt of q_2 to the first body node in the query graph when q_3 was queried for the first time, and the path back from the input salt of q_4 to the answer salt of query q_1 in the query graph when q_1 is queried. This tuple is stored in a list ordered by j.

- A tuple consisting of the minimum index of the query q_3 in $\mathsf{Q}_{\mathsf{fresh}}$, the path back from the input salt of q_3 to the first body node of the mouse structure in the query graph when q_3 is queried for the first time. This tuple is stored in an unordered set.

It removes the answers of queries q_3, q_4, q_1 from $\tilde{h}$. We can do similar calculations like the previous scenario and show here that we save at least $y(\log N/T^2 + 2\log N/uTB^2m_0^{2(B-2)}) - y\log e - u$ bits.

The argument that decoding will succeed is similar to the earlier scenarios, and we omit it here.

Scenario 4. And finally, for the other scenario when a different query q_5 (without loss of generality in the same body curve as q_2) which is not at the tip of the mouse structure is the last query to be made for the first time among the body queries, the encoding procedure will store the following.

- A tuple consisting of the minimum index of the queries q_2, q_3 in $\mathsf{Q}_{\mathsf{fresh}}$, index of query q_1 in Q_j, the minimum index of the query q_4 in $\mathsf{Q}_{\mathsf{fresh}}$, the path back to the answer salt of q_5 from the input salt of q_2 when q_5 was queried for the first time, the path back from the input salt of q_3 to the first body node when q_5 was queried for the first time, the path back to the answer salt of q_4 from the first body node of the mouse structure when q_5 was queried for the first time, and the path back from the input salt of q_4 to the answer salt of query q_1 in the query graph when q_4 is queried for the first time. This tuple is stored in a list ordered by j.

- A tuple consisting of the minimum index of the query q_5 in $\mathsf{Q}_{\mathsf{fresh}}$, the path back from the input salt of q_5 to the first body node when q_5 was queried for the first time. This is stored in an unordered set.

It removes the answers of queries q_3, q_4, q_5, q_1 from $\tilde{h}$. We can do similar calculations like before and show here that we save at least the following number of bits:

$$y(\log N/T^2 + 3\log N/uTB^2m_0^{2(B-2)}) - y\log e - u\ .$$

.

The argument that decoding will succeed is similar to the earlier scenarios, and we omit it here.

Remark 8.3.13 (Total penalties incurred throughout sub-cases c and d.)**.** *There are two scenarios we considered in sub-case c and in each case, the penalty incurred was* $u + y \log e$ *bits. For sub-case d, we considered four scenarios, and in each scenario the penalty incurred was* $u + y \log e$ *bits. So the total penalty we incur is at most* $6u + u \log e$ *bits. Overall, case 6 has* $8\ (= 1 + 1 + 2 + 4)$ *different further categorizations if there are no multi-collisions and* 2 *categorizations concerning multi-collisions, leading to* 10 *sub-categories as claimed earlier.*

8.3.5 Description of our Encoding and Decoding Procedures of Theorem 9.3.5

This section describes the encoding and decoding procedure used in the proof of Theorem 9.3.5. The encoding and decoding procedures for Theorem 8.4.1 require very small changes, and so the modifications are given in Section 8.4.1.

8.3.6 The Encoding Procedure

This section gives the decoding procedure. Recall that the sets $\mathsf{Q}_{\mathsf{fresh}}, \mathsf{SIts}, \mathsf{Qrs}, \mathsf{Q}_j, \mathsf{MouseSIts}_j$ are defined as mentioned in the setup in Section 9.3. We do not explicitly repeat these definitions to avoid cluttering.

The encoding algorithm $\mathsf{Encode}(\sigma, U, h)$:

1. Parse U as $\{a_1, \ldots, a_u\}$ where $a_1, \ldots, a_u$ are in lexicographical order.

2. Set $j \leftarrow 0$.

3. Initialize the set P to be empty, the lists Cases and $\tilde{h}$ to be empty, and the set of nodes in the query graph to $[N]$ and the set of edges to be empty.

4. For $i = 1, \ldots, u$, do:

   (a) If $a_i \in \mathsf{SIts}(a)$ for some $a \in U_{\mathsf{fresh}}$, let $r \in [jT]$ be the smallest index such that $\mathsf{Q}_{\mathsf{fresh}}[r] = (a_j, *)$. Add r to P, and continue to next iteration.

   (b) Otherwise, increment j, set $a'_j \leftarrow a_i$. Run $\mathcal{A}_2$ on (σ, a'_j), answering its queries to h. For every query made by $\mathcal{A}_2$ to h on (a, α) with answer a', add an edge (a, a') with label α to the query graph. Add $h(a, \alpha)$ to the list $\tilde{h}$ if it was not already added earlier. Classify the query as NEW, REPEATEDUSED, SEMIFAMILIARREPEATED, REPEATEDUNUSED as defined in Section 9.3.

   (c) Determine the category of the mouse structure using the classification in Section 8.3.1. According to the categorization do the following.

- If the mouse structure gets categorized in case 1, then the mouse structure contains a NEW query (a, α) such that $h(a, \alpha) = a$. Let the index of the query in Q_j be n. Add n to the list L_1, set $\mathsf{Cases}[j] \leftarrow 1$ and remove the entry $h(a, \alpha)$ from $\tilde{h}$.
- Else, if the mouse structure gets categorized in case 2, the mouse structure contains two distinct NEW queries (a_1, α_1), (a_2, α_2) such that $h(a_1, \alpha_1) = h(a_2, \alpha_2)$, and (a_1, α_1) is queried before (a_2, α_2). Let the indices of the two queries be n_1, n_2, respectively, in Q_j. Add (n_1, n_2) to the list L_2, set $\mathsf{Cases}[j] \leftarrow 2$ and remove the entry $h(a_2, \alpha_2)$ from $\tilde{h}$. Continue to next iteration.
- Else, if the mouse structure gets categorized in case 3, then it contains a NEW query (a, α) such that $h(a, \alpha) = a'$ and $a' \in \mathsf{MouseSIts}_k$ for some $k < j$. Let the index of the NEW query in Q_j be n and the lexicographic order of the salt a' in $\mathsf{MouseSIts}_k$ be o. Add (k, o, n) to a list L_3, set $\mathsf{Cases}[j] \leftarrow 3$ and remove the entry $h(a, \alpha)$ from $\tilde{h}$. Continue to next iteration.
- Else, if the mouse structure gets categorized in case 4, then it contains a REPEATEDUNUSED query (a, α) such that $h(a, \alpha) = a$. Let n the smallest index of the query (a, α) in $\mathsf{Q}_{\mathsf{fresh}}$. Add n to the set S_4, set $\mathsf{Cases}[j] \leftarrow 3$ and remove the entry $h(a, \alpha)$ from $\tilde{h}$. Continue to next iteration.
- Else, if the mouse structure is categorized in case 5, there is at least one salt d such that $d \in \mathsf{MouseSIts}_i$ for some $i < j$ and there are at most $B - 2$ edges in the mouse structure between d and s where s is an answer to a NEW query. If there are several possible candidate pairs, choose one where the number of edges is the smallest between the source and the destination salt. Let the path back from d to s in the query graph when (a, α) is queried be P. If we have "encountered a large multi-collision" (as defined in Section 8.3.3) in P, then set $\mathsf{Cases}[j] \leftarrow 5m$ and goto MULCOL below. Else, suppose that the NEW query that results with the source salt s is (a, α), the destination salt is d and $d \in \mathsf{MouseSIts}_i$ for some $i < j$. Construct the tuple consisting of the following.[9]

Element	Size in bits
i	$\log u$
index of query (a, α) in Q_j	$\log T$
lexicographical index of d in $\mathsf{MouseSIts}_i$	$\log B$
path P	$\log Bm_0^{(B-2)}$

[9] We list the elements of the tuple in a table along with the sizes of each element to enable a reader cross check the size calculations we did in section 9.3.

Store the tuple in a list L_5, set $\mathsf{Cases}[j] \leftarrow 5a$ it removes the entry $h(a, \alpha)$ from $\tilde{h}$. Continue to next iteration.

- Else, if the mouse structure gets categorized in case 6, sub-categorize into sub-cases a to d as specified in Section 8.3.4.
  - If the mouse structure is categorized in a, define queries q_1, q_2, q_3 as defined under the heading "Handling sub-case a" in Section 6. Let the path back from input salt of q_3 to the answer of q_1 in the query graph when q_1 was queried be path P. If there was a large multi-collision encountered on path P, set $\mathsf{Cases}[j] = 6m$ and goto line MulCol.
    Otherwise define a tuple consisting of the following.

Element	Size in bits
the indices of q_1, q_2 in Q_j	$2 \log T$
the smallest index of q_3 in $\mathsf{Q}_{\mathsf{fresh}}$	$\log FT$
path P	$\log Bm_0^{(B-2)}$

    Add the tuple to the list L_{6a}. Remove the answers corresponding to queries q_1, q_2 from $\tilde{h}$ and set $\mathsf{Cases}[j] = 6a$.
  - If the mouse structure is categorized in b, define queries q_1, q_2, q_3, q_4 as defined under the heading "Handling sub-case b" in Section 6. Let the path back from the input salt of q_3 to the answer of q_1 in the query graph when q_1 was queried be P_1. Let the path back from the input salt of q_4 to the answer of q_2 in the query graph when q_2 was queried be P_2. If there was a large multi-collision encountered on either P_1 or P_2, set $\mathsf{Cases}[j] = 6m$ and goto MulCol. Otherwise define the following tuple.

Element	Size in bits
the indices of q_1, q_2 in Q_j	$2 \log T$
the smallest indices of q_3, q_4 in $\mathsf{Q}_{\mathsf{fresh}}$	$2 \log FT$
paths $\mathsf{P}_1, \mathsf{P}_2$	$\log B^2 m_0^{2(B-2)}$

    Add this tuple to the list L_{6b}. Remove the answers corresponding to queries q_1, q_2, q_4 from $\tilde{h}$ and set $\mathsf{Cases}[j] = 6b$.
  - If the mouse structure is categorized in c, define queries q_1, q_2, q_3 as defined under the heading "Handling sub-case c" in Section 6. Further, we categorize to scenarios 1 or 2, as defined there.
    i. If it is in scenario 1 let the path back to the answer salt of q_1 from answer salt of q_3 in the query graph when q_1 was queried be P_1 and let the path back from the input salt

of q_3 to the input salt of q_2 in the query graph when q_3 was queried be P_2. If there is a large multi-collision encountered either P_1 or P_2, set $\mathsf{Cases}[j] = 6m$ and goto MulCol. Otherwise define the following tuple.

Element	Size in bits
the index of q_1 in Q_j	$\log T$
the smallest index of q_2 in $\mathsf{Q}_{\mathsf{fresh}}$	$\log FT$
path P_1	$\log Bm_0^{(B-2)}$

Add the tuple to the list L_{6c1}. Define the following tuple.

Element	Size in bits
the smallest index of q_3 in $\mathsf{Q}_{\mathsf{fresh}}$	$\log FT$
path P_2	$\log Bm_0^{(B-2)}$

Add the tuple to the set S_{6c1}.

Remove the answers corresponding to queries q_1, q_3 from $\tilde{h}$ and set $\mathsf{Cases}[j] = 6c1$.

ii. Otherwise if it is in scenario 2, define q_4 as mentioned in scenario 2 under heading "Handling sub-case c" in Section 6. Let the path back to the answer salt of q_1 from the answer salt of q_3 in the query graph when q_1 was queried be P_1. Let the path back to the answer salt of q_4 from the input salt of q_4 in the query graph when q_4 was queried be P_2. If there is a large multi-collision encountered on either P_1 or P_2, set $\mathsf{Cases}[j] = 6m$ and goto MulCol. Otherwise, define the following tuple.

Element	Size in bits
the index of q_1 in Q_j	$\log T$
the smallest index of q_3 in $\mathsf{Q}_{\mathsf{fresh}}$	$\log FT$
paths P_1	$\log Bm_0^{(B-2)}$

Add the tuple to the list L_{6c2}. Define the following tuple.

Element	Size in bits
the smallest index of q_4 in $\mathsf{Q}_{\mathsf{fresh}}$	$\log FT$
path P_2	$\log Bm_0^{(B-2)}$

Add this tuple to the list L_{6c2}. Remove the answers corresponding to queries q_1, q_4 from $\tilde{h}$ and set $\mathsf{Cases}[j] = 6c2$.

– If the mouse structure is categorized in d, define queries q_1, q_2, q_3 as defined under the heading "Handling sub-case d" in Section 6. Further, categorize into scenarios 1, 2, 3 or

4, as defined there.

i. If it is in scenario 1 let the path back from the input salt of q_2 to the salt that is the answer of q_1 when in the query graph when q_3 is queried be P_1 and let the path back from the input salt of q_3 to the salt that is the answer of q_1 when in the query graph when q_3 is queried be P_2 If there is a large multi-collision encountered either P_1 or P_2, set $\mathsf{Cases}[j] = 6m$ and goto MulCol. Otherwise define the following tuple.

Element	Size in bits
the index of q_1 in Q_j	$\log T$
the smallest index of q_2 in $\mathsf{Q}_{\mathsf{fresh}}$	$\log FT$
path P_1	$\log Bm_0^{(B-2)}$

Add the tuple to the list L_{6d1}. Define the following tuple.

Element	Size in bits
the smallest index of q_3 in $\mathsf{Q}_{\mathsf{fresh}}$	$\log FT$
path P_2	$\log Bm_0^{(B-2)}$

Add the tuple to the set S_{6d1}.

Remove the answers corresponding to queries q_1, q_3 from $\tilde{h}$ and set $\mathsf{Cases}[j] = 6d1$.

ii. If it is in scenario 2, define q_4 as mentioned in scenario 2 under heading "Handling subcase d" in Section 6. Let the path back to the answer salt of q_4 from the input salt of query q_2 in the query graph when q_4 is queried be P_1. Let the path back from the input salt of q_3 to the answer salt of query q_1 in the query graph when q_4 is queried be P_2. Let the path back from the input salt of q_4 to the answer salt of query q_1 in the query graph when q_4 is queried be P_3. If there is a large multi-collision encountered either $\mathsf{P}_1, \mathsf{P}_2$ or P_3, set $\mathsf{Cases}[j] = 6m$ and goto MulCol. Otherwise define the following tuple.

Element	Size in bits
the index of q_1 in Q_j	$\log T$
the smallest indices of q_2, q_3 in $\mathsf{Q}_{\mathsf{fresh}}$	$2\log FT$
paths $\mathsf{P}_1, \mathsf{P}_2$	$\log B^2 m_0^{2(B-2)}$

Add the tuple to the list L_{6d2}. Define the following tuple.

Element	Size in bits
the smallest index of q_4 in $\mathsf{Q}_{\mathsf{fresh}}$	$\log FT$
path P_3	$\log Bm_0^{(B-2)}$

Add the tuple to the set S_{6d2}.

Remove the answers corresponding to queries q_1, q_3, q_4 from $\tilde{h}$ and set $\mathsf{Cases}[j] = 6d2$.

iii. If it is in scenario 3, let the path back to the answer salt of q_4 from the first body node of the mouse structure in the query graph when q_4 is queried be P_1. Let the path back from the input salt of q_2 to the first body node in the query graph when q_3 was queried for the first time be P_2. Let the path back from the input salt of q_4 to the answer salt of query q_1 in the query graph when q_1 is queried be P_3. Let the path back from the input salt of q_3 to the first body node of the mouse structure in the query graph when q_3 is queried for the first time be P_4. If there is a large multi-collision encountered either $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$ or P_4, set $\mathsf{Cases}[j] = 6m$ and goto MulCol. Otherwise define the following tuple.

Element	Size in bits
the index of q_1 in Q_j	$\log T$
the smallest indices of q_2, q_4 in $\mathsf{Q}_{\mathsf{fresh}}$	$2\log FT$
paths $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$	$3\log Bm_0^{(B-2)}$

Add the tuple to the list L_{6d3}. Define the following tuple.

Element	Size in bits
the smallest index of q_3 in $\mathsf{Q}_{\mathsf{fresh}}$	$\log FT$
path P_4	$\log Bm_0^{(B-2)}$

Add the tuple to the set S_{6d3}. Remove the answers corresponding to queries q_1, q_3, q_4 from $\tilde{h}$ and set $\mathsf{Cases}[j] = 6d3$.

iv. If it is in scenario 4, define q_5 as mentioned in scenario 4 under heading "Handling sub-case d" in Section 6. Let the path back to the answer salt of q_5 from the input salt of q_2 in the query graph when q_5 was queried for the first time be P_1. Let the path back from the input salt of q_3 to the first body node in the query graph when q_5 was queried for the first time be P_2. Let the path back to the answer salt of q_4 from the first body node of the mouse structure in the query graph when q_5 was queried for the first time be P_3. Let the path back from the input salt of q_4 to the answer salt of query q_1 in the query graph when q_4 is queried for the first time be P_4. Let the path back to

the input salt of q_5 to the first body node in the query graph when q_5 was queried for the first time be P_5. If there is a large multi-collision encountered either $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3, \mathsf{P}_4$, or P_5, set $\mathsf{Cases}[j] = 6m$ and goto MulCol. Otherwise define the following tuple.

Element	Size in bits
the index of q_1 in Q_j	$\log T$
the smallest indices of q_2, q_3, q_4 in $\mathsf{Q}_{\mathsf{fresh}}$	$3 \log FT$
paths $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3, \mathsf{P}_4$	$2 \log B^2 m_0^{2(B-2)}$

Add the tuple to the list L_{6d4}. Define the following tuple.

Element	Size in bits
the smallest index of q_5 in $\mathsf{Q}_{\mathsf{fresh}}$	$\log FT$
paths P_5	$\log B m_0^{(B-2)}$

Add the tuple to the set S_{6d4}. Remove the answers corresponding to queries q_1, q_3, q_4, q_5 from $\tilde{h}$ and set $\mathsf{Cases}[j] = 6d4$.

– Continue to next iteration.

- MulCol: If there is a query in this mouse structure whose answer is the center of the multi-collision, and this query was in an earlier mouse structure, do nothing. Else if there is a query in this mouse structure whose answer is the center of the multi-collision, and this query was not in any earlier mouse structure. If the center multi-collision of the multi-collision is s and $(s, \mathsf{Q}_s) \in S_{\mathsf{MC}}$, add the first index in $\mathsf{Q}_{\mathsf{fresh}}$ of all the queries in the large multi-collision into the set that were not already in Q_s to it. Otherwise, set s to be the center of the multi-collisions and Q_s to be the set of the first index in $\mathsf{Q}_{\mathsf{fresh}}$ of the queries involved in the large multi-collision., and remove the answer of all these queries from $\tilde{h}$. Add (s, Q_s) to S_{MC}, set $\mathsf{Cases}[j] \leftarrow m$.

5. Add all the evaluations of $h(a, \alpha)$ to $\tilde{h}$ in lexicographical order for all (a, α) that were never queried.

6. Output F, U', P, L_1, L_2, L_3, L_4, L_5, S_{MC}, L_{6a}, L_{6b}, L_{6c1}, L_{6c2}, L_{6d1}, L_{6d2}, L_{6d3}, L_{6d4}, S_{6c1}, S_{6c2}, S_{6d1}, S_{6d2}, S_{6d3}, S_{6d4}, $\tilde{h}$.

8.3.7 The Decoding Procedure

This section gives the decoding procedure.

The decoding algorithm Decode:

1. Parse U' as $\{a'_1, \ldots, a'_F\}$ where $a'_1, \ldots, a'_F$ are in lexicographical order.

2. Set $j \leftarrow 0$.

3. Initialize the set of nodes in the query graph to $[N]$ and the set of edges to empty.

4. For $j = 1, \ldots, F$ do:

   i Run $\mathcal{A}_2$ on (σ, a'_j), answering its queries to h.

   ii Answer the i-th query made by $\mathcal{A}_2$ to h on (a, α) as follows.

   iii If for some $(a', \mathsf{Q}_{a'}) \in S_{\mathsf{MC}}$, $((j-1) \cdot T + i) \in \mathsf{Q}_a$, then set $h(a, \alpha) \leftarrow a'$. Return a'.

   iv If $((j-1) \cdot T + i) \in \mathsf{P}$ then add a to U.

   v If $((j-1) \cdot T + i) \in S_4$ then set $h(a, \alpha) = a$ and return a.

   vi If $((j-1) \cdot T + i)$ appears as q_4 in any tuple of L_{6b}, then let a' be the answer of the query q_3 listed in the tuple. Set $h(a, \alpha) \leftarrow a'$ and return a'.

   vii If $((j-1) \cdot T + i)$ appears as q_3 in any tuple of S_{6c1}, then let the path P_2 in the tuple starting from a end in a'. Set $h(a, \alpha) \leftarrow a'$ and return a'.

   viii If $((j-1) \cdot T + i)$ appears as q_4 in any tuple of S_{6c2}, then let the path P_2 in the tuple starting from a end in a'. Set $h(a, \alpha) \leftarrow a'$ and return a'.

   ix If $((j-1) \cdot T + i)$ appears as q_3 in any tuple of S_{6d1}, then let the path P_2 in the tuple starting from a end in a'. Check for which tuple in the list L_{6d1} the path P_1 from the answer of the query q_2 of the tuple ends in a'. Let a'' be the answer of the query q_2 of that tuple. Set $h(a, \alpha) \leftarrow a''$ and return a''.

   x If $((j-1) \cdot T + i)$ appears as q_3 in any tuple of S_{6d2}, then let a' be the answer of the query q_2 of that tuple. Set $h(a, \alpha) \leftarrow a'$ and return a'.

   xi If $((j-1) \cdot T + i)$ appears as q_4 in any tuple of S_{6d2}, then let the path P_3 in the tuple starting from a end in a'. Check for which tuple in the list L_{6d2} the path P_2 from the answer of the query q_2 of the tuple ends in a'. Let a'' be the answer of the query q_2 of that tuple. Let the path P_1 in the tuple starting from a'' end in a'''. Set $h(a, \alpha) \leftarrow a'''$ and return a'''.

   xii If $((j-1) \cdot T + i)$ appears as q_3 in any tuple of S_{6d3}, then let the path P_4 in the tuple starting from a end in a'. Check for which tuple in the list L_{6d2} the path P_2 from the answer of the query q_2 of the tuple ends in a'. Let a'' be the answer of the query q_2 of that tuple. Set $h(a, \alpha) \leftarrow a''$ and return a''.

xiii If $((j-1)\cdot T+i)$ appears as q_4 in any tuple of L_{6d3}, then let a' be the first body node of the mouse structure located when answering the query q_3 corresponding to the tuple. Let the path P_1 in the tuple starting from a' end in a''. Set $h(a,\alpha) \leftarrow a''$ and return a''.

xiv If $((j-1)\cdot T+i)$ appears as q_3 in any tuple of L_{6d4}, then let a' be the answer of the query q_2 of the tuple. Set $h(a,\alpha) \leftarrow a'$ and return a'.

xv If $((j-1)\cdot T+i)$ appears as q_5 in any tuple of S_{6d4}, then let the path P_5 in the tuple starting from a end in a'. Check for which tuple in the list L_{6d2} the path P_2 from the answer of the query q_3 of the tuple ends in a'. Let a'' be the input of the query q_2 of that tuple. Let the path P_1 in the tuple starting from a'' end in a'''. Set $h(a,\alpha) \leftarrow a'''$ and return a'''.

xvi If $((j-1)\cdot T+i)$ appears as q_4 in any tuple of L_{6d4}, then let a' be the first body node of the mouse structure located when answering the query q_5 corresponding to the tuple. Let the path P_3 in the tuple starting from a' end in a''. Set $h(a,\alpha) \leftarrow a''$ and return a''.

xvii If $\mathsf{Cases}[j] = 1$ and i is the front element of the list L_1, then remove i from L_1, set $h(a,\alpha) \leftarrow a$ and return a.

xviii If $\mathsf{Cases}[j] = 2$ and (k,i) is the front element of the list L_2 for some $k < i$, let a' be the answer of the k-th query. Remove (k,i) from L_2, set $h(a,\alpha) \leftarrow a'$ and return a'.

xix If $\mathsf{Cases}[j] = 3$ and (k,o,i) is the front element of the list L_3 for some $k < j$, let a' be the salt that has lexicographic order o in $\mathsf{MouseSlts}_k$-th query. Remove (k,o,i) from L_3, set $h(a,\alpha) \leftarrow a'$ and return a'.

xx If $\mathsf{Cases}[j] = 5a$ and (k,o,i) is the front element of the list L_3 for some $k < j$, let a' be the salt that has lexicographic order o in $\mathsf{MouseSlts}_k$-th query. Remove (k,o,i) from L_5, set $h(a,\alpha) \leftarrow a'$ and return a'.

xxi If $\mathsf{Cases}[j] = 6a$ and if the tuple which is at the front of list L_{6a} has i as the index of q_2, then suppose the query listed as q_3 had answer a', set $h(a,\alpha) \leftarrow a'$ and return a'. Remove the tuple from the list L_{6a} if both the queries q_1, q_2 in the tuple have been answered.

xxii If $\mathsf{Cases}[j] = 6a$ and the tuple which is at the front of list L_{6a} has i as the index of q_1, then suppose the query listed as q_3 had answer a'. Let the path P in the tuple from a' in the query graph end in salt a''. Set $h(a,\alpha) \leftarrow a''$ and return a''. Remove the tuple from the list L_{6a} if both the queries q_1, q_2 in the tuple have been answered.

xxiii If $\mathsf{Cases}[j] = 6b$ and if the tuple which is at the front of list L_{6b} has i as the index of q_1, then suppose the query listed as q_3 had answer a'. Let the path P_1 in the tuple from a' in the query

graph end in salt a''. Set $h(a,\alpha) \leftarrow a''$ and return a''. Remove the tuple from the list L_{6b} if both the queries q_1, q_2 in the tuple have been answered.

xxiv If $\mathsf{Cases}[j] = 6b$ and if the tuple which is at the front of list L_{6b} has i as the index of q_2, then suppose the query listed as q_4 had answer a'. Let the path P_2 in the tuple from a' in the query graph end in salt a''. Set $h(a,\alpha) \leftarrow a''$ and return a''. Remove the tuple from the list L_{6b} if both the queries q_1, q_2 in the tuple have been answered.

xxv If $\mathsf{Cases}[j] = 6c1$ and if the tuple which is at the front of list L_{6c1} has i as the index of q_1, then suppose the query listed as q_3 had answer a'. Let the path P_1 in the tuple from a' in the query graph end in salt a''. Set $h(a,\alpha) \leftarrow a''$ and return a''. Remove the tuple from the list L_{6c1}.

xxvi If $\mathsf{Cases}[j] = 6c2$ and if the tuple which is at the front of list L_{6c2} has i as the index of q_1, then suppose the query listed as q_3 had answer a'. Let the path P_1 in the tuple from a' in the query graph end in salt a''. Set $h(a,\alpha) \leftarrow a''$ and return a''. Remove the tuple from the list L_{6c2}.

xxvii If $\mathsf{Cases}[j] = 6d1$ and if the tuple which is at the front of list L_{6d1} has i as the index of q_1, then let a' be the salt a' computed when answering the query q_3 that corresponded to this tuple. Set $h(a,\alpha) \leftarrow a'$ and return a'. Remove the tuple from the list L_{6d1}.

xxviii If $\mathsf{Cases}[j] = 6d2$ and if the tuple which is at the front of list L_{6d2} has i as the index of q_1, then then let a' be the salt a' computed when answering the query q_3 that corresponded to this tuple. Set $h(a,\alpha) \leftarrow a'$ and return a'. Remove the tuple from the list L_{6d3}.

xxix If $\mathsf{Cases}[j] = 6d3$ and if the tuple which is at the front of list L_{6d3} has i as the index of q_1, then suppose the path P_2 from the input salt of the query listed as q_3 ended in salt a'. Set $h(a,\alpha) \leftarrow a'$ and return a'. Remove the tuple from the list L_{6d3}.

xxx If $\mathsf{Cases}[j] = 6d3$ and if the tuple which is at the front of list L_{6d3} has i as the index of q_1, then suppose the query listed as q_4 had answer a'. Let the path P_3 in the tuple from a' in the query graph end in salt a''. Set $h(a,\alpha) \leftarrow a''$ and return a''. Remove the tuple from the list L_{6d3}.

xxxi If $\mathsf{Cases}[j] = 6d4$ and if the tuple which is at the front of list L_{6d3} has i as the index of q_1, then suppose the query listed as q_4 had answer a'. Let the path P_4 in the tuple from a' in the query graph end in salt a''. Set $h(a,\alpha) \leftarrow a''$ and return a''. Remove the tuple from the list L_{6d4}.

5. Populate the rest of h by remaining elements in $\tilde{h}$ in lexicographical order.

6. Output (U, h).

8.4 Proving the STB Conjecture for $SB \ll T$

This section proves another upper bound on the advantage of any auxiliary-input adversary in the bounded-length collision resistance game of salted MD hash in the AI-ROM. The main theorem is stated next.

Theorem 8.4.1. *Let $C = 2^9 \cdot 6 \cdot e^4$. For any $N, M, B, S, T \in \mathbb{N}_{>0}$ and fixing $\hat{S} := S + \log N$, it holds that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{MD},N,M,B}(S,T) \leqslant C \cdot \max\left\{\left(\frac{T^2}{N}\right), \left(\frac{\hat{S}^4 T B^2}{N}\right)\right\} + \frac{1}{N} \,.$$

The proof of this theorem mostly mirrors that of Theorem 9.3.5, except in the way that few of the cases are handled in the compression argument. Technically, we derive the following Lemma 8.4.2 (an analogue of Lemma 9.3.6), and combine it with Lemma 10.4.1 to get the claimed bound in Theorem 8.4.1.

Lemma 8.4.2. *Fix $N, M, B, S, T, u \in \mathbb{N}_{>0}$, $\sigma \in \{0,1\}^S$. Then, for any $\mathcal{A}_2$ that makes at most T queries to its oracle, it holds that*

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{MD},N,M,B,u}(\sigma, \mathcal{A}_2) \leqslant \left(2^9 e^4 \max\left\{\left(\frac{u^4 T B^2}{N}\right), \left(\frac{T^2}{N}\right)\right\}\right)^u \,.$$

The proof of this lemma is in the same spirit as the proof of Lemma 9.3.6 with several key differences. The main difference is how we encode the path back from a given destination node to the associated source node. In Section 9.3, we do this in a somewhat straightforward manner by encoding the length of the path and then the index of every edge to take, where the index might be large if there is a large multi-collision associated with that node. Large-enough multi-collisions were handled separately, so we had a bound ($m_0 \approx \log u / \log\log u$) for the range of the index of each back-edge. In this section, we encode the source node by just writing its lexicographic index among all possible sources within a given distance in the query graph. Of course, there might be too many possible sources at a given distance, making it too expensive to encode. But, and this is our main technical observation in this section, if there are too many possible sources, then there must be many large multi-collision, and therefore we can save enough bits by taking advantage of it.

We proceed by setting up the graph-theoretic definitions and lemmas. In the lemma below, we show that if in the query graph there is a node v such that there are at least p nodes which have a shortest path of length d to the node v, then there must be at least $p - 1$ edges involved in a multi-collision in the induced sub-graph.

Definition 8.4.3 (d-neighborhood of a vertex)**.** *Let $G = (V, E)$ be a directed graph. We say that a node $v_1 \in V$ is in the d-neighborhood of $v_2 \in V$ if the shortest directed path from v_1 to v_2 in G consists of (exactly) d edges.*

Definition 8.4.4 (multi-collision of edges in a graph)**.** *For an edge $e = (a, b)$, we refer to b as the target of the edge. We say that edges $e_1, \cdots, e_m$ form a m-multi-collision if all of them share a common target. We refer to the common target as the center of the multi-collision. We refer to m as the size of the multi-collision.*

We prove the following lemma.

Lemma 8.4.5. *Let $G = (V, E)$ be a directed graph. Let $d \in \mathbb{N}_{>0}$ and $p \in \mathbb{N}_{>0}$ such that $p \geqslant 2$. Suppose that there is a node $v \in V$ such that there are p distinct nodes in its d-neighborhood. Then, there are $t \geqslant 1$ distinct nodes in G, such that each of them have in-degree $\beta_i \geqslant 2$ for $i = 1, \ldots, t$, and $\sum_{i=1}^{t}(\beta_i - 1) \geqslant p - 1$.*

Proof. The proof is via induction on p. The base case is for $p = 2$. If there are at least 2 nodes in the d-neighborhood of v, it is easy to see that the graph has to have a node with in-degree 2. Here $t = 1$ and $\beta_1 = 2$ and therefore, the statement holds.

For the induction hypothesis, assume that the statement holds for $p = k$, and we will show that it also holds for $p = k + 1$. Since $p = k + 1$, the graph G has a node v with $k + 1$ nodes in its d-neighborhood. Let this set of $k + 1$ nodes be S. Consider the subgraph G' of G with only the edges on the shortest directed path from the nodes in S to the node v (choose one shortest path strategically if there are multiple paths of the same length– making sure that internal node in G' has out-degree 1. One way to do this is by following a BFS tree). By definition, in G', the node v still has $k + 1$ nodes in its d-neighborhood. Consider a node $v' \in S$ such that it has in-degree 0 in G' (note that such a v' must exist as there are no cycles in G'). Further, consider the set of edges P' that are on the path from v' to v but not on the path from the other edges in S to v. In other words, denoting $E_{v^* \to v}$ the set of edges on the path from v^* to v, we define

$$P' = E_{v' \to v} \setminus \bigcup_{v'' \in S \setminus \{v'\}} E_{v'' \to v}.$$

Definition 8.4.6 (Frontal edge of path)**.** *For a directed path in a graph, we define the edge whose target node has out-degree zero to be the frontal edge of the path.*

Define a subgraph H' of G' by removing the edges in P' from G'. Observe that P' will consist of a path and the frontal edge of the path will necessarily be incident on the target of an edge in H' – this is true because the first possibility is that this path is incident directly on v. In this case, there has to be at least one other path incident of v since $p > 2$ and we did not remove edges on the shortest path from other nodes in S to v. Hence, it is incident on the target of some edge in H'. The other possibility is that the frontal edge of this path incident on the end-point of some other edge in H' because we know this is part of a path from v' to v. The end-point has to be the target of some edge because otherwise, the length of the path from v' to v would be more than d.

It follows from the definition of P' that v has p distinct nodes in its d-neighborhood in H'. Using the induction hypothesis H' has t nodes with in-degree β_i such that $\beta_i \geqslant 2$ and $\sum_{i=1}^{t}(\beta_i - 1) \geqslant k - 1$. When we add back the edges in G', there are two possible cases. In the first case, one of the edges we add back is incident to one of the t edges, in which case G' has t edges each of size β_i' such that $\beta_i' \geqslant 2$ and

$\sum_{i=1}^{t}(\beta'_i - 1) \geqslant k - 1 + 1 = k$, because one of the β'_i will be larger than β_i by 1. In the other case, one of the edges we add back is incident on the target of an edge in G which was not incident on one of those t nodes. But this edge now creates a node of in-degree 2 in G'. So G' has $t+1$ nodes each of in-degree β'_i such that $\beta'_i \geqslant 2$ and $\sum_{i=1}^{t}(\beta'_i - 1) \geqslant k - 1 + 1 = k$. Since G' is a subgraph of G, it follows that for some t, G has t nodes with in-degree β'_i such that $\beta'_i \geqslant 2$ and $\sum_{i=1}^{t}\beta'_i \geqslant k$.

■

A direct corollary is that in the query graph, if for any salt s there are at least $p+1$ salts in its d-neighborhood, then there must be p queries involved in multi-collisions on the paths from the nodes in its d-neighborhood to s. We obtain non-trivial compression for large enough p by encoding those multi-collisions.

Non-trivial encoding for multiple multi-collisions. We consider t multi-collisions of sizes $\beta_1, \ldots, \beta_t$. We encode these t multi-collisions by encoding the t centers of the multi-collision, and for each center, we encode the index of the queries in $\mathsf{Q}_{\mathsf{fresh}}$ that form the multi-collision in a set. Recall that the indices of the queries in $\mathsf{Q}_{\mathsf{fresh}}$ are in $[FT]$. The total number bits we need to encode is

$$\log\left(\binom{N}{t}\binom{FT}{\beta_1}\binom{FT-\beta_1}{\beta_2}\cdots\binom{FT-\sum_{i=1}^{t-1}\beta_i}{\beta_t}\right) \leqslant \log\left(\frac{N^t(uT)^{\sum_{i=1}^{t}\beta_i}}{t!\beta_1!\beta_2!\cdots\beta_t!}\right)$$
$$= \log\left(\frac{N^t(u^4T)^{\beta-2t}T^{2t}u^{8t-3\beta}}{t!\beta_1!\beta_2!\cdots\beta_t!}\right), \tag{8.8}$$

where the inequality follows by using $\binom{n}{r} \leqslant \frac{n^r}{r!}$ and $F \leqslant u$, and the equality follows by letting $\beta = \sum_{i=1}^{t}\beta_i$ and rearranging.

Claim 8.4.7. *If $\beta \geqslant e^3u^2/2$, then Eq. (8.8) $\leqslant \log\left(N^t(u^4T)^{\beta-2t}T^{2t}\right)$.*

Proof. If $8t < 3\beta$, then the claim is immediate. Otherwise, we use the fact that $\beta_1!\beta_2!\cdots\beta_t! \geqslant (\beta/(te))^{\beta}$ (see Claim 8.4.8 for a proof) and the fact that $t! \geqslant (t/e)^t$ in order to get

$$(8.8) \leqslant \log\left(\frac{N^t(u^4T)^{\beta-2t}T^{2t}u^{8t-3\beta}t^{\beta-t}e^{\beta+t}}{\beta^{\beta}}\right).$$

Since $3\beta/8 \leqslant t \leqslant \beta/2$ (with the upper bound since each $\beta_i \geqslant 2$ and the lower bound by our assumption), we can show that $\beta^{\beta} \geqslant u^{8t-3\beta}t^{\beta-t}e^{\beta+t}$; See Claim 8.4.9 for a proof. Then, by plugging this bound into the above inequality we get the claim. ■

From this claim it follows that when $\beta \geqslant e^3u^2/2$, by storing the multi-collisions as above, the amount of bits saved is at least

$$\log N^{\beta} - \log\left(N^t(u^4T)^{\beta-2t}T^{2t}\right) = t\log\left(\frac{N}{T^2}\right) + (\beta-2t)\log\left(\frac{N}{u^4T}\right)$$
$$\geqslant (\beta-t)\log\left(\min\left\{\frac{N}{T^2}, \frac{N}{u^4T}\right\}\right) \geqslant u\log\left(\min\left\{\frac{N}{T^2}, \frac{N}{u^4T}\right\}\right),$$

where the second inequality follows since $t \leqslant \beta/2$, and $\beta/2 \geqslant e^3u^2/4 \geqslant u^2 \geqslant u$.

For the bound that we want to get in this section, the amount of bits we need to save *per mouse structure* is $\log\left(\min\left\{\frac{N}{T^2}, \frac{N}{u^4TB^2}\right\}\right)$ (see explanation below), and so *we indeed save enough from encoding one such set of multi-collisions.*

The compression argument. We encode a source node from a destination node by encoding the distance and which of the nodes at the given distance from the destination node is the source node. If the number of candidate nodes at the specified distance is larger than $e^3u^2/2$, by Lemma 8.4.5, we are guaranteed that there exist t multi-collisions of size $\beta_1, \beta_2, \ldots, \beta_t$ such that $\beta \geqslant \sum_{i=1}^{t}(\beta_i - 1) \geqslant e^3u^2/2$. This implies, by Claim 8.4.7, that we can already save enough by only encoding this set of multi-collisions. Using this argument we prove Lemma 8.4.2 which in turn implies Theorem 8.4.1, as explained above. We sketch the proof of Lemma 8.4.2 below. (We note that in Section 8.4.1 we state the changes in the pseudocode of encoding and decoding algorithms, compared to the algorithms used in the proof of Theorem 9.3.5.)

Proof. [Sketch of Lemma 8.4.2] We continue using the setup and notation we defined in the proof of Lemma 9.3.6. After encoding U into $U_{\mathsf{fresh}}, \mathsf{P}$ and accounting for the list that classifies the mouse structure into the various cases, here the budget of bits per mouse structure will be

$$\log\left(\min\left\{\frac{N}{e^3 4T^2}, \frac{N}{4e^3u^4TB^2}\right\}\right).$$

The first difference here is how we encode source salts from destination salts, an action that we need to do to handle Cases 5 and 6. Except this, there are no differences in the argument, namely, handling Cases 1 to 4 remains the same.

Encoding source salts. We let d be the minimum number of edges between the source and destination nodes. If there are at most $e^3u^2/2$ possible salts such that their shortest path to the source node in the query graph so far has d edges, we encode the distance and which of the nodes is the actual destination node. This would take $\log(e^3u^2B/2)$ bits. By repeating the same proof with this term, we obtain that the term $B(m_0)^{B-2}$ is replaced by $e^3u^2B/2$. In Case 5 and sub-cases a and b of Case 6, this strategy will save enough directly.

If there are more than $e^3u^2/2$ possible salts such that their shortest path to the destination node in the query graph so far has d edges, we encode the set of multi-collisions that we are guaranteed by Lemma 8.4.5 and save at least $u\log\left(\min\left\{\frac{N}{e^3T^2}, \frac{N}{e^3u^4T}\right\}\right)$ bits, which is more than what we need across all mouse structures. In this case, the entire encoding just removes the answers of queries from $\tilde{h}$ involved in these multi-collisions and just outputs the rest of $\tilde{h}$ and these multi-collisions.

Handling sub-cases c and d of Case 6. For sub-cases c and d of Case 6, if we use the new strategy to encode the destination nodes, we might not save enough. We explain why this is the case. Recall that in

sub-case c of Case 6, per mouse structure, our savings was

$$\log \frac{N}{T^2} + \log \frac{N}{uTB^2 m_0^{2(B-2)}}$$

bits. Now, when we replace Bm_0^{B-2} with $e^3u^2B/2$, the per mouse structure savings is

$$\log \frac{N}{T^2} + \log \frac{N}{e^3u^5TB^2}$$

bits. For our current budget, this is not enough. To accommodate this, we could increase our budget per mouse structure by replacing the u^4 term with a u^5 term.[10] But, by slightly modifying the encoding/decoding procedures, we can use the allocated budget.

The idea here is that because we have a larger budget in terms of u, we no longer need to use the unordered set trick. In more detail, we were encoding indices of some queries in unordered sets and paths related to that query to figure out the order of these queries. Here we can encode all queries in their right order and no longer encode the paths required to recover the order of these queries. We explain this through the example of how we handle Scenario 1 of sub-case c.

The encoding procedure now stores the following:

- Minimum index of the query q_2, q_3 in $\mathsf{Q}_{\mathsf{fresh}}$, index of query q_1 in Q_j. These are stored in lists.

- The number of edges d on the shortest path from the answer salt of query q_3 in the query graph when q_1 was queried and the lexicographic index i of the answer salt of q_1 among the salts whose shortest path to the answer salt of query q_3 is d. This is stored in a list ordered by j.

The answers of queries q_1 and q_3 are removed from $\tilde{h}$.

The decoding procedure answers q_3 with the input salt of q_2. On query q_1, it locates the salts in the query graph such that they have d edges on the shortest path answers with the salt with lexicographic index i among these salts.

Here the total number of bits saved per mouse structure is at least

$$\begin{aligned}
&\log N^2 - 2\log FT - \log T - \log B - \log(e^3u^2/2) \\
&\geqslant \log \frac{N}{T^2} + \log \frac{N}{e^3F^2u^2TB} \\
&\geqslant \log \frac{N}{T^2} + \log \frac{N}{e^3u^4TB},
\end{aligned}$$

[10] This would result in an S^5 term in the total upper bound on the advantage of the attacker. Our improved compression strategy gives an S^4 term.

where the last inequality follows since $F \leqslant u$. Note that this is within our budget. We can similarly show that we can save enough bits (as required by our budget) for other scenarios of sub-cases c and d. We avoid repetition and omit the full details. Otherwise, there are no other differences in this proof compared to Lemma 9.3.6. ■

Finally, we prove the two claims we used in calculations earlier.

Claim 8.4.8. *Let $t \in \mathbb{N}_{>0}$ and $a_1, a_2, \ldots, a_t \in \mathbb{N}_{>0}$. Let $a = a_1 + a_2 + \ldots + a_t$. Then*

$$a_1! a_2! \cdots a_t! \geqslant \left(\frac{a}{te}\right)^a$$

Proof. From Sterling's approximation, we have that $a_i! \geqslant \left(\frac{a_i}{e}\right)^{a_i}$. So it suffices to prove that

$$\prod_{i=1}^{t} a_i^{a_i} \geqslant \left(\frac{a}{t}\right)^a .$$

Since $a = a_1 + a_2 + \ldots + a_t$, this is equivalent to proving

$$\prod_{i=1}^{t} \left(\frac{a}{ta_i}\right)^{a_i} \leqslant 1 .$$

We have that

$$\frac{a}{ta_i} = \left(\frac{1}{t} + \sum_{j \in [t] \setminus \{i\}} \frac{a_j}{ta_i}\right) = 1 + \left(\sum_{j \in [t] \setminus \{i\}} \frac{a_j}{ta_i} - \frac{t-1}{t}\right) .$$

Using $1 + x \leqslant e^x$ we get that

$$\prod_{i=1}^{t} \left(\frac{a}{ta_i}\right)^{a_i} \leqslant \prod_{i=1}^{t} e^{a_i \sum_{j \in [t] \setminus \{i\}} \frac{a_j}{ta_i} - \frac{t-1}{t}} .$$

Simplifying we have

$$\prod_{i=1}^{t} \left(\frac{a}{ta_i}\right)^{a_i} \leqslant e^{\sum_{j=1}^{t} a_j \left(\frac{t-1}{t} - \frac{t-1}{t}\right)} = e^0 = 1 .$$

Hence, $\prod_{i=1}^{t} \left(\frac{a}{ta_i}\right)^{a_i} \leqslant 1$, as needed. ■

Claim 8.4.9. *Let $\beta, u, t \in \mathbb{N}_{>0}$ such that $3\beta/8 \leqslant t \leqslant \beta/2$ and $\beta \geqslant e^3 u^2/2$. Then*

$$\beta^\beta \geqslant u^{8t-3\beta} t^{\beta-t} e^{\beta+t} .$$

Proof. Let $g(t) = u^{8t-3\beta} t^{\beta-t} e^{\beta+t} / \beta^\beta$. Let $f(t) = \ln g(t)$ where ln is the natural logarithm. Since ln is an increasing function, $g(t)$ is maximized where $f(t)$ is maximized and vice versa. We show that $g(t)$ is maximized at $t = \beta/2$ (see why below). For this, it suffices to show that $f(t)$ is maximized at $t = \beta/2$ for $3\beta/8 \leqslant t \leqslant \beta/2$ and $\beta \geqslant e^3 u^2/2$.

We have

$$f(t) = (8t - 3\beta) \ln u + (\beta - t) \ln t + (\beta + t) - \beta \ln \beta .$$

Differentiating with respect to t, we have

$$f'(t) = 8 \ln u + \frac{\beta - t}{t} - \ln t + 1 = \frac{\beta}{t} + \ln \frac{u^8}{t} .$$

Since $\beta \geqslant e^3 u^2/2$, and $3\beta/8 \leqslant t \leqslant \beta/2$, $f'(t)$ is positive throughout in the domain we are interested in. So, the maximum occurs for $t = \beta/2$. At $t = \beta/2$,

$$g(t) = \frac{u^\beta \beta^{\frac{\beta}{2}} e^{\frac{3\beta}{2}}}{2^{\frac{\beta}{2}} \beta^\beta} = \left(\frac{e^3 u^2}{2\beta} \right)^{\frac{\beta}{2}} .$$

Since $\beta \geqslant e^3 u^2/2$, this is at most 1. Since at $t = \beta/2$ the value of $g(t)$ is maximized for the range we are interested in, the claim follows. ■

8.4.1 Changes for Theorem 8.4.1

In the encoding procedure, all the checks for multi-collisions are replaced by the check for the number of possible destination salts at the specified distance d from the source salt: if there are more than $e^3 u^2/2$ possible salts such that their shortest path to the destination salt in the query graph so far has d edges, we encode the set of multi-collisions. For cases $6c$ and $6d$, the indices of queries q_1, q_2, q_3 are all stored in lists (i.e. we do not need the sets which thereby reduces the number of paths). The code for the decoding procedure is modified accordingly.

Chapter 9

TIME-SPACE TRADEOFFS FOR SPONGE HASHING: ATTACKS AND LIMITATIONS FOR SHORT COLLISIONS

In this chapter, we study the time-space tradeoff attacks for finding short collisions in the Sponge constructions. We give several new attacks and prove several limitations on the best possible attacks. This chapter is based on our work in [FGK22].

We start off this chapter by providing a high-level overview of our techniques. We first describe the generic attack for finding B-block collisions for $B \geqslant 2$. This attack is a variant of an analogous attack for MD, given by ACDW [ACDW20]. We also recall the known best attack for $B = 1$. Then, we describe our new attack for finding 1-block collisions. In particular, our attack outperforms the optimal analogous attacks for MD for specific regimes of parameter settings. Lastly, we overview the techniques used to prove limitations on the best possible attacks for finding short collisions.

SPONGE NOTATION. A sponge function is a keyed hash function that takes as input an a c-bit initialization vector IV along with an arbitrary size input and outputs an r-bit string: $\mathsf{Sp}\colon \{0,1\}^c \times \{0,1\}^* \to \{0,1\}^r$. The second input is parsed as a sequence of r-bit blocks, denoted $(m_1, m_2, \ldots)$. On such an input $\mathsf{Sp}(\mathsf{IV}, (m_1, m_2, \ldots))$ is defined as follows. The function Sp is defined relative to a permutation $\Pi\colon \{0,1\}^{r+c} \to \{0,1\}^{r+c}$. An input to or an output of this permutation, denoted $\sigma \in \{0,1\}^{r+c}$, contains an r-bit block, denoted $\sigma[1]$, and a c-bit block, denoted $\sigma[2]$. We sometimes use $(\sigma[1], \sigma[2])$ to mean $\sigma[1] \| \sigma[2] = \sigma$.

On input $m_1, m_2, \ldots, m_\ell$ to Sp, it works as follows:

1. Initialize $\sigma^{(0)} = (\sigma^{(0)}[1], \sigma^{(0)}[2]) = (0, \mathsf{IV})$.
2. For $i = 1, \ldots, \ell$, compute $\sigma^{(i)} = \Pi((\sigma^{(i-1)}[1] \oplus m_i) \parallel \sigma^{(i-1)}[2])$.
3. Output $\sigma^{(\ell)}[1]$.

Attacks

GENERIC ATTACK FOR FINDING LENGTH B COLLISIONS. In the preprocessing phase the adversary randomly samples $t \approx S$ different IVs $\mathsf{IV}_1, \ldots, \mathsf{IV}_t$ and for $i = 1, \ldots, t$ it does as follows.

1. Compute $\sigma_{i,j}$ for $j \in [B/2 - 1]$ as $\sigma_{i,j} = \Pi(0, \sigma_{i,j-1}[2])$, where $\sigma_{i,0} = (0, \mathsf{IV})$.

The sequence $\sigma_{i,0}, \ldots, \sigma_{i,B/2-1}$ forms a "zero-walk" on IV_i.

2. Find m_i, m_i' such that $\Pi(m_i, \sigma_{i,B/2-1}[2])[1] = \Pi(m_i', \sigma_{i,B/2-1}[2])[1]$.

The preprocessing phase outputs $(\sigma_{i,B/2-1}[2], m_i, m_i')_{i=1,\ldots,t}$. In Fig. 9.1, we depict the preprocessing phase of the attack. In the online phase, the adversary gets a challenge IV as input. For $i = 1, \ldots, T/B$, it computes

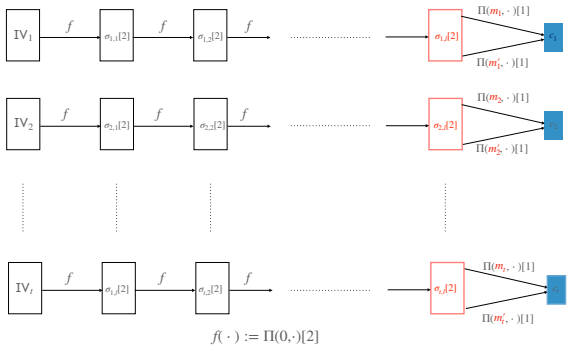


Figure 9.1: An illustration of the preprocessing phase of the generic attack. In red, we depict the components that are part of the output of the preprocessing phase. In blue we see the collisions that will be outputted in the online phase if some chain is hit. Notice that we denote $f(\cdot) := \Pi(0, \cdot)[2]$.

$\mathsf{IV}_i = \Pi(i, \mathsf{IV})[2]$ (for simplicity, we assume that i is in its bit representation). For each of the IV_i's, it does a zero-walk of length $B - 2$. Formally, it sets $\sigma_{i,0} \leftarrow \Pi(i, \mathsf{IV}_i)$ and then for $j = 1, \ldots, B - 1$ it does the following.

1. If there is a tuple of the form $(\sigma_{i,j-1}[2], m, m')$ in the preprocessing output, then return

$$(\sigma_{i,0}[1] \,\|\, \ldots \sigma_{i,j-1}[1] \,\|\, m), (\sigma_{i,0}[1] \,\|\, \ldots \sigma_{i,j-1}[1] \,\|\, m').$$

2. Set $\sigma_{i,j} \leftarrow \Pi(0, \sigma_{i,j-1}[2])$.

Correctness is easy to verify. We next discuss the success probability of the adversary. Suppose that the online phase of the adversary computes a $\sigma_{i,j}$ during the first half of any of the T/B zero-walks such $\sigma_{i,j}[2]$ matches the last c bits of one of the $\sigma_{i',j'}$s defined in the preprocessing phase. Then, it is guaranteed to stumble on $\sigma_{i',B/2-1}[2]$ during its zero walk. Hence, in this case, it would output a collision.

Since the adversary encounters roughly $\Omega(SB)$ distinct $\sigma_{i,j}[2]$'s in expectation during the preprocessing phase, this suffices to prove that with probability roughly $\Omega(STB/2^c)$ the online phase will win. The term $\Omega(T^2/2^c + T^2/2^r)$ appears due to birthday-style collisions. We refer the reader to Section 9.2.1 for details.

ATTACK FOR $B = 1$. As described in the introduction, the best attack known so far for $B = 1$ has an advantage of $O(S/2^c + T^2/2^r)$. The analogous attack for MD is provably optimal, as mentioned. However, in contrast to the setting in MD where the ideal object is a random function, here the ideal object is a random permutation, which gives us the additional ability to make inverse queries. This is precisely the leverage that we utilize to get our improved attack. We remark that we are not aware of any prior work that takes advantage of making inverse queries in related contexts.

For $B = 1$, recall that the goal is, given a random IV, to find m, m' such that $\Pi(m, \mathsf{IV})[1] = \Pi(m', \mathsf{IV})[1]$. Our first step is a bit counter-intuitive since we actually aim to solve a *harder* task. Specifically, rather than finding an arbitrary collision, we set out to find a collision on 0, that is, find m and m' such that $\Pi(m, \mathsf{IV})[1] = \Pi(m', \mathsf{IV})[1] = 0$. This step helps us since a natural way to use inverse queries arises, as we argue next.

Main observation: Finding a collision on 0 can be obtained by finding distinct y and y' such that

$$\Pi^{-1}(0, y)[2] = \Pi^{-1}(0, y')[2] = \mathsf{IV}.$$

In other words, it suffices to find two pre-images of IV with respect to the function $f_\Pi \colon \{0,1\}^c \to \{0,1\}^c$ where $f_\Pi(x)$ outputs the last c bits of $\Pi^{-1}(0, x)$. In Fig. 9.2, we show the partite representations of $\Pi(\cdot)$ and $\Pi^{-1}(0, \cdot)$. Note that while $\Pi(\cdot)$ is a perfect matching, the function $f_\Pi(\cdot) = \Pi^{-1}(0, \cdot)$ has several elements in its co-domain with multiple pre-images.

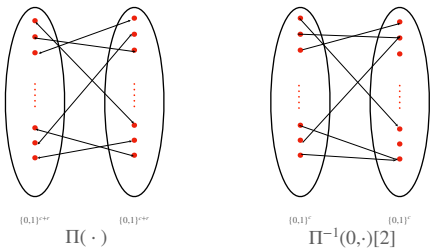


Figure 9.2: Partite representation of $\Pi(\cdot)$ and $\Pi^{-1}(0, \cdot)$. Notice that Π is a permutation and thus forms a perfect matching while $\Pi^{-1}(0, \cdot)$ is not a permutation and in expectation a random image will have several pre-images.

At this point, we made some progress: we reduced the problem of collision finding to a function inversion problem (for the function $f_\Pi \colon \{0,1\}^c \to \{0,1\}^c$). Indeed, preprocessing attacks for function inversion have been well studied since the 80's. Hellman [Hel80] described an algorithm that gets S bits of preprocessing on the random function $f \colon \{0,1\}^a \to \{0,1\}^b$ as input and inverts it at a point in its image making T queries to

the function. It was later formally analyzed by Fiat and Naor [FN99] and shown to have advantage $\varepsilon(a,b)$ at inverting $y = f(x)$ for a random $x \leftarrow\!\!\$\ \{0,1\}^a$, where

$$\varepsilon(a,b) = \Omega\left(\min\left\{1, \frac{ST}{2^{\min(a,b)}}, \left(\frac{S^2T}{2^{2\min(a,b)}}\right)^{1/3}\right\}\right) \tag{9.1}$$

We are almost done; three technical challenges remain. First, the result of Hellman applies only to random functions. On the other hand, our function is a restriction of a random permutation (which is not a random function). Fiat and Naor [FN99] showed a clever method to extend Hellman's algorithm to support any function (rather than only random ones), but this improvement is more complicated and comes with a cost in efficiency, which we would like to avoid. To this end, we re-do and adapt the analysis of Hellman to our setting by using the fact that restrictions of permutations are "close enough" to random functions. Our analysis achieves the same parameters as the original one of Hellman, up to constants.

The second problem is that we want to find a pre-image of $\mathsf{IV} \leftarrow\!\!\$\ \{0,1\}^c$ under f_Π, but IV may not even have any pre-images under f_Π, let alone two which are required for our attack. Fortunately, as f_Π is at least "close" to a random function, we can show via a balls-into-bins analysis that a constant fraction of the co-domain will have at least two distinct pre-images. Still, could it be the case that Hellman's attack somehow fails on this fraction of the co-domain? Via a closer analysis of Hellman's attack, we show that for any function $f\colon \{0,1\}^a \to \{0,1\}^b$ and fixed element $y \in \{0,1\}^b$, the attack succeeds at finding a pre-image $x' \in f^{-1}(y)$ with probability $\varepsilon(a,b)$ where

$$\varepsilon(a,b) = \Omega\left(\frac{1}{b}\min\left(1, \frac{ST \cdot |f^{-1}(y)|}{2^a}, \left(\frac{S^2T \cdot |f^{-1}(y)|^2}{2^{2a}}\right)^{1/3}\right)\right) \tag{9.2}$$

The last problem we face is that we need to find two *distinct* pre-images for IV. However, applying an inversion algorithm in a black box fashion does not guarantee that distinct inverses will be found. Thus, we also prove that Hellman's inversion algorithm finds a uniform pre-image among all possible pre-images for a given element in the co-domain.

After resolving the above technical challenges, we show that if we run Hellman's attack twice independently for the function f_Π on the image IV, if IV has at least two pre-images (which it does with constant probability), then we will find two distinct pre-images with at least $1/2$ probability times the probability that both attacks succeed. Thus, our overall success probability is roughly $\Omega(\varepsilon(c,c)^2)$ where ε is defined in (9.2). We refer the reader to section 9.2.2 for the details.

Impossibility Results for Best Attacks

When giving new attacks for finding short collisions, the natural question is how far we can go. In other words, what are the best possible attacks? For $B = 1, 2$ in the case of MD, optimal attacks are known. Our goal here is to prove an upper bound on the advantage for the best-possible adversary that has S bits

of preprocessing as input and can make T queries to Π, Π^{-1} in finding collisions of length 1 and 2 for the sponge construction.

IMPOSSIBILITY RESULT FOR $B = 1$. We use the pre-sampling technique proposed by [Unr07] and later optimized and adapted to the AI-RPM by [CDG18] to get an advantage upper bound of roughly $O(ST/2^c + T^2/2^r)$. However, we note that this bound does not match the best $B = 1$ attacks, so it is open which side can be improved. Ideally, one could use a compression-based technique as done in [DGK17] to get a tight bound for the $B = 1$ case for MD, but it is not clear how to adapt this argument to handle inverse queries in the AI-RPM model, as we shall see below.

IMPOSSIBILITY RESULT FOR $B = 2$. The presampling technique of [Unr07, CDG18] *provably* cannot give an advantage upper bound better than $O(ST^2/2^c + T^2/2^r)$ for $B = 2$. Since we can prove this advantage upper bound even for unbounded length collisions, it is natural to ask whether we can prove that 2-block collisions are, in fact, harder to find than collisions of arbitrary length. Aside from presampling techniques, the main technique used to rule out attacks is via a compression argument [GT00, Wee05], which we turn to for our impossibility result. As a warm up, we first give an overview for the $B = 1$ compression argument for MD from [DGK17] to highlight the key challenges in our setting.

Overview of $B = 1$ compression argument for MD. In a compression argument, the main idea is to use an adversary $\mathcal{A}$ that succeeds at some task involving a random object $\mathcal{O}$, to compress $\mathcal{O}$ beyond what is information theoretically possible. This clearly establishes a contradiction, which gives an upper bound in the success probability of $\mathcal{A}$.

Let $h\colon [N] \times [M] \to [N]$ be a hash function that is modeled as a random oracle, and $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an (S, T) adversary that tries to find a 1-block collision in h. $\mathcal{A}_1$ gets h as input and can output S bits of advice σ. $\mathcal{A}_2$ gets σ along with a random salt $a \in [N]$, can make T queries to h, and tries to output $m, m' \in [M]$ such that $h(a, m) = h(a, m')$ and $m \neq m'$. We show that if $\mathcal{A}$ succeeds at this task for many salts a, then we can describe h with fewer bits than possible.

To encode h, we first compute $\sigma \leftarrow \mathcal{A}_1(h)$. We let $G \subseteq [N]$ be the set of elements for which $\mathcal{A}_2$ succeeds on inputs (σ, a) for all $a \in G$. We run $\mathcal{A}_2$ on (σ, a) for all $a \in G$ in lexicographic order. The hope is that whenever $\mathcal{A}_2$ succeeds in finding a collision, we can use the corresponding queries for the collision it makes to compress the function h.

For example, if $\mathcal{A}_2(\sigma, a)$ outputs a collision (m, m'), then we can assume that $\mathcal{A}_2$ must have queried $h(a, m)$ and $h(a, m')$ at some point (we assume without loss of generality that (a, m) was queried before (a, m')). So whenever it queries $h(a, m')$, rather than encoding the output of h, we write down information to indicate that it's the same output as the query for $h(a, m)$. It is easy to see by a counting argument that at least half

of the $a \in G$ cannot be queried by $\mathcal{A}_2$ more than $2T$ times. For all such a, we can refer back to the previous query $h(a, m)$ using $\log 2T$ bits, and we use another $\log 2T$ bits to identify the query $h(a, m')$. Thus, we save $\log N - 2 \log 2T$ bits per each of these a values in G, which gives a non-trivial compression of h if $T^2 < N$.

The problem with inverse queries. As we stated before, the first major roadblock we encounter when adapting this framework to the AI-RPM is the existence of inverse queries. Let's try to adapt the argument above to the setting of Sponge with 1-block collision. Now the preprocessing adversary $\mathcal{A}_1$ is given a random permutation Π and outputs some state σ with $|\sigma| \leqslant S$. The online adversary $\mathcal{A}_2$ receives σ and a random IV, and tries to find m, m' such that $\Pi(m, \mathsf{IV})[1] = \Pi(m', \mathsf{IV})[1]$.

Now suppose that $\mathcal{A}_2$ outputs a collision (m, m') with respect to the sponge construction. We can no longer even assume that $\mathcal{A}_2$ queries both $\Pi(m, \mathsf{IV})$ and $\Pi(m', \mathsf{IV})$! For example, it may have first queried $\Pi(m, \mathsf{IV}) = (y, u_1)$ and then queried $\Pi^{-1}(y, u_2) = (m', \mathsf{IV})$. At first glance, this doesn't seem like a problem, we can again note that part of the output of query (y, u_2) is the same as the input to the query for (m, IV). So maybe we can use the same trick as before and instead of storing the whole answer of $\Pi^{-1}(y, u')$, store information indicating that the last c bits of the answer is the same as the input of the query $\Pi(m, \mathsf{IV})$. This intuition is misleading. We can no longer do a counting argument to show that this information is short. It is not clear how to identify the query $\Pi^{-1}(y, u')$ with few bits to be able to point back to the $\Pi(m, \mathsf{IV})$ query. For example, the adversary may just query $\Pi^{-1}(y, *)$ many times and hope to hit IV twice. We hope that this example sheds light on why, at a minimum, inverse queries significantly complicate the situation and deserve extra attention.

Compression for $B = 2$ via multi-instance games. The above compression approach is not known to generalize to the case of $B \geqslant 2$ collisions for MD. To overcome this limitation, Akshima et al. [ACDW20] propose a beautiful framework that gives non-trivial bounds $B \geqslant 2$ for the case of MD. Their framework reduces the problem to a related "multi-instance" game.

In a multi-instance game, the adversary has an arbitrary size string σ of S-bits hard-coded, and its goal is to find a 2-block collision for a set of $u \approx S$ uniformly random a's. The adversary $\mathcal{A}_2$ can make T queries to h when running on each of the u IVs. The key distinctions in this multi-instance game is that (1) the advice sigma that $\mathcal{A}_2$ receives is *independent* of h, and (2) we only need to analyze $\mathcal{A}_2$'s success probability for a random set of u IVs. The core of the proof is a compression argument to upper bound the advantage of this adversary. This framework unfortunately is not strong enough to deal with $B = 1$, as at best it gives the same bound as bit-fixing. However, we adapt this framework to the setting of random permutations to give a non-trivial bound for $B = 2$.

In our case, we need to build a compression argument to compress Π and a set of u random IVs, $\mathsf{IV}_1, \ldots, \mathsf{IV}_u$, (for $u \approx S$) using an adversary $\mathcal{A}_2$ which has some fixed hard-coded advice. $\mathcal{A}_2$ runs on the IVs one by one and succeeds in finding 2-block collisions for all of them. The encoding avoids storing some of the values of Π explicitly, and instead stores information about the queries of $\mathcal{A}_2$ to Π and Π^{-1} which help during the decoding procedure to recover these particular values of Π.

For $B = 2$ collisions, there are possibly 4 "crucial" queries that the adversary might make that correspond to a 2-block collision (two for each message). We possibly need to consider all combinations of ways that the queries could have been made in either the forward or the reverse direction. For the case of this overview, we zoom in on a single case where inverse queries complicate the situation, and explain how we overcome this.

Suppose on an input IV_j (where $\mathcal{A}_2$ had previously been run on inputs $\mathsf{IV}_1, \ldots, \mathsf{IV}_{j-1}$), $\mathcal{A}_2$ arrives at a collision by making the crucial queries q_1, q_2, q_3, q_4 (not necessarily in that order) such that

1. q_1 was a query to Π on (m_1, IV_j) and returned (x_1, IV'_1)
2. q_2 was a query to Π on (m_2, IV_j) and returned (x_2, IV'_2)
3. q_3 was a query to Π on $(m_3 \oplus x_1, \mathsf{IV}'_1)$ and returned (y, IV'_3)
4. q_4 was a query to Π on $(m_4 \oplus x_2, \mathsf{IV}'_2)$ and returned (y, IV'_4)

Clearly, (m_1, m_3) and (m_2, m_4) hash to the same output and hence are a collision. Now suppose queries q_1 and q_2 were first made during $\mathcal{A}_2(\mathsf{IV}_j)$,[1] while queries q_3 and q_4 were each made previously while running $\mathcal{A}_2$ on an earlier IV_i value. Now, the strategy to compress on the lines of [ACDW20] is not to include the last c bits of the answers of q_1, q_2 and the last r bits of the answer of q_4 in the encoding. Instead, we can store the index of the queries q_3, q_4 among all queries (these indices will be in $[uT]$ since there are u IVs and T queries for each of them) and store the indices of the queries q_1, q_2 among the queries made while running $\mathcal{A}_2$ on IV_j (these indices will be in $[T]$). This leads to a saving of roughly $2c + r - 2\log T - 2\log uT$ bits. For reasonable parameters of S, T, c, r, this implies a compression of at least $c - \log uT$ bits, so if $uT \approx ST < 2^c$, this gives non-trivial compression. This implies an upper bound of $ST/2^c$ on the advantage for this case.

However, with inverse queries allowed, things get more complicated. Suppose instead that queries q_3 and q_4 were made in the reverse direction, so $\mathcal{A}_2$ queries $\Pi^{-1}(y, \mathsf{IV}'_3)$ and $\Pi^{-1}(y, \mathsf{IV}'_4)$ prior to running $\mathcal{A}_2(\mathsf{IV}_j)$. In this case, we can still save the c bits from the answers of q_1, q_2. But there is no clear way to save in storing the answer to query q_4 since its answer $(m_4 \oplus x_2, \mathsf{IV}'_2)$ has seemingly no relation to either the answer

[1] Note that this assumption is easy to remove as otherwise we can achieve compression by not including IV_j in the encoding and recovering it from $\mathcal{A}_2$'s queries during decoding

or input of q_3. So, in this case, we are only able to save $2c - 2\log T - 2\log uT$ bits, which leads to non-trivial compression only if $u^2T^4 \approx S^2T^4 < 2^c$. This implies an upper bound of $S^2T^4/2^{2c}$ on the attacker's advantage for this case. Note that this is actually *better* than the $ST/2^c$ bound we got when considering only forward queries whenever $ST^3 < 2^c$. However, it is important to note that we still need to consider all possible ways in which the attacker may find a collision. We need to show even in the worst case, we can compress Π in order to get an upper bound on the advantage.

The above highlights just one of the several subtleties that inverse queries introduce in the proof. The ability of the adversary to make queries in two directions makes the encoding and decoding procedures significantly more complicated and lengthy.

Chapter Organization. We describe the details of this result in the rest of this chapter. In Section 9.1 we give the necessary preliminaries for this chapter. In Section 9.2, we give our new attacks and in Section 9.3, we prove limitations on the best possible time-space tradeoff attacks.

9.1 Preliminaries

We let $[N] = \{1, 2, \ldots, N\}$ for $N \in \mathbb{N}$ and for $k \in \mathbb{N}$ such that $k \leqslant N$, let $\binom{S}{k}$ denote the set of k-sized subsets of S. We use $|X|$ to denote the size of a set X and use X^+ to denote one or more elements of X. The set of all permutations on D is denoted by $\mathsf{Perm}(D)$. We let $*$ denote a wildcard element. For example $(*, z) \in L$ is true if there is an ordered pair in L where z is the second element (the type of the wildcard element shall be clear from the context). For a random variable X we use $\mathbb{E}[[]X]$ to denote its expected value.

We use $x \xleftarrow{\$} \mathcal{D}$ to denote sampling x according to the distribution $\mathcal{D}$. If D is a set, we overload notation and let $x \xleftarrow{\$} D$ denote uniformly sampling from the elements of D. For a bit-string s we use $|s|$ to denote the number of bits in s.

All logarithms in this chapterr are for base 2 unless otherwise specified.

Sponge-based hashing. For $c, r \in \mathbb{N}$, let $\Pi : \{0,1\}^{c+r} \to \{0,1\}^{c+r}$ be a permutation. We define sponge-based hashing $\mathsf{Sp}_\Pi : \{0,1\}^c \times (\{0,1\}^r)^+ \to \{0,1\}^r$ as follows. For $s \in \{0,1\}^{r+c}$ we use $s[1]$ to denote its first r bits and $s[2]$ to denote its last c bits.

$\underline{\mathsf{Sp}_\Pi(\mathsf{IV}, m = (m_1, \ldots, m_B))}$
$s_0 \leftarrow 0^r \,\|\, \mathsf{IV}$
For $i = 1, \ldots, B$
  $s_i[1] \,\|\, s_i[2] \leftarrow \Pi\left((m_i \oplus s_{i-1}[1]) \,\|\, s_{i-1}[2]\right)$
Return $s_B[1]$

The elements of $\{0,1\}^r$ shall be referred to as *blocks* and IV refers to the *initialization vector* (also referred to as *salt* in the literature). This is the same abstraction of sponge-based hashing as the one used in [CDG18].

Game $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}(\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2))$	Subroutine $\mathsf{AI\text{-}CR}_{\Pi,\mathsf{IV}}(\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2))$
1. $\Pi \leftarrow\!\!\$\ \mathsf{Perm}(\{0,1\}^{c+r})$	1. $\sigma \leftarrow\!\!\$\ \mathcal{A}_1(\Pi)$
2. $\mathsf{IV} \leftarrow\!\!\$\ \{0,1\}^c$	2. $(\alpha,\alpha') \leftarrow\!\!\$\ \mathcal{A}_2^{\Pi,\Pi^{-1}}(\sigma,\mathsf{IV})$
3. Return $\mathsf{AI\text{-}CR}_{\Pi,\mathsf{IV}}(\mathcal{A})$	3. Return true if: (a) $\alpha \neq \alpha'$, (b) $\lvert\alpha\rvert, \lvert\alpha'\rvert$ are at most B blocks long and (c) $\mathsf{Sp}_\Pi(\mathsf{IV},\alpha) = \mathsf{Sp}_\Pi(\mathsf{IV},\alpha')$
	4. Else, return false

Figure 9.3: The bounded-length collision resistance game of salted sponge based hash in the AI-RPM, denoted $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}$.

Auxiliary-input Random Permutation Model (AI-RPM). We use the Auxiliary-Input Random Permutation Model (AI-RPM) introduced by Coretti, Dodis and Guo [CDG18] to study non-uniform adversaries in the Random Permutation Model (this was a natural extension of the AI-ROM model proposed by Unruh in [Unr07]). This model is parameterized by two non-negative integers S and T and an adversary $\mathcal{A}$ is divided into two stages $(\mathcal{A}_1,\mathcal{A}_2)$. Adversary $\mathcal{A}_1$, referred to as the preprocessing phase of $\mathcal{A}$ has unbounded access to the random permutation Π and it outputs an S-bit auxiliary input σ. Adversary $\mathcal{A}_2$, referred to as the online phase, gets σ as input and can make a total of T queries to Π,Π^{-1}, and attempts to accomplish some goal involving Π. Formally, we say that $\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)$ is an (S,T)-AI adversary if $\mathcal{A}_1$ outputs S bits and $\mathcal{A}_2$ issues T queries to its oracles. We next formalize the collision resistance of sponge-based hash functions in AI-RPM.

Short collision resistance of sponge-based hashing in AI-RPM. We formalize the hardness of bounded-length collision resistance of sponge-based hash functions in the AI-RPM. The game is parameterized by c,r. The game first samples a permutation Π uniformly at random from $\mathsf{Perm}(\{0,1\}^{c+r})$ and IV uniformly at random from $\{0,1\}^c$. Then, $\mathcal{A}_1$ is given unbounded access to Π, and it outputs σ. At this time, $\mathcal{A}_2$ gets σ and IV as input and has oracle access to Π,Π^{-1}. It needs to find $\alpha\neq\alpha'$ such that (1) $\mathsf{Sp}_\Pi(\mathsf{IV},\alpha)=\mathsf{Sp}_\Pi(\mathsf{IV},\alpha')$ and (2) α,α' consist of $\leqslant B$ blocks from $\{0,1\}^r$. This game, denoted $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}$, is explicitly written in Fig. 9.3. In Fig. 9.3, we write the adversary's execution in its own subroutine only for syntactical purposes (as we shall use it later).

Definition 9.1.1 (AI-CR Advantage)**.** *For parameters $c, r, B \in \mathbb{N}$, the advantage of an adversary $\mathcal{A}$ against the bounded-length collision resistance of sponge in the AI-RPM is*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,B}(\mathcal{A}) = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}(\mathcal{A}) = \mathsf{true}\right]$$

For parameters $S, T \in \mathbb{N}$, we overload notation and denote

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,B}(S,T) = \max_{\mathcal{A}} \left\{\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,B}(\mathcal{A})\right\},$$

where the maximum is over all (S,T)-AI adversaries.

THE COMPRESSION LEMMA. Our proof of the impossibility result for $B = 2$ uses the well-known technique of finding an "impossible compression". The main idea, formalized in the following proposition, is that it is impossible to compress a random element in set $\mathcal{X}$ to a string shorter than $\log|\mathcal{X}|$ bits long, even relative to a random string.

Proposition 9.1.2 (E.g., [DTT10])**.** *Let* Encode *be a randomized map from $\mathcal{X}$ to $\mathcal{Y}$ and let* Decode *be a randomized map from $\mathcal{Y}$ to $\mathcal{X}$ such that*

$$\mathsf{Pr}_{x \xleftarrow{\$} \mathcal{X}}\left[\mathsf{Decode}(\mathsf{Encode}(x)) = x\right] \geqslant \varepsilon.$$

Then, $\log|\mathcal{Y}| \geqslant \log|\mathcal{X}| - \log(1/\varepsilon)$.

9.2 Attacks

In this section, we first provide the generic attack for finding B-block collisions inspired by the analogous attack for MD in [ACDW20]. We then provide our new AI-RPM attack for finding 1-block collision (Section 9.2.2). Additionally, in Section 9.2.3, we prove the key lemma for our attack. The key lemma is a preprocessing attack for inverting a function f which is a restricted random permutation. The attack is closely related to that of Hellman [Hel80], but we provide rigorous analysis for our specific application for completeness.

9.2.1 Generic Attack for B-Block Collisions

We give a (S,T) adversary $\mathcal{A}$ that has advantage $O(STB/2^c + T^2/2^c + T^2/2^r)$ against $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}$. The main idea for this attack is similar to the zero-walk attack for finding B-block collisions in the Merkle-Damgård construction introduced in [ACDW20] which was in turn inspired by an attack in [CDGS18].

HIGH LEVEL IDEA. In the preprocessing phase the adversary randomly samples $t \approx S$ different IVs $\mathsf{IV}_1, \ldots, \mathsf{IV}_t$ and for each of them computes $\sigma_{i,j}$ for $j \in [B/2-1]$ as $\sigma_{i,j} = \Pi(0, \sigma_{i,j-1}[2])$, where $\sigma_{i,0} = (0, \mathsf{IV})$. The sequence $\sigma_{i,0}, \ldots, \sigma_{i,B/2-1}$ forms a "zero-walk" on IV_i. It then finds m_i, m_i' such that $\Pi(m_i, \sigma_{i,B/2-1}[2])[1] = \Pi(m_i', \sigma_{i,B/2-1}[2])[1]$ for $i = 1, \ldots, t$. It outputs

$$(\sigma_{i,B/2-1}[2], m_i, m_i')_{i=1,\ldots,t} \ .$$

In the online phase, the adversary gets a challenge IV as input. For $i = 1, \ldots, T/B$, it computes $\mathsf{IV}_i = \Pi(i, \mathsf{IV})[2]$. For each of the IV_i's, it does a zero-walk of length $B-2$. If on any of the walks it hits an IV that the preprocessing phase output then it outputs a collision. The reason this attack achieves an advantage of $\Omega(STB/2^c)$ is because in the preprocessing phase the adversary roughly hits $\Omega(SB)$ distinct IVs and in the online phase if it hits any of these IV's in the first half of its T/B (i.e., in roughly $T/2$ of the queries) walks it finds a collision.

We formally state our result below.

Theorem 9.2.1. *Let $S, T, B, c, r \in \mathbb{N}$ such that $SB \leqslant 2^{c-1}$, $T \leqslant \min\{(2^{c-1}, 2^{r-1})\}$, $T \geqslant 2B$. There exists an (S, T) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,B}(\mathcal{A}) \geqslant \left\lfloor \frac{S}{c+2r} \right\rfloor \left\lfloor \frac{B}{2} - 1 \right\rfloor \frac{T}{2^{c+3}} + \frac{(T-B)(T-B-1)}{2^{c+1}} + \frac{3(T-B)(T-B-1)}{2^{r+3}} - \frac{S}{e^{(2^r-1)}} \ .$$

Proof. The adversary is $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ defined in Figure 9.4.

We first note that the probability that the preprocessing phase aborts in any of the iterations is at most

$$\frac{\prod_{i=0}^{2^r-1}(2^{c+r} - i2^c)}{\prod_{i=0}^{2^r-1}(2^{c+r} - i)} \ .$$

This follows since the numerator indicates the number of ways to sample $\Pi(\cdot, \mathsf{IV})$ such that no two outputs match in the first r bits, while the denominator is the total number of ways to sample $\Pi(\cdot, \mathsf{IV})$. The above product is upper bounded by

$$\frac{\prod_{i=0}^{2^r-1}(2^{c+r} - i2^c)}{\prod_{i=0}^{2^r-1} 2^{c+r-1}} = \prod_{i=0}^{2^r-1}(1 - i2^{1-r}) \leqslant e^{-2^{1-r}\sum_{i=0}^{2^r-1} i} = e^{-(2^r-1)} \ .$$

Using the union bound it follows that the preprocessing phase will abort with probability at most $S/e^{(2^r-1)}$.

We next calculate the probability that $\mathcal{A}_2$ succeeds in finding a collision. Let R be the event that the adversary $\mathcal{A}_2$ returns due to the if statements in line 9 or line 14 and let Q be the event that it returns due to the if statement in line 11. Clearly, adversary $\mathcal{A}_2$ succeeds with probability $\mathsf{Pr}[R] + \mathsf{Pr}[Q]$. First we compute $\mathsf{Pr}[R]$. Note that R happens if all the $\mathsf{IV}_{k,j}$'s are not distinct or if all the $\mathsf{IV}_{k,j}$'s are distinct but all

Algorithm $\mathcal{A}_1(\Pi)$

1. $t \leftarrow \lfloor S/(c+2r) \rfloor$
2. $B' \leftarrow \lfloor B/2 \rfloor - 1$
3. For $i = 1, \ldots, t$
4. $\quad \mathsf{IV}_{p,i,0} \leftarrow\!\!\$\ \{0,1\}^c$
5. $\quad$ For $j = 0, \ldots, B'$:
6. $\quad\quad \mathsf{IV}_{p,i,j} \leftarrow \Pi(0, \mathsf{IV}_{p,i,j-1})[2]$
7. $\quad \mathsf{IV}_{p,i} \leftarrow \mathsf{IV}_{p,i,B/2-1}$
8. $\quad$ Compute (m_i, m_i') such that $\Pi(m_i \,\|\, \mathsf{IV}_{p,i})[1] = \Pi(m_i' \,\|\, \mathsf{IV}_{p,i})[1]$
9. $\quad$ If no such m_i, m_i' found, ABORT
10. Return $\{(\mathsf{IV}_{p,i}, m_i, m_i')\}_{i=1,\ldots,B/2-1}$

Algorithm $\mathcal{A}_2(\mathsf{IV}, \sigma)$

1. $t \leftarrow \lfloor S/(c+2r) \rfloor$
2. $B' \leftarrow \lfloor B/2 \rfloor - 1$
3. $T' \leftarrow \lfloor T/B \rfloor$
4. Parse σ as $(\mathsf{IV}_{p,i}, m_i, m_i')_{i=1}^{t}$
5. If $\exists i \in [t] : \mathsf{IV}_{p,i} = \mathsf{IV}$: return (m_i, m_i')
6. For $k = 1, \ldots, T'$:
7. $\quad m_{k,0} \,\|\, \mathsf{IV}_{k,0} \leftarrow \Pi(k, \mathsf{IV})$
8. $\quad$ For $j = 1, \ldots, B-1$:
9. $\quad\quad$ If $\mathsf{IV}_{k',j'} = \mathsf{IV}_{k,j-1}$ for some $(k', j') \neq (k, j-1)$:
10. $\quad\quad\quad$ Return $(k' \,\|\, m_{k',0} \,\|\, m_{k',1} \,\|\, \ldots \,\|\, m_{k',j'}, k \,\|\, m_{k,0} \,\|\, m_{k,1} \,\|\, \ldots \,\|\, m_{k,j-1})$
11. $\quad\quad$ If $\mathsf{IV}_{k,j-1} = \mathsf{IV}_{p,i}$ for some $i \in [t]$:
12. $\quad\quad\quad$ Return $(k \,\|\, m_{k,0} \,\|\, \ldots \,\|\, m_{k,j-1} \,\|\, m_i, k \,\|\, m_{k,0} \,\|\, \ldots \,\|\, m_{k,j-1} \,\|\, m_i')$
13. $\quad\quad m_{k,j} \,\|\, \mathsf{IV}_{k,j} \leftarrow \Pi(0 \,\|\, \mathsf{IV}_{k,j-1})$
14. If $\exists (k, k', j, j') : m_{k,j} = m_{k'j'}$:
15. $\quad$ Return $(k \,\|\, m_{k,0} \,\|\, \ldots \,\|\, m_{k,j-1}, k' \,\|\, m_{k',0} \,\|\, \ldots \,\|\, m_{k',j'-1})$

Figure 9.4: Adversary $\mathcal{A}$ used in the proof of Theorem 9.2.1.

the $m_{k,j}$'s are not distinct. The probability that all the $\mathsf{IV}_{k,j}$'s are not distinct is at least

$$1 - \sum_{i=1}^{T'B} \left(1 - \frac{i}{2^c - i}\right) \geqslant 1 - e^{-\sum_{i=1}^{T'B} \frac{i}{2^c}} \geqslant 1 - e^{-\frac{(T-B)(T-B-1))}{2^{c+1}}} \geqslant \frac{(T-B)(T-B-1)}{2^{c+1}}$$

The second inequality follows using $1 - x \leqslant e^{-x}$ and the third inequality follows using $T'B \geqslant T - B$, and the last inequality follows using $e^{-2x} \leqslant 1 - x$ for $x \leqslant 1/2$ (this can be verified using elementary calculus). It is also easy to see that the probability that all the $\mathsf{IV}_{k,j}$'s are not distinct is at most $T^2/2^{c+1}$ using an union bound. Using $T^2 \leqslant 2^{c-1}$ we have that this probability is at most $1/4$. Hence the probability that all the $\mathsf{IV}_{k,j}$'s are distinct is at least $3/4$. Similarly we can show that the probability that all the $m_{k,j}$'s are not distinct given all $\mathsf{IV}_{k,j}$'s are distinct is at least $\frac{(T-B)(T-B-1)}{2^{r+1}}$. This would give us that

$$\mathsf{Pr}[R] \geqslant \frac{(T-B)(T-B-1)}{2^{c+1}} + \frac{3(T-B)(T-B-1)}{2^{r+3}} .$$

It is also easy to see that the probability that all the $m_{k,j}$'s are not distinct is at most $T^2/2^{r+1}$ using an union bound. Using $T^2 \leqslant 2^{r-1}$ we have that this probability is at most $1/4$. Hence the $\mathsf{Pr}[R] \leqslant 1/2$.

We next note that

$$\mathsf{Pr}[Q] + \mathsf{Pr}[R] \geqslant \mathsf{Pr}[Q|\neg R](1 - \mathsf{Pr}[R]) + \mathsf{Pr}[R] \geqslant 1/2\mathsf{Pr}[Q|\neg R] + \mathsf{Pr}[R] .$$

Above we used $\mathsf{Pr}[R] \leqslant 1/2$. Now we shall show a lower bound on $\mathsf{Pr}[Q|\neg R]$. First note that the event $\neg R$ implies that all the $\mathsf{IV}_{k,j}$'s are distinct.

Define

$$A = \{\mathsf{IV}_{p,i,j} : 1 \leqslant i \leqslant t, 0 \leqslant j \leqslant B'\} .$$

Let $F_{k,j}$ be the event that $\mathsf{IV}_{k,j} \in A$ and for all $k' < k, 0 < j' < B - 1$, $\mathsf{IV}_{k',j'} \notin A$ and for all $j' < j$, $x_{k,j'} \notin A$. We have that

$$\mathsf{Pr}[F_{k,j}|\neg R] = \frac{|A|}{2^c - ((k-1)B + j)} \prod_{i=1}^{(i-1)B+j} \left(1 - \frac{|A|}{2^c - i + 1}\right) .$$

Using $T \leqslant 2^{c-1}$ we have that $2^c - i + 1 \geqslant 2^{c-1}$. Therefore

$$\mathsf{Pr}[F_{k,j}|\neg R] \geqslant \frac{|A|}{2^c}\left(1 - \frac{2|A|}{2^c}\right)^{(k-1)B+j} \geqslant \frac{|A|}{2^c}\left(1 - \frac{2|A|((k-1)B+j)}{2^c}\right) \geqslant |A|/2^{c+1} .$$

It is clear that $F_{k,j}$ are disjoint, and given that R does not happen, Q happens if for any $k \in [T']$, in the first $B - B'$ iterations of the loop in j, $x_{i,j} \in A$. Therefore

$$\begin{aligned}\mathsf{Pr}[Q|\neg R] &\geqslant \sum_{k\in[T']} \sum_{j\in\{0,\ldots,B-1-B'\}} \mathsf{Pr}[F_{k,j}|\neg R] = (B - 1 - B')T'(|A|/2^{c+1}) \\ &= (\lceil B/2 \rceil)\lfloor T/B \rfloor |A|/2^{c+1} \geqslant T|A|/2^{c+1} .\end{aligned}$$

The last inequality follows using $T \geqslant 2B$.

Hence we have that the advantage of the adversary $\mathcal{A}$ is

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,B}(\mathcal{A}) &\geqslant \sum_{i=1}^{t(B'+1)} \frac{\mathsf{Pr}[|A| = i]Ti}{2^{c+2}} + \frac{(T-B)(T-B-1)}{2^{c+1}} + \frac{3(T-B)(T-B-1)}{2^{r+3}} - \frac{S}{e^{(2^r-1)}} \\ &= \frac{\mathbb{E}[[]A]T}{2^{c+2}} + \frac{(T-B)(T-B-1)}{2^{c+1}} + \frac{3(T-B)(T-B-1)}{2^{r+3}} - \frac{S}{e^{(2^r-1)}} \,.\end{aligned}$$

The second equality follows from the definition of expectation. We next need to show that $\mathbb{E}[[]A] \geqslant \frac{1}{2}\lfloor\frac{S}{c+2r}\rfloor\lfloor\frac{B}{2}-1\rfloor$ to finish the proof.

Let $E_{i,j}$ be the event that $\mathsf{IV}_{p,i,j}$ is a new IV discovered while preprocessing. We compute a lower bound on $\mathsf{Pr}[E_{i,j}]$ as follows.

$$\begin{aligned}\mathsf{Pr}[E_{i,j}] &\geqslant \mathsf{Pr}[E_{i,0} \cap E_{i,1} \cap \ldots \cap E_{i,j}] = \prod_{k=0}^{j} \mathsf{Pr}[E_{i,k} | E_{i,0} \cap \ldots \cap E_{i,k-1}] \\ &\geqslant \prod_{k=0}^{j} \frac{2^{c+r} - ((i-1)(B'+1)+k)2^r}{2^{c+r} - ((i-1)(B'+1)+k)} \geqslant \prod_{k=0}^{j} \frac{2^{c+r} - 2^{c+r-1}}{2^{c+r}} = 1/2 \,.\end{aligned}$$

The second inequality uses $SB \leqslant 2^{c-1}$. Hence we have that

$$\mathbb{E}[[]A] = \sum_{i=1}^{t}\sum_{j=0}^{B'} \mathsf{Pr}[E_{i,j}] \geqslant \frac{1}{2}\left\lfloor\frac{S}{c+2r}\right\rfloor\left\lfloor\frac{B}{2}-1\right\rfloor \,.$$

This concludes the proof. ∎

9.2.2 Preprocessing Attack for $B = 1$

We give a new AI-RPM attack for finding 1-block collisions in the Sponge construction. The key ingredient in our attack is an (S,T) adversary for a function f finds two distinct pre-images of a random element of the co-domain under f. We construct this adversary in Lemma 9.2.3 based on the adversary from Lemma 9.2.5 that finds a single pre-image of a random element of the co-domain under f.

Theorem 9.2.2. *Let $c, r \in \mathbb{N}$. For any $S, T \in \mathbb{N}$ such that $S \geqslant 24c$, $2^c \geqslant 24S$, and $2^c \geqslant (S/(T-2))\cdot 24^3$, there exists an (S,T) attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that on input $\{0,1\}^c$ outputs a valid 1-block collision with probability ε, where*

$$\varepsilon \geqslant \left(\frac{1}{20\cdot 288^2\cdot c^2}\right)\cdot \min\left(1, \frac{S^2(T-2)^2}{2^{2c+2}}, \left(\frac{S^2(T-2)}{2^{2c+1}}\right)^{2/3}\right).$$

Proof. Let $\Pi\colon \{0,1\}^{c+r} \to \{0,1\}^{c+r}$ be a random permutation. Define the function $f_\Pi\colon \{0,1\}^c \to \{0,1\}^c$ as $f_\Pi(x) = \Pi^{-1}(0^r \,\|\, x)[2]$. Note that f_Π is equivalent to the function that outputs the first c bits of the permutation $\Pi'(x \,\|\, 0^r)$, where $\Pi'(x \,\|\, y)$ for $x \in \{0,1\}^c, y \in \{0,1\}^r$ computes $\Pi^{-1}(y \,\|\, x)$ and shifts the first r

bits of the output to the end of its output. Thus, we can invoke Lemma 9.2.3 for the function f_Π, which implies an $(S, T-2)$ attacker $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ for finding two distinct pre-images of a random $y \leftarrow\!\!\$\ \{0,1\}^c$. The attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is defined as follows.

- $\mathcal{A}_1(\Pi)$:

  1. Output $\sigma \leftarrow \mathcal{B}_1(f_\Pi)$.

- $\mathcal{A}_2(\mathsf{IV}, \sigma)$:

  1. Compute $(x_1, x_2) \leftarrow \mathcal{B}_1(\mathsf{IV}, \sigma)$.
  2. If $f_\Pi(x_1) = f_\Pi(x_2) = \mathsf{IV}$ and $x_1 \neq x_2$, compute $m_1 \,\|\, \mathsf{IV} = \Pi^{-1}(0^r \,\|\, x_1)$ and $m_2 \,\|\, \mathsf{IV} = \Pi^{-1}(0^r \,\|\, x_2)$ and output (m_1, m_2).
  3. Otherwise, output $\perp$.

For correctness, we note that if $\mathcal{A}_2$ outputs a non-$\perp$ value, then $\mathcal{A}_2$ succeeds in finding a 1-block collision. Recall that $\mathsf{Sp}_\Pi(\mathsf{IV}, m_1) = \Pi(m_1 \,\|\, \mathsf{IV})[1]$ and $\mathsf{Sp}_\Pi(\mathsf{IV}, m_2) = \Pi(m_2 \,\|\, \mathsf{IV})[1]$. By construction, $f_\Pi(x_1) = f_\Pi(x_2) = \mathsf{IV}$ implies $m_1 \,\|\, \mathsf{IV} = \Pi^{-1}(0^r \,\|\, x_1)$ and $m_2 \,\|\, \mathsf{IV} = \Pi^{-1}(0^r \,\|\, x_2)$ for some $m_1, m_2 \in \{0,1\}^r$. But this in turn implies that $\Pi(m_1 \,\|\, \mathsf{IV})[1] = \Pi(m_2 \,\|\, \mathsf{IV})[1] = 0^r$. Since Π is a permutation and $x_1 \neq x_2$, it must be the case that $m_1 \neq m_2$, so (m_1, m_2) is a valid 1-block collision, as required.

Whenever $\mathcal{B}$ succeeds, $\mathcal{A}$ succeeds, so the success probability follows immediately from Lemma 9.2.3. ■

Lemma 9.2.3. *Let $n \geqslant 1$, $a \leqslant n-3$ and Π be a random permutation over $\{0,1\}^n$. Let $f\colon \{0,1\}^a \to \{0,1\}^a$ such that $f(x)$ consists of the first a bits output by $\Pi(x \,\|\, 0^{n-a})$. For any $S, T \in \mathbb{N}$ such that $S \geqslant 24a$, $2^a \geqslant 24S$, and $2^a \geqslant (S/T) \cdot 24^3$, there exists an (S,T) attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that on input $y \leftarrow\!\!\$\ \{0,1\}^a$ outputs x_1, x_2 such that $f(x_1) = f(x_2) = y$ and $x_1 \neq x_2$ with probability ε, where*

$$\varepsilon \geqslant \left(\frac{1}{20 \cdot 288^2 \cdot a^2}\right) \cdot \min\left(1, \frac{S^2T^2}{2^{2a+2}}, \left(\frac{S^2T}{2^{2a+1}}\right)^{2/3}\right).$$

Proof. Let $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ be an $(S/2, T/2)$ adversary from Lemma 9.2.5. In the offline phase, $\mathcal{A}_1$ on input the function f runs $\mathcal{B}_1(f)$ twice and gets σ_1, σ_2. $\mathcal{A}_1$ outputs $\sigma = (\sigma_1, \sigma_2)$. In the online phase, $\mathcal{A}_2$ on input σ and $y = f(x)$ for $x \leftarrow\!\!\$\ \{0,1\}^a$, computes $x_1 = \mathcal{B}_2(y, \sigma_1)$ and $x_2 = \mathcal{B}_2(y, \sigma_2)$. If $f(x_1) = f(x_2) = y$ and $x_1 \neq x_2$, $\mathcal{A}_2$ outputs (x_1, x_2) and otherwise outputs $\perp$. It directly follows that $\mathcal{A}$ uses space $|\sigma| = |\sigma_1| + |\sigma_2| \leqslant S$ and makes at most $2 \cdot (T/2) = T$ queries. So it remains to analyze the advantage of $\mathcal{A}$.

We define the following events that are relevant to the analysis. Let $\mathsf{Success}_1$, $\mathsf{Success}_2$ be the events that $\mathcal{B}_1(f, \sigma_1)$ and $\mathcal{B}_1(f, \sigma_2)$ output a valid pre-image, respectively. Let $\mathsf{Inverse}$ be the event that $|f^{-1}(y)| \geqslant 2$

for the challenge $y \leftarrow\!\!\$\ \{0,1\}^a$. Let $\mathsf{Distinct}$ be the event that the outputs x_1 and x_2 are distinct. Thus, the probability of success is given by

$$\varepsilon = \mathsf{Pr}[\mathsf{Success}_1 \wedge \mathsf{Success}_2 \wedge \mathsf{Inverse} \wedge \mathsf{Distinct}].$$

Note that $\mathsf{Success}_1$ and $\mathsf{Success}_2$ are identical and independently distributed given a fixed value for y. Thus, we can rewrite the success probability as

$$\begin{aligned}\varepsilon &= \mathsf{Pr}[\mathsf{Inverse}] \cdot \mathsf{Pr}[\mathsf{Success}_1 \mid \mathsf{Inverse}] \cdot \mathsf{Pr}[\mathsf{Success}_2 \mid \mathsf{Inverse}] \\ &\quad \cdot \mathsf{Pr}[\mathsf{Distinct} \mid \mathsf{Success}_1 \wedge \mathsf{Success}_2 \wedge \mathsf{Inverse}]. \\ &= \mathsf{Pr}[\mathsf{Inverse}] \cdot \mathsf{Pr}[\mathsf{Success}_1 \mid \mathsf{Inverse}]^2 \\ &\quad \cdot \mathsf{Pr}[\mathsf{Distinct} \mid \mathsf{Success}_1 \wedge \mathsf{Success}_2 \wedge \mathsf{Inverse}]\end{aligned}$$

We analyze each of these terms separately.

In Claim 9.2.4, we show that $\mathsf{Pr}[\mathsf{Inverse}] \geqslant 1/10$ as long as $a \leqslant n-3$. $\mathsf{Pr}[\mathsf{Success}_1 \mid \mathsf{Inverse}]$ is given in Lemma 9.2.5 using $S' = S/2$ and $T' = T/2$. As $|f^{-1}(y)| \geqslant 2$ by assumption, it holds that

$$\mathsf{Pr}[\mathsf{Success}_1 \mid \mathsf{Inverse}] \geqslant \left(\frac{1}{288 \cdot a}\right) \cdot \min\left(1, \frac{ST}{2^{a+1}}, \left(\frac{S^2T}{2^{2a+1}}\right)^{1/3}\right).$$

For the event $\mathsf{Distinct}$, note that in the worst case $|f^{-1}(y)| = 2$. In this case, it is equally as likely that $x_1 = x_2$ compared to $x_1 \neq x_2$ since there are only two equally likely values for x_1, x_2. Thus,

$$\mathsf{Pr}[\mathsf{Distinct} \mid \mathsf{Success}_1 \wedge \mathsf{Success}_2 \wedge \mathsf{Inverse}] = 1/2.$$

Combining the above, we conclude that the attackers probability of success is at least

$$\varepsilon \geqslant \frac{1}{2} \cdot \frac{1}{10} \cdot \left(\frac{1}{288^2 \cdot a^2}\right) \cdot \min\left(1, \frac{S^2T^2}{2^{2a+2}}, \left(\frac{S^2T}{2^{2a+1}}\right)^{2/3}\right),$$

as required. ∎

Claim 9.2.4. *Let $n \geqslant 1$, $a \leqslant n-3$ and Π be a random permutation over $\{0,1\}^n$. Let $f\colon \{0,1\}^a \to \{0,1\}^a$ such that $f(x)$ consists of the first a bits output by $\Pi(x \,\|\, 0^{n-a})$. Then, $\mathsf{Pr}[y \leftarrow\!\!\$\ \{0,1\}^a : |f^{-1}(y)| \geqslant 2] \geqslant 1/10$.*

Proof. We analyze this as a balls-into-bins problem. Specifically, we imagine each output $y \in \{0,1\}^a$ as a bin, and for each $x \in \{0,1\}^a$, we put a ball in the bin corresponding to $f(x)$. Let $x_1, \ldots, x_{2^a}$ be an arbitrary ordering of $\{0,1\}^a$ that denotes the order we place the balls into the bins. Let $X_{i,b}$ be a random variable corresponding to the set of bins with i balls inside after placing b balls via this process. Since $\sum_{i=0}^{2^a} |X_{i,2^a}| = 2^a$, to get a lower bound on $\sum_{i=2}^{2^a} \mathbb{E}[[]|X_{i,2^a}|]$, it suffices to upper bound $\mathbb{E}[[]|X_{0,2^a}| + |X_{1,2^a}|]$.

First, note that $X_{0,0} = 2^a$. In general when adding the bth ball, $X_{0,b}$ either decreases if the ball hits $X_{0,b-1}$ or stays the same otherwise. This gives the following recursive expression for $\mathbb{E}[[\,]|X_{0,b}|]$.

$$\mathbb{E}[[\,]|X_{0,b}|] = \mathbb{E}[[\,]|X_{0,b-1}|] - \mathsf{Pr}[f(x_b) \in X_{0,b-1}].$$

We claim that $\mathsf{Pr}[f(x_b) \in X_{0,b-1}] = |X_{0,b-1}| \cdot (2^{n-a})/(2^n - (b-1))$. This is because each $y \in X_{0,b-1}$ has not been mapped to, so there are 2^{n-a} values $y \,\|\, u$ that $\Pi(x_b \,\|\, 0^{n-a})$ can take that cause $f(x_b) = y$ out of $2^n - (b-1)$ total values to choose from. We can thus rewrite the recursive expression as

$$\begin{aligned}\mathbb{E}[[\,]|X_{0,b}|] &= \mathbb{E}[[\,]|X_{0,b-1}|] - \mathbb{E}[[\,]|X_{0,b-1}|] \cdot \left(\frac{2^{n-a}}{2^n - (b-1)}\right) \\ &= \mathbb{E}[[\,]|X_{0,b-1}|] \cdot \left(1 - \frac{2^{n-a}}{2^n - (b-1)}\right).\end{aligned}$$

Solving for this expression, we get

$$\begin{aligned}\mathbb{E}[[\,]|X_{0,b}|] &= \mathbb{E}[[\,]|X_{0,0}|] \cdot \prod_{i=1}^{b} \left(1 - \frac{2^{n-a}}{2^n - (i-1)}\right) \\ &\leqslant 2^a \cdot \left(1 - \frac{2^{n-a}}{2^n}\right)^b = 2^a \cdot \left(1 - \frac{1}{2^a}\right)^b.\end{aligned}$$

Thus, $\mathbb{E}[[\,]|X_{0,2^a}|] \leqslant 2^a/e$.

For $X_{1,b}$, we first note that $X_{1,0} = 0$. Using our result for $X_{0,b}$, we can again write a recursive expression for $\mathbb{E}[[\,]|X_{1,b}|]$. The key point is that $X_{1,b-1}$ increases by one if $f(x_b)$ lands in $X_{0,b-1}$ and decreases by one if $f(x_b)$ lands in $X_{1,b-1}$.

$$\mathbb{E}[[\,]|X_{1,b}|] = \mathbb{E}[[\,]|X_{1,b-1}|] - \mathsf{Pr}[f(x_b) \in X_{1,b-1}] + \mathsf{Pr}[f(x_b) \in X_{0,b-1}].$$

We already have an expression for $\mathsf{Pr}[f(x_b) \in X_{0,b-1}]$. For $\mathsf{Pr}[f(x_b) \in X_{1,b-1}]$, we note that now each $y \in X_{1,b-1}$ has been mapped to exactly once. So there are only $2^{n-a} - 1$ possible $y \,\|\, u$ values that $\Pi(x_b \,\|\, 0^{n-a})$ can take to cause $f(x_b) = y$. It follows that $\mathsf{Pr}[f(x_b) \in X_{1,b-1}] = |X_{1,b-1}| \cdot (2^{n-a} - 1)/(2^{n-a} - (b-1))$, so we can rewrite the recursive expression as

$$\begin{aligned}\mathbb{E}[[\,]|X_{1,b}|] &= \mathbb{E}[[\,]|X_{1,b-1}|] - \mathbb{E}[[\,]|X_{1,b-1}|] \cdot \left(\frac{2^{n-a} - 1}{2^n - (b-1)}\right) + \mathbb{E}[[\,]|X_{0,b-1}|] \cdot \left(\frac{2^{n-a}}{2^n - (b-1)}\right) \\ &= \mathbb{E}[[\,]|X_{1,b-1}|] \cdot \left(1 - \frac{2^{n-a} - 1}{2^n - (b-1)}\right) + 2^a \cdot \prod_{i=1}^{b-1} \left(1 - \frac{2^{n-a}}{2^n - (i-1)}\right) \cdot \left(\frac{2^{n-a}}{2^n - (b-1)}\right).\end{aligned}$$

We use the fact that $(1 - \frac{2^{n-a}}{2^n - i}) \leqslant (1 - \frac{2^{n-a}-1}{2^n - i})$ and $X_{1,0} = 0$, we can solve for an upper bound on this

recurrence as follows.

$$\begin{aligned}
\mathbb{E}[[]|X_{1,b}|] &\leqslant \mathbb{E}[[]|X_{1,0}|] \cdot \prod_{i=1}^{b}\left(1 - \frac{2^{n-a}-1}{2^n-(i-1)}\right) \\
&\quad + \left(\sum_{i=1}^{b}\frac{2^n}{2^n-(i-1)}\right)\cdot\prod_{i=1}^{b}\left(1-\frac{2^{n-a}-1}{2^n-(i-1)}\right). \\
&= \left(\sum_{i=1}^{b}\frac{2^n}{2^n-(i-1)}\right)\cdot\prod_{i=1}^{b}\left(1-\frac{2^{n-a}-1}{2^n-(i-1)}\right) \\
&\leqslant \frac{b\cdot 2^n}{2^n-2^a}\cdot\left(1-\frac{2^{n-a}-1}{2^n}\right)^b \\
&\leqslant \frac{8b}{7}\cdot\left(1-\frac{7}{8\cdot 2^a}\right)^b,
\end{aligned}$$

where the last inequality follows since $a \leqslant n-3$. Thus,

$$\mathbb{E}[[]|X_{1,2^a}|] \leqslant 2^a \cdot (8/7)\cdot e^{-7/8} \leqslant 2^a/2.$$

It follows that

$$\begin{aligned}
\sum_{i=2}^{2^a}\mathbb{E}[[]|X_{i,2^a}|] &= 2^a - \mathbb{E}[[]|X_{0,2^a}|] - \mathbb{E}[[]|X_{0,2^a}|] \\
&\geqslant 2^a\cdot(1-1/2-1/e) \\
&\geqslant 2^a/10.
\end{aligned}$$

Define the set S_i to be the set of functions f such that $|\{y : |f^{-1}(y)| \geqslant 2\}| = i$. Then

$$\begin{aligned}
\mathsf{Pr}[|f^{-1}(y)| \geqslant 2] &= \sum_{i=2}^{2^a}\mathsf{Pr}[f \in S_i]\cdot\frac{i}{2^a} \\
&= \frac{1}{2^a}\cdot\mathbb{E}[[]|\{y : |f^{-1}(y)| \geqslant 2\}|] \\
&\geqslant \frac{1}{2^a}\cdot\frac{2^a}{10} = \frac{1}{10},
\end{aligned}$$

as required. ∎

9.2.3 Time-Space Tradeoffs for Inverting a Restricted Permutation

In this section, we prove a time-space tradeoff for inverting a restricted permutation. Let $n \in \mathbb{N}$, $a, b < n$, and let $\Pi \leftarrow \mathsf{Perm}(n)$ be a randomly chosen permutation. Consider the function $f\colon \{0,1\}^a \to \{0,1\}^b$ defined such that $f(x)$ outputs the first b bits of $\Pi(x \,\|\, 0^{n-a})$. We show that there exists an (S,T) adversary $\mathcal{A}$ that inverts f with advantage roughly $\Omega(\min(1, ST/2^{\min(a,b)}, (S^2T/2^{2\min(a,b)})^{1/3}))$. Additionally, we show that on input $y = f(x)$ for a random $x \leftarrow\!\!\$\ \{0,1\}^a$, $\mathcal{A}$ outputs a uniformly random pre-image $x' \in f^{-1}(y)$ if it succeeds.

We note that our attack closely follows the approach of Hellman [Hel80] and its extension from Fiat and Naor [FN99]. We provide the full details of the attack and analysis for completeness. We emphasize that our analysis differs from Hellman's analysis since our function f is not quite a random function. Still, we do not need the full generality of the result of Fiat and Naor that works for arbitrary functions. We also note that we show how to instantiate and analyze the "g" functions (see the proof for full details) used in Hellman's attack using only pairwise independence, whereas Fiat and Naor's result for arbitrary functions required k-wise independence for $k \approx T$.

Lemma 9.2.5. *Let $n \geqslant 1$, $a, b \leqslant n$ and Π be a random permutation over $\{0,1\}^n$. Let $f\colon \{0,1\}^a \to \{0,1\}^b$ such that $f(x)$ consists of the first b bits output by $\Pi(x \,\|\, 0^{n-a})$. For any $S, T \in \mathbb{N}$ such that $S \geqslant 24\max(a,b)$, $2^{\min(a,b)} \geqslant 24S$, and $2^{\min(a,b)} \geqslant (S/T) \cdot 24^3$, there exists an (S,T) attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that succeeds in inverting f on input $y = f(x)$ for $x \leftarrow\!\!\$\, \{0,1\}^a$ with probability ε, where*

$$\varepsilon \geqslant \left(\frac{1}{288 \cdot b}\right) \cdot \min\left(1, \frac{ST}{2^{\min(a,b)}}, \left(\frac{S^2T}{2^{2\min(a,b)}}\right)^{1/3}\right).$$

Additionally, the following hold:

- *If the attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ succeeds at inverting $y = f(x)$, it outputs a uniform pre-image $x' \in f^{-1}(y)$ over the randomness of $\mathcal{A}_1$ and $\mathcal{A}_2$.*

- *For any fixed $x \in \{0,1\}^a$ and $y = f(x)$, the attack succeeds with probability at least*

$$\varepsilon \geqslant \left(\frac{1}{288 \cdot b}\right) \cdot \min\left(1, \frac{ST \cdot |f^{-1}(y)|}{2^a}, \left(\frac{S^2T|f^{-1}(y)|^2}{2^{2a}}\right)^{1/3}\right).$$

Proof. We start by describing a "Hellman table," which is the main building block in our attack. It is helpful to think of a Hellman table as the preprocessing phase of an (S,T) adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ for a restricted range of S and T. Our full adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ uses many Hellman tables to boost its success probability for any range of S and T.

Hellman tables. For any $m, \ell \in \mathbb{N}$, a Hellman table consists of m chains each of length ℓ and makes use of a common function $g\colon \{0,1\}^b \to \{0,1\}^a$ that we provide as input to $\mathcal{B}_1$ and $\mathcal{B}_2$. We will specify the function g to be used in our full construction. Let $h\colon \{0,1\}^b \to \{0,1\}^b$ be the function such that $h(y) = f(g(y))$. Denote by $h^{(j)}(y)$ the jth iteration of h on input y, so $h^{(j)}(y) = h(h(\ldots h(y)\ldots))$ consisting of j invocations of h. For the ith of m chains, we sample a random $y_{i,0} \leftarrow\!\!\$\, \{0,1\}^b$, and compute the remaining ℓ entries $y_{i,j} = h^{(j)}(y_{i,0})$ for all $j \in [\ell]$. The preprocessing phase $\mathcal{B}_1$ outputs the starting point $y_{i,0}$ and the final chain entry $y_{i,\ell}$ for each chain, $\sigma = \{(y_{i,0}, y_{i,\ell}) : i \in [m]\}$.

The online attacker $\mathcal{B}_2$ receives as input σ, some $y = f(x)$ for a random $x \leftarrow\!\!\$\ \{0,1\}^a$, and the description of a function g. $\mathcal{B}_2$ computes $h^{(j)}(y)$ for each $j \in [\ell]$ number of steps. If, for any $j \geqslant 1$, $h^{(j)}(y) = y_{i,\ell}$ for any final chain entry stored in σ, $\mathcal{B}_2$ computes $y_{i,k} = h^{(k)}(y_{i,0})$ for each $k \in [\ell]$. If $y_{i,k} = y$ for some $k \in [\ell]$, $\mathcal{B}_2$ outputs $x' = g(y_{i,k-1})$, and outputs $\perp$ if no such k is found.

To see the correctness guarantee, consider the set of points $\mathsf{Table} = \{y_{i,j} : i \in [m], j \in [\ell]\}$. We claim that if the challenge y received by $\mathcal{B}_2$ is in the set Table, then $\mathcal{B}_2$ outputs a valid pre-image $x' \in f^{-1}(y)$. Suppose $y \in S$. Then, by definition of S, we know that $y = h^{(j)}(y_{i,0})$ for some $i \in [m]$, $j \in [\ell]$. But that implies that $y_{i,\ell} = h^{(\ell)}(y_{i,0}) = h^{(\ell-j)}(y)$, so $\mathcal{B}_2$ will compute $y_{i,\ell}$ during its online attack. At this point, $\mathcal{B}_2$ will then compute $y_{i,k} = h^{(k)}(y_{i,0})$ for each $k \in [\ell]$ and see that $y_{i,j} = y$. It outputs $x' = g(y_{i,j-1})$, and it holds that $f(x') = y$ by construction, as required.

The full construction. Our full adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is as follows. For any space bound S and time bound T, we compute the parameter $\gamma \in \mathbb{N}$ as follows,

$$\gamma = \left\lceil \max\left(\frac{ST}{2^{\min(a,b)}}, \left(\frac{ST^2}{2^{\min(a,b)}} \right)^{1/3} \right) \right\rceil .$$

In the pre-processing phase, we sample functions $g_1, \ldots, g_\gamma \colon \{0,1\}^b \to \{0,1\}^a$ from a pairwise independent hash family where additionally g_i and g_j are independent for all $i \neq j \in [\gamma]$. This requires at most $6 \cdot \max(a,b)$ bits to describe, using the construction of Claim 9.2.9. Using these functions, we construct γ Hellman tables of m chains each of length ℓ where,

$$m = \left\lfloor \frac{S - 6\max(a,b)}{2\gamma b} \right\rfloor , \ell = \left\lfloor \frac{T}{6\gamma} \right\rfloor - 1.$$

For each $t \in [\gamma]$, the function g_t induces a new function $h_t(x) = f(g_t(x))$ used to construct the chains for the tth Hellman table. For all $t \in [\gamma]$, let $\sigma_t \leftarrow \mathcal{B}_1(f, g_t)$ be the output of $\mathcal{B}_1$, which depends on the function g_t as well as the choice of starting points $y_{t,1,0}, \ldots, y_{t,m,0}$ sampled by $\mathcal{B}_1$ for the tth table. $\mathcal{A}_1$ outputs $\{\sigma_t : t \in [\gamma]\}$ plus the descriptions of $g_1, \ldots, g_\gamma$.

The online attacker $\mathcal{A}_2$ receives pre-processing $\sigma = (\{\sigma_t : t \in [\gamma]\}, (g_1, \ldots, g_\gamma))$ and the challenge $y = f(x)$ for a random $x \leftarrow\!\!\$\ \{0,1\}^a$. For all $t \in [k]$, it computes a candidate pre-image $x' = \mathcal{B}_2(\sigma_t, y, g_t)$. If all such x' are invalid, $\mathcal{A}_2$ outputs $\perp$. Otherwise $\mathcal{A}_2$ outputs the first x' such that $f(x') = y$.

By the correctness guarantee of $\mathcal{B}$, it follows that if $\mathcal{A}_2$ outputs a non-$\perp$ value, it is valid pre-image of y. Let $x' \neq \perp$ be the pre-image output by $\mathcal{A}_2$, we claim that x' is a uniformly random pre-image of y. Suppose x' was first output by $\mathcal{B}_2(\sigma_t, y, g_t)$. Let $P = f^{-1}(y)$. We want to show that for all $\tilde{x} \in P$, $\mathsf{Pr}[x' = \tilde{x}] = 1/|P|$. By construction, it holds that $x' = g_t(y_{t,i,j-1})$ for some index $j \in [\ell]$ in chain $i \in [m]$. Thus, x' is uniformly distributed in $\{0,1\}^a$, so it is equally like to be equal to any $\tilde{x} \in P$, as required.

Analyzing the attack's efficiency. For space efficiency, we remark that $|\sigma_t| = 2mb$ for each table $t \in [k]$. Additionally, by Claim 9.2.9, the functions $g_1, \ldots, g_k$ requires $6 \cdot \max(a, b)$ bits to store. So, the space usage overall is

$$|\sigma| = 2mb\gamma + 6 \cdot \max(a, b) \leqslant S,$$

since $m \leqslant (S - 6 \cdot \max(a, b))/(2\gamma b)$ by construction.

To analyze the time efficiency, consider the attacker $\tilde{\mathcal{A}}_2$ that does not stop at T queries. We will revisit the analysis of $\mathcal{A}_2$ as opposed to $\tilde{\mathcal{A}}_2$ at the end of the proof after analyzing $\tilde{\mathcal{A}}_2$'s success probability. Let $\tilde{T}$ be the random variable describing the number of queries that $\tilde{\mathcal{A}}_2$ makes. Let $\widetilde{\mathsf{Success}}$ be the event that $\tilde{\mathcal{A}}_2$ finds a valid pre-image in this experiment. We analyze the expected number of queries $\tilde{T}$ conditioned on $\tilde{\mathcal{A}}_2$ finding a valid pre-image, $\mathbb{E}[[]\tilde{T} \mid \widetilde{\mathsf{Success}}]$.

For each $t \in [\gamma]$, $\mathcal{B}_2(\sigma_t, y, g_t)$ always makes at least ℓ oracle calls to f in computing $h_t^{(k)}(y)$ for all $k \in [\ell]$. For each $i \in [m]$ and $j \in [\ell]$ such that $h_t^{(j)}(y) = y_{t,i,\ell}$, $\mathcal{B}_2$ potentially makes ℓ more oracle calls to f. However, each time that $\mathcal{B}_2$ starts to search for y in one of these chains, it is either because (a) y is actually in Table or (b) $\mathcal{B}_2$ got unlucky and hit some $y_{t,i,\ell}$ value that was not preceded by y in the chain. Whenever case (a) occurs, $\mathcal{B}_2$ can terminate early at a cost of at most ℓ extra queries to f. Case (b) is known as a "false alarm" [Hel80] and causes $\mathcal{B}_2$ to make ℓ more queries to f. We show in Claim 9.2.7 that the probability a single chain in any table results in a false alarm is at most $(\ell + 1)^2/(2^{\min(a,b)})$, so the expected number of false alarms per table is at most $m(\ell + 1)^2/(2^{\min(a,b)})$. Thus, given that $\tilde{\mathcal{A}}_2$ succeeds, it will make at most $2\ell\gamma$ queries plus ℓ times the number of false alarms across all tables, so

$$\mathbb{E}[[]\tilde{T} \mid \widetilde{\mathsf{Success}}] \leqslant 2\ell\gamma + \frac{\gamma m\ell \cdot (\ell + 1)^2}{2^{\min(a,b)}}.$$

Since $m \leqslant (S/(2\gamma b))$ and $\ell + 1 \leqslant T/(6\gamma)$, it follows that $m(\ell + 1)^2 \leqslant (S/(2\gamma b)) \cdot (T^2/(36\gamma^2)) \leqslant N$, by construction. This implies that

$$\mathbb{E}[[]\tilde{T} \mid \widetilde{\mathsf{Success}}] \leqslant 3\ell\gamma \leqslant T/2,$$

where the last inequality follows since $\ell \leqslant T/(6\gamma)$.

Analyzing the attack's success probability. Again, we first analyze the hypothetical attack $\tilde{\mathcal{A}}_2$ that does not stop at T queries, and we will analyze $\mathcal{A}_2$'s success probability at the end of the proof. Recall that $\widetilde{\mathsf{Success}}$ is the probability that $\tilde{\mathcal{A}}_2$ outputs a pre-image $x' \in f^{-1}(y)$.

For each table $t \in [k]$, let $\mathsf{Table}_t = \{y_{t,i,j} = h_t^{(j)}(y_{t,i,0}) : i \in [m], j \in [\ell], y_{t,i,0} \in \sigma_t\}$ be the set of points that table t covers. For the full construction, we define $\mathsf{Table} = \cup_{t \in [\gamma]} \mathsf{Table}_t$. Since $\mathcal{A}_2$ succeeds whenever some $\mathcal{B}_2(\sigma_t, y, g_t)$ succeeds, it follows that $\mathcal{A}_2$ succeeds if $y \in \mathsf{Table}_t$ for any $t \in [\gamma]$. We first analyze the success probability for $y = f(x)$ for a random $x \xleftarrow{\$} \{0,1\}^a$, and then describe why it holds for any fixed x

as well. Note that over a random permutation Π, $y = f(x)$ for $x \leftarrow\!\!\$\ \{0,1\}^a$ is uniformly distributed over the range of f. Therefore, it follows that $\mathsf{Pr}[\widetilde{\mathsf{Success}}] \geqslant \mathbb{E}[[]|\mathsf{Table}|]/2^{\min(a,b)}$ for a random $x \leftarrow\!\!\$\ \{0,1\}^a$, so it suffices to lower bound the expected size of $|\mathsf{Table}|$. By Claim 9.2.8, it follows that

$$\mathsf{Pr}[\widetilde{\mathsf{Success}}] = \frac{\mathbb{E}[[]|\mathsf{Table}|]}{2^{\min(a,b)}} \geqslant \frac{\gamma m\ell}{6 \cdot 2^{\min(a,b)}}.$$

Now for any fixed $x \in \{0,1\}^a$ and $y = f(x)$, recall that each $y_{t,i,j}$ value added to Table from Claim 9.2.8 is of the form $f(g_t(y_{t,i,j-1}))$. So $g_t(y_{t,i,j-1})$ is a uniformly random value in $\{0,1\}^a$. Thus, the probability that $f(x) = y_{t,i,j}$ is at least $|f^{-1}(y)|/2^a$, and the probability that $f(x) \in \mathsf{Table}$ is at least $\mathbb{E}[[]|\mathsf{Table}|] \cdot |f^{-1}(y)|/2^a$.

Putting it all together. Let ε be the probability that $\mathcal{A}_2$ succeeds at inverting $y = f(x)$ for $x \leftarrow\!\!\$\ \{0,1\}^a$.

$$\begin{aligned}
\varepsilon &= \mathsf{Pr}[\mathsf{Success}] = \mathsf{Pr}[\widetilde{\mathsf{Success}} \wedge T' < T] \\
&= \mathsf{Pr}[T' < T \mid \widetilde{\mathsf{Success}}] \cdot \mathsf{Pr}[\widetilde{\mathsf{Success}}] \\
&\geqslant \left(1 - \frac{\mathbb{E}[[]T' \mid \widetilde{\mathsf{Success}}]}{T}\right) \cdot \frac{\gamma m\ell}{6 \cdot 2^{\min(a,b)}} \\
&\geqslant \frac{\gamma m\ell}{12 \cdot 2^{\min(a,b)}} \\
&\geqslant \frac{\gamma}{12 \cdot 2^{\min(a,b)}} \cdot \frac{S - 6\max(a,b)}{2\gamma b} \cdot \left(\frac{T}{6\gamma} - 1\right) \\
&\geqslant \frac{ST - 6\gamma S - 6T\max(a,b)}{144\gamma b \cdot 2^{\min(a,b)}} \\
&\geqslant \frac{ST}{288\gamma b \cdot 2^{\min(a,b)}},
\end{aligned}$$

where the last line holds if $T \geqslant 24\gamma$ and $S \geqslant 24\max(a,b)$. We note that $T \geqslant 24\gamma$ as long as

$$T \geqslant 24 \cdot \lceil \max(ST/2^{\min(a,b)}, (ST^2/2^{\min(a,b)})^{1/3}) \rceil,$$

which holds as long as $2^{\min(a,b)} \geqslant 24S$ and $2^{\min(a,b)} \geqslant (S/T) \cdot 24^3$. We assume these hold in the statement of our lemma. Note that for a fixed $x \in \{0,1\}^a$ and $y = f(x)$, the bound holds for $\varepsilon \geqslant \frac{ST \cdot |f^{-1}(y)|}{288\gamma b \cdot 2^a}$.

Finally, we note that either $\gamma = 1$, or we set γ such that

$$\gamma \geqslant \lceil \max(ST/2^{\min(a,b)}, (ST^2/2^{\min(a,b)})^{1/3}) \rceil.$$

$\gamma = 1$ corresponds to the case where $ST^2 \leqslant 2^{\min(a,b)}$. In this case, we get,

$$\varepsilon \geqslant \left(\frac{1}{288 \cdot b}\right) \cdot \left(\frac{ST}{2^{\min(a,b)}}\right)$$

Otherwise if $\gamma > 1$, which corresponds to the case where $ST^2 > 2^{\min(a,b)}$, we get the following bound,

$$\begin{aligned}\varepsilon &\geqslant \min\left(\frac{ST}{288 \cdot (ST/2^{\min(a,b)}) \cdot b \cdot 2^{\min(a,b)}}, \frac{ST}{288 \cdot ((ST^2/2^{\min(a,b)})^{1/3}) \cdot b \cdot 2^{\min(a,b)}}\right) \\ &= \frac{1}{288 \cdot b} \cdot \min\left(1, \left(\frac{S^2T}{2^{2\min(a,b)}}\right)^{1/3}\right).\end{aligned}$$

However, note that if $ST^2 > 2^{\min(a,b)}$, then

$$\left(\frac{S^2T}{2^{2\min(a,b)}}\right)^{1/3} > \left(\frac{ST}{2^{\min(a,b)}}\right),$$

so the second bound dominates in this case. This implies that the probability of success is always at least

$$\varepsilon \geqslant \left(\frac{1}{288 \cdot b}\right) \cdot \min\left(1, \frac{ST}{2^{\min(a,b)}}, \left(\frac{S^2T}{2^{2\min(a,b)}}\right)^{1/3}\right),$$

as required.

Doing the same analysis for fixed $x \in \{0,1\}^a$ and $y = f(x)$, we get that the probability of success it as least

$$\varepsilon \geqslant \left(\frac{1}{288 \cdot b}\right) \cdot \min\left(1, \frac{ST|f^{-1}(y)|}{2^a}, \left(\frac{S^2T|f^{-1}(y)|^2}{2^{2a}}\right)^{1/3}\right).$$

■

Claim 9.2.6. *Let $n \geqslant 1$, $a, b \leqslant n$ and Π be a random permutation over $\{0,1\}^n$. Let $f\colon \{0,1\}^a \to \{0,1\}^b$ such that $f(x)$ consists of the first b bits output by $\Pi(x \,\|\, 0^{n-a})$. Then*

$$\mathsf{Pr}\left[\begin{array}{l} x_1, x_2 \leftarrow\!\!\$\, \{0,1\}^a, \\ \Pi \leftarrow \mathsf{Perm}\left(\{0,1\}^n\right) \end{array} : f(x_1) = f(x_2)\right] \leqslant \frac{1}{2^a} + \frac{1}{2^b}.$$

Proof. Let Π be a random permutation, $x_1, x_2 \leftarrow\!\!\$\, \{0,1\}^a$, and $y \,\|\, u = \Pi(x_1 \,\|\, 0^{n-a})$ for $y \in \{0,1\}^b$. First, if $x_1 = x_2$, then clearly $f(x_1) = f(x_2)$. This happens with probability $1/2^a$. Otherwise, for any $x_2 \in \{0,1\}^a$ such that $x_1 \neq x_2$, we note that $\Pi(x_2 \,\|\, 0^{n-a})$ can take any value other that $y \,\|\, u$. There are $(2^{n-b} - 1)$ such values of the form $y \,\|\, u'$ for $u' \neq u$, so the probability over a random Π that the first b bits of $\Pi(x_2 \,\|\, 0^{n-a}) = y$ is $\frac{2^{n-b}-1}{2^n-1}$. This is at most $1/2^b$ since

$$\frac{1}{2^b} = \frac{2^{n-b}-1}{2^n-1} \cdot \frac{2^n-1}{2^n-2^b},$$

and $(2^n - 1)/(2^n - 2^b) \geqslant 1$ for $b \geqslant 1$. Thus,

$$\mathsf{Pr}[f(x_1) = f(x_2)] \leqslant \mathsf{Pr}[x_1 = x_2] + \mathsf{Pr}[f(x_1) = f(x_2) \mid x_1 \neq x_2] \leqslant \frac{1}{2^a} + \frac{1}{2^b},$$

as required. ■

Claim 9.2.7. *The probability that a chain results in a false alarm is at most* $\frac{(\ell+1)^2}{2^{\min(a,b)}}$.

Proof. Let $t \in [k]$ be the table considered and $i \in [m]$ the chain in the table. Let $y = f(x)$ for a random $x \leftarrow\!\!\$\, \{0,1\}^a$ be the challenge. Let $\tilde{y}_0 = y$ and define $\tilde{y}_j = h_t^{(j)}(y)$ for all $j \in [\ell]$ be the steps computed in the online phase. Recall that $y_{t,i,0} \leftarrow\!\!\$\, \{0,1\}^a$ is the initial point of the chain, and $y_{t,i,j} = h_t^{(j)}(y_{t,i,0})$ for each $j \in [\ell]$ are the subsequent points.

A false alarm occurs if $\tilde{y}_j = y_{t,i,j'}$ for any $j \in [\ell]$ and $j' \geqslant j$. This happens if the initial chain points are equal, $\tilde{y}_0 = y_{t,i,0}$, since then the chains immediately collide and we cannot find a pre-image for y. Note that if $y = \tilde{y}_0 = y_{t,i,j'}$ for some $j' > 0$, then this is not a false alarm as this allows the attacker to invert. Otherwise, there is a false alarm whenever $\tilde{y}_j = y_{t,i,j'}$ for any $j' \geqslant j > 0$, given that the preceding values are not equal. By a union bound, we upper bound this probability as

$$\begin{aligned}
&\mathsf{Pr}[\tilde{y}_0 = y_{t,i,0}] + \mathsf{Pr}[\exists j \in [\ell], j' \geqslant j, \tilde{y}_j = y_{t,i,j'}] \\
&= \mathsf{Pr}[\tilde{y}_0 = y_{t,i,0}] + \sum_{j' \geqslant j > 0} \mathsf{Pr}[\tilde{y}_j = y_{t,i,j'} \mid \forall \hat{j}' \geqslant \hat{j} > 0 \text{ s.t. } \hat{j}' < j' \wedge \hat{j} < j,\ \tilde{y}_{\hat{j}} \neq y_{t,i,\hat{j}'}].
\end{aligned}$$

First, note that the probability that $\tilde{y}_0 = y_{t,i,0}$ is at most $1/2^b$, since $y_{t,i,0}$ is a uniformly random value in $\{0,1\}^b$. Next, for any $j' \geqslant j > 0$, $y_{t,i,j'} = f(g_t(y_{t,i,j'-1}))$ and $\tilde{y}_j = f(g_t(\tilde{y}_{j-1}))$. Given that $y_{t,i,j'-1} \neq \tilde{y}_{j-1}$, it holds that $g_t(y_{t,i,j'-1})$ and $g_t(\tilde{y}_{j-1})$ are uniform and independent by pairwise independence of g_t. Thus, they collide with probability at most $1/2^a + 1/2^b$. It follows that each term in the sum above is at most $1/2^a + 1/2^b$, so the probability that a chain results in a false alarm is at most

$$\left(1 + \frac{\ell \cdot (\ell+1)}{2}\right) \cdot \left(\frac{1}{2^a} + \frac{1}{2^b}\right) \leqslant \frac{(\ell+1)^2}{2^{\min(a,b)}}.$$

■

Claim 9.2.8. $\mathbb{E}[[]|\mathsf{Table}|] \geqslant \gamma m\ell/6$.

Proof. We consider the process of adding each element in each chain for each table one at a time. To capture this, we define two sets, $\mathsf{Prog}_{t,i,j}$ and $\mathsf{TableProg}_{t,i,j}$. For each table t, chain $i \in [m]$ with starting point $y_{t,i,0} \in \sigma_t$, and value $j \in [\ell]$, recall that $y_{t,i,j} = h_t^{(j)}(g_t(y_{t,i,0}))$ is the jth element in the chain. Let $\mathsf{Prog}_{t,i,j}$ be the set containing all preceding $y_{t',i',j'}$ values before $y_{t,i,j}$ is computed. We also define a similar set per table, $\mathsf{TableProg}_{t,i,j}$, which is the set of $y_{t',i',j'}$ values for fixed $t' = t$ before $y_{t,i,j}$ is computed. For convenience, we define $\mathsf{Prog}_{t,i,0} = \mathsf{Prog}_{t,i,1}$ and $\mathsf{TableProg}_{t,i,0} = \mathsf{TableProg}_{t,i,1}$. Thus, it follows that the

expected size of Table is the probability that each value $y_{t,i,j}$ is new and not contained in the set $\mathsf{Prog}_{t,i,j}$, so

$$\begin{aligned}\mathbb{E}[[]|\mathsf{Table}|] &= \sum_{t=1}^{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{\ell}\mathsf{Pr}[y_{t,i,j} \notin \mathsf{Prog}_{t,i,j}] \\ &= \sum_{t=1}^{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{\ell}\mathsf{Pr}[y_{t,i,j} \notin \mathsf{Prog}_{t,1,1} \wedge y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j}] \\ &= \sum_{t=1}^{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{\ell}\mathsf{Pr}[y_{t,i,j} \notin \mathsf{Prog}_{t,1,1}] \cdot \mathsf{Pr}[y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j}],\end{aligned}$$

where the last equality holds since the function g_t is independent of each previous $g_{t'}$ for $t' < t$. We analyze each of these two terms in the summation separately.

First, we analyze $\mathsf{Pr}[y_{t,i,j} \notin \mathsf{Prog}_{t,1,1}]$. Recall that $y_{t,i,j} = f(g_t(y_{t,i,j-1}))$ and consider $y_{t',i',j'} = f(g_{t'}(y_{t',i',j'-1}))$ for any $t' < t, i' \in [m], j' \in [\ell]$. Since the function g_t and $g_{t'}$ are universal and pairwise independent for all $t' < t$, it holds that $g_t(y_{t,i,j-1})$ and $g_t(y_{t',i',j'-1})$ are independent and uniform on $\{0,1\}^a$. Thus, by Claim 9.2.6,

$$\mathsf{Pr}[y_{t,i,j} = y_{t',i',j'}] = \mathsf{Pr}[f(g_t(y_{t,i,j-1})) = f(g_{t'}(y_{t',i',j'-1}))] \leqslant \frac{1}{2^a} + \frac{1}{2^b}.$$

Taking a union bound over all such t', i', j', we have that

$$\mathsf{Pr}[\exists t' < t, i \in [m], j \in [\ell], y_{t,i,j} = y_{t',i',j'}] = \mathsf{Pr}[y_{t,i,j} \in \mathsf{Prog}_{t,1,1}] \leqslant (t-1)m\ell \cdot \left(\frac{1}{2^a} + \frac{1}{2^b}\right).$$

Next, we analyze $\mathsf{Pr}[y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j}]$. Note that $y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j}$ holds only if $y_{t,i,j'} \notin \mathsf{TableProg}_{t,i,j'}$ for all $1 \leqslant j' < j$ and $y_{t,i,0} \notin \mathsf{TableProg}_{t,i,1}$, since otherwise the chain has already collided with a previous chain in the table. It follows that, for all $t \in [\gamma]$, $i \in [m]$,

$$\begin{aligned}&\mathsf{Pr}[y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j}] \\ &\quad\geqslant \mathsf{Pr}[\forall 0 \leqslant j' \leqslant j, y_{t,i,j'} \notin \mathsf{TableProg}_{t,i,j'}] \\ &\quad= \mathsf{Pr}[y_{t,i,0} \notin \mathsf{TableProg}_{t,i,0}] \\ &\qquad\cdot \mathsf{Pr}[y_{t,i,1} \notin \mathsf{TableProg}_{t,i,1} \mid y_{t,i,0} \notin \mathsf{TableProg}_{t,i,0}] \cdot \ldots \\ &\qquad\cdot \mathsf{Pr}[y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j} \mid \forall 0 \leqslant j' < j, y_{t,i,j'} \notin \mathsf{TableProg}_{t,i,j'}].\end{aligned}$$

We analyze each individual term in the product above. Specifically, consider the term corresponding to $\hat{j}$, $\mathsf{Pr}[y_{t,i,\hat{j}} \notin \mathsf{TableProg}_{t,i,\hat{j}} \mid \forall j' < \hat{j}, y_{t,i,j'} \notin \mathsf{TableProg}_{t,i,j'}]$. First, we consider the case where $\hat{j} = 0$. In this case, $y_{t,i,0} \leftarrow\!\!\$\ \{0,1\}^b$ is a uniformly random value. So the probability is it in $\mathsf{TableProg}_{t,i,0}$ is at most $|\mathsf{TableProg}_{t,i,0}|/2^b \leqslant i\ell/2^b$. For $\hat{j} > 0$, we are given that $y_{t,i,\hat{j}-1} \notin \mathsf{TableProg}t, i, \hat{j} - 1$. Thus, by pairwise independence of g_t, we have that $g_t(y_{t,i,\hat{j}-1})$ and $g_t(y_{t,i',j'})$ are uniform and independent for all earlier values of i', j' in table t. Thus, $\mathsf{Pr}[y_{t,i,\hat{j}} = y_{t,i',j'}] = \mathsf{Pr}[f(g_t(y_{t,i,j-1})) = f(g_{t'}(y_{t',i',j'-1}))] \leqslant \frac{1}{2^a} + \frac{1}{2^b}$ by Claim 9.2.6,

and by a union bound

$$\mathsf{Pr}[y_{t,i,j} \in \mathsf{TableProg}_{t,i,j} \mid \forall 0 \leqslant j' < j, y_{t,i,j'} \notin \mathsf{TableProg}_{t,i,j'}] \leqslant i\ell \cdot \left(\frac{1}{2^a} + \frac{1}{2^b}\right).$$

Multiplying out all $j+1$ terms in the product, we get that

$$\mathsf{Pr}[y_{t,i,j} \notin \mathsf{TableProg}_{t,i,j}] \geqslant \left(1 - i\ell \cdot \left(\frac{1}{2^a} + \frac{1}{2^b}\right)\right)^{j+1}.$$

Combining the previous two results, we get that

$$\begin{aligned}
\mathbb{E}[[]|\mathsf{Table}|] &\geqslant \sum_{t=1}^{\gamma}\sum_{i=1}^{m}\sum_{j=1}^{\ell}\left(1-(t-1)m\ell\cdot\left(\frac{1}{2^a}+\frac{1}{2^b}\right)\right)\cdot\left(1-i\ell\cdot\left(\frac{1}{2^a}+\frac{1}{2^b}\right)\right)^{j+1} \\
&\geqslant \sum_{t=1}^{\gamma}\left(1-\frac{2(t-1)m\ell}{2^{\min(a,b)}}\right)\cdot\sum_{i=1}^{m}\sum_{j=1}^{\ell}\left(1-\frac{2i\ell}{2^{\min(a,b)}}\right)^{j+1} \\
&\geqslant \sum_{t=1}^{\gamma}\left(1-\frac{2(t-1)m\ell}{2^{\min(a,b)}}\right)\cdot\sum_{i=1}^{m}\sum_{j=1}^{\ell}e^{-(2i\ell(j+1))/(2^{\min(a,b)}-2i\ell)} \\
&\geqslant \gamma m\ell\cdot\left(1-\frac{(\gamma-1)m\ell}{2^{\min(a,b)}}\right)\cdot e^{-(4m\ell^2)/(2^{\min(a,b)}-2m\ell)}.
\end{aligned}$$

In the third inequality above, we use the fact that $(1-1/n)^{n-1} > 1/e$ for all $n \in \mathbb{N}$, so if we set $n = 2i\ell/2^{\min(a,b)}$, it follows that $\left(1-\frac{2i\ell}{2^{\min(a,b)}}\right)^{(2^{\min(a,b)}-2i\ell)/(2i\ell)} \geqslant e^{-1}$.

For the above, we note that $e^{-(4m\ell^2)/(2^{\min(a,b)}-2m\ell)} \geqslant 1/e$ by the following sequence of inequalities,

$$4m\ell^2 + 2m\ell \leqslant 6m\ell^2 \leqslant 6\left(\frac{S}{2\gamma b}\right)\left(\frac{T^2}{36\gamma^2}\right) \leqslant \frac{ST^2}{\gamma^3}\cdot\frac{1}{12b} \leqslant 2^{\min(a,b)},$$

where the last inequality holds since $\gamma \geqslant ((ST^2)/(2^{\min(a,b)}))^{1/3}$. Next, we note that $(1-(\gamma-1)m\ell/(2^{\min(a,b)}) \geqslant 1/2$ since

$$2(\gamma-1)m\ell \leqslant 2\gamma\left(\frac{S}{2\gamma b}\right)\left(\frac{T}{6\gamma}\right) \leqslant \frac{ST}{6\gamma} \leqslant 2^{\min(a,b)},$$

where the last inequality holds since $\gamma \geqslant ST/2^{\min(a,b)}$. Combining these two facts, it follows that

$$\mathbb{E}[[]|\mathsf{Table}|] \geqslant \gamma m\ell \cdot (1/2) \cdot (1/e) \geqslant \gamma m\ell/6,$$

as required. ■

Claim 9.2.9 ([FN99])**.** *Let $a, b \in \mathbb{N}$. For any $q \in \mathbb{N}$, there exists a family G of universal q-wise independent functions from $\{0,1\}^b$ to $\{0,1\}^a$ such that:*

1. *k functions $g_1, \ldots, g_k$ from G can be sampled using $2(q+1)\cdot\max(a,b)$ bits, and*

2. *for each $i \neq j \in [k]$, g_i and g_j are independent.*

Proof. The construction is taken from [FN99]. Consider the finite field $\mathbb{F}$ of size $2^{\max(a,b)}$. The construction samples $2(q+1)$ random elements from $\mathbb{F}$, $\alpha_0, \ldots, \alpha_q$ and $\beta_0, \ldots, \beta_q$. Let g'_i be the polynomial over F whose jth coefficient is $\alpha_j \cdot i + \beta_j$, and then set $g_i(x) = g'_i(x) \mod 2^a$. Pairwise independence follows since each g_i is a random degree-2 polynomial. Independence of g_i, g_j for $i \neq j$ holds since the coefficients themselves are uniform and independent between any two functions. ■

9.3 Impossibility Results

We next give impossibility results for attacks for 1-block and 2-block collisions for sponge hashing. This consists of upper bounding the best possible advantage of any (S, T) adversary.

9.3.1 Advantage Upper Bound for B=1

We prove an upper bound for the advantage of an adversary in finding a 1-block collision for the sponge construction. We use the result of [CDG18] which relates the advantage upper bound of an adversary in the AI-RPM to that in BF-RPM (bit-fixing RPM). We also argue why the bit-fixing technique cannot be used to prove a better advantage upper bound for $B = 2$ than the upper bounded for collisions of unbounded length.

In the BF model, the first stage of a two stage adversary fixes P points of the permutation Π. The permutation Π is then sampled uniformly at random from the set of permutations with the restriction of the points fixed by the first stage adversary. The second stage adversary can make T queries to Π, Π^{-1} and has to accomplish some given task involving Π. In Fig. 9.5, we formally define the game $\mathsf{G}^{\mathsf{bf\text{-}cr}}$ which captures the the bounded length collision resistance in the BF-RPM.

For parameters $c, r, B \in \mathbb{N}$, the advantage of an adversary $\mathcal{A}$ against the bounded-length collision resistance of sponge in the BF-RPM is

$$\mathsf{Adv}^{\mathsf{bf\text{-}cr}}_{\mathsf{Sp},c,r,B}(\mathcal{A}) = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}(\mathcal{A}) = \mathsf{true}\right] .$$

For parameters $P, T \in \mathbb{N}$, we overload notation and denote

$$\mathsf{Adv}^{\mathsf{bf\text{-}cr}}_{\mathsf{Sp},c,r,B}(P,T) = \max_{\mathcal{A}} \left\{ \mathsf{Adv}^{\mathsf{bf\text{-}cr}}_{\mathsf{Sp},c,r,B}(\mathcal{A}) \right\},$$

where the maximum is over all (P, T)-AI adversaries.

We restate the following proposition from [CDG18] that relates upper bounds $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,B}(S,T)$ in terms of $\mathsf{Adv}^{\mathsf{bf\text{-}cr}}_{\mathsf{Sp},c,r,B}(P,T)$.

Proposition 9.3.1. *For any $S, T, P, B \in \mathbb{N}$ and $\gamma > 0$ such that $P \geqslant (S + \log \gamma^{-1})T$ we have that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,B}(S,T) \leqslant \mathsf{Adv}^{\mathsf{bf\text{-}cr}}_{\mathsf{Sp},c,r,B}(P,T) + \gamma .$$

Game $\mathsf{G}^{\mathsf{bf\text{-}cr}}_{c,r,B}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2))$

1. $(I, Q) \leftarrow \mathcal{A}_1$ // $|I| = |Q|$
2. $S \leftarrow \varnothing$
3. For $j \in [|I|]$ set $\Pi(I[j]) \leftarrow Q[j]$, $\mathsf{S}_1 \leftarrow \mathsf{S}_1 \cup \{Q[j]\}$, $\mathsf{S}_2 \leftarrow \mathsf{S}_2 \cup \{I[j]\}$
4. For $j \in [2^{c+r}] \backslash \mathsf{S}_2$ sample $\Pi[j] \xleftarrow{\$} [2^{c+r}] \backslash \mathsf{S}_1$, set $\mathsf{S}_1 \leftarrow \mathsf{S}_1 \cup \{\Pi[j]\}$
5. $(\alpha, \alpha') \xleftarrow{\$} \mathcal{A}_2^{\Pi, \Pi^{-1}}(I, Q, \mathsf{IV})$
6. Return true if:
7. $\alpha \neq \alpha'$ and $|\alpha|, |\alpha'|$ are at most B blocks long, and $\mathsf{Sp}_\Pi(\mathsf{IV}, \alpha) = \mathsf{Sp}_\Pi(\mathsf{IV}, \alpha')$

Figure 9.5: The bounded-length collision resistance game of salted sponge based hash in the BF-RPM, denoted $\mathsf{G}^{\mathsf{bf\text{-}cr}}_{c,r,B}$.

We prove the following theorem that upper bounds the advantage of finding 1-block collisions for sponge in the AI-RPM.

Theorem 9.3.2. *For all $S, T, c, r \in \mathbb{N}$*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,1}(S, T) \leqslant \frac{2(S + c)T + 1}{2^c} + \frac{T^2}{2^r} .$$

Proof. The proof of the theorem follows directly from from setting $P = (S+c)T$, $\gamma = 2^{-c}$ in Proposition 9.3.1 and the claim below.

Claim 9.3.3. *For all $P, T, c, r \in \mathbb{N}$*

$$\mathsf{Adv}^{\mathsf{bf\text{-}cr}}_{\mathsf{Sp},c,r,1}(P, T) \leqslant P/2^c + T^2/2^{r+1} .$$

Proof. If IV which is given as input to $\mathcal{A}_2$ is such that $(*, \mathsf{IV})$ not among any of the P points, IV's then the probability that $\mathcal{A}_2$ finds a 1-block collision is bounded by the birthday bound i.e., $T^2/2^{r+1}$. The probability that $(*, \mathsf{IV})$ is among the P fixed points is at most $P/2^c$ and hence the claim follows. ■

■

Bit-fixing upper bound for $B = 2$ is trivial. We note that we cannot prove an advantage upper bound better than $O(ST^2/2^c)$ for $B = 2$, since there is a (P, T) adversary that has advantage $PT/2^c$ in finding 2-block collisions. This adversary chooses $P/2$ tuples (y_i, x_i, x'_i) for $i \in [P/2]$ such that $y_i \in \{0,1\}^c$, $x_i, x'_i \in \{0,1\}^r$, $x_i \neq x'_i$. It sets the P points of Π such that $\Pi(x_i, y_i)[1] = \Pi(x'_i, y_i)[1]$ for $i \in [P/2]$.

The second stage of the adversary on getting IV as input queries (m, IV) for T different m's. With probability roughly $PT/2^c$ it succeeds in finding m such that $\Pi(m, \mathsf{IV}) = (*, y_i)$ for some y_i chosen by the first stage adversary. In this case it can output a 2 block collision of $(m, \Pi(m, \mathsf{IV})[1] \oplus x_i), (m, \Pi(m, \mathsf{IV})[1] \oplus x_i')$ with probability $\Omega(PT/2^c)$. This would translate to an upper bound no better than $O(ST^2/2^c)$.

9.3.2 Advantage Upper Bound for $B = 2$

In this section we prove an upper-bound the advantage of an adversary in finding a 2-block collision for the sponge construction in the AI-RPM, according to the game $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{c,r,B}$ described in Fig. 9.3. First, without loss of generality, in what follows we assume that the adversary is deterministic. This is because we can transform any probabilistic attacker into a deterministic one by hard-wiring the best randomness (see Adleman [Adl78]).

We reduce the task of bounding the advantage of an attacker in finding a 2-block collision in the sponge construction, to a "multi-instance" game where the adversary does not have a preprocessing phase but rather only has non-uniform auxiliary input, chosen *before* the random permutation Π. The latter game is easier to analyze. This is in line with the work of Akshima et al. [ACDW20].

We define the following "multi-instance" game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{c,r,B,u}(\sigma, \mathcal{A}_2)$, where the preprocessing part of the adversary $\mathcal{A}_1$ is degenerate and outputs the fixed string σ. More precisely, the game has the following steps:

1. $\Pi \leftarrow\!\!\$\ \mathsf{Perm}(\{0,1\}^{c+r})$
2. $U \leftarrow\!\!\$\ \binom{\{0,1\}^c}{u}$
3. Define $\mathcal{A}_1$ to be the algorithm that always outputs the string σ.
4. Return true if $\mathsf{AI\text{-}CR}_{\Pi,\mathsf{IV}}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)) = \mathsf{true}$ for every $\mathsf{IV} \in U$. Otherwise, return false.

For a string σ and an adversary $\mathcal{A}_2$, define

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{Sp},c,r,B,u}(\sigma, \mathcal{A}_2) = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{mi\text{-}cr}}_{c,r,B,u}(\sigma, \mathcal{A}_2)\right].$$

Lemma 9.3.4 (Reducing the problem to the multi-instance game)**.** *Fix $c, r, B, S, T, u \in \mathbb{N}$. Then,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,B}(S,T) \leqslant 6 \cdot \left(\max_{\sigma,\mathcal{A}_2}\left\{\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{Sp},c,r,B,u}(\sigma,\mathcal{A}_2)\right\}\right)^{\frac{1}{u}} + 2^{S-u},$$

where the maximum is taken over all $\sigma \in \{0,1\}^S$ and T-query algorithms $\mathcal{A}_2$.

We next prove an upper bound on the advantage of any auxiliary-input adversary in finding a 2-block collision for the sponge construction. The main theorem is stated next.

Theorem 9.3.5. *For any $c, r, S, T \in \mathbb{N}$ and fixing $\hat{S} := S + c$, it holds that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{Sp},c,r,2}(S,T) \leqslant \left(2^7 e \cdot \max\left\{\frac{T^2}{2^{c-1}}, \frac{T^2}{2^{r-1}}, \frac{\hat{S}T}{2^{c-3}}, \frac{\hat{S}T}{2^{r-1}}, \frac{\hat{S}^2T^4}{2^{2c-2}}\right\}\right) + \frac{1}{2^c}\ .$$

Theorem 9.3.5 follows as a direct corollary of Lemma 10.4.1 together with the following lemma, setting $u = S + c$ and observing that **(1)** the lemma holds trivially when $\frac{T^2}{2^{r-1}} > 1$, **(2)** $\frac{uT^3}{2^{c+r-2}} \leqslant \frac{uT}{2^{c-1}}$ whenever $\frac{T^2}{2^{r-1}} \leqslant 1$, **(3)** the lemma holds trivially when $\frac{uT^2}{2^{c-1}} > 1$, and **(4)** $\frac{u^2T^3}{2^{c+r-2}} \leqslant \frac{uT}{2^{c-1}}$ whenever $\frac{uT^2}{2^{c-1}} \leqslant 1$.

Lemma 9.3.6 (Hardness of the multi-instance game)**.** *Fix c, r, T, $u \in \mathbb{N}$ and $\sigma \in \{0,1\}^S$. Then, for any $\mathcal{A}_2$ that makes at most T queries to its oracle, it holds that*

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{Sp},c,r,2,u}(\sigma,\mathcal{A}_2) \leqslant \left(2^7 \cdot e \cdot \max\left\{\frac{T^2}{2^{c-1}}, \frac{T^2}{2^{r-1}}, \frac{uT}{2^{c-3}}, \frac{uT^3}{2^{c+r-2}}, \frac{u^2T^3}{2^{c+r-2}}, \frac{u^2T^4}{2^{2c-2}}\right\}\right)^u.$$

The rest of this section is devoted to the proof of Lemma 9.3.6.

We are interested in bounding the advantage of the best strategy, i.e., a pair $(\sigma, \mathcal{A}_2)$ where $\sigma \in \{0,1\}^S$ is a fixed string and $\mathcal{A}_2$ is a T-query algorithm, of finding collisions of length 2 in a sponge with respect to the game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{c,r,2,u}(\sigma,\mathcal{A}_2)$. Recall that in this game $\mathcal{A}_2$ needs to find proper collisions for u randomly chosen IVs, denoted U. The main idea in the proof is to use any such adversary $(\sigma, \mathcal{A}_2)$ in order to represent the permutation Π *as well as* the set of random IVs U with as few bits as possible. If the adversary is "too good to be true" we will get an impossible representation, contradicting Proposition 11.1.1.

Setup. Denote

$$\zeta^* := \log\left(\left(2^5 \cdot 4e \cdot \max\left\{\frac{T^2}{2^{c-1}}, \frac{T^2}{2^{r-1}}, \frac{uT}{2^{c-3}}, \frac{uT^3}{2^{c+r-2}}, \frac{u^2T^3}{2^{c+r-2}}, \frac{u^2T^4}{2^{2c-2}}\right\}\right)^u \cdot \binom{2^c}{u} \cdot (2^{c+r})!\right).$$

Assume the existence of an adversary $\mathcal{A} = (\sigma, \mathcal{A}_2)$, where $\sigma \in \{0,1\}^S$ is a string and $\mathcal{A}_2$ is a T-query adversary, that contradict the inequality stated in the lemma. That is, there is $\zeta > \zeta^*$ such that

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{\mathsf{Sp},c,r,2,u}(\mathcal{A}) := \zeta > \zeta^*. \tag{9.3}$$

Define $\mathcal{G}$ to be the set of permutations-sets of IV pairs for which the attacker succeeds in winning the game for every IV in the set relative to the permutation, That is,

$$\mathcal{G} = \left\{(U,\Pi) \;\middle|\; \begin{array}{c} U \in \binom{\{0,1\}^c}{u}, \\ \Pi \in \mathsf{Perm}(\{0,1\}^{c+r}), \end{array} \forall \mathsf{IV} \in U : \mathsf{AI\text{-}CR}_{\Pi,\mathsf{IV}}(\mathcal{A}) = \mathsf{true}\right\}.$$

Recall that ζ is defined to be the advantage of $\mathcal{A}$ in the game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{c,r,2,u}(\mathcal{A})$ in which Π and U are chosen uniformly, and then $\mathcal{A}$ needs to find a collision with respect to every one of the u IVs in U. Therefore,

$$|\mathcal{G}| = \zeta \cdot \binom{2^c}{u} \cdot (2^{c+r})!.$$

In what follows we define an encoding and a decoding procedure such that the encoding procedure gets as input U, Π such that $U \in \binom{\{0,1\}^c}{u}$ and $\Pi \in \mathsf{Perm}(\{0,1\}^{c+r})$, and it outputs an L bit string, where $L = \log\left(\zeta^* \cdot \binom{2^c}{u} \cdot (2^{c+r})!\right)$. The decoding procedure takes as input the string L and outputs U^*, Π^*. It will hold that $U^* = U$ and $\Pi^* = \Pi$ with probability ζ.[2] Using Proposition 11.1.1, this would give us that

$$\log \zeta \leqslant L - \log\left(\binom{2^c}{u} \cdot (2^{c+r})!\right) \Longrightarrow \zeta \leqslant \zeta^*$$

which is a contradiction to the assumption (see (9.3)).

Using $\mathcal{A}$, we shall define procedures $\mathsf{Encode}, \mathsf{Decode}$ such that for every $(U, \Pi) \in \mathcal{G}$, $\mathsf{Decode}(\mathsf{Encode}(U, \Pi)) = (U, \Pi)$ and the size of the output of $\mathsf{Encode}(U, \Pi)$ is at most L bits where

$$L = \log\left(\left(2^5 \cdot 4e \cdot \max\left\{\frac{T^2}{2^{c-1}}, \frac{T^2}{2^{r-1}}, \frac{uT}{2^{c-3}}, \frac{uT^3}{2^{c+r-2}}, \frac{u^2T^3}{2^{c+r-2}}, \frac{u^2T^4}{2^{2c-2}}\right\}\right)^u \cdot \binom{2^c}{u} \cdot (2^{c+r})!\right) .$$

Using Proposition 11.1.1, this would give us that

$$\varepsilon \leqslant \left(2^7 \cdot e \cdot \max\left\{\frac{T^2}{2^{c-1}}, \frac{T^2}{2^{r-1}}, \frac{uT}{2^{c-3}}, \frac{uT^3}{2^{c+r-2}}, \frac{u^2T^3}{2^{c+r-2}}, \frac{u^2T^4}{2^{2c-2}}\right\}\right)^u .$$

This immediately gives the bound claimed in the statement of the lemma. The rest of the proof of the lemma would define $\mathsf{Encode}, \mathsf{Decode}$, show an upper bound on the size of the output of Encode and that $\mathsf{Decode}(\mathsf{Encode}(U, \Pi)) = (U, \Pi)$ for all $(U, \Pi) \in \mathcal{G}$.

Notation and Definitions. Fix $(U, \Pi) \in \mathcal{G}$. Let $U = \{\mathsf{IV}_1, \ldots, \mathsf{IV}_u\}$ where the IV_i's are ordered lexicographically. Let $\mathsf{Qrs}(\mathsf{IV}) \in (\{0,1\}^{r+c})^T$ be the list of queries that $\mathcal{A}_2$ makes to Π or Π^{-1} when executed with input (σ, IV). Namely, for $\mathsf{IV} \in \{0,1\}^c$,

$$\mathsf{Qrs}(\mathsf{IV}) = \left\{s \in \{0,1\}^{c+r} \mid \mathcal{A}_2(\sigma, \mathsf{IV}) \text{ queries } \Pi \text{ or } \Pi^{-1} \text{ on } s\right\}$$

Note that $\mathsf{Qrs}(\mathsf{IV})$ is indeed a set as we can assume (without loss of generality) that $\mathcal{A}_2$ never repeats queries in a single execution (since $\mathcal{A}_2$ can just remember all of its past queries).

Let $\mathsf{Ans}(\mathsf{IV}) \in (\{0,1\}^{r+c})^T$ be the list of answers to the queries of that $\mathcal{A}_2$ to Π or Π^{-1} when executed with input (σ, IV). Namely, for $\mathsf{IV} \in \{0,1\}^c$,

$$\mathsf{Ans}(\mathsf{IV}) = \left\{s \in \{0,1\}^{c+r} \mid \mathcal{A}_2(\sigma, \mathsf{IV}) \text{ queries } \Pi \text{ or } \Pi^{-1} \text{ on } s\right\}$$

We say that $\mathsf{IV}' \in \mathsf{SIVs}(\mathsf{IV})$ if there is some $s[2] \in \{0,1\}^r$ such that $s[2] \,\|\, \mathsf{IV}'$ is an entry in $\mathsf{Qrs}(\mathsf{IV})$ or $\mathsf{Ans}(\mathsf{IV})$. Namely, for $\mathsf{IV}, \mathsf{IV}' \in \{0,1\}^c$,

$$\mathsf{IV}' \in \mathsf{SIVs}(\mathsf{IV}) \Longleftrightarrow \exists s[2] \in \{0,1\}^r \text{ s.t. } s[2] \,\|\, \mathsf{IV}' \in \mathsf{Qrs}(\mathsf{IV}) \cup \mathsf{Ans}(\mathsf{IV}).$$

---

[2] Essentially, we will show that for all $(U, \Pi) \in |\mathcal{G}|$, if the encoding procedure produces output L, then the decoding procedure on input L outputs U^*, Π^* such that $U^* = U$ and $\Pi^* = \Pi$.

We define the set of *fresh* IVs in U. An IV IV_i for $i \in [u]$ is called fresh if it was never an IV in either input or output of any query performed by $\mathcal{A}_2$ while being executed on IV_j for $j \leqslant i-1$ which are fresh. The first IV IV_1 is always fresh. An IV IV_i for $i \geqslant 2$ is fresh if for any fresh IV_j for $j \leqslant i-1$, $\mathsf{IV}_i \notin \mathsf{SIVs}(\mathsf{IV}_j)$. Namely, denoting the set of fresh IVs by U_{fresh}, we have the following inductive (on $i \in [u]$) definition:

$$\mathsf{IV}_i \in U_{\mathsf{fresh}} \iff \forall j \leqslant i-1, \mathsf{IV}_j \in U_{\mathsf{fresh}} \colon \mathsf{IV}_i \notin \mathsf{SIVs}(\mathsf{IV}_j).$$

Looking ahead, we define U_{fresh} like this because we run $\mathcal{A}_2$ on the IVs in U_{fresh} in lexicographical order, and this definition ensures that each IV that $\mathcal{A}_2$ is executed on was not queried by it previously. Denote

$$F := |U_{\mathsf{fresh}}| \quad \text{and} \quad U_{\mathsf{fresh}} = \{\mathsf{IV}'_1, \ldots, \mathsf{IV}'_F\} \text{ (ordered lexicographically).}$$

Denote

$$\forall i \in [F] \colon \mathsf{Q}_i := \mathsf{Qrs}(\mathsf{IV}'_i) \quad \text{and} \quad \mathsf{Q}_{\mathsf{fresh}} := \mathsf{Q}_1 \,\|\, \mathsf{Q}_2 \,\|\, \ldots \,\|\, \mathsf{Q}_F,$$

where $\|$ is the concatenation operator. Let $\mathsf{Q}_{\mathsf{fresh}}[r]$ be the rth query in the list $\mathsf{Q}_{\mathsf{fresh}}$. Note that $r \in [F \cdot T]$. For every $\mathsf{IV} \in U \backslash U_{\mathsf{fresh}}$, let t_{IV} be the minimum value such that $\mathsf{Q}_{\mathsf{fresh}}[t_{\mathsf{IV}}]$ is a query either with input or output of the form $(*, \mathsf{IV})$. Let $b_{\mathsf{IV}} = 0$ if input of $\mathsf{Q}_{\mathsf{fresh}}[t_{\mathsf{IV}}]$ was of the form $(*, \mathsf{IV})$ and 1 otherwise. Define the set of *prediction* queries as

$$\mathsf{P} := \{2t_{\mathsf{IV}} - b_{\mathsf{IV}} \mid \mathsf{IV} \in U \backslash U_{\mathsf{fresh}}\}.$$

The encoding algorithm will output $U_{\mathsf{fresh}}, \mathsf{P}$, which suffices to recover the set U by running $\mathcal{A}_2$.

Structure of collisions. Since adversary $\mathcal{A}_2$ succeeds on all of the IVs in U, it holds that for every $j \in [F]$, the output of the adversary is (α_j, α'_j) such that $\alpha_j \neq \alpha'_j$, $\mathsf{Sp}_\Pi(\mathsf{IV}'_j, \alpha_j) = \mathsf{Sp}_\Pi(\mathsf{IV}'_j, \alpha'_j)$ and both $\alpha_j \neq \alpha'_j$. We can assume without loss of generality that the last blocks of α_j and α'_j are distinct (because otherwise we can trim α_j, α'_j to obtain a shorter collision).

Definition 9.3.7 (Crucial queries)**.** *The queries to Π, Π^{-1} in Q_j include a subset of queries that we call the* crucial queries*. The subset consists of earliest appearing queries in Q_j that are required to compute $\mathsf{Sp}_\Pi(\mathsf{IV}'_j, \alpha_j)$ and $\mathsf{Sp}_\Pi(\mathsf{IV}'_j, \alpha'_j)$. It follows that for 2-block collisions, this subset consists of at most four queries.*

We say that a query made by running while running on (σ, IV'_j) is NEW if either of the following hold.

- the query is $\Pi(m, \mathsf{IV})$ with answer (m', IV') and neither $\Pi(m, \mathsf{IV})$ or $\Pi^{-1}(m', \mathsf{IV}')$ had been queried by $\mathcal{A}_2$ while running on $\mathsf{IV}'_1, \ldots, \mathsf{IV}'_{j-1}$.

- the query is $\Pi^{-1}(m, \mathsf{IV})$ with answer (m', IV') and neither $\Pi^{-1}(m, \mathsf{IV})$ or $\Pi(m', \mathsf{IV}')$ had been queried by $\mathcal{A}_2$ while running on $\mathsf{IV}'_1, \ldots, \mathsf{IV}'_{j-1}$.

If a query is not NEW we classify it into one of 2 types: REPEATEDUSED, and REPEATEDUNUSED. A REPEATEDUSED query is one such that it was a crucial query for IV'_i where $i < j$. A REPEATEDUNUSED query is one such that it is not a NEW or a REPEATEDUSED query.

Our goal is to compress (U, Π) and we are going to achieve this by using our collision finding adversary $\mathcal{A}_2$. The encoding procedure shall output the set U_{fresh}, the set P, the list $\tilde{\Pi}$ with some entries removed and some additional lists and sets. We will be describing the details of these lists and sets below and which entries we remove from $\tilde{\Pi}$. It is instructive to think about how decoding would work to understand the intuition behind out encoding strategy. The decoding procedure will run $\mathcal{A}_2$ on the IVs in U_{fresh} and answer its queries using the entries in $\tilde{\Pi}$ and the other lists and sets it gets as input. We need to encode enough information in these lists and sets so that decoding procedure can correctly answer the queries whose answers will be removed from $\tilde{\Pi}$. The IVs in $U \backslash U_{\mathsf{fresh}}$ will be recovered using P. Our main goal is to show that when we remove entries of $\tilde{\Pi}$ and instead using additional lists and set, we are actually compressing. Our ways to compress will depend on the crucial queries in each Q_j for $j \in [F]$.

We classify the IV'_jth for each $j \in [F]$ into the first of the following cases it satisfies, e.g., if the crucial queries for IV'_j satisfies both cases 1 and 2, we categorize it into 1.

1. One of the crucial queries for IV'_j is a query such that the last c bits of the answer is IV'_j
2. All of the crucial queries to are NEW.
3. At least one of the crucial queries is REPEATEDUSED.
4. There is exactly one REPEATEDUNUSED crucial query.
5. There are exactly two REPEATEDUNUSED crucial queries.

Claim 9.3.8. *We claim that each IV'_j will be categorized into one of the above cases.*

Proof. To begin with, observe that given how we define *fresh* IV'_j, there will have to be a NEW crucial query such that either it is a query to Π with input of the form $(*, \mathsf{IV}'_j)$ or it is a query to Π^{-1} with answer of the form $(*, \mathsf{IV}'_j)$, because we have assumed without loss of generality that $\mathcal{A}_2$ makes all the queries while running on IV'_j required to find the collision. If there is a NEW crucial query to Π^{-1} with answer of the form $(*, \mathsf{IV}'_j)$ then case 1 is satisfied. Also observe that there are at most two crucial queries that are not NEW since we are

looking at 2-block collisions because any query whose input or output is of the form $(*, \mathsf{IV}'_j)$ is NEW for all IV'_j as they are fresh. So it follows if IV'_j is not categorized into 1, it will either have all NEW crucial queries (case 2) or have at least one REPEATEDUSED query (case 3) or have one (case 4) or two REPEATEDUNUSED (case 5) queries. This proves the claim. ■

COMPRESSION BUDGET. Recall that we need to prove that the size of the output of the encoding procedure is

$$L = \log\left(\left(2^5 \cdot 4e \cdot \max\left\{\frac{T^2}{2^{c-1}}, \frac{T^2}{2^{r-1}}, \frac{uT}{2^{c-3}}, \frac{uT^3}{2^{c+r-2}}, \frac{u^2T^3}{2^{c+r-2}}, \frac{u^2T^4}{2^{2c-2}}\right\}\right)^u \cdot \binom{2^c}{u} \cdot (2^{c+r})!\right)$$

bits.

In other words, we need to show that the encoding procedure saves at least

$$\begin{aligned} & u \cdot \min\{\min\{(c-1),(r-1)\} - 2\log T, (c-1) - \log 4uT, \\ & \qquad c + r - 2 - \log uT - 2\log T, c + r - 2 - 2\log uT - \log T, 2c - 2 - 2\log T - 2\log uT\} \\ & \qquad - 5u - u\log 4e \end{aligned} \tag{9.4}$$

bits overall.

We make the following claim.

Claim 9.3.9. *To show that the compression indeed achieves the savings from (9.4), for every* IV *in* U_{fresh}*, it suffices to save at least the following number of bits.*

$$\begin{aligned} & \min\{\min\{(c-1),(r-1)\} - 2\log T, (c-1) - \log 4FT, \\ & \qquad c + r - 2 - \log FT - 2\log T, c + r - 2 - \log 2FT - \log T, 2c - 2 - 2\log T - 2\log FT\}\ . \end{aligned}$$

Proof. First we look at the amount of savings we get from $\tilde{\Pi}$.

Required savings in $\tilde{\Pi}$. As mentioned earlier, the output of the encoding algorithm will consist of $U_{\mathsf{fresh}}, \mathsf{P}, \tilde{\Pi}$, and some additional sets and lists. The lists U_{fresh} and P will suffice to recover the set U. The list $\tilde{\Pi}$ along with the additional sets and lists are used to recover Π.

First we compute the amount of bits we save for encoding U, then that would give us the amount of bits we need to save for encoding Π. Denoting $|U_{\mathsf{fresh}}| = F$ and $|U| = u$, we can describe P using $\binom{FT}{u-F}$ bits. Therefore, U, which is trivially described using $\log\binom{2^c}{u}$ bits, can be encoded using $\log\left(\binom{2FT}{u-F}\binom{2^c}{F}\right)$ bits.

Therefore, the saving in bits in the description of U is at least

$$\begin{aligned}
&\log\binom{2^c}{u} - \log\left(\binom{2FT}{u-F}\binom{2^c}{F}\right) \geqslant \log\left(\frac{\left(\frac{2^c}{u}\right)^u}{\left(\frac{2eFT}{u-F}\right)^{u-F}\left(\frac{e2^c}{F}\right)^F}\right) \\
&= \log\left(\frac{2^{c(u-F)}(u-F)^{u-F}F^F}{e^u 2^{u-F}(FT)^{u-F}u^u}\right) = (u-F)\log\left(\frac{2^{c-1}}{FT}\right) + \log\left(\frac{(u-F)^{u-F}F^F}{e^u u^u}\right) \\
&= (u-F)\log\left(\frac{2^{c-1}}{FT}\right) - \log\left(e^u\left(\frac{u}{F}\right)^F\left(\frac{u}{u-F}\right)^{u-F}\right) \\
&\geqslant (u-F)\log\left(\frac{2^{c-1}}{uT}\right) - u\log 4e\,. \qquad (9.5)
\end{aligned}$$

where the first inequality uses the basic bounds for binomial coefficients $(n/r)^r \leqslant \binom{n}{r} \leqslant (en/r)^r$, and the last inequality follows since $x \leqslant 2^x$ for every $x \geqslant 0$, and $u \geqslant F$.

By subtracting (9.5) (how much we save in U) from (9.4) (how much we need to save in total), it suffices to show that we save at least

$$\begin{aligned}
&u \cdot \min\{\min\{(c-1),(r-1)\} - 2\log T, (c-1) - \log 4uT, c+r-2-\log uT - 2\log T, \\
&\quad c+r-2-2\log uT - \log T, 2c-2-2\log T - 2\log uT\} - 5u - (u-F)\log\left(\frac{2^c}{uT}\right) \qquad (9.6)
\end{aligned}$$

bits while encoding Π. Observe that

$$\begin{aligned}
&\log(2^c/uT) = c - \log uT \geqslant \min\{\min\{(c-1),(r-1)\} - 2\log T, \\
&\quad (c-1) - \log 4uT, c+r-2-\log uT - 2\log T, 2c-2-2\log T - 2\log uT\}\,.
\end{aligned}$$

Using this observation and $u \geqslant F$, we have that the expression in (9.6) is at least

$$\begin{aligned}
&F \cdot \min\{\min\{(c-1),(r-1)\} - 2\log T, (c-1) - \log 4FT, \\
&\quad c+r-2-\log FT - 2\log T, c+r-2-2\log FT - \log T, 2c-2-2\log T - 2\log FT\} - 5u \qquad (9.7)
\end{aligned}$$

bits.

We will label each of the IVs in F with few bits that described its "type" (according to the cases described above – case 1 will have 2 subcategories, case 2 will have 4 subcategories, case 3 will have 2 subcategories, case 4 will have 13 subcategories and case 5 will have 8 subcategories), and for this 5 bits will suffice. This will cost, in total, another $5u$ bits, and therefore for each IV in U_{fresh} it suffices to save the following number of bits.

$$\begin{aligned}
&\min\{\min\{(c-1),(r-1)\} - 2\log T, (c-1) - \log 4FT, \\
&\quad c+r-2-\log FT - 2\log T, c+r-2-2\log FT - \log T, 2c-2-2\log T - 2\log FT\}\,.
\end{aligned}$$

■

We make the following observation- the values of Π which are never queried by the adversary will be stored at the end of $\tilde{\Pi}$ and these will be at least $2^{c+r} - T \cdot 2^c$ values (since on every IV there will be T queries made). Since we have assumed $r \geqslant 2$ and $T \leqslant 2^{r/2}$, it follows that at least 2^{c+r-1} values will be at the end of $\tilde{\Pi}$- meaning for every value of Π we will remove from $\tilde{\Pi}$ to recover from m bits of other data structures, we make a saving of $c + r - 1 - m$ bits.

For each of these cases we will devise strategies to compress the amount of specified amount of bits. Before diving into the details, we give a brief glimpse of the complexity of this task in terms of handling a large number of subcases for some of the cases. Cases 1 to 3 are fairly simple in this regard (they do not lead to a large number of sub-cases). Handling cases 4 and 5 requires way more care since they give rise to many subcases.

For instance in case 4 where there is one crucial query that is REPEATEDUNUSED, one of these possibilities arise: namely **a)** there are only two crucial queries **b)** there are three crucial queries, or **c)** there are four crucial queries. For the possibility **b)** we have three different scenarios depending on the structure formed by the crucial queries – the collision could be caused by either the answer of the two NEW queries sharing common bits with the input and answer of the REPEATEDUNUSED query, or the input of one NEW query sharing bits with the input/answer of the REPEATEDUNUSED query and the output of the same NEW query sharing bits with the answer/input of the REPEATEDUNUSED query or the answer of one NEW query sharing common bits with the answer or input of the other NEW query.

Further depending on whether queries were made in the forward direction or the reverse direction, even more possibilities come up. Similarly for case 5, there are numerous possibilities that arise. We need to give a strategy covering all of these possibilities, making the compression argument extremely cumbersome by design.

We now describe the details of how we handle each case. Assuming that the crucial queries for IV'_j satisfies a particular case, we describe the encoding and decoding procedures, and the amount of savings. We provide this locally verifiable view for ease of parsing- in Section 9.3.3, we also include the full descriptions of the encoding and decoding procedures for the sake of completeness.

HANDLING CASE 1. Assume that the query for which the answer is of the form (m, IV'_j) is a query to Π^{-1} on (m', IV'). This query has to be NEW since IV'_j is fresh. Let the index of this query among all the queries made by $\mathcal{A}_2$ when running on IV'_j be i. The encoding procedure stores (i, m) in a list and removes the entry for $\Pi(m, \mathsf{IV}'_j)$ from $\tilde{\Pi}$.

In decoding, if the IV'_j is categorized as case 1, then it removes the front element (i, m) the list. It answers the ith query made while running on IV'_j with (m, IV'_j), and appropriately sets $\Pi(m, \mathsf{IV}'_j)$ to (m', IV').

If the query for which the answer is of the form $(*, \mathsf{IV}'_j)$ is a query to Π, the encoding and decoding

procedure will be similar with appropriate changes (e.g., what is removed from $\tilde{\Pi}$ etc).

In this case we save at least $(c+r-1)-r-\log T = (c-1)-\log T$ bits, which is more than what we need.

HANDLING CASE 2. If IV'_j gets categorized into this case then there will always be two NEW queries q_1, q_2 with q_1 queried before q_2 such that either the last c bits in the answer of q_2 is the same as the last c bits in the input or answer of q_1 or the first r bits in the answer of q_2 is the same as the first r bits in the input or answer of q_1 (giving rise to four sub-cases as claimed).

The encoding procedure stores in a list the tuple consisting of the indices of q_1, q_2, and the other c or r bits. It removes the entry corresponding to q_2 from $\tilde{\Pi}$.

In decoding, if the IV'_j is categorized as case 2, the front element of list is removed and the information is sufficient to answer the NEW query whose entry was removed from $\tilde{\Pi}$.

In this case we save at least $\min\{(c-1),(r-1)\} - 2\log T$ bits, which suffices.

HANDLING CASE 3. In this case it is easy to see that there is a NEW query to Π on (m', IV'_j) such that the last c bits of the answer is same as that of the last c bits of the answer or input of a REPEATEDUSED query q (giving rise to two sub-cases as claimed). Let the index of the query $\Pi(m', \mathsf{IV}'_j)$ among all the queries made by $\mathcal{A}_2$ when running on IV'_j be i and its answer be (m, IV). Let the REPEATEDUSED query q be a crucial query for IV'_k such that $k < j$ and let it be the nth crucial query ($n < 4$).

The encoding procedure stores (i, k, n, m) in a list and removes the entry for $\Pi(m', \mathsf{IV}'_j)$ from $\tilde{\Pi}$.

In decoding, if the IV'_j is categorized as case 3, then it removes the front element (i, k, n, m) the list. It answers the ith query made while running on IV'_j with (m', IV'_j), where IV'_j is computed using (k, n) (and the relevant sub-case categorization) and appropriately sets $\Pi(m', \mathsf{IV}'_j)$ to (m, IV).

In this case we save at least $(c+r-1)-\log T-\log F-\log 4-r = (c-1)-\log 4FT$ bits, which suffices.

HANDLING CASE 4. First we observe that if IV'_j is categorized into this case, then there are five following possibilities.

(a) There is a NEW crucial query q_1 of the form $(*, \mathsf{IV}'_j)$ to Π with answer (m', IV') and there is only one other crucial query q_2 which is REPEATEDUNUSED such that either it is a query to Π with input of the form $(*, \mathsf{IV}')$ and answer of the form $(m', *)$ or it is a query to Π^{-1} with input of the form $(m', *)$ and answer of the form $(*, \mathsf{IV}')$ (i.e., two sub-sub-cases).

(b) There are two NEW crucial queries q_1, q_2 to Π with input $(*, \mathsf{IV}'_j)$ and a REPEATEDUNUSED crucial query q_3 such that if q_3 is a query with input of the form $(*, \mathsf{IV}')$ and output of the form $(m', *)$ then the answer of q_1 is of the form $(*, \mathsf{IV}')$ and the answer to q_2 is of the form $(m', *)$.

(c) There is a NEW crucial query q_1 to Π with input of the form $(*, \mathsf{IV}'_j)$ and answer of the form $(*, \mathsf{IV}')$ and there is a second NEW crucial query q_2 and a REPEATEDUNUSED crucial query q_3 such that one of the following happen.

i q_2 is a query to Π on input of the form $(*, \mathsf{IV}')$ and answer of the form $(m', *)$ and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and answer $(m', *)$.

ii q_2 is a query to Π on input of the form $(*, \mathsf{IV}')$ and answer of the form $(m', *)$ and q_3 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}')$.

iii q_2 is a query to Π^{-1} on input of the form $(m', *)$ and answer of the form $(*, \mathsf{IV}')$ and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and answer $(m', *)$.

iv q_2 is a query Π^{-1} on input of the form $(m', *)$ and answer of the form $(*, \mathsf{IV}')$ and q_3 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}')$.

(d) There are two NEW crucial queries q_1, q_2 such that q_1 is made before q_2 and the last c bits of the answer of q_2 is same as the last c bits of the input or answer of q_1

(e) There are two NEW crucial queries q_1, q_2 to Π both with input of the form $(*, \mathsf{IV}'_j)$ and answer of the form $(*, \mathsf{IV}')$, $(*, \mathsf{IV}'')$ respectively and there is a third NEW crucial query q_3 made after q_1, and a REPEATEDUNUSED crucial query q_4 such that one of the following happen.

i q_3 is a query to Π on input of the form $(*, \mathsf{IV}')$ and answer of the form $(m', *)$ and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and answer $(m', *)$.

ii q_3 is a query to Π on input of the form $(*, \mathsf{IV}')$ and answer of the form $(m', *)$ and q_4 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}'')$.

iii q_3 is a query to Π^{-1} on input of the form $(m', *)$ and answer of the form $(*, \mathsf{IV}')$ and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and has answer $(m', *)$.

iv q_3 is a query to Π^{-1} on input $(m', *)$ and has answer $(*, \mathsf{IV}')$ and q_4 is a query to Π^{-1} on input $(m', *)$ and has answer $(*, \mathsf{IV}'')$.

First we argue why these possibilities are exhaustive for an IV'_j that has gotten categorized in case 4. Since we are looking for 2-block collisions, there can be only one or two other crucial queries that do not involve IV'_j- these will be REPEATEDUNUSED or NEW in this case. If there are only two crucial queries- one NEW query and one REPEATEDUNUSED query then it will satisfy sub-case a. Otherwise if there are three crucial queries: two NEW queries and one REPEATEDUNUSED query such that both new queries are to Π

and of the form $(*, \mathsf{IV}'_j)$, then it will satisfy sub-case b. It is easy to see that sub-case c covers the other possibility of two NEW queries and one REPEATEDUNUSED query when there are three crucial queries because for a 2-block collision, the NEW query which is not of the form $(*, \mathsf{IV}'_j)$ and the REPEATEDUNUSED query will share either the the first c or last r bits in input or output (hence four sub-sub cases) and share the c bits of either input or output with the NEW query of the form $(*, \mathsf{IV}'_j)$. Similarly it is easy to see that if there are four crucial queries- three NEW and one REPEATEDUNUSED then it is covered by sub-cases d, e.

For 4a, the encoding algorithm stores in a list the tuple consisting of the index of the NEW crucial query among the queries made while running on IV'_j and the index of the first occurrence of the REPEATEDUNUSED crucial query among all queries made by $\mathcal{A}_2$. The answer of the NEW crucial query is removed from $\tilde{\Pi}$.

During decoding the REPEATEDUNUSED and NEW queries can be correctly identified and from the input and answer of the REPEATEDUNUSED query both m', IV' can be inferred and the NEW query can be answered correctly.

It follows that savings for this sub-case is at least $c + r - 2 - \log FT - \log T$ bits.

For 4b, the encoding algorithm stores in a list the tuple consisting of the indices of q_1, q_2 among the queries made while running on IV'_j and the index of the first occurrences of the queries q_3 among all queries made by $\mathcal{A}_2$ and the remaining c and r bits of the answers of q_1, q_2 respectively . It removes the answers of q_1, q_2 from $\tilde{\Pi}$.

During decoding queries q_1, q_2, q_3 can be identified correctly using the information from the encoding and q_1, q_2 can be correctly answered using the answer of q_3 and whatever is present in the encoding.

It follows that savings for this sub-case is at least $c + r - 2 - 2\log T - \log FT$ bits.

For 4c, the encoding algorithm stores in a list the tuple consisting of the indices of q_1, q_2 among the queries made while running on IV'_j and the index of the first occurrences of the queries q_3 among all queries made by $\mathcal{A}_2$ and the remaining c and r bits of the answers of q_1, q_2 (depending on the sub-sub case) . It removes the answers of q_1, q_2 from $\tilde{\Pi}$.

During decoding queries q_1, q_2, q_3 can be identified correctly using the information from the encoding and q_1, q_2 can be correctly answered using the answer of q_3 and whatever is present in the encoding.

It follows that savings for this sub-case is at least $\min\{2c - 2, c + r - 2\} - 2\log T - \log FT$ bits.

For 4d, the encoding algorithm stores in a list the tuple consisting of the indices of q_1, q_2 among the queries made while running on IV'_j and the remaining c or r bits of the answer of q_2 (depending on the sub-sub case). It removes the answer of q_2 from $\tilde{\Pi}$.

During decoding queries q_1, q_2, can be identified correctly using the information from the encoding and q_2 can be correctly answered using the answer of q_1 and whatever is present in the encoding.

It follows that savings for this sub-case is at least $\min\{c - 1, r - 1\} - 2\log T$ bits.

For 4e, the encoding algorithm stores in a list the tuple consisting of the indices of q_2, q_3 among the queries made while running on IV'_j and the indices of the first occurrence of the query q_4 among all queries made by $\mathcal{A}_2$ and the remaining c and r bits of the answers of q_2, q_3 (depending on the sub-sub case) . It removes the answers of q_2, q_3 from $\tilde{\Pi}$.

During decoding queries q_2, q_3, q_4 can be identified correctly using the information from the encoding and q_2, q_3 can be correctly answered using the answer of q_3 and whatever is present in the encoding.

It follows that savings for this sub-case is at least $\min\{2c-2, c+r-2\} - 2\log T - \log FT$ bits.

To conclude, in this case we save at least the following number of bits which suffices.

$$\min\{\min\{2c-2, c+r-2\} - \log FT - 2\log T, \min\{c-1, r-1\} - 2\log T\} .$$

HANDLING CASE 5. First we observe that if IV'_j is categorized into this case, then one of the following must happen.

(a) There is a NEW crucial query q_1 of the form $(*, \mathsf{IV}'_j)$ to Π with answer (m', IV') and there are two other crucial queries q_2, q_3 (q_2 is made for the first time before q_3 is made for the first time) both of which are REPEATEDUNUSED such that one of the following happen.

i q_2 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$.

ii q_2 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_3 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$.

iii q_2 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$.

iv q_2 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_3 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$.

(b) There are two NEW crucial queries q_1, q_2 to Π both with input of the form $(*, \mathsf{IV}'_j)$ and answer of the form $(*, \mathsf{IV}')$, $(*, \mathsf{IV}'')$ respectively and there are two crucial REPEATEDUNUSED queries q_3, q_4 (q_3 is made for the first time before q_4 is made for the first time) such that one of the following happen.

i q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and with answer $(m', *)$.

ii q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_4 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}'')$.

iii q_3 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and with answer $(m', *)$.

iv q_3 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_4 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}'')$.

First we argue why these possibilities are exhaustive for an IV'_j that has gotten categorized in case 5. Since we are focusing on two block collisions and we are not in the case 3, the queries other than NEW queries will be REPEATEDUNUSED queries. Again from the definition of fresh IVs and since IV'_j did not get categorized into 1, there must be one or two crucial NEW queries (sub-cases 5a, 5b) on $(*, \mathsf{IV}'_j)$. Since we are looking for 2-block collisions, there can be only two other crucial queries that do not involve IV'_j- these will be REPEATEDUNUSED in this case. It is easy to see that these two REPEATEDUNUSED queries will satisfy one of the four sub-sub-cases of 5a and 5b. Hence this categorization is exhaustive for this case.

For 5a, the encoding algorithm stores in a list the tuple consisting of the index of q_1 among the queries made while running on IV'_j and the indices of the first occurrences of the queries q_2, q_3 (assume wlog that q_2 is queried earlier) among all queries made by $\mathcal{A}_2$, the message m' and r or c bits of rest of the answer of q_2 or q_3 depending the on the sub-sub-case. The answer of q_1 and q_3 is removed from $\tilde{\Pi}$.

During decoding the queries q_1, q_2, q_3 can be inferred from what is present in the encoding and hence the answers of q_1 and whichever of q_2 or q_3 was removed can be computed correctly.

It follows that savings for this sub-case is at least $\min\{2c-2, c+r-2\} - \log T - 2\log FT$ bits.

For 5b, the encoding algorithm stores in a list the tuple consisting of the indices of q_1, q_2 among the queries made while running on IV'_j and the indices of the first occurrences of the queries q_3, q_4 among all queries made by $\mathcal{A}_2$, the messages m', m'' and r or c bits of rest of the answer of q_3 or q_4 depending the on the sub-sub-case.

During decoding the queries q_1, q_2, q_3, q_4 can be inferred from what is present in the encoding and hence the answers of q_1 and q_2 can be computed correctly using things present in the encoding and answers of q_3, q_4.

The decoding for this is similar to that for 5a.

It follows that savings for this sub-case is at least $2c - 2 - 2\log T - 2\log FT$ bits.

In this case we save at least the following number of bits which suffices.

$$\min\{c + r - 2 - \log T - 2\log FT, 2c - 2 - 2\log T - 2\log FT\} .$$

9.3.3 Encoding and decoding procedure pseudocodes for Theorem 9.3.5

The encoding algorithm $\mathsf{Encode}(\sigma, U, \Pi)$:

- Parse U as $\{\mathsf{IV}_1, \ldots, \mathsf{IV}_u\}$ where $\mathsf{IV}_1, \ldots, \mathsf{IV}_u$ are in lexicographical order.

- Set $j \leftarrow 0$.

- Initialize the set P to be empty, the lists Cases and $\tilde{\Pi}$ to be empty

- For $i = 1, \ldots, u$, do:

  – If $\mathsf{IV}_i \in \mathsf{SIVs}(a)$ for some $a \in U_{\mathsf{fresh}}$, let $r \in [jT]$ be the smallest index such that either $\mathsf{Q}_{\mathsf{fresh}}[r]$ or its answer is of the form $(*, \mathsf{IV}_j)$. If $\mathsf{Q}_{\mathsf{fresh}}[r] = (*, \mathsf{IV}_j)$ then set $b \leftarrow 1$ and 0. Add $2r - b$ to P, and continue to next iteration.

  – Otherwise, increment j, set $\mathsf{IV}'_j \leftarrow \mathsf{IV}_i$. Run $\mathcal{A}_2$ on (σ, IV'_j), answering its queries to Π, Π^{-1}. If the query was one to Π with input (m, IV), add $\Pi(m, \mathsf{IV})$ to the list $\tilde{\Pi}$ if it was not already added earlier. If the query was one to Π^{-1} with output (m, IV), add $\Pi(m, \mathsf{IV})$ to the list $\tilde{\Pi}$ if it was not already added earlier. Classify the queries as NEW, REPEATEDUSED, REPEATEDUNUSEDas defined above.

  – Determine the crucial queries as specified in Definition 9.3.7. Determine which category IV'_j belongs to among Cases 1 to 5 based on the crucial queries. According to the categorization do the following.

    * If IV'_j gets categorized into case 1

      (a) If the query for which the answer is of the form (m', IV'_j) is a query to Π^{-1} on (m'', IV'), then let index of this query among all the queries made by $\mathcal{A}_2$ when running on IV'_j be d. Set $\mathsf{Cases}[j] \leftarrow 1a$, add (d, m') to a list L_{1a} and remove the entry for $\Pi(m', \mathsf{IV}'_j)$ from $\tilde{\Pi}$.

      (b) If the query for which the answer is of the form (m', IV'_j) is a query to Π on (m'', IV'), then let index of this query among all the queries made by $\mathcal{A}_2$ when running on IV'_j be d. Set $\mathsf{Cases}[j] \leftarrow 1a$, add (d, m') to a list L_{1b} and remove the entry for $\Pi(m'', \mathsf{IV}')$ from $\tilde{\Pi}$.

      Continue to next iteration

    * Else if IV'_j gets categorized into case 2, define q_1, q_2 as specified above under heading "Handling case 2". Let the input of q_1 be (m', IV') and its answer be (m'', IV''). Let the indices of queries q_1, q_2 among the queries made by $\mathcal{A}_2$ when running on IV'_j be d_1, d_2.

      (a) If the answer of q_2 is (m''', IV') for some m''', set $\mathsf{Cases}[j] \leftarrow 2a$, add (d_1, d_2, m''') to list L_{2a}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_2 (i.e. if q_2 was a query to Π on (m, IV) remove $\Pi(m, \mathsf{IV})$ and if q_2 was a query to Π^{-1} with answer (m, IV) remove $\Pi(m, \mathsf{IV})$ from $\tilde{\Pi}$).

(b) If the answer of q_2 is (m', IV''') for some IV''', set $\mathsf{Cases}[j] \leftarrow 2b$, add $(d_1, d_2, \mathsf{IV}''')$ to list L_{2a}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_2.

(c) If the answer of q_2 is (m''', IV'') for some m''', set $\mathsf{Cases}[j] \leftarrow 2c$, add (d_1, d_2, m''') to list L_{2c}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_2.

(d) If the answer of q_2 is (m'', IV''') for some IV''', set $\mathsf{Cases}[j] \leftarrow 2d$, add (d_1, d_2, m''') to list L_{2d}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_2.

Continue to the next iteration.

* Else if IV'_j gets categorized into case 3, define q as specified above under heading "Handling case 3". Let q be a crucial query for IV'_k and let it be the n-th crucial query (in query order) among the crucial queries for IV'_k. Let the input of q be (m', IV') and its answer be (m'', IV''). Let the NEW crucial query for IV'_j the last c bits of whose answer is same as the last c bits of the answer or input of q. Let the index of query q' among the queries made by $\mathcal{A}_2$ when running on IV'_j be d.

(a) If the answer of q' is (m''', IV') for some m''', set $\mathsf{Cases}[j] \leftarrow 3a$, add (d, k, n, m''') to list L_{3a}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q'.

(b) If the answer of q' is (m''', IV'') for some m''', set $\mathsf{Cases}[j] \leftarrow 3b$, add (d, k, n, m''') to list L_{3b}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q'.

* Else if IV'_j gets categorized into case 4, determine which of the sub-cases IV'_j belongs to according to the categorization specified above under heading "Handling case 4".

(a) If IV'_j gets categorized into sub-case a, define $q_1, q_2, (m', \mathsf{IV}')$ as specified above for the sub-case. Let d be the index of q_1 among all the queries made by $\mathcal{A}_2$ when running on IV'_j and let e be the smallest index of the query q_2 in $\mathsf{Q}_{\mathsf{fresh}}$.

i If q_2 was a Π query with input of the form $(*, \mathsf{IV}')$ and answer of the form $(m', *)$, set $\mathsf{Cases}[j] \leftarrow 4a1$, add (d, e) to the list L_{4a1}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_1.

ii If q_2 was a Π^{-1} query with input of the form $(m', *)$ and answer of the form $(*, \mathsf{IV}')$, set $\mathsf{Cases}[j] \leftarrow 4a2$, add (d, e) to the list L_{4a2}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_1.

(b) If IV'_j gets categorized into sub-case b, define $q_1, q_2, q_3, m', \mathsf{IV}'$ as specified above for the sub-case. Let d_1, d_2 be the indices of q_1, q_2 among all the queries made by $\mathcal{A}_2$ when running on IV'_j and let e be the smallest index of the query q_3 in $\mathsf{Q}_{\mathsf{fresh}}$. Suppose the answer of q_1 is (m'', IV') and that of q_2 is (m', IV'').Set $\mathsf{Cases}[j] \leftarrow 4b$, add $(d_1, d_2, e, m'', \mathsf{IV}'')$ to the list L_{4b}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

(c) If IV'_j gets categorized into sub-case c, define $q_1, q_2, q_3, \mathsf{IV}'$ as specified above for the sub-case. Let d_1, d_2 be the indices of q_1, q_2 among all the queries made by $\mathcal{A}_2$ when running on IV'_j and let e be the smallest index of the query q_3 in $\mathsf{Q}_{\mathsf{fresh}}$. Let the answer of q_1 be (m''', IV').

i If q_2 is a query to Π on input (m'', IV') and with answer (m', IV'') and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and answer $(m', *)$, then set $\mathsf{Cases}[j] \leftarrow 4c1$, add $(d_1, d_2, e, m''', \mathsf{IV}'')$ to the list L_{4c1}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

ii If q_2 is a query to Π on input (m'', IV') and with answer (m', IV'') and q_3 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}')$, then set $\mathsf{Cases}[j] \leftarrow 4c2$, add $(d_1, d_2, e, m''', \mathsf{IV}'')$ to the list L_{4c2}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

iii If q_2 is a query to Π^{-1} on input (m', IV'') and with answer (m'', IV') and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and answer $(m', *)$, then set $\mathsf{Cases}[j] \leftarrow 4c3$, add (d_1, d_2, e, m''', m'') to the list L_{4c3}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

iv If q_2 is a query to Π^{-1} on input (m', IV'') and with answer (m'', IV') and q_3 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}')$, then set $\mathsf{Cases}[j] \leftarrow 4c4$, add (d_1, d_2, e, m''', m'') to the list L_{4c4}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

(d) If IV'_j gets categorized into sub-case d, define q_1, q_2 as specified above for the sub-case. Let d_1, d_2 be the indices of q_1, q_2 among all the queries made by $\mathcal{A}_2$ when running on IV'_j.

i If the answer of q_2 is (m', IV') and the input of q_1 is of the form $(*, \mathsf{IV}')$, set $\mathsf{Cases}[j] \leftarrow 4d1$, add (d_1, d_2, m') to list L_{4d1}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_2.

ii If the answer of q_2 is (m', IV') and the answer of q_1 is of the form $(*, \mathsf{IV}')$, set $\mathsf{Cases}[j] \leftarrow 4d2$, add (d_1, d_2, m') to list L_{4d2}, remove the entry in $\tilde{\Pi}$ corresponding to the answer of q_2.

(e) If IV'_j gets categorized into sub-case e, define q_1, q_2, q_3, q_4 as specified above for the sub-case. Let d_1, d_2 be the indices of q_2, q_3 among all the queries made by $\mathcal{A}_2$ when running on IV'_j. Let e be the smallest index of the query q_4 in $\mathsf{Q}_{\mathsf{fresh}}$. Suppose the answer of q_2 is (m'', IV'').

i If q_3 is a query to Π on input of the form $(*, \mathsf{IV}')$ and with answer (m', IV''') and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and answer $(m', *)$, then set $\mathsf{Cases}[j] \leftarrow 4e1$, add $(d_1, d_2, e, m'', \mathsf{IV}''')$ to the list L_{4e1}, remove the entry in $\tilde{\Pi}$ corresponding to the answers

of q_2, q_3.

ii If q_3 is a query to Π on input of the form $(*, \mathsf{IV}')$ and with answer (m', IV''') and q_4 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}'')$, then set $\mathsf{Cases}[j] \leftarrow 4e2$, add $(d_1, d_2, e, m'', \mathsf{IV}''')$ to the list L_{4e2}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_2, q_3.

iii If q_3 is a query to Π^{-1} on input of the form $(m', *)$ and with answer (m''', IV') and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and answer $(m', *)$, then set $\mathsf{Cases}[j] \leftarrow 4e3$, add (d_1, d_2, e, m'', m''') to the list L_{4e3}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_2, q_3.

iv If q_3 is a query to Π^{-1} on input of the form $(m', *)$ and with answer (m''', IV') and q_4 is a query to Π^{-1} on input $(m', *)$ and answer $(*, \mathsf{IV}'')$, then set $\mathsf{Cases}[j] \leftarrow 4e4$, add (d_1, d_2, e, m'', m''') to the list L_{4e4}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_2, q_3.

* Else if IV'_j gets categorized into case 5, determine which of the sub-cases IV'_j belongs to according to the categorization specified above under heading "Handling case 5".

(a) If IV'_j gets categorized into sub-case a, define q_1, q_2, q_3 as specified above for the sub-case. Let d be the index of q_1 among all the queries made by $\mathcal{A}_2$ when running on IV'_j. Let e_1, e_2 be the smallest indices of q_2, q_3 in $\mathsf{Q}_{\mathsf{fresh}}$. Let the answer of q_1 be (m'', IV').

i If q_2 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer (m', IV''), then set $\mathsf{Cases}[j] \leftarrow 5a1$, add $(d, e_1, e_2, m'', \mathsf{IV}'')$ to the list L_{5a1}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_3.

ii If q_2 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_3 is a query to Π^{-1} on input $(m', *)$ and with answer (m''', IV'), then set $\mathsf{Cases}[j] \leftarrow 5a2$, add (d, e_1, e_2, m'', m''') to the list L_{5a2}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_3.

iii If q_2 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer (m', IV''), then set $\mathsf{Cases}[j] \leftarrow 5a3$, add $(d, e_1, e_2, m'', \mathsf{IV}'')$ to the list L_{5a3}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_3.

iv If q_2 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_3 is a query to Π^{-1} on input $(m', *)$ and with answer (m''', IV'), then set $\mathsf{Cases}[j] \leftarrow 5a4$, add (d, e_1, e_2, m'', m''') to the list L_{5a4}, remove the entry in $\tilde{\Pi}$ corresponding to the answers

of q_1, q_3.

(b) If IV'_j gets categorized into sub-case b, define q_1, q_2, q_3, q_4 as specified above for the sub-case. Let d_1, d_2 be the indices of q_1, q_2 among all the queries made by $\mathcal{A}_2$ when running on IV'_j. Let e_1, e_2 be the smallest indices of q_3, q_4 in $\mathsf{Q}_{\mathsf{fresh}}$. Let the answer of q_1 be (m'', IV') and that of q_2 be (m''', IV'').

i If q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and with answer $(m', *)$, then set $\mathsf{Cases}[j] \leftarrow 5b1$, add $(d_1, d_2, e_1, e_2, m'', m''')$ to the list L_{5b1}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

ii If q_3 is a query to Π on input $(*, \mathsf{IV}')$ and with answer $(m', *)$ and q_4 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}'')$, then set $\mathsf{Cases}[j] \leftarrow 5b2$, add $(d_1, d_2, e_1, e_2, m'', m''')$ to the list L_{5b2}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

iii If q_3 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_4 is a query to Π on input $(*, \mathsf{IV}'')$ and with answer $(m', *)$, then set $\mathsf{Cases}[j] \leftarrow 5b3$, add $(d_1, d_2, e_1, e_2, m'', m''')$ to the list L_{5b3}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

iv If q_3 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}')$ and q_4 is a query to Π^{-1} on input $(m', *)$ and with answer $(*, \mathsf{IV}'')$, then set $\mathsf{Cases}[j] \leftarrow 5b4$, add $(d_1, d_2, e_1, e_2, m'', m''')$ to the list L_{5b4}, remove the entry in $\tilde{\Pi}$ corresponding to the answers of q_1, q_2.

– Add all the evaluations of $\Pi(m, \mathsf{IV})$ to $\tilde{\Pi}$ in lexicographical order for all (m, IV)'s such that no Π query was not made on them and no Π^{-1} query had (m, IV) as answer.

– Output F, U', P, L_{1a},L_{1b}, L_{2a}, L_{2b}, L_{2c}, L_{2d}, L_{3a}, L_{3b}, L_{4a1}, L_{4a2}, L_{4b}, L_{4c1}, L_{4c2}, L_{4c3}, L_{4c4}, L_{4d1}, L_{4d2}, L_{4e1}, L_{4e2}, L_{4e3}, L_{4e4}, L_{5a1}, L_{5a2}, L_{5a3}, L_{5a4}, L_{5b1}, L_{5b2}, L_{5b3}, L_{5b4}, $\tilde{\Pi}$.

The decoding algorithm Decode:

1. Parse U' as $\{\mathsf{IV}'_1, \ldots, \mathsf{IV}'_F\}$ where $\mathsf{IV}'_1, \ldots, \mathsf{IV}'_F$ are in lexicographical order.

2. Set $j \leftarrow 0$.

3. For $j = 1, \ldots, F$ do:

   i Run $\mathcal{A}_2$ on (σ, IV'_j), answering its queries to Π, Π^{-1}.

   ii Answer the ith query made by $\mathcal{A}_2$ on (m, IV) as follows.

iii If $2 \cdot ((j-1) \cdot T + i) \in \mathsf{P}$ then add IV to U.

iv If $2 \cdot ((j-1) \cdot T + i) - 1 \in \mathsf{P}$ then add the IV of the answer to U before returning it.

v If $(*, d, c, *, \mathsf{IV}'') \in L_{5a1}$ then let the answer of the query with index d in $\mathsf{Q}_{\mathsf{fresh}}$ be $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}'')$, return (m', IV'')

vi If $(*, d, c, *, m''') \in L_{5a2}$ then let the input of the query with index d in $\mathsf{Q}_{\mathsf{fresh}}$ be $(*, \mathsf{IV}')$. Set $\Pi(m''', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, return (m''', IV')

vii If $(*, d, c, *, \mathsf{IV}'') \in L_{5a3}$ then let the input of the query with index d in $\mathsf{Q}_{\mathsf{fresh}}$ be $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}'')$, return (m', IV'')

viii If $(*, d, c, *, m''') \in L_{5a4}$ then let the answer of the query with index d in $\mathsf{Q}_{\mathsf{fresh}}$ be $(*, \mathsf{IV}')$. Set $\Pi(m''', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, return (m''', IV')

ix If $\mathsf{Cases}[j] = 1a$ and the tuple in front of list L_{1a} is (i, m'). Set $\Pi(m', \mathsf{IV}'_j) \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{1a}. Return (m', IV'_j).

x Else if $\mathsf{Cases}[j] = 1b$ and the tuple in front of list L_{1b} is (i, m'). Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}'_j)$, and remove the tuple from L_{1b}. Return (m', IV'_j).

xi Else if $\mathsf{Cases}[j] = 2a$ and the tuple in front of list L_{2a} is (d, i, m'''), let the query with index d in Q_i have input (m', IV') and answer (m'', IV''). Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}')$, and remove the tuple from L_{2a}. Return (m''', IV').

xii Else if $\mathsf{Cases}[j] = 2b$ and the tuple in front of list L_{2a} is (d, i, IV'''), let the query with index d in Q_i have input (m', IV') and answer (m'', IV''). Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}''')$, and remove the tuple from L_{2b}. Return (m', IV''').

xiii Else if $\mathsf{Cases}[j] = 2c$ and the tuple in front of list L_{2a} is (d, i, m'''), let the query with index d in Q_i have input (m', IV') and answer (m'', IV''). Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}'')$, and remove the tuple from L_{2c}. Return (m''', IV'').

xiv Else if $\mathsf{Cases}[j] = 2d$ and the tuple in front of list L_{2a} is (d, i, IV'''), let the query with index d in Q_i have input (m', IV') and answer (m'', IV''). Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}''')$, and remove the tuple from L_{2d}. Return (m'', IV''').

xv Else if $\mathsf{Cases}[j] = 3a$ and the tuple in front of list L_{3a} is (i, k, n, m'''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}')$, and remove the tuple from L_{3a}. Return (m''', IV').

xvi Else if $\mathsf{Cases}[j] = 3b$ and the tuple in front of list L_{3b} is (i, k, n, m'''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}'')$, and remove the tuple from L_{3b}. Return (m''', IV'').

xvii Else if $\mathsf{Cases}[j] = 4a1$ and the tuple in front of list L_{4a1} is (i, e), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}')$, and remove the tuple from L_{4a1}. Return (m', IV').

xviii Else if $\mathsf{Cases}[j] = 4a2$ and the tuple in front of list L_{4a2} is (i, e), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}')$ and input $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}')$, and remove the tuple from L_{4a2}. Return (m', IV').

xix Else if $\mathsf{Cases}[j] = 4b$ and the tuple in front of list L_{4b} is $(i, d_2, e, m'', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{4b} if $d_2 < i$. Return (m'', IV').

xx Else if $\mathsf{Cases}[j] = 4b$ and the tuple in front of list L_{4b} is $(d_1, i, e, m'', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}'')$, and remove the tuple from L_{4b} if $d_1 < i$. Return (m', IV'').

xxi Else if $\mathsf{Cases}[j] = 4c1$ and the tuple in front of list L_{4c1} is $(i, d_2, e, m''', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}')$, and remove the tuple from L_{4c1} if $d_2 < i$. Return (m''', IV').

xxii Else if $\mathsf{Cases}[j] = 4c1$ and the tuple in front of list L_{4c1} is $(d_1, i, e, m''', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}'')$, and remove the tuple from L_{4c1} if $d_1 < i$. Return (m', IV'').

xxiii Else if $\mathsf{Cases}[j] = 4c2$ and the tuple in front of list L_{4c2} is $(i, d_2, e, m''', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}')$, and remove the tuple from L_{4c2} if $d_2 < i$. Return (m''', IV').

xxiv Else if $\mathsf{Cases}[j] = 4c2$ and the tuple in front of list L_{4c2} is $(d_1, i, e, m''', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}'')$, and remove the tuple from L_{4c2} if $d_1 < i$. Return (m', IV'').

xxv Else if $\mathsf{Cases}[j] = 4c3$ and the tuple in front of list L_{4c1} is (i, d_2, e, m''', m''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}')$, and remove the tuple from L_{4c3} if $d_2 < i$. Return (m''', IV').

xxvi If $\mathsf{Cases}[j] = 4c3$ and the tuple in front of list L_{4c3} is (d_1, i, e, m''', m''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$ and answer $(m', *)$. Set $\Pi(m'', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{4c3} if $d_1 < i$. Return (m'', IV').

xxvii Else if $\mathsf{Cases}[j] = 4c4$ and the tuple in front of list L_{4c4} is (i, d_2, e, m''', m''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}')$, and remove the

tuple from L_{4c4} if $d_2 < i$. Return (m''', IV').

xxviii Else if $\mathsf{Cases}[j] = 4c4$ and the tuple in front of list L_{4c4} is $(d_1, i, e, m''', \mathsf{IV}'')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}')$. Set $\Pi(m'', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{4c4} if $d_1 < i$. Return (m'', IV').

xxix Else if $\mathsf{Cases}[j] = 4d1$ and the tuple in front of list L_{4d1} is (d_1, i, m'), let the query with index d_1 in Q_j have input $(*, \mathsf{IV}')$. If the query is a query to Π, set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}')$ else set $\Pi(m', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{4d1}. Return (m', IV').

xxx Else if $\mathsf{Cases}[j] = 4d2$ and the tuple in front of list L_{4d2} is (d_1, i, m'), let the query with index d_1 in Q_j have answer $(*, \mathsf{IV}')$. If the query is a query to Π, set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}')$ else set $\Pi(m', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{4d2}. Return (m', IV').

xxxi Else if $\mathsf{Cases}[j] = 4e1$ and the tuple in front of list L_{4e1} is $(i, d_2, e, m'', \mathsf{IV}''')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}'')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}'')$, and remove the tuple from L_{4e1} if $d_2 < i$. Return (m'', IV'').

xxxii Else if $\mathsf{Cases}[j] = 4e1$ and the tuple in front of list L_{4e1} is $(d_1, i, e, m'', \mathsf{IV}''')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}'')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}''')$, and remove the tuple from L_{4e1} if $d_2 < i$. Return (m', IV''').

xxxiii Else if $\mathsf{Cases}[j] = 4e2$ and the tuple in front of list L_{4e2} is $(i, d_2, e, m'', \mathsf{IV}''')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}'')$, and remove the tuple from L_{4e2} if $d_2 < i$. Return (m'', IV'').

xxxiv Else if $\mathsf{Cases}[j] = 4e2$ and the tuple in front of list L_{4e2} is $(d_1, i, e, m'', \mathsf{IV}''')$, let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m', \mathsf{IV}''')$, and remove the tuple from L_{4e2} if $d_2 < i$. Return (m', IV''').

xxxv Else if $\mathsf{Cases}[j] = 4e3$ and the tuple in front of list L_{4e3} is (i, d_2, e, m'', m'''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}'')$ and answer $(m', *)$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}'')$, and remove the tuple from L_{4e3} if $d_2 < i$. Return (m'', IV'').

xxxvi Else if $\mathsf{Cases}[j] = 4e3$ and the tuple in front of list L_{4e3} is (d_1, i, e, m'', m'''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}'')$ and answer $(m', *)$. Set $\Pi(m''', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{4e3} if $d_2 < i$. Return (m''', IV').

xxxvii Else if $\mathsf{Cases}[j] = 4e4$ and the tuple in front of list L_{4e4} is (i, d_2, e, m'', m'''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}'')$, and remove the tuple from L_{4e4} if $d_2 < i$. Return (m'', IV'').

xxxviii Else if $\mathsf{Cases}[j] = 4e4$ and the tuple in front of list L_{4e4} is (d_1, i, e, m'', m'''), let the query with index e in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(m', *)$ and answer $(*, \mathsf{IV}'')$. Set $\Pi(m''', \mathsf{IV}') \leftarrow (m, \mathsf{IV})$, and remove the tuple from L_{4e4} if $d_2 < i$. Return (m''', IV'').

xxxix Else if $\mathsf{Cases}[j] = 5a1$ and the tuple in front of list L_{5a1} is $(i, e_1, e_2, m'', \mathsf{IV}'')$, let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5a1}. Return (m'', IV').

xl Else if $\mathsf{Cases}[j] = 5a2$ and the tuple in front of list L_{5a2} is (i, e_1, e_2, m'', m'''), let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5a2}. Return (m'', IV').

xli Else if $\mathsf{Cases}[j] = 5a3$ and the tuple in front of list L_{5a3} is $(i, e_1, e_2, m'', \mathsf{IV}'')$, let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5a3}. Return (m'', IV').

xlii Else if $\mathsf{Cases}[j] = 5a4$ and the tuple in front of list L_{5a4} is (i, e_1, e_2, m'', m'''), let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5a4}. Return (m'', IV').

xliii Else if $\mathsf{Cases}[j] = 5b1$ and the tuple in front of list L_{5b1} is $(i, d_2, e_1, e_2, m'', m''', \mathsf{IV}''')$, let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5b1} if $i > d_2$. Return (m'', IV').

xliv Else if $\mathsf{Cases}[j] = 5b1$ and the tuple in front of list L_{5b1} is $(d_1, i, e_1, e_2, m'', m''', \mathsf{IV}''')$, let the query with index e_2 in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}'')$, and remove the tuple from L_{5b1}if $i > d_1$. Return (m''', IV'').

xlv Else if $\mathsf{Cases}[j] = 5b2$ and the tuple in front of list L_{5b2} is $(i, d_2, e_1, e_2, m'', m''', m'''')$, let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5b2} if $i > d_2$. Return (m'', IV').

xlvi Else if $\mathsf{Cases}[j] = 5b2$ and the tuple in front of list L_{5b2} is $(d_1, i, e_1, e_2, m'', m''', m'''')$, let the query with index e_2 in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}'')$, and remove the tuple from L_{5b2} if $i > d_1$. Return (m''', IV'').

xlvii Else if $\mathsf{Cases}[j] = 5b3$ and the tuple in front of list L_{5b3} is $(i, d_2, e_1, e_2, m'', m''', \mathsf{IV}''')$, let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5b3} if $i > d_2$. Return (m'', IV').

xlviii Else if $\mathsf{Cases}[j] = 5b3$ and the tuple in front of list L_{5b3} is $(d_1, i, e_1, e_2, m'', m''', \mathsf{IV}''')$, let the query with index e_2 in $\mathsf{Q}_{\mathsf{fresh}}$ have input $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}'')$, and remove the tuple from

L_{5b3} if $i > d_1$. Return (m''', IV'').

xlix Else if $\mathsf{Cases}[j] = 5b4$ and the tuple in front of list L_{5b4} is $(i, d_2, e_1, e_2, m'', m''', m'''')$, let the query with index e_1 in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m'', \mathsf{IV}')$, and remove the tuple from L_{5b4} if $i > d_2$. Return (m'', IV').

l Else if $\mathsf{Cases}[j] = 5b4$ and the tuple in front of list L_{5b4} is $(i, d_2, e_1, e_2, m'', m''', m'''')$, let the query with index e_2 in $\mathsf{Q}_{\mathsf{fresh}}$ have answer $(*, \mathsf{IV}'')$. Set $\Pi(m, \mathsf{IV}) \leftarrow (m''', \mathsf{IV}'')$, and remove the tuple from L_{5b4} if $i > d_1$. Return (m''', IV'').

4. Populate the rest of Π by remaining elements in $\tilde{\Pi}$ in lexicographical order.

5. Output (U, Π).

Chapter 10

OPTIMAL SECURITY FOR KEYED HASH FUNCTIONS: AVOIDING TIME-SPACE TRADEOFFS FOR FINDING COLLISIONS

In this chapter, we present several new constructions of keyed hash functions that are provably resistant to any non-trivial time-space tradeoff attacks for finding collisions. Our constructions provide various tradeoffs between their efficiency and the range of parameters where they achieve optimal security for collision resistance. This chapter is based on our work in [FGK23].

We start this chapter by giving a high level overview of our main techniques used for this result. Our goal is to construction a variable-input length, keyed, hash function $H^h\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ from an idealized, fixed-input length hash function $h\colon \{0,1\}^a \to \{0,1\}^n$.

Non-uniform security in the AI-ROM. We consider non-uniform (S,T)-attackers $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in the auxiliary-input random oracle model (AI-ROM) [Unr07] with the following structure. First, h is randomly sampled from the space of all a-bit to n-bit functions. Then, in the preprocessing phase, $\mathcal{A}_1$ has unbounded access to h and outputs an advice string σ such that $|\sigma| \leqslant S$. The online phase $\mathcal{A}_2$ receives auxiliary input σ and a random key $\mathsf{k} \leftarrow \{0,1\}^\kappa$ as input, and then has to find two distinct messages $\mathsf{msg}, \mathsf{msg}'$ such that $H^h(\mathsf{k}, \mathsf{msg}) = H^h(\mathsf{k}, \mathsf{msg}')$ while making at most T queries to h. Our goal is to give constructions H^h such that no (S,T)-attacker as above can find a collision with better than $O(S/2^\kappa + T^2/2^n)$ probability. This is "optimal" in the sense that this matches a naive attack against a purely random H: the preprocessing attacker stores collisions for $\Omega(S)$ keys, and the online attacker either gets "lucky" and receives one of those keys as input or performs a standard birthday-style attack.

Merkle-Damgård framework with a keyed inner hash. We consider a general framework based on the Merkle-Damgård (MD) transformation where we instantiate the inner hash function with a keyed one. Let $s \in \mathbb{N}$ be the desired message block size. Then, given a function $g\colon \{0,1\}^{\kappa+n+s} \to \{0,1\}^n$, we can build a function H^g where any attack on the collision resistance of H^g implies an attack on g. The idea behind H^g is as follows. We first break our message up into blocks $m_1, \ldots, m_\ell$ of size $a - \kappa - n$. We initialize the value $y_0 = 0^n$, and for $i = 1, 2, \ldots, \ell$ we compute $y_i = g(\mathsf{k} \| y_{i-1} \| m_i)$. Finally, we output y_ℓ. It is known (e.g. see Section 8.5 of [GB08]) that if you can find a collision in the MD construction H^g for a keyed g in the uniform

setting, then this implies you can find a collision in g. Indeed, this reasoning extends to the non-uniform setting with (S,T)-attackers in the AI-ROM. Hence, this shows that H^g is as secure as g. So, our new goal is to construct such a g with "optimal security" given an idealized hash function $h\colon \{0,1\}^a \to \{0,1\}^n$.

Our first observation is that if $a \geqslant \kappa + n + s$, then we can simply use $g = h$. Furthermore, h has optimal security $O(S/2^\kappa + T^2/2^n)$ (first formalized by [DGK17] in the AI-ROM), so we are done! So our next goal is to try to use an h from minimal assumptions. Namely, can we get a keyed hash function with arbitrary length input from any $h\colon \{0,1\}^a \to \{0,1\}^n$ where a is much smaller, i.e. even $a = n + 1$? This will allow us to focus on building as simple a primitive as possible which we can bootstrap to a full variable-input length hash function with optimal security.

Next, we note that for any $a > \max(\kappa, n)$, we can always do the standard Merkle-Damgård transformation using h to construct g, where k is not fed into every invocation of h. For standard MD, it makes sense to set $\kappa = n$ since we use the key k as the initialization vector. Recall, the MD^h construction sets $y_0 = \mathsf{k}$, computes $y_i = h(y_{i-1} \| m_i)$ for $i = 1, \ldots, \ell$, and outputs y_ℓ. This approach has a major downside in that instantiating MD without inserting k into each invocation of h suffers non-trivial time-space tradeoffs. In general, there is an attack on the general MD^h construction with advantage $\Omega(ST^2/2^n)$. This is strictly worse than the optimal bound of $O(S/2^n + T^2/2^n)$ for any setting of parameters with $S, T \gg 1$. However, this attack finds very large—roughly length T—collisions. In our setting, we only care about using MD^h to get a function g with inputs of size $2n + s$. Thus, we leverage a recent line of work (see [CDGS18, ACDW20, CGLQ20, GK22, AGL22]) that shows that if you only use MD^h on ℓ-block messages, then the best known attack has advantage at most $O(ST\ell/2^n + T^2/2^n)$ (and further this is provably tight for constant ℓ [ACDW20, GK22] and, when $ST^2 \leqslant 2^n$, is provably tight for all ℓ [AGL22]).

So, if we instantiate g in our framework with a fixed-length MD^h construction where we set κ and s to be equal to the output length n, we get a construction for H^h where any (S,T)-attacker has advantage at most $O(ST/2^n + T^2/2^n)$ (up to $\mathrm{poly}(n)$ factors). This is only "optimal," however, under the (strong) assumption that $S \leqslant T$.

Our main technical contribution is instantiating g in the framework above using a new keyed Merkle tree approach, which does not require the assumption that $S \leqslant T$.

Keyed Merkle Tree Analysis

For the rest of this technical overview, we focus on our analysis of the keyed Merkle tree construction.

Construction. We start by defining the 2-level construction. We want a keyed function $g\colon \{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$ from a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ where $a \geqslant 2n$. To do so, we split the $(n+s)$-bit input into two parts, call them m_L, m_R, of size at most $a - \kappa$ (hence we require here that $a - \kappa \geqslant (n+s)/2$, or

$a \geqslant \kappa + n/2 + s/2$). We concatenate each part with the key k and compute $y_L \leftarrow h(\mathsf{k} \| m_L)$ and $y_R \leftarrow h(\mathsf{k} \| m_R)$. We then concatenate y_L with y_R and feed the resulting string into h to get the output $z \leftarrow h(y_L \| y_R)$ (we require here that $a \geqslant 2n$). For technical purposes, we "domain separate" each call to h, so $y_L = h_1(\mathsf{k} \| m_L)$, $y_R = h_2(\mathsf{k} \| m_R)$, and $z = h_3(y_L \| y_R)$.

Analysis. We want to bound the probability that any (S, T)-attacker can find a collision in this keyed Merkle tree construction of g. To simplify the analysis here, we consider the case where κ and s are equal to n. Hence, each call to h takes two n-bit inputs and has one n-bit output. We also assume from the start that $ST^2 \leqslant 2^n$.

We start with the following observation. The probability that any (S, T)-attacker finds a collision at the leaves, corresponding to the calls to h_1, h_2, is at most $O(S/2^n + T^2/2^n)$. Because these calls include the key k, if such an event happened, we could reduce to finding a collision in h directly. So, the challenge is to reason about the advantage of an (S, T)-attacker finding a collision at the second level of the Merkle tree, which is only implicitly related to the key k in the Merkle tree construction. We have to somehow characterize all possible ways that an (S, T)-attack can encode information about h in its advice string from the preprocessing phase.

One of the first tools one often turns to in such analysis is to use the *presampling* technique from [Unr07, CDGS18]. We note that if we were to use the presampling technique, we would obtain a term of the form $ST/2^n$ in out bound, which is optimally secure only in the range $S \leqslant T$. Our main technical contribution is getting an optimally secure protocol for the range $S \gg T$, which therefore requires techniques other than presampling.

The AGL [AGL22] framework: reducing to multi-instance games. To make our lives significantly easier, we use the multi-instance framework of [AGL22], previously used in somewhat different forms in [IK10, ACDW20, CGLQ20, GK22]. At a very high level, this framework gives a way to reason about (S, T)-attackers using an *average-case* advice string rather than a worst-case one. In more detail, they show how to bound the advantage that any (S, T)-attacker $\mathcal{A}$ finds a collision in g^h by the advantage of a (uniform) attacker $\mathcal{B}$ in the following game. First, a random function h is sampled. Then, $\mathcal{B}$ has to win the following game for all $i = 1, \ldots, S$ sequentially, where it is allowed to maintain arbitrary state (that it generates) between each successive game. In each game i, the attacker $\mathcal{B}$ receives a random key k_i, its state from the previous games, and has to come up with a pair of messages $\mathsf{msg}_i, \mathsf{msg}'_i$ such that $g^h(\mathsf{k}_i, \mathsf{msg}_i) = g^h(\mathsf{k}_i, \mathsf{msg}'_i)$ using at most T queries to h. [CGLQ20, AGL22] show that if the advantage of $\mathcal{B}$ is at most δ^S, then the advantage of the (S, T)-attacker $\mathcal{A}$ is at most 2δ. The magic of this framework is that we can analyze the advantage of $\mathcal{B}$ in each game i only given its state from the previous games, instead of having to reason about arbitrary

advice strings as in the case of (S, T)-attackers. Namely, we can lazily sample h on any point that $\mathcal{B}$ has not queried, in a way that is independent of $\mathcal{B}$'s current state.

Note that it suffices to show that the advantage of $\mathcal{B}$ "in game i" is at most δ *given* it has won all previous games. Let W_i be the event that $\mathcal{B}$ wins game i and $W_{<i}$ be the event that $\mathcal{B}$ wins all games before i. This follows since $\mathsf{Pr}[W_1 \wedge \ldots \wedge W_S] = \prod_{i=1}^{S} \mathsf{Pr}[W_i | W_{<i}] \leqslant \delta^S$ if $\mathsf{Pr}[W_i | W_{<i}] \leqslant \delta$ for all $i \in [S]$. Hence, our goal is to show that $\mathsf{Pr}[W_i | W_{<i}] \leqslant O(S/2^n + T^2/2^n)$, up to $\mathrm{poly}(n)$ factors.

Knowledge gaining event: bounding "hitting" queries. Now, to even further simplify the analysis of $\mathsf{Pr}[W_i | W_{<i}]$, we define a key "knowledge gaining event" (based on the techniques of [AGL22]) representing the kind of information that $\mathcal{B}$ may have encoded into its state based on the queries it made to h before game i has started. At a high level, this is an event that we show happens with very small probability (technically at most $2^{-2i \cdot n}$) for an average-case advice string for $\mathcal{B}$ at the start of game i. Then, assuming this event does not occur, we can more easily characterize the strategies of $\mathcal{B}$.

To define this event, we introduce some notation to characterize $\mathcal{B}$'s queries. We refer to all $(i-1) \cdot T$ queries $\mathcal{B}$ makes before the start of game i as "offline" queries, and we refer to the T queries made in game i as "online" queries. An offline query is said to be "hitting" if its output is equal to an output or either of the two inputs to some prior query, i.e. it "hits" a previous query. We are now ready to state our key knowledge gaining event.

- We say that $\mathsf{E}^i_{\mathsf{hit}}$ holds if there are more than $i \cdot \mathrm{poly}(n)$ hitting queries among the $(i-1) \cdot T$ offline queries.

Briefly, we justify why $\mathsf{E}^i_{\mathsf{hit}}$ holds with very small probability. The output for each query is uniformly sampled, and there are at most $3 \cdot (i-1) \cdot T$ values to hit across inputs/outputs in the previous $(i-1) \cdot T$ offline queries. So the probability each offline query is a hitting query is at most $3iT/2^n$, meaning we expect at most $3i^2T^2/2^n$ hitting queries accounting for all $(i-1) \cdot T$ offline queries. Furthermore, we show using a Chernoff bound that there will not be more than $i \cdot \mathrm{poly}(n) \cdot \max(1, iT^2/2^n) = O(i)$ hitting queries (assuming $ST^2 \leqslant 2^n$) with high probability (recall that we ignore $\mathrm{poly}(n)$ terms).

A case analysis based on collision queries. Now, assuming there are at most $i \cdot \mathrm{poly}(n)$ hitting queries, we are ready to show that $\mathsf{Pr}[W_i | W_{<i}] \leqslant O(S/2^n + T^2/2^n)$. To do so, we look at the following "collision" queries corresponding to the valid collision $\mathsf{msg}_i = (m_L, m_R) \neq \mathsf{msg}'_i = (m'_L, m'_R)$ that $\mathcal{B}$ outputs in game i (we assume that $\mathcal{B}$ makes all of these queries at some point during or before game i).

- Q_1, Q_2, Q_3 are the queries $y_L \leftarrow h_1(\mathsf{k}_i \| m_L)$, $y_R \leftarrow h_2(\mathsf{k}_i \| m_R)$, $z \leftarrow h_3(y_L \| y_R)$, respectively.

- Q'_1, Q'_2, Q'_3 are the queries $y'_L \leftarrow h_1(\mathsf{k}_i \| m'_L)$, $y'_R \leftarrow h_2(\mathsf{k}_i \| m'_R)$, $z \leftarrow h_3(y'_L \| y'_R)$, respectively.

Recall that we assumed the collision occurs among Q_3, Q'_3 (not at the leaves), so it must be the case that $(y_L, y_R) \neq (y'_L, y'_R)$ and queries Q_3, Q'_3 are distinct.

If all of these collision queries are online (were first made during game i), then clearly $\mathsf{Pr}[W_i | W_{<i}] \leqslant O(T^2/2^n)$. Specifically, as Q_3 and Q'_3 are online (and distinct by assumption) and form a collision, this follows by a birthday bound on at most T online—and hence lazily sampled—queries that $\mathcal{B}$ makes during game i. The challenge comes when analyzing the cases where $\mathcal{B}$ may have made some of these queries *before* game i, so it could have encoded information about these queries in its state. To do so, we consider the following remaining cases, which cover all possible strategies that $\mathcal{B}$ may employ.

As we already considered when both Q_3, Q'_3 are online queries, it must be the case that one of Q_3, Q'_3 must be an offline query. Assume without loss of generality that Q_3 is offline. Then either **(A)** both Q_1, Q_2 are online, **(B)** exactly one of Q_1, Q_2 are online, or Q_1, Q_2 are both offline. The latter case implies that Q_1, Q_2, Q_3 are all offline. Then either is the case that **(C)** Q'_3 is online, or Q'_3 is also offline. If Q'_3 is also offline, then either we reduce to case (A) or (B) above by symmetry, or it holds that **(D)** all $Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3$ are offline. So, it suffices to show in cases (A-D) that $\mathsf{Pr}[W_i | W_{<i}] \leqslant \delta \leqslant O(S/2^n + T^2/2^n)$, at least assuming $ST^2 \leqslant 2^n$. We proceed to give the main ideas behind each of these cases.

(A) Q_3 is offline but Q_1, Q_2 are online.

There are at most $(i-1) \cdot T$ options for the offline query Q_3. Both online queries Q_1 and Q_2 have to hit such a query, which happens with at most $(i-1) \cdot T \cdot (T/2^n)^2 = O(iT^3/2^{2n}) \leqslant O(T/2^n)$ when $ST^2 \leqslant 2^n$ since $i \leqslant S$.

(B) Q_1, Q_3 are offline but Q_2 is online (symmetrically for Q_1 online and Q_2 offline).

In this case, we claim we can "associate" the key k_i in game i to the query Q_3 since both Q_1 and Q_3 are offline queries. If we can associate k_i to at most k possible Q_3 queries, then this implies the probability the output of some online query hits an input of such an associated query is at most $k \cdot T/2^n$. But how many Q_3 queries can we associate to a key k_i?

In the worst case, k_i may be associated to $(i-1) \cdot T$ many Q_3 queries, but this implies a suboptimal bound of $O(iT^2/2^n)$. But this cannot be true for too many values of k simultaneously. In particular, if a Q_3 query is associated with more than k possible values of k, this means there are k hitting queries, so each Q_3 query can be associated with at most $O(i)$ keys. This implies there are at most $O(i^2 T)$ pairs of associated k values with potential Q_3 queries, meaning a random k_i value will be associated with at

most $O(i^2T/2^n)$ potential Q_3 values on average. Plugging this average-case bound into k above, this implies a bound of $O(i^2T^2/2^{2n}) \leqslant O(S/2^n)$ assuming $ST^2 \leqslant 2^n$ given $i \leqslant S$.

(C) Q_1, Q_2, Q_3 are offline but Q_3' is online.

In this case, Q_3' is a distinct query from Q_3 by assumption, but again we can "associate" Q_3 with k_i as above. Then, since Q_3' must share an output with Q_3, the same argument as above gives a bound of $O(S/2^n)$ in this case.

(D) All collision queries $Q_1, Q_2, Q_3, Q_1', Q_2', Q_3'$ are offline.

In this case, we show that every full collision structure with respect to some k among the offline queries leads to a hitting query. Furthermore, two collision structures cannot share the same hitting query. So if there are at most $O(i)$ hitting queries, the probability $Q_1, Q_2, Q_3, Q_1', Q_2', Q_3'$ are all offline for a random k_i is at most $O(i/2^n) \leqslant O(S/2^n)$.

Thus, we showed that $\mathsf{Pr}[W_i|W_{<i}] \leqslant O(S/2^n + T^2/2^n)$ no matter when $Q_1, Q_2, Q_3, Q_1', Q_2', Q_3'$ were queried before or during game i. Further, recall the last case where there is a collision at one of the leaves (corresponding to Q_1, Q_1' or Q_2, Q_2'), which can happen with at most $O(S/2^n + T^2/2^n)$ probability since such a collision directly involves k_i. Thus, in all possible cases, we have shown $\mathsf{Pr}[W_i|W_{<i}] \leqslant \delta \leqslant O(S/2^n + T^2/2^n)$, at least assuming $ST^2 \leqslant 2^n$. Finally, by the framework of [CGLQ20, AGL22], this implies the same bound (up to a multiplicative factor of 2) on the advantage of finding a collision for any (S, T)-attacker.

Chapter Organization. In the rest of the chapter we provide the technical details of these results. In Section 10.1, we cover the required preliminaries for this chapter. In Section 10.2, we lay out the general framework for our main results, based on the Merkle-Damgård transform using a keyed inner hash. In 10.3, we present different approaches of instantiating the keyed inner hash using Merkle-Damgård itself. Section 10.4 contains our main result where we instantiate the inner hash using a two-level Merkle tree. Finally in Section 10.5, we lay out an alternative framework based on Merkle trees with a keyed inner hash.

10.1 Preliminaries

We let $\mathbb{N} = \{1, 2, 3, \ldots\}$ denote the natural numbers. The set of all functions with domain D and range R is denoted by $\mathsf{Fcs}(D, R)$. We let $*$ denote a wildcard element. For example $(*, z) \in L$ is true if there is an ordered pair in L where z is the second element (the type of the wildcard element shall be clear from the context). For a random variable X we use $\mathsf{E}[X]$ to denote its expected value. We use $x \leftarrow\!\!\$\, D$ to denote sampling x uniformly sampling from the elements of D. All logarithms in this chapter are for base 2 unless otherwise specified.

For a bit-string s, we use $|s|$ to denote the number of bits in s. For two strings s_1, s_2, we use $s_1 \| s_2$ to denote the concatenation of two strings. We use standard regular expression notation where s^* denotes 0 or more copies of s, s^+ denotes one or more copies of s, and s^k denotes k copies of s. Similarly, for a set S, we use S^*, S^+, and S^k to represent 0 or more, 1 or more, of k elements takes from a set S. In particular, we use $\{0,1\}^*$ to represent any arbitrary string of bits. We use the notation $\{0,1\}^{\leqslant k}$ to represent a string of length at most k.

Chernoff bound. We state a Chernoff bound which we use in the technical part of the chapter.

Proposition 10.1.1. *Let $n \in \mathbb{N}$. Let $X_1, X_2, \ldots, X_n$ be independent 0-1 random variables. Let $X = \sum_{i=1}^{n} X_i$. Let μ' be such that $\mathsf{E}[X] \leqslant \mu'$. Then we have that*

$$\mathsf{Pr}\left[X \geqslant (1+\delta)\mu'\right] \leqslant e^{-\delta\mu'/3} .$$

Notice that this version is somewhat non-standard as it even works when we know only an upper bound on the expectation (usually, in standard formulations of Chernoff bound, we need to know the expectation exactly). We refer the interested reader to [FGK23] for proof of this.

Auxiliary-input Random Oracle Model (AI-ROM). The auxiliary-input random oracle model, introduced by Unruh [Unr07], captures the power of non-uniform adversaries against random oracles. An attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in this model is formalized as a two stage adversary. In its first stage, which is referred to as the preprocessing phase, $\mathcal{A}_1$ has unbounded access to the random oracle h, and outputs any arbitrary S-bit advice string or auxiliary input σ. In the second stage, referred to as the online phase, gets σ as input, $\mathcal{A}_2$ can make at most T queries to its oracle h. Its aim is to accomplish some task involving h, e.g. find a collision in a construction based on h. We refer to such an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as an (S,T)-attacker.

Collision resistance of g^h in AI-ROM. We next formalize the keyed-collision resistance of an iterated hash function construction g relative to a hash function $h : \{0,1\}^a \to \{0,1\}^n$ in the AI-ROM. The construction g has a parameter κ associated with it, where κ is the bit length of k used in g. It first samples a random function $h : \{0,1\}^a \to \{0,1\}^n$. The adversary $\mathcal{A}_1$ gets unbounded access to h, and it outputs an advice string σ. At this time, $\mathcal{A}_2$ is given the auxiliary input σ, a randomly sampled k from $\{0,1\}^\kappa$, as well as oracle access to h, and it needs to find $\mathsf{msg} \neq \mathsf{msg}'$ such that $g^h(\mathsf{k}, \mathsf{msg}) = g^h(\mathsf{k}, \mathsf{msg}')$. This game, denoted $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{g^h}$ is formally defined in Fig. 10.1.

Definition 10.1.2 (AI-CR Advantage)**.** *The advantage of an adversary $\mathcal{A}$ against the collision resistance of g^h in the AI-ROM is*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{g^h}(\mathcal{A}) = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{ai\text{-}cr}}_{g^h}(\mathcal{A}) = \mathsf{true}\right].$$

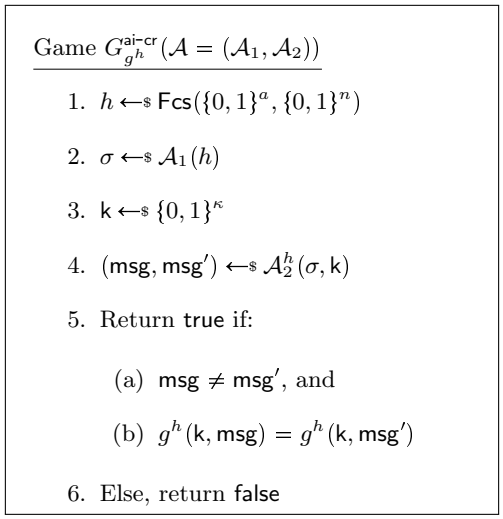


Figure 10.1: The collision resistance game $G^{\mathsf{ai\text{-}cr}}_{g^h}$ in AI-ROM for a function g^h based on a random oracle $h : \{0,1\}^a \to \{0,1\}^n$. The construction g^h has a parameter κ associated with it, where κ is the bit length of k used in g.

For parameters $S, T \in \mathbb{N}$, we overload notation and denote

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{g^h}(S,T) = \max_{\mathcal{A}} \left\{ \mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{g^h}(\mathcal{A}) \right\},$$

where the maximum is over all (S,T)-attackers.

Throughout the chapter, for any (S,T)-attacker $\mathcal{A}$ that outputs messages $\mathsf{msg}, \mathsf{msg}'$ that causes $\mathsf{G}^{\mathsf{ai\text{-}cr}}_{g^h}(\mathcal{A})$ to output true on a key k, we assume that $\mathcal{A}$ has fully queried $g^h(\mathsf{k}, \mathsf{msg})$ and $g^h(\mathsf{k}, \mathsf{msg}')$. This is true without loss of generality (up to constant factors in the advantage) as if there exists any (S,T)-attacker $\mathcal{A}$ that does not, you can construct an $(S, T + 2\ell)$-attacker $\mathcal{B}$ that does, where g^h requires at most ℓ invocations of h to compute either $g^h(\mathsf{k}, \mathsf{msg})$ or $g^h(\mathsf{k}, \mathsf{msg}')$. As $\ell \leqslant T$, the resulting attacker will have comparable advantage up to constant factors in T. We note that this is a standard assumption in existing related works in the AI-ROM.

On padding. The variable-input length hash functions we consider of this work all act on messages which have been parsed into many fixed size blocks of some specified size s. We therefore need a padding function that takes arbitrary length inputs and converts them to a sequence of fixed-size blocks. We need to ensure that this padding function maintains certain properties like injectivity in order to guarantee that if an adversary finds a collision on the padded versions of messages, then it implies a collision with respect to the

underlying messages as well. For the purpose of this chapter, we define the following padding function, which is a slightly simplified version of the padding function used by the SHA family of hash functions (see [GB08, Section 8.5] for more discussion on MD-compliant padding functions). The function pad we use takes in a message $\mathsf{msg} \in \{0,1\}^*$, an integer $s \in \mathbb{N}$ representing the size of each block, and an integer n that stipulates that $|\mathsf{msg}| \leqslant 2^n$. The construction is formally defined as follows.

$\underline{\mathsf{pad}(\mathsf{msg}, s, n)}$:

1. Let $k = s - ((|\mathsf{msg}| + n) \bmod s + 1)$.
2. Interpret $|\mathsf{msg}| \in [2^n]$ as an n-bit string.
3. Output $(m_1, \ldots, m_\ell) \in (\{0,1\}^s)^\ell$ where $m_1 \| \ldots \| m_\ell = \mathsf{msg} \| |\mathsf{msg}| \| 1 \| 0^k$.

We formalize the guarantees we use for this padding function in the following theorem.

Theorem 10.1.3 (Padding)**.** *Let $s, n \in \mathbb{N}$. The function $\mathsf{pad}(\mathsf{msg}, s, n)$ on messages $\mathsf{msg} \in \{0,1\}^{2^n}$ satisfies the following properties:*

1. $|\mathsf{pad}(\mathsf{msg}, s, n)| \in (\{0,1\}^s)^\ell$ *for* $\ell = \lceil (|\mathsf{msg}| + n + 1)/s \rceil$.
2. *There is a unique decoding procedure that outputs* msg *given* $\mathsf{pad}(\mathsf{msg}, s, n)$*, and outputs* $\perp$ *on invalid padded messages.*
3. *If* $\mathsf{pad}(\mathsf{msg}, s, n) = \mathsf{pad}(\mathsf{msg}', s, n)$*, then* $\mathsf{msg} = \mathsf{msg}'$.
4. *If* $|\mathsf{msg}| < |\mathsf{msg}'|$*, then* $\mathsf{pad}(\mathsf{msg}, s, n)$ *is not a suffix of* $\mathsf{pad}(\mathsf{msg}', s, n)$.

Proof. The length of a padded message follows by construction.

For the decoding process, note that to recover msg from $\mathsf{pad}(\mathsf{msg}, s, n)$, you simply need to remove the trailing 0s and preceding 1. Parse the remaining final n bits as an integer $|\mathsf{msg}|$ and check that there are exactly $|\mathsf{msg}|$ bits leftover. If such a recovery process fails, we output $\perp$ and say that the padded message is invalid (as it corresponds to no message msg).

To see injectivity, suppose two messages $\mathsf{msg}, \mathsf{msg}'$ are padded to the same value. This means they both end in $1\|0^k$ for $k = (|\mathsf{msg}| + n + 1) \bmod s$ and $|\mathsf{msg}| = |\mathsf{msg}'|$. Thus, if both are valid padded messages, they only consist of $|\mathsf{msg}|$ remaining bits, which are the same, implying that $\mathsf{msg} = \mathsf{msg}'$, as required.

Now suppose that $|\mathsf{msg}| < |\mathsf{msg}'|$ but $\mathsf{pad}(\mathsf{msg}, s, n)$ is a suffix of $\mathsf{pad}(\mathsf{msg}', s, n)$. This implies that both strings end in $1\|0^k$ for $k = (|\mathsf{msg}| + n + 1) \bmod s$ and $|\mathsf{msg}| = |\mathsf{msg}'|$. Thus, one of the encodings must be invalid as we assumed $|\mathsf{msg}| < |\mathsf{msg}'|$. ■

Construction $\mathsf{KMD}^h(\mathsf{k}, \mathsf{msg})$:

1. $(m_1, m_2, \ldots, m_\ell) \leftarrow \mathrm{pad}(\mathsf{msg}, s, n)$ where $\ell = \lceil (|\mathsf{msg}| + n + 1)/s \rceil$ given by Theorem 10.1.3.
2. Initialize $y_0 = 0^n$, and compute $y_i \leftarrow h(\mathsf{k}, y_{i-1} \| m_i)$ for $i = 1, \ldots, \ell$.
3. Output y_ℓ.

Figure 10.2: The keyed Merkle-Damgård construction $\mathsf{KMD}^h \colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ given any underlying function $h \colon \{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$, where κ is the key length, n is the output length, and s is the message block size.

10.2 Merkle-Damgård Framework with a Keyed Inner Hash

In this section, we lay out the general framework for our main results, based on the Merkle-Damgård transform using a keyed inner hash. We note that this framework has been explicitly considered in the uniform setting in Section 8.5 of the lecture notes of Goldwasser and Bellare [GB08].[1] We extend this framework to the preprocessing setting, modeled by the AI-ROM of Unruh [Unr07], noting that the high level ideas are similar.

For a key length κ, output length n, and message block size s, we assume an underlying primitive $h \colon \{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$. In other words, viewing the primitive h as a function from $\{0,1\}^a$ to $\{0,1\}^n$, this implies that $a = \kappa + n + s$.

Given such a primitive h, we define the following keyed Merkle-Damgård hash function $\mathsf{KMD}^h \colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$. On input key $\mathsf{k} \in \{0,1\}^\kappa$ and message $\mathsf{msg} \in \{0,1\}^*$ of length at most 2^n, the function KMD^h first pads the message to split it into $\ell = \lceil (b+n+1)/s \rceil$ message blocks of size s as in Theorem 10.1.3. It then essentially computes the Merkle-Damgård hash function using the underlying hash function h, except that the key k is inserted into every invocation of h. This is formalized in Figure 10.2.

Since the key k is included in every call to the underlying primitive h, it follows that any (S,T)-attacker that finds a collision in KMD^h with respect to a key k also finds a collision in h with respect to k. This is formalized via a reduction, which gives the following theorem. Again, we note that the following theorem very closely follows the reduction given in [GB08, Section 8.5], but we give the full details in the AI-ROM for completeness.

Theorem 10.2.1. *Let $\kappa, n, s \in \mathbb{N}$. Let $h \colon \{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$ be any function, and let $\mathsf{KMD}^h \colon \{0,1\}^\kappa \times$*

[1] The existence of this variant of the Merkle-Damgård transform has gone completely unnoticed in recent works studying non-uniform security of this transformation [CDGS18, ACDW20, GK22, AGL22].

$\{0,1\}^* \to \{0,1\}^n$. *Then, for every* $S, T \in \mathbb{N}$, *it holds that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{KMD}^h}(S,T) \leqslant \mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{h}(S,T).$$

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be any (S,T)-attacker against the collision resistance of the construction KMD^h. Recall that we assume for simplicity that if $\mathcal{A}_2$ outputs a successful collision $\mathsf{msg}_{\mathcal{A}} \neq \mathsf{msg}'_{\mathcal{A}}$ such that $\mathsf{KMD}^h(\mathsf{k}, \mathsf{msg}_{\mathcal{A}}) = \mathsf{KMD}^h(\mathsf{k}, \mathsf{msg}'_{\mathcal{A}})$, then $\mathcal{A}_2$ queried all the necessary values to fully compute each of these functions. We construct an (S,T)-attacker $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against the collision resistance of h as follows.

In the preprocessing phase, $\mathcal{B}_1(h)$ simply computes $\sigma \leftarrow \mathcal{A}_1(h)$ and outputs σ where $|\sigma| \leqslant S$. In the online phase, $\mathcal{B}_2^h(\sigma, \mathsf{k})$ is given a random key $\mathsf{k} \leftarrow \{0,1\}^\kappa$ that it needs to find a collision for. $\mathcal{B}_2$ computes $(\mathsf{msg}_{\mathcal{A}}, \mathsf{msg}'_{\mathcal{A}}) \leftarrow \mathcal{A}_2^h(\sigma, \mathsf{k})$, where it uses its own query access to the function h to answer queries that $\mathcal{A}_2$ makes to h. $\mathcal{B}_2$ sets $(m_1, \ldots, m_\ell) \leftarrow \mathrm{pad}(\mathsf{msg}_{\mathcal{A}}, s, n)$ and $(m'_1, \ldots, m'_{\ell'}) \leftarrow \mathrm{pad}(\mathsf{msg}'_{\mathcal{A}}, s, n)$, where $\ell = \lceil (|\mathsf{msg}_{\mathcal{A}}| + n + 1)/s \rceil$ and $\ell' = \lceil (|\mathsf{msg}'_{\mathcal{A}}| + n + 1)/s \rceil$. It initializes $y_0 = y'_0 = 0^n$ as in the construction of KMD^h. For $i = 1, \ldots, \ell$, it looks for the queries $y_i \leftarrow h(\mathsf{k}, y_{i-1} \| m_i)$, and for $j = 1, \ldots, \ell'$, it looks for the queries $y'_j \leftarrow h(\mathsf{k}, y'_{j-1} \| m'_j)$ made by $\mathcal{A}_2$ (which must exist if $\mathcal{A}_2$ succeeds in outputting a collision by assumption). Let $i^\star, j^\star$ be the minimal $i, j > 0$ such that $y_i = y'_j$ but $y_{i-1} \| m_i \neq y'_{j-1} \| m'_{j-1}$. If such an $i^\star, j^\star$ exists, $\mathcal{B}_2$ outputs $(\mathsf{msg}, \mathsf{msg}')$ where $\mathsf{msg} = y_{i^\star - 1} \| m_{i^\star}$ and $\mathsf{msg}' = y'_{j^\star - 1} \| m'_{j^\star}$. Otherwise, $\mathcal{B}_2$ aborts and outputs $\perp$.

It remains to analyze the complexity and success probability of $\mathcal{B}$. First, as $\mathcal{B}$ simply runs $\mathcal{A}$ in the preprocessing and online phases, it follows that if $\mathcal{A}$ is an (S,T)-attacker, then so is $\mathcal{B}$.

For the success probability, we claim that whenever $\mathcal{A}$ succeeds, then $\mathcal{B}$ also succeeds. To see this, first observe that $y_\ell = y'_{\ell'}$ by assumption. If $y_{\ell-1} \| m_{\ell-1} \neq y'_{\ell'-1} \| m'_{\ell'-1}$, that implies that such an $i^\star, j^\star$ exist leading to a valid collision, and we are done. Otherwise, we have that $y_{\ell-1} = y'_{\ell'-1}$. Assume without loss of generality that $\ell' \geqslant \ell$ and let $\ell' = \ell + i$ for $i \geqslant 0$. Peeling back message blocks one at a time, $\mathcal{B}$ will either find a collision in h relative to k, or we will have two sequences of identical message blocks $(m_1, \ldots, m_\ell)$ and $(m'_{i+1}, \ldots, m'_{i+\ell})$. We proceed to show that if $\mathcal{B}$ does not find a collision, then $\mathcal{A}$ must have not actually succeeded in finding a collision.

First, consider the case where $\ell' = \ell$ so $i = 0$. By injectivity of pad of Theorem 10.1.3, this implies that $\mathsf{msg}_{\mathcal{A}} = \mathsf{msg}'_{\mathcal{A}}$, so $\mathcal{A}$ in fact did not succeed at finding a valid collision. Next, consider the case where $\ell' > \ell$ so $i > 0$. Theorem 10.1.3 guarantees that $\mathrm{pad}(\mathsf{msg}_{\mathcal{A}}, s, n)$ is not a suffix of $\mathrm{pad}(\mathsf{msg}'_{\mathcal{A}}, s, n)$, so one of the padded messages must be invalid.

Thus, whenever $\mathcal{A}$ succeeds, $\mathcal{B}$ will also succeed, so it follows that

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{KMD}^h}(S,T) \leqslant \mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{h}(S,T),$$

as required. ∎

Next, we recall that if h is a keyed function modeled as a random oracle (in the AI-ROM of Unruh [Unr07]), Dodis, Guo, and Katz [DGK17] give the following bound on the success probability that any (S,T)-attacker can find a collision in h with respect to a random key.

Theorem 10.2.2 ([DGK17])**.** *Let $h\colon \{0,1\}^\kappa \times \{0,1\}^b \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, for any $S,T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{h}(S,T) \leqslant \frac{2S+2\kappa}{2^\kappa} + \frac{50T^2}{2^n}.$$

Combining Theorems 10.2.1 and 10.2.2, we get the following result.

Theorem 10.2.3. *Let $a,\kappa,n \in \mathbb{N}$ be such that $a > \kappa + n$. Let $h\colon \{0,1\}^a \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, there is an $H^h\colon \{0,1\}^\kappa \times \{0,1\}^{<2^n} \to \{0,1\}^n$ such that:*

1. *For any $S,T \in \mathbb{N}$,*
$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S,T) \leqslant \frac{2S+2\kappa}{2^\kappa} + \frac{50T^2}{2^n}.$$
2. *One evaluation of H^h on messages of length b requires $\lceil (b+n+1)/s \rceil$ queries to h, where $s = a-\kappa-n$.*

Proof. As $a > \kappa + n$, we define $s = a - \kappa - n > 0$. Then, we view h as a function from $\{0,1\}^\kappa \times \{0,1\}^{n+s}$ to $\{0,1\}^n$ and use it in the construction KMD^h of Figure 10.2 to get the hash function H^h required by the theorem.

The bound on the advantage immediately follows as a corollary of Theorems 10.2.1 and 10.2.2. As for efficiency, we note that padding a message $\mathsf{msg} \in \{0,1\}^b$ via Theorem 10.1.3 results in a message consisting of ℓ blocks each of length s, where $\ell = \lceil (b+n+1)/s \rceil$. Each block of the message requires a single invocation of h, so evaluating KMD^h on msg requires $\ell = \lceil (b+n+1)/s \rceil$ queries to h as required. ∎

10.3 Instantiating the Inner Hash: Standard MD

We next consider instantiating the Merkle-Damgård framework of Section 10.2 whenever the underlying hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ has input length a such that $n < a < \kappa + n + s$, where κ is the key length, n is the output length, and s is the desired message block size. Specifically, our goal is to use such a primitive $h\colon \{0,1\}^a \times \{0,1\}^n$ to build a larger (fixed-length) hash function $g^h\colon \{0,1\}^\kappa \times \{0,1\}^{n+s}$ that *can* be plugged into the construction KMD^{g^h}.

The first approach we consider is by simply building g from h using the standard Merkle-Damgård construction where h is not keyed in every invocation. We emphasize that in the standard version of MD, a random initialization vector/ key is still included in the first invocation of h. However, it is not included in the subsequent invocations of h, allowing h to take in smaller inputs overall.

Construction $\mathsf{FMD}_b^h(\mathsf{k}, \mathsf{msg})$:

1. Parse msg into ℓ blocks $(m_1, m_2, \ldots, m_\ell)$ each of size s where $\ell = \lceil b/s \rceil$ and m_ℓ is padded with 0s if needed.
2. Initialize $y_0 = \mathsf{k}$, and compute $y_i \leftarrow h(y_{i-1}, m_i)$ for $i = 1, \ldots, \ell$.
3. Output y_ℓ.

Figure 10.3: The standard Merkle-Damgård construction with fixed input message length b $\mathsf{FMD}_b^h \colon \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$ given any underlying function $h \colon \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^n$, where n is the key and output length, and s is the message block size.

Given an underlying hash function $h \colon \{0,1\}^a \to \{0,1\}^n$ where $a > \max(\kappa, n)$, we define the standard Merkle-Damgård hash function with fixed input message length b $\mathsf{FMD}_b^h \colon \{0,1\}^\kappa \times \{0,1\}^b \to \{0,1\}^n$ as follows. For sake of simplicity and due to the nature of the MD construction, we will assume that the key length κ is equal to the output length n. Let $s = a - n$ be the message block size we will include in each invocation of the underlying h. On input a key $\mathsf{k} \in \{0,1\}^n$ and a message $\mathsf{msg} \in \{0,1\}^b$, the function FMD_b^h splits the message msg into $\ell = \lceil b/s \rceil$ message blocks of size s (adding 0s to the last block if needed). It initializes $y_0 = \mathsf{k}$, and for $i = 1, \ldots, \ell$, computes y_i as the hash of y_{i-1} concatenated with m_i using h. The output of FMD_b^h is then y_ℓ. This is formalized in Figure 10.3.

If we instantiate $g^h := \mathsf{FMD}_b^h$ only on input messages of size at most $b = n + s$ as required to instantiate it inside KMD^{g^h}, then we only need to worry about ℓ-block collisions for $\ell = \lceil (n+s)/(a-n) \rceil$. Akshima, Guo, and Liu [AGL22] currently show the best known upper bound on the advantage of finding general ℓ-block collisions in MD^h, given by the following theorem.

Theorem 10.3.1 ([AGL22]). *Let $b, n, s \in \mathbb{N}$ such that $b > s$, and set $\ell = \lceil b/s \rceil$. Let $h \colon \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, for any $S, T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{FMD}_b^h}(S,T) \leqslant \begin{cases} \frac{200n \cdot (ST + T^2)}{2^n} & \text{if } \ell = 2, \text{ and} \\ \frac{34n \cdot ST\ell}{2^n} \cdot \max\left(1, \frac{ST^2}{2^n}\right) + \frac{2 \cdot T^2}{2^n} & \text{if } \ell > 2. \end{cases}$$

We note that Ghoshal and Komargodski [GK22] give a bound of $O(ST/2^n + T^2/2^n)$ whenever ℓ is a constant, which doesn't require the assumption that $ST^2 \leqslant 2^n$. However, their bound does not extend to super constant ℓ.

Combined with Theorem 10.2.1, we get the following result.

Theorem 10.3.2. *Let $a, n, s \in \mathbb{N}$ be such that $a > n$. Let $\ell = \lceil (n+s)/(a-n) \rceil$. Let $h\colon \{0,1\}^a \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, there is an $H^h\colon \{0,1\}^\kappa \times \{0,1\}^{<2^n} \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S,T) \leqslant \begin{cases} \frac{200n\cdot(ST+T^2)}{2^n} & \text{if } \ell = 2, \text{ and} \\ \frac{34n\cdot ST\ell}{2^n} \cdot \max\left(1, \frac{ST^2}{2^n}\right) + \frac{2\cdot T^2}{2^n} & \text{if } \ell > 2. \end{cases}$$

2. *One evaluation of H^h on messages of length b requires $\ell \cdot \lceil (b+n+1)/s \rceil$ queries to h.*

Proof. As $a > n$, we parse $h\colon \{0,1\}^n \times \{0,1\}^{a-n} \to \{0,1\}^n$ and use it to construct FMD^h_{n+s} as defined in Figure 10.3. We then use FMD^h_{n+s} as the primitive underlying our MD-based hash function of Figure 10.2. So, we set

$$H^h := \mathsf{KMD}^{\mathsf{FMD}^h_{n+s}}.$$

The bound on the advantage follows as a corollary to Theorems 10.2.1 and 10.3.1, where the message length required for the FMD^h_{n+s} construction is only $n+s$. Furthermore, this implies that FMD^h_{n+s} requires $\ell = \lceil (n+s)/(a-n) \rceil$ invocations of h per invocation of FMD^h_{n+s}, and $\mathsf{KMD}^{\mathsf{FMD}^h_{n+s}}$ requires $\lceil (b+n+1)/s \rceil$ invocations of FMD^h_{n+s}, giving the resulting efficiency bound. ∎

We emphasize that because our reduction in Theorem 10.2.1 is generic, any improvement on the bound of [AGL22, GK22] for finding an ℓ-block collision in MD will immediately imply an improved bound in Theorem 10.3.2.

We next state a corollary of Theorem 10.3.2 where we restrict to using $\ell = 2$ invocations of h in the underlying FMD^h_{n+s} construction. For this setting, the bound of [AGL22] above is optimal (up to $\mathrm{poly}(n)$ factors), and we observe that the resulting bound matches our desired bound of $O(S/2^n + T^2/2^n)$ whenever $S \leqslant T$.

Corollary 10.3.3. *Let $a, n, s \in \mathbb{N}$ be such that $a \geqslant 3n/2 + s/2$. Let $h\colon \{0,1\}^a \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, there is an $H^h\colon \{0,1\}^\kappa \times \{0,1\}^{<2^n} \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S,T) \leqslant \frac{200n\cdot(ST+T^2)}{2^n}$$

2. *One evaluation of H^h on messages of length b requires $2 \cdot \lceil (b+n+1)/s \rceil$ queries to h.*

Proof. Restricting to $\ell = 2$ in Theorem 10.3.2, we require that $\ell = \lceil (n+s)/(a-n) \rceil \leqslant 2$. This holds as long as $a \geqslant 3n/2 + s/2$, as required. ∎

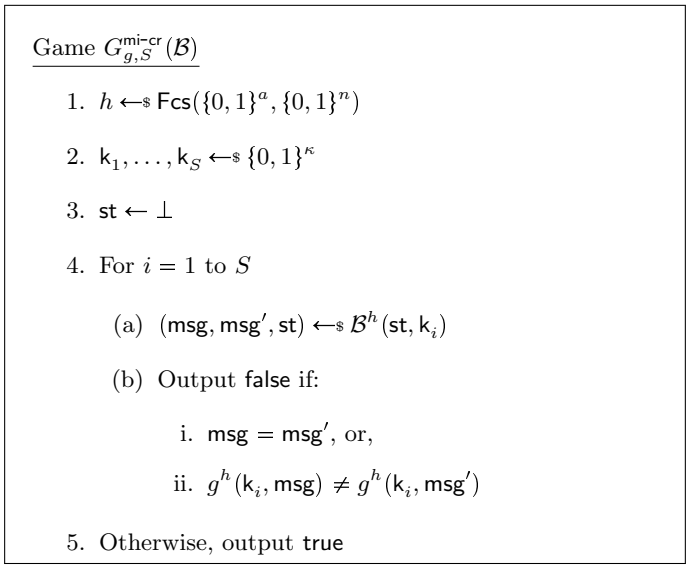


Figure 10.4: The multi-instance game $G^{\mathsf{mi\text{-}cr}}_{g^h,S}$, where $\mathcal{B}$ is a uniform adversary with oracle access to the function h.

10.4 Instantiating the Inner Hash: Two-Level Merkle Tree

In this section, we instantiate the Merkle-Damgård framework of Section 10.2, that uses a keyed inner hash, in the setting where the underlying hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ satisfies $a \geqslant \max(2n+2, \kappa + \lceil n/2 \rceil + 3)$.

The compression function in this instantiation of the MD-based framework, is a Merkle tree with two leaves, where we additionally input the key into each leaf. We describe next the framework introduced in [AGL22] that we use to analyze the collision-resistance of this construction in AI-ROM.

10.4.1 The [AGL22] Framework

In this section, we briefly introduce the framework given by [AGL22] which is useful in analyzing non-uniform security. An earlier version of this framework was introduced by [CGLQ20, ACDW20] inspired by techniques used in proving constructive Chernoff bounds in [IK10] and later refined by [ACDW20, GK22, AGL22] to upper bound $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{g^h}(S,T)$. This framework involves upper bounding the advantage of an (S,T)-attacker using the advantage of a uniform adversary for a multi-instance game that has to find collisions for S randomly chosen values of k.

We define the "multi-instance" game $\mathsf{G}^{\mathsf{mi\text{-}cr}}_{g^h,S}(\mathcal{B})$ in Fig. 10.4. We refer an adversary playing $G^{\mathsf{mi\text{-}cr}}_{g^h,S}$ and

making at most T queries for each k as a (S,T)-MI adversary. For any (S,T)-MI adversary, define

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(\mathcal{B}) = \mathsf{Pr}\left[\mathsf{G}^{\mathsf{mi\text{-}cr}}_{g^h,S}(\mathcal{B}) = \mathsf{true}\right].$$

Further,

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(S,T) = \max_{\mathcal{B}} \mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(\mathcal{B}),$$

where the maximum is taken over all (S,T)-MI adversaries. The following key lemma relates $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{g^h}(S,T)$ to $\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(S,T)$, which is proven in [AGL22].

Lemma 10.4.1 ([AGL22])**.** *Fix $S,T \in \mathbb{N}$ and $0 \leqslant \delta \leqslant 1$, if $\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(S,T) \leqslant \delta^S$, then $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{g^h}(S,T) \leqslant 2\delta$.*

Offline and Online queries. Since the adversary in the multi-instance game is stateful, we can assume without loss of generality that it does not repeat queries since they can simply remember the answers. Additionally, [AGL22] formalized the notion of "offline" and "online" queries during a particular instance of the game. When running the adversary on k_i, the queries that were made while the adversary was run on $\mathsf{k}_1, \ldots, \mathsf{k}_{i-1}$ are collectively known as the "offline" queries, and the queries made while running on k_i are "online queries".

10.4.2 Two-Level Merkle Tree

In this section, we present our construction of a keyed Merkle tree and analyze its collision resistance in the AI-ROM using the framework in the previous section. Specifically, given an underlying hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ where $a \geqslant \max(2n+2, \kappa + \lceil n/2 \rceil + 3)$, we define a keyed, 2-level Merkle tree $\mathsf{2MT}^h_b$ for message length $b \leqslant 2a - 2\kappa - 4$.

Before we define $\mathsf{2MT}^h_b$, we introduce notation that allows use to "domain-separate" h into three separate functions. Given a fixed hash function $h\colon \{0,1\}^a \to \{0,1\}^n$, we define three domain-separated functions $h_1, h_2, h_3\colon \{0,1\}^{a-2} \to \{0,1\}^n$, where $h_i(x)$ outputs $h(\hat{i} \| x)$ where $\hat{i} \in \{0,1\}^2$ is the 2-bit binary representation of i. Moreover, we refer to a query $h(\hat{i} \| *)$ as a query to the function h_i (which is also clearly a query to h).

To construct $\mathsf{2MT}^h_b\colon \{0,1\}^\kappa \times \{0,1\}^b \to \{0,1\}^n$, we use h_1 and h_2 above to process the two leaves of the depth-2 Merkle tree, where we include $\mathsf{k} \in \{0,1\}^\kappa$ in each leaf. We then feed those outputs as input to h_3 to get the output of $\mathsf{2MT}^h_b$. This construction is formalized in Figure 10.5.

Our main result of this section is the following theorem, which bounds the probability that any (S,T)-attacker finds a collision in $\mathsf{2MT}^h_b$.

Theorem 10.4.2. *Let $a, \kappa, n \in \mathbb{N}$ be such that $a \geqslant \max(\kappa + \lceil n/2 \rceil + 3, 2n+2)$. Let $h\colon \{0,1\}^a \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, for $s = 2a - 2\kappa - n - 4$, the construction $\mathsf{2MT}^h_{n+s}\colon \{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$ of Figure 10.5 satisfies the following.*

$\underline{\mathsf{2MT}_b^h(\mathsf{k}, \mathsf{msg})}$

1. Parse $\mathsf{msg} \in \{0,1\}^{2a-2\kappa-4}$ as $\mathsf{msg}_L \| \mathsf{msg}_R$ where $\mathsf{msg}_L, \mathsf{msg}_R \in \{0,1\}^{a-\kappa-2}$.
2. Compute $y_L \leftarrow h_1(\mathsf{k} \| \mathsf{msg}_L)$.
3. Compute $y_R \leftarrow h_2(\mathsf{k} \| \mathsf{msg}_R)$.
4. Output $z \leftarrow h_3(y_L \| y_R)$.

Figure 10.5: The two-level, keyed Merkle tree construction $\mathsf{2MT}_b^h \colon \{0,1\}^\kappa \times \{0,1\}^b \to \{0,1\}^n$ with fixed input message length b given any underlying function $h \colon \{0,1\}^a \to \{0,1\}^n$, where $a \geqslant \max(2n+2, \kappa+3)$ and $b \leqslant 2a-2\kappa-4$. h_1, h_2, h_3 are all domain-separated using the first two bits of h to encode $1, 2, 3$, respectively.

- *For any* $S, T \in \mathbb{N}$,

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{g^h}(S,T) \leqslant \left(\frac{S}{2^\kappa} \cdot \left(14n + 42n\gamma + 42n\gamma^2\right) + \frac{T^2}{2^n} \cdot \left(2 + \frac{2\gamma}{T} + \frac{2}{T^2}\right)\right),$$

where $\gamma = ST^2/2^n$.

We prove this theorem using the framework described in Section 10.4.1. So, by Lemma 10.4.1, it suffices to prove the following lemma which bounds the advantage of an (S,T)-MI adversary.

Lemma 10.4.3. *Let* $S, T \in \mathbb{N}$. *Then*

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(S,T) \leqslant \left(\frac{S}{2^\kappa} \cdot \left(7n + 21n \cdot \gamma + 21n \cdot \gamma^2\right) + \frac{T^2}{2^n} \cdot \left(1 + \frac{\gamma}{T} + \frac{1}{T^2}\right)\right)^S,$$

where $\gamma = ST^2/2^n$.

Proof. Following the techniques of [AGL22], we reduce the task of bounding $\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(S,T)$ to that of bounding any T-query adversaries advantage of succeeding in iteration i given that it has succeeded in all previous iterations. Fix any (S,T)-MI attacker $\mathcal{A}$. Let W_i be the indicator random variable that $\mathcal{A}$ wins on k_i in $G^{\mathsf{mi\text{-}cr}}_{g^h,S}$. Define the random variable $W_{<i} := W_1 \wedge \ldots \wedge W_{i-1}$. We have that

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(\mathcal{A}) = \mathsf{Pr}\left[W_1 \wedge W_2 \wedge \ldots \wedge W_S\right] = \prod_{i=1}^{S} \mathsf{Pr}\left[W_i | W_{<i}\right] .$$

We will prove that for every $\mathcal{A}$ and each $i \in [S]$, $\mathsf{Pr}\left[W_{<i+1}\right] \leqslant (\delta_S)^i$ where

$$\delta_S = \frac{T^2}{2^n} + 7n \cdot \frac{S}{2^\kappa} + 21n \cdot \frac{S^2T^2}{2^{n+\kappa}} + \frac{ST^3}{2^{2n}} + 21n \cdot \frac{S^3T^4}{2^{2n+\kappa}} + \frac{1}{2^n}.$$

Note that δ_S is equal to the term in the lemma statement raised to the S power. Also, we observe that if $\mathsf{Pr}[W_{<i}] \leqslant (\delta_S)^i$, then we are done since $\mathsf{Pr}[W_{<i+1}] \leqslant \mathsf{Pr}[W_{<i}]$. Therefore, we assume from here on that $\mathsf{Pr}[W_{<i}] > (\delta_S)^i$.

To simplify the analysis of $\mathsf{Pr}[W_i|W_{<i}]$, we define the "knowledge-gaining event" $\mathsf{E}^i_{\mathsf{hit}}$ as follows. We say that an offline query to h (via h_1, h_2, or h_3) is a "hitting" query if its output is equal to the output or the left/ right inputs of a prior h query (after removing the first two bits if necessary). We define $\mathsf{E}^i_{\mathsf{hit}}$ to be the event that there are more than $7n \cdot \max(i, 3i^2T^2/2^n)$ hitting queries among the first $(i-1) \cdot T$ offline queries.

Conditioning on this event, we upper bound $\mathsf{Pr}[W_i|W_{<i}]$ for any $i \in [S]$ as follows.

$$\begin{aligned}
\mathsf{Pr}[W_i|W_{<i}] &= \mathsf{Pr}\left[W_i|W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}\right] \cdot \mathsf{Pr}\left[\neg \mathsf{E}^i_{\mathsf{hit}}|W_{<i}\right] \\
&\quad + \mathsf{Pr}\left[W_i|W_{<i} \wedge \mathsf{E}^i_{\mathsf{hit}}\right] \cdot \mathsf{Pr}\left[\mathsf{E}^i_{\mathsf{hit}}|W_{<i}\right] \\
&\leqslant \mathsf{Pr}\left[W_i|W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}\right] + \mathsf{Pr}\left[\mathsf{E}^i_{\mathsf{hit}}|W_{<i}\right] \\
&\leqslant \mathsf{Pr}\left[W_i|W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}\right] + \frac{\mathsf{Pr}\left[\mathsf{E}^i_{\mathsf{hit}}\right]}{\mathsf{Pr}[W_{<i}]} \\
&\leqslant \mathsf{Pr}\left[W_i|W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}\right] + \mathsf{Pr}\left[\mathsf{E}^i_{\mathsf{hit}}\right] \cdot (2^{n \cdot i}),
\end{aligned}$$

where the last line follows by the assumption that $\mathsf{Pr}[W_{<i}] > (\delta_S)^i \geqslant (1/2^n)^i$. Hence, it suffices to separately bound $\mathsf{Pr}[\mathsf{E}^i_{\mathsf{hit}}]$ and $\mathsf{Pr}[W_i|W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}]$, which we do in Claims 10.4.4 and 10.4.5, respectively, below.

Specifically, we show in Claim 10.4.4 that $\mathsf{Pr}\left[\mathsf{E}^i_{\mathsf{hit}}\right] \leqslant (2^{-n \cdot 2i})$, so $\mathsf{Pr}\left[\mathsf{E}^i_{\mathsf{hit}}\right] \cdot (2^{n \cdot i}) \leqslant 1/2^n$. Then in Claim 10.4.5, we show that $\mathsf{Pr}[W_i|W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}] \leqslant \delta_S - 1/2^n$.

It follows that for any (S,T)-MI attacker $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(\mathcal{A}) = \mathsf{Pr}[W_{<S+1}] = \prod_{i=1}^{S} \mathsf{Pr}[W_i|W_{<i}] \leqslant (\delta_S)^S.$$

As this holds for any such $\mathcal{A}$, it follows that $\mathsf{Adv}^{\mathsf{mi\text{-}cr}}_{g^h}(S,T) \leqslant (\delta_S)^S$, as required by the lemma statement. It remains to prove the claims.

Claim 10.4.4. $\mathsf{Pr}\left[\mathsf{E}^i_{\mathsf{hit}}\right] \leqslant (2^n)^{-2i}$.

Proof. Let the random variable Z_j be an indicator random variable for the j-th query among the first $(i-1) \cdot T$ queries being a hitting query. Define the random variable $Z := \sum_{j=1}^{(i-1)\cdot T} Z_j$ to denote the total number of hitting queries. Thus, it suffices to show that $\mathsf{Pr}\left[Z \geqslant 7n \cdot \max(i, 3i^2T^2/2^n)\right] \leqslant 2^{-2n \cdot i}$.

First, note that output of each query can hit either the output of a previous query, or one of the left/right inputs. As each new output is lazily sampled and has to match n bits of some previous value, this implies that

$$\mathsf{E}[Z_j] = \mathsf{Pr}[Z_j = 1] \leqslant 3iT/2^n = 3iT/2^n.$$

Using linearity of expectation over at most iT possible offline queries, we have that

$$\mathsf{E}[Z] \leqslant 3i^2T^2/2^n.$$

We next give a general tail bound on Z for any upper bound $\mu_{\max}$ on $\mathsf{E}[Z]$. Let $\delta := 7n - 1$, which is at least 1 as $n \geqslant 1$. So, we can apply the Chernoff bound of Proposition 10.1.1 to get the following tail bound on Z.

$$\begin{aligned}\mathsf{Pr}[Z \geqslant 7n \cdot \mu_{\max}] &\leqslant \mathsf{Pr}[Z \geqslant (1+\delta) \cdot \mu_{\max}] \\ &\leqslant \exp(-\mu_{\max} \cdot \delta/3) \\ &\leqslant 2^{-\mu_{\max} \cdot (7n/3 - 1/3) \cdot \log e} \\ &\leqslant 2^{-2n \cdot \mu_{\max}}.\end{aligned}$$

We consider two cases: (a) $i \geqslant 3i^2T^2/2^n$, or (b) $i < 3i^2T^2/2^n$. In case (a), this implies that $\mathsf{E}[Z] \leqslant i = \mu_{\max}$, so by the above tail bound,

$$\mathsf{Pr}\left[Z \geqslant 7n \cdot i \mid i \geqslant 3i^2T^2/2^n\right] \leqslant 2^{-2n \cdot i}.$$

In case (b), we use $\mu_{\max} = 3i^2T^2/2^n$ and the fact that $\mu_{\max} \leqslant i$ in the tail bound above. Furthermore, $i < \mu_{\max}$, so it follows that

$$\mathsf{Pr}\left[Z \geqslant 7n \cdot (3i^2T^2/2^n) \mid i < 3i^2T^2/2^n\right] \leqslant 2^{-2n \cdot i}.$$

Combining the two cases, it holds that

$$\mathsf{Pr}\left[Z \geqslant 7n \cdot \max(i, 3i^2T^2/2^n)\right] \leqslant 2^{-2n \cdot i},$$

as required.

■

Claim 10.4.5. $\mathsf{Pr}\left[W_i | W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}\right] \leqslant \delta_S - 1/2^n$.

Proof. Suppose that the adversary $\mathcal{A}$ wins in game i on key k_i. We can assume without loss of generality that the adversary makes all the queries that are required to compute a collision. We define the queries $Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3$ corresponding to a collision on k_i as follows:

1. $Q_1 = (1, (\mathsf{k}_i \| x_1), y_1)$, where $y_1 = h_1(\mathsf{k}_i \| x_1)$ was queried.

2. $Q_2 = (2, (\mathsf{k}_i \| x_2), y_2)$, where $y_2 = h_1(\mathsf{k}_i \| x_2)$ was queried.

3. $Q_3 = (3, (y_1 \| y_2), z)$, where $z = h_3(y_1 \| y_2)$ was queried.

4. $Q'_1 = (1, (\mathsf{k}_i \| x'_1), y'_1)$ where $y'_1 = h_1(\mathsf{k}_i \| x'_1)$ was queried.

5. $Q'_2 = (2, (\mathsf{k}_i \| x'_2), y'_2)$, where $y'_2 = h_1(\mathsf{k}_i, x'_2)$ was queried.

6. $Q'_3 = (3, (y'_1 \| y'_2), z)$, where $z = h_3(y'_1 \| y'_2)$ was queried.

If there are multiple such candidates of queries $(Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3)$, we fix the candidate that is the lexicographically smallest.

We categorize the possibilities for the queries of k_i into one of the following cases and sub-cases for the following analysis. Recall that an offline query is one made before game i started, and an online query is one made during game i.

1. The queries Q_3, Q'_3 are identical queries. Assume without loss of generality that the queries Q_1, Q'_1 are different (the other symmetric possibility is that the queries Q_2, Q'_2 are different, since otherwise if Q_1, Q'_1 are same, Q_2, Q'_2 are same, Q_3, Q'_3 are same then by definition they cannot lead to a collision). We further have the following sub-cases.

   (a) Q_1, Q'_1 are both online queries.

   (b) One of Q_1, Q'_1 is offline, and the other is online.

   (c) Q_1, Q'_1 are both offline.

2. Q_3, Q'_3 are different queries. We have the following sub-cases.

   (a) Q_3, Q'_3 are both online queries.

   (b) Q_3 is offline, but Q_1, Q_2 are online.

   (c) Q_3 is offline, and one of Q_1, Q_2 is online, and the other is offline.

   (d) Q_1, Q_2, Q_3 are offline but Q'_3 is online.

   (e) $Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3$ are all offline.

First, we argue that these cases are exhaustive. If the queries Q_3, Q'_3 are identical, we can assume without loss of generality that Q_1, Q'_1 are different queries that have the same answer. So the three possibilities are both are online (sub-case 1a), exactly one of them is online (sub-case 1b), and both are offline (sub-case 1c). When Q_3, Q'_3 are different, the possibility that they are both online is captured by sub-case 2a. If one of them

Case	Probability
1a	$\frac{T^2}{2^n}$
1b	$\frac{iT^2}{2^{n+\kappa}}$
1c	$7n \cdot \frac{i}{2^\kappa} + 21n \cdot \frac{i^2T^2}{2^{n+\kappa}}$
2a	$\frac{T^2}{2^n}$
2b	$\frac{iT^3}{2^{2n}}$
2c	$7n \cdot \frac{i^2T^2}{2^{n+\kappa}} + 21n \cdot \frac{i^3T^4}{2^{2n+\kappa}}$
2d	$7n \cdot \frac{i^2T^2}{2^{n+\kappa}} + 21n \cdot \frac{i^3T^4}{2^{2n+\kappa}}$
2e	$7n \cdot \frac{i}{2^\kappa} + 21n \cdot \frac{i^2T^2}{2^{n+\kappa}}$

Table 10.1: Upper bounds on probability of the cases

is offline–say Q_3 without loss of generality– then either one of Q_1, Q_2 is online (captured by sub-case 2b), or one of Q_1, Q_2 is offline and the other online (captured by sub-case 2c), or they are both offline (this is captured by sub-case 2d since we know Q'_3 is online). Finally when Q_3, Q'_3 are both offline – if any of Q_1, Q_2, Q'_1, Q'_2 is online, this is captured by one of the sub-cases 2b and 2c. The final possibility is $Q_1, Q_2, Q_3, Q'_1, Q'_2, Q'_3$ are all old which is captured by sub-case 2e.

We will upper bound $\mathsf{Pr}[W_i | W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}]$ by upper bounding the probability of each of these cases happening. We will prove the upper bound on the probability of the cases as shown in Table 10.1. As $\mathsf{Pr}\left[W_i | W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}\right]$ is at most the maximum of the probabilities in Table 10.1, it follows that

$$\mathsf{Pr}\left[W_i | W_{<i} \wedge \neg \mathsf{E}^i_{\mathsf{hit}}\right] \leqslant \frac{T^2}{2^n} + 7n \cdot \frac{i}{2^\kappa} + 21n \cdot \frac{i^2T^2}{2^{n+\kappa}} + \frac{iT^3}{2^{2n}} + 21n \cdot \frac{i^3T^4}{2^{2n+\kappa}}.$$

We proceed to prove the upper bound on the probability of each case happening one by one.

Case 1a: ($Q_3 = Q'_3$, and Q_1, Q'_1 are both online queries) We can simply upper bound the probability of this case happening by the probability of two h_1 online queries producing a collision, which is at most $T^2/2^n$ by a birthday bound.

Case 1b: ($Q_3 = Q'_3$, and exactly one of Q_1, Q'_1 are online) Assume without loss of generality that Q_1 is the offline query and Q'_1 is online. Q_1 must correspond to some offline h_1 query that contains k_i as a prefix. If there are k such offline queries, then one of the online queries has to hit the same output as one of those k

queries to be the Q'_1 query. This implies a bound of $k \cdot T/2^n$ probability. But, the expected number of offline h_1 queries that had k_i as a prefix is at most $(i-1) \cdot T/2^\kappa$. Thus, this case happens with probability at most

$$\begin{aligned}
&\sum_{k=0}^{(i-1)\cdot T} \mathsf{Pr}[\text{there are } k \text{ offline } h_1 \text{ queries prefixed by } \mathsf{k}_i] \cdot k \cdot T/2^n \\
&= \mathsf{E}\left[\text{number of offline } h_1 \text{ queries prefixed by } \mathsf{k}_i\right] \cdot T/2^n \\
&\leqslant iT^2/2^{n+\kappa}.
\end{aligned}$$

Case 1c: ($Q_3 = Q'_3$, and both of Q_1, Q'_1 are offline) If this case happens, there are two different offline h_1 queries on $(\mathsf{k}\|*)$ such that they both produce the same answer, i.e., in particular there is hitting query of the form $h_1(\mathsf{k}\|*)$. If this happens for k different ks, there are at least k hitting queries. Since we know that there are at most $7n \cdot \max(i, 3i^2T^2/2^n)$ hitting queries, and k is sampled uniformly at random this case happens with probability at most $7n \cdot i/2^\kappa + 7n \cdot (3i^2T^2/2^n)/2^\kappa = 7n \cdot i/2^\kappa + 21n \cdot i^2T^2/2^{\kappa+n}$.

Case 2a: ($Q_3 \neq Q'_3$, and both of Q_3, Q'_3 are online) We can simply upper bound the probability of this case happening by the probability of two h_3 online queries producing a collision, which is at most $T^2/2^n$ by a birthday bound.

Case 2b: ($Q_3 \neq Q'_3$, Q_3 is offline, but Q_1, Q_2 are online) For this case to occur, the adversary needs to make distinct online queries Q_1, Q_2 whose outputs hit the input of some offline query Q_3. Since there are $(i-1) \cdot T$ offline queries, and T online queries, this happens with probability at most $(i-1) \cdot T \cdot T^2/2^{2n} \leqslant iT^3/2^{2n}$.

Case 2c: ($Q_3 \neq Q'_3$, Q_3 is offline, and exactly one of Q_1, Q_2 is online) Assume without loss of generality that Q_2, Q_3 are offline and Q_1 is online. We say that an offline h_3 query on $(x\|y)$ is "associated" with k_i if either there exists an offline query $h_1(\mathsf{k}_i\|*) = x$ or there exists an offline query $h_2(\mathsf{k}_i\|*) = y$. Given that the event $\mathsf{E}^i_{\mathsf{hit}}$ does not hold, there cannot exist a set of more than $k = 7n \cdot \max(i, 3i^2T^2n/2^n)$ distinct h_1 or h_2 queries that all have the same output, since otherwise there will be k hitting queries. Hence, an h_3 query can be associated with at most $7n \cdot \max(i, 3i^2T^2n/2^n)$ distinct k values. Given that there are at most $(i-1) \cdot T$ offline h_3 queries, it follows that there are at most $7n \cdot iT \cdot \max(i, 3i^2T^2n/2^n)$ different pairs of associated h_3 queries and k values. Therefore, for any randomly sampled k, the expected number of h_3 queries it is associated with it is at most $7n \cdot iT \cdot \max(i, 3i^2T^2n/2^n)/2^\kappa$.

For the specific k_i value sampled, if there are k h_3 queries associated with k_i, then there are at most k possible values of the output of some potential online Q_1 query. Hence, the probability of this case happening

is at most

$$\begin{aligned}
&\sum_{k=0}^{(i-1)T} \mathsf{Pr}\left[\text{there are } k \ h_3 \text{ queries associated with } \mathsf{k}_i\right] \cdot k \cdot (T/2^n) \\
&= T/2^n \cdot \mathsf{E}\left[\text{number of } h_3 \text{ queries associated with } \mathsf{k}_i\right] \\
&\leqslant 7n \cdot i^2T^2/2^{n+\kappa} + 21n \cdot i^3T^4/2^{2n+\kappa}
\end{aligned}$$

Case 2d: ($Q_3 \neq Q_3'$, Q_1, Q_2, Q_3 are offline but Q_3' is online) Suppose for the value of k_i sampled there are k h_3 queries associated with k_i as above. Then, there are at most k possible values that some potential online Q_3' query has to hit to cause this case to happen. Thus, the analysis follows identically to the previous case, giving a bound of $7n \cdot i^2T^2/2^{n+\kappa} + 21n \cdot i^3T^4/2^{2n+\kappa}$.

Case 2e: ($Q_3 \neq Q_3'$ and $Q_1, Q_2, Q_3, Q_1', Q_2', Q_3'$ are all offline) We refer to a collision structure for k as a set of offline queries of the form $Q_1, Q_2, Q_3, Q_1', Q_2', Q_3'$ corresponding to k. We claim that if there are k values of k with a corresponding collision structures queried, then there must be at least k different hitting queries.

First it is easy to see that for every k that gets classified into this case, at least one out of Q_3, Q_3' is a hitting query because they have the same answer. It remains to show that a set of k collision structures implies k distinct hitting queries.

To do so, we build a set of hitting queries as follows assuming there are k values of k with queried collision structures. For every such k value in lexicographic order, do the following.

- If one of Q_1, Q_2, Q_1', Q_2' is a hitting query, add it to this set,

- Otherwise, Q_3 or Q_3' must be a hitting query, so add it to the set.

We now claim this set has size at least k. Note that if a Q_1, Q_2, Q_1', Q_2' is added to the set, it is unique since it is an h_1 or h_2 query that contains k, so it increases the size of the set by one. Otherwise, a Q_3 or Q_3' query is added. Assume without loss of generality this Q_3 is added. Either this increases the size of the set by one, or Q_3 was already in the set. In the latter case, there must exist some previous key $\tilde{\mathsf{k}}$ for which some h_3 query corresponding to Q_3 was added. We use $\tilde{Q}_1, \tilde{Q}_2, \tilde{Q}_3, \tilde{Q}_1', \tilde{Q}_2', \tilde{Q}_3'$ to denote the corresponding queries in the collision structure for $\tilde{\mathsf{k}}$. By assumption $Q_3 = \tilde{Q}_3$ and neither of the two collision structures have a h_1 or h_2-type hitting query. But this is impossible since there must exist a pair of h_1 or h_2 queries, containing k and $\tilde{\mathsf{k}}$, with output contained in the input of $Q_3 = \tilde{Q}_3$, so there must be a h_1 or h_2-type hitting query for either k or $\tilde{\mathsf{k}}$. Therefore, for k collision structures, there will be at least k distinct hitting queries.

Given the above claim regarding collision structures, and assuming $\mathsf{E}_{\mathsf{hit}}^i$ doesn't hold, this implies there are at most $7n \cdot \max(i, 3i^2T^2/2^n)$ distinct k values with collision structures queried. But this implies that a

randomly sampled k_i gets classified into this case is at most

$$7n \cdot \max(i, 3i^2T^2/2^n)/2^\kappa = 7n \cdot i/2^\kappa + 21n \cdot i^2T^2/2^{n+\kappa}.$$

■

This completes the proof of Lemma 10.4.3. ■

10.4.3 Variable-Input Length Hash from Two-Level Merkle Trees

We combine the construction of Section 10.4.2 with the framework of Section 10.2 to get a variable-input length hash function. This construction is optimally secure as long as $ST^2 \leqslant 2^n$ and requires an underlying function $h\colon \{0,1\}^a \to \{0,1\}^n$ where $a \geqslant \max(\kappa + \lceil n/2 \rceil + 3, 2n + 2)$. This results in the following theorem.

We note if we modify values of κ, n by additive constant factors, we can get the same result as below with only $O(1)$ multiplicative loss in security. In this sense, we can achieve the theorem below assuming a function $h\colon \{0,1\}^{2n} \to \{0,1\}^n$, i.e. $a = 2n$ so only compressing by a factor exactly two.

Theorem 10.4.6. *Let $a, \kappa, n \in \mathbb{N}$ be such that $a \geqslant \max(\kappa + \lceil n/2 \rceil + 3, 2n + 2)$. Let $h\colon \{0,1\}^a \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, there is an $H^h\colon \{0,1\}^\kappa \times \{0,1\}^{<2^n} \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S,T) \leqslant \left(\frac{S}{2^\kappa} \cdot \left(14n + 42n\gamma + 42n\gamma^2\right) + \frac{T^2}{2^n} \cdot \left(2 + \frac{2\gamma}{T} + \frac{2}{T^2}\right)\right),$$

   where $\gamma = ST^2/2^n$.

2. *One evaluation of H^h on a message b bits long requires $3 \cdot \lceil (b + n + 1)/s \rceil$ queries to h where $s = 2a - 2\kappa - n - 4$.*

Proof. We define $H := \mathsf{KMD}^{\mathsf{2MT}^h_{n+s}}$ where $\mathsf{2MT}^h_{n+s}$ is defined in Figure 10.5. From Theorem 10.2.1, we have that $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S,T)$ is upper bounded by $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{2MT}^h_{n+s}}(S,T)$. Therefore, the bound on the advantage of any (S,T)-attacker on H^h follows from Theorem 10.4.2.

We have that $\mathsf{2MT}^h_{n+s}\{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$, for $s = 2a - 2\kappa - n - 4$. A b bit message, after padding, will result in $\lceil (b+n+1)/s \rceil$ message blocks that are fed into $\mathsf{2MT}^h_{n+s}$. For each call of $\mathsf{2MT}^h_{n+s}$, we need 3 calls to h, which implies the bound on the efficiency of H^h.

■

10.5 Merkle Tree Framework with a Keyed Inner Hash

In this section, we give another framework similar to the MD-based framework in Section 10.3. This framework is a keyed version of the Merkle tree construction where the key is used in every invocation of the inner hash function, not just at the leaves.

For a key length κ, output length n, and message block size s (where $s \geqslant 2n$), we assume an underlying primitive $h\colon \{0,1\}^\kappa \times \{0,1\}^{s+1} \to \{0,1\}^n$. Given h, we define the keyed Merkle tree hash function $\mathsf{KMT}^h\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$. On input key $\mathsf{k} \in \{0,1\}^\kappa$, and message msg. The function KMT^h first pads the message to split it into $\ell = \lceil (b+1)/s \rceil$ message blocks of size s using padding pad' which is defined as follows.

$\underline{\mathsf{pad}'(\mathsf{msg}, s)}$:

1. Let $k = s - ((|\mathsf{msg}| \bmod s) + 1)$.

2. Output $(m_1, \ldots, m_\ell) \in (\{0,1\}^s)^\ell$ where $m_1 \| \ldots \| m_\ell = \mathsf{msg} \| 1 \| 0^k$.

Similar to Theorem 10.1.3 we can prove the following theorem for pad'.

Theorem 10.5.1. *Let $s \in \mathbb{N}$. The function $\mathsf{pad}'(\mathsf{msg}, s)$ on messages $\mathsf{msg} \in \{0,1\}^*$ satisfies the following properties:*

1. $|\mathsf{pad}'(\mathsf{msg}, s)| \in (\{0,1\}^s)^\ell$ *for* $\ell = \lceil (|\mathsf{msg}| + 1)/s \rceil$.

2. *There is a unique decoding procedure that outputs* msg *given* $\mathsf{pad}'(\mathsf{msg}, s)$*, and outputs* $\perp$ *on invalid padded messages.*

Note that the second point implies that pad' is injective. We omit the proof of this theorem because it follows from the proof of Theorem 10.1.3.

After padding the message msg to get message blocks $(m_1, \ldots, m_\ell)$, the construction essentially computes a standard Merkle tree with message blocks at the leaves, using the hash function $h(\mathsf{k}, \cdot)$ as the compression function. For technical reasons, we additionally use a single bit appended to k to indicate whether a block in the Merkle tree corresponds to a leaf (specified by a 0) or an interior node (specified by a 1). This is done to avoid trivial collisions that can be obtained by just outputting a subtree of a given tree (say the root and its two children). To deal with trees with ℓ leaves where ℓ is not a power of 2, we simply "push" a value up the levels of the tree until it needs to be merged. This is formalized in Figure 10.5. We illustrate an example of how the Merkle tree is built for $\ell = 7$ in Fig. 10.6. Since the key k is included in every call to the underlying primitive h, it follows that any (S, T)-attacker that finds a collision in KMT^h with respect

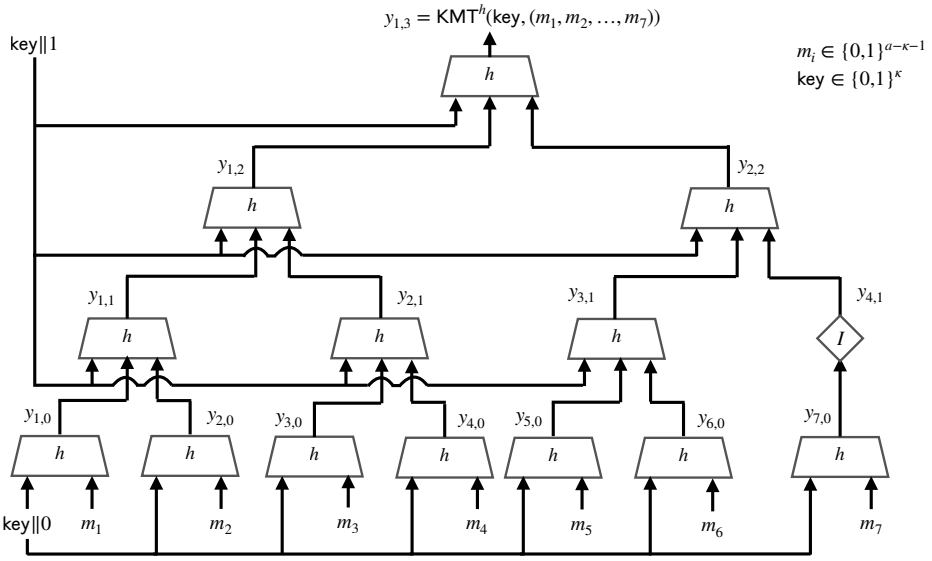


Figure 10.6: The construction KMT^h given a hash function $h\colon \{0,1\}^a \to \{0,1\}^n$ for $a \geqslant \max(\kappa + 2, 2n + 1)$. Let $s = a - \kappa - 1$. The inputs to h are padded with appropriate number of zeros if the size of the input is less than $\kappa + s + 1$ bits. I is the identity function that is used to "push" a value up the tree until it needs to be merged.

to a key k also finds a collision in h with respect to key. This is formalized via a reduction, which gives the following theorem.

Theorem 10.5.2. *Let $\kappa, n, s \in \mathbb{N}$ such that $s \geqslant 2n$. Let $h\colon \{0,1\}^\kappa \times \{0,1\}^{s+1} \to \{0,1\}^n$ be any function, and let $\mathsf{KMT}^h\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ as shown in Figure 10.7. Then, for every $S, T \in \mathbb{N}$, it holds that*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{KMT}^h}(S,T) \leqslant \mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{h}(S,T).$$

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be any (S,T)-attacker against the collision resistance of the construction KMT^h. Recall that we assume for simplicity that if $\mathcal{A}_2$ outputs a successful collision $\mathsf{msg}_{\mathcal{A}} \neq \mathsf{msg}'_{\mathcal{A}}$ such that $\mathsf{KMD}^h(\mathsf{k}, \mathsf{msg}_{\mathcal{A}}) = \mathsf{KMD}^h(\mathsf{k}, \mathsf{msg}'_{\mathcal{A}})$, then $\mathcal{A}_2$ queried all the necessary values to fully compute each of these functions. We construct an (S,T)-attacker $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against the collision resistance of h as follows.

In the preprocessing phase, $\mathcal{B}_1(h)$ simply computes $\sigma \leftarrow \mathcal{A}_1(h)$ and outputs σ where $|\sigma| \leqslant S$. In the online phase, $\mathcal{B}_2^h(\sigma, \mathsf{k})$ is given a random key $\mathsf{k} \leftarrow \{0,1\}^\kappa$ that it needs to find a collision for. $\mathcal{B}_2$ computes $(\mathsf{msg}_{\mathcal{A}}, \mathsf{msg}'_{\mathcal{A}}) \leftarrow \mathcal{A}_2^h(\sigma, \mathsf{k})$, where it uses its own query access to the function h to answer queries that $\mathcal{A}_2$ makes to h. $\mathcal{B}_2$ sets $(m_1, \ldots, m_\ell) \leftarrow \mathsf{pad}'(\mathsf{msg}_{\mathcal{A}}, s)$ and $(m'_1, \ldots, m'_{\ell'}) \leftarrow \mathsf{pad}'(\mathsf{msg}'_{\mathcal{A}}, s)$, where $\ell = \lceil (|\mathsf{msg}_{\mathcal{A}}| + 1)/s \rceil$ and $\ell' = \lceil (|\mathsf{msg}'_{\mathcal{A}}| + 1)/s \rceil$. Assume without loss of generality that $\ell' \geqslant \ell$.

Construction $\mathsf{KMT}^h(\mathsf{k}, \mathsf{msg})$:

1. $(m_1, m_2, \ldots, m_\ell) \leftarrow \mathsf{pad}'(\mathsf{msg}, s)$ where $\ell = \lceil (|\mathsf{msg}| + 1)/s \rceil$.
2. Initialize $y_{i,0} \leftarrow h(\mathsf{k} \| 0, m_i)$ for $i = 1, \ldots, \ell$.
3. $\ell' \leftarrow \ell$.
4. For $j = 1$ to $\lceil \log \ell \rceil$.
   (a) Compute $y_{i,j} = h(\mathsf{k} \| 1, y_{2i-1,j-1} \| y_{2i,j-1} \| 0^{s-2n})$ for $i = 1, \ldots, \lfloor \ell'/2 \rfloor$.
   (b) If $\ell' \bmod 2 = 1$ then $y_{(\ell'+1)/2,j} = y_{\ell',j-1}$, $\ell' \leftarrow (\ell' + 1)/2$.
   (c) Else $\ell' \leftarrow \ell'/2$.
5. Output $y_{1,\lceil \log \ell \rceil}$.

Figure 10.7: The Merkle tree construction with a keyed inner hash.

$\mathcal{B}$ looks at the queries made by $\mathcal{A}_2$ and recovers the corresponding Merkle trees with leaves $(m_1, \ldots, m_\ell)$ and $(m'_1, \ldots, m'_{\ell'})$ ($\mathcal{A}_2$ must have made the queries which are required to compute these trees since $\mathcal{A}_2$ succeeds in outputting a collision by assumption). It looks for any collisions in h among these queries in the Merkle tree. If it finds one, it outputs the collision, otherwise aborts and outputs $\perp$.

It remains to analyze the complexity and success probability of $\mathcal{B}$. First, as $\mathcal{B}$ simply runs $\mathcal{A}$ in the preprocessing and online phases and makes no additional queries to h, it follows that if $\mathcal{A}$ is an (S, T)-attacker, then so is $\mathcal{B}$.

To analyze the success probability of $\mathcal{B}$, we claim that whenever $\mathcal{A}$ succeeds, then $\mathcal{B}$ also succeeds. To help in our analysis, we define a recursive procedure CollisionSearch (in Figure 10.8) that takes as input two different nodes in a Merkle tree with the same value, treated as the roots, and outputs true if there is a collision and false otherwise.

First, we claim that CollisionSearch outputs true only if it finds a collision. This is because it is always called on two roots which have the same value and returns true only if the two children of the roots have different values (in which case it has found a collision because the roots have the same value), or if one root is a leaf and the other is not (in this case it has to be the collision because the leaves have input prefixed by $\mathsf{k} \| 0$ while other nodes have inputs prefixed by $\mathsf{k} \| 1$).

As a result, it remains to argue that whenever CollisionSearch returns false, $\mathcal{A}$ fails to find a collision. If $\ell = \ell'$, then the two Merkle trees will have the same structure, so CollisionSearch will return false only

Procedure $\mathsf{CollisionSearch}(\mathsf{root}_1, \mathsf{root}_2)$:

1. Let the number of leaves of the tree rooted at root_1 be ℓ, and the number of leaves of the tree rooted at root_2 be ℓ'.
2. If $\ell = \ell' = 1$, then return false.
3. If $\ell = 1, \ell' > 1$ or $\ell' = 1, \ell > 1$, then return true.
4. Find the left and the right child of root_1, (going down the I nodes if required), call these $\mathsf{root}_{1,L}$ and $\mathsf{root}_{1,R}$. Find the left and the right child of root_2, (going down the I nodes if required), call these $\mathsf{root}_{2,L}$ and $\mathsf{root}_{2,R}$.
5. If the value at $\mathsf{root}_{1,L}$ and $\mathsf{root}_{2,L}$ are different or if the value at $\mathsf{root}_{1,R}$ and $\mathsf{root}_{2,R}$ are different return true.
6. Make a recursive call to $\mathsf{CollisionSearch}(\mathsf{root}_{1,L}, \mathsf{root}_{2,L})$, and return true if it returns true.
7. Make a recursive call to $\mathsf{CollisionSearch}(\mathsf{root}_{1,R}, \mathsf{root}_{2,R})$, and return true if it returns true.
8. Otherwise return false.

Figure 10.8: Recursive collision search procedure used in the proof of Theorem 10.5.2

if all the leaves of the two trees are same. This means that $(m_1, \ldots, m_\ell) = (m'_1, \ldots, m'_\ell)$, so $\mathsf{msg} = \mathsf{msg}'$ by injectivity of pad'. Therefore, $\mathcal{A}$ did not find a valid collision, as required. If one of ℓ or ℓ' is 1 and $\mathsf{CollisionSearch}$ returns false, then it must be that $\ell = \ell' = 1$, meaning $\mathcal{A}$ has failed to find a collision as above. Finally, consider the case where $\ell' > \ell \geqslant 2$. Let ℓ_L, and ℓ_R be the number of leaves in the left and right subtrees of the tree with ℓ leaves, i.e., $\ell_L + \ell_R = \ell$. Similarly define ℓ'_L, ℓ'_R so that $\ell'_L + \ell'_R = \ell'$. Then it must be the case that either $\ell_L \neq \ell'_L$ or $\ell_R \neq \ell'_R$ at every step. It follows inductively that at every step of the recursion, either the left or the right subtree will have a different number of leaves, so eventually, $\mathsf{CollisionSearch}$ will arrive at a case when one subtree has one leaf and the other has more than one, returning true, in contradiction.

Thus, whenever $\mathcal{A}$ succeeds, $\mathcal{B}$ will also succeed, so it follows that

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{KMT}^h}(S,T) \leqslant \mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{h}(S,T),$$

as required. ■

We combine the construction of Section 10.4.2 with the framework presented above to get a variable-input length hash function. This construction is optimally secure as long as $ST^2 \leqslant 2^n$ and requires an underlying

function $h\colon \{0,1\}^a \to \{0,1\}^n$ where $a \geqslant \max(\kappa + \lceil n/2 \rceil + 3, 2n+2, \kappa + 3\lceil n/2 \rceil - 1)$. This results in the following theorem.

We note if we modify values of κ, n by additive constant factors, we can get the same result as below with only $O(1)$ multiplicative loss in security. In this sense, we can achieve the theorem below assuming a function $h\colon \{0,1\}^{2n} \to \{0,1\}^n$, i.e. $a = 2n$ so only compressing by a factor exactly two.

Theorem 10.5.3. *Let $a, \kappa, n \in \mathbb{N}$ be such that $a \geqslant \max(\kappa + 3\lceil n/2 \rceil + 2, 2n+2)$. Let $h\colon \{0,1\}^a \to \{0,1\}^n$ be modeled as a random oracle in the AI-ROM. Then, there is an $H^h\colon \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^n$ such that:*

1. *For any $S, T \in \mathbb{N}$,*

$$\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S,T) \leqslant \left(\frac{S}{2^\kappa}\left(14n + 42n\gamma + 42n\gamma^2\right) + \frac{T^2}{2^n} \cdot \left(2 + \frac{2\gamma}{T} + \frac{2}{T^2}\right)\right),$$

   where $\gamma = ST^2/2^n$.

2. *One evaluation of H^h on a message b bits long requires at most $6 \cdot \lceil (b+1)/s \rceil - 3$ queries to h where $s = 2a - 2\kappa - n - 4$.*

Proof. Let $s = 2a - 2\kappa - n - 4$. We define $H := \mathsf{KMT}^{\mathsf{2MT}^h_{n+s}}$ where $\mathsf{2MT}^h_{n+s}$ is defined in Figure 10.5. Since we have that $a \geqslant \kappa + 3\lceil n/2 \rceil + 2$, it implies $s = 2a - 2\kappa - n - 4 \geqslant 2n$. Therefore, from Theorem 10.5.3, we have that $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{H^h}(S,T)$ is upper bounded by $\mathsf{Adv}^{\mathsf{ai\text{-}cr}}_{\mathsf{2MT}^h_{n+s}}(S,T)$. Therefore, the bound on the advantage of any (S,T)-attacker on H^h follows from Theorem 10.4.2 since $a \geqslant \max(\kappa + 3\lceil n/2 \rceil + 2, 2n+2) \geqslant \max(\kappa + \lceil n/2 \rceil + 3, 2n+2)$.

We have that $\mathsf{2MT}^h_{n+s}\{0,1\}^\kappa \times \{0,1\}^{n+s} \to \{0,1\}^n$, for $s = 2a - 2\kappa - n - 4$. A b bit message, after padding, will result in $\lceil (b+1)/s \rceil$ message blocks that are fed into $\mathsf{2MT}^h_{n+s}$. To build a Merkle tree on $\lceil (b+1)/s \rceil$ blocks we need $2\lceil (b+1)/s \rceil$ calls to $\mathsf{2MT}^h_{n+s}$, and for each call of $\mathsf{2MT}^h_{n+s}$, we need 3 calls to h, which implies the bound on the efficiency of H^h.

■

Chapter 11

THE QUERY-COMPLEXITY OF PREPROCESSING ATTACKS

In this chapter, we focus on the question of how much *time* is needed to produce the advice for preprocessing attacks. This question is highly non-trivial and often overlooked. As in the case of space-time tradeoffs, we focus in particular on settings with ideal primitives, where both the offline and online time-complexities are approximated by the number of queries to the given primitive. We give generic results that highlight the benefits of salting to generically increase the offline costs of preprocessing attacks. The majority of our pauper presents several results focusing on *salted* hash functions. In particular, we provide a fairly involved analysis of the pre-image- and collision-resistance security of the (two-block) Merkle-Damgård construction in our model. This chapter is based on our work in [GT23].

In Section 11.1, we introduce the preliminaries for this chapter. In Section 11.2, we prove a result that generically justifies the practice of salting cryptographic primitives to defeat preprocessing attacks. Then in Section 11.3, we introduce new non-generic techniques specific results for offline-online attacks on salted random oracles. Finally, in Section 11.3, we analyze the security of two-block Merkle-Damgård against offline-online attacks.

11.1 Preliminaries

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ denote the set of all natural numbers and $\mathbb{N}_{>0} = \mathbb{N}\backslash\{0\}$. For $N \in \mathbb{N}_{>0}$, let $[N] = \{1, 2, \ldots, N\}$. For a set X, let $|X|$ be its size and X^+ denote one or more elements of X. For a set S and $r \in \mathbb{N}_{>0}$ such that $r \leqslant |\mathsf{S}|$, we denote using $\binom{\mathsf{S}}{r}$ the set of subsets of S with r elements. We denote $\mathsf{Fcs}(\mathsf{D}, \mathsf{R})$ the set of all functions mapping elements in D to the elements of R. Security notions are defined via games; for an example see Fig. 11.2. The probability that a game G outputs true is denoted using $\mathsf{Pr}[\mathsf{G}]$.

We let $x \leftarrow\!\!\$\, \mathcal{D}$ denote sampling x according to the distribution $\mathcal{D}$. If D is a set, we overload notation and let $x \leftarrow\!\!\$\, \mathsf{D}$ denote uniformly sampling from the elements of D. For a bit-string s we use $|s|$ to denote the number of bits in s. For a random variable X, we use $\mathsf{E}[X]$ to denote its expectation.

Merkle-Damgård. We recall the Merkle-Damgård hashing mechanism. For $n, \ell \in \mathbb{N}_{>0}$, let $\mathrm{h} : \{0,1\}^n \times (\{0,1\}^\ell)^+ \to \{0,1\}^n$ be a compression function. We recursively define Merkle-Damgård (MD) hashing $\mathsf{MD}^{\mathrm{h}} : \{0,1\}^n \times (\{0,1\}^\ell)^+ \to \{0,1\}^n$ as

$$\mathsf{MD}^{\mathrm{h}}(a, M) = \mathrm{h}(a, M)$$

for $a \in \{0,1\}^n, M \in \{0,1\}^\ell$ and

$$\mathsf{MD}^{\mathrm{h}}(a,(M_1,M_2,\ldots,M_B)) = \mathrm{h}(\mathsf{MD}^{\mathrm{h}}(a,(M_1,M_2,\ldots,M_{B-1})),M_B)$$

for $a \in \{0,1\}^n$ and $M_1,\ldots,M_B \in \{0,1\}^\ell$. We refer to a as the *salt*.

THE COMPRESSION LEMMA. The compression lemma states that it is impossible to compress a random element in set $\mathcal{X}$ to a string shorter than $\log|\mathcal{X}|$ bits long, even relative to a random string.

Proposition 11.1.1 (E.g., [DTT10])**.** *Let* Encode *be a randomized map from* $\mathcal{X}$ *to* $\mathcal{Y}$ *and let* Decode *be a randomized map from* $\mathcal{Y}$ *to* $\mathcal{X}$ *such that*

$$\mathsf{Pr}_{x \xleftarrow{\$} \mathcal{X}}\left[\mathsf{Decode}(\mathsf{Encode}(x)) = x\right] \geqslant \varepsilon.$$

Then, $\log|\mathcal{Y}| \geqslant \log|\mathcal{X}| - \log(1/\varepsilon)$.

MARKOV'S INEQUALITY. We use Markov's inequality multiple times in this chapter. We state it here for the sake of completeness.

Proposition 11.1.2. *Let* X *be a non-negative random variable and* $a > 0$*. Then*

$$\mathsf{Pr}\left[X \geqslant a\right] \leqslant \frac{\mathsf{E}\left[X\right]}{a}\ .$$

11.2 Offline-Online Tradeoffs and the Role of Salting

We present some basic facts about offline-online tradeoffs and discuss the role of salting. To do this, we define a notational framework that captures the generality of our statements.

11.2.1 A general framework for offline-online attacks

GAMES. We formalize security guarantees in cryptography using *games*, which we also use for security proofs. A game G describes an environment an adversary $\mathcal{A}$ can interact with, and the combination of G and $\mathcal{A}$ results in a random experiment $\mathsf{G}(\mathcal{A})$ (we refer to this as $\mathcal{A}$ "playing" the game G) which produces a Boolean output. We also denote this output as $\mathsf{G}(\mathcal{A})$.

GAMES WITH IDEAL PRIMITIVES. We are interested in a special class of games that depend on an *ideal primitive*, such as a random oracle, random permutation, ideal cipher, etc. We model this via a distribution $\mathcal{I}$ on a set of functions. For example, a *random oracle* with (finite) domain D and range R would be modeled by the uniform distribution on $\mathsf{Fcs}(\mathsf{D},\mathsf{R})$.[1] Similarly, an ideal cipher with key space K and domain X can

[1] As usual, one must be more precise when formally defining a random oracle with $\mathsf{D} = \{0,1\}^*$, but we remain intentionally informal on this front; all of our examples can be assumed to work on a finite domain.

be modeled as a uniformly chosen function e from the set $\mathsf{Fcs}(\mathsf{K} \times \mathsf{X} \times \{-1,1\}, \mathsf{X})$ such that $e(k,\cdot,1)$ is a permutation on X for all $k \in \mathsf{K}$, and $e(k,\cdot,-1)$ is its inverse. We can also model a variant of the generic-group model (GGM) [Sho97] by looking at the uniform distribution of functions $f \in \mathsf{Fcs}(\mathbb{Z}_p \times \mathsf{X} \times \mathsf{X}, \mathsf{X} \times \mathsf{X})$, where $|\mathsf{X}| = p$, and

$$\pi(x, l_1, l_2) = (\phi(x), \phi(\phi^{-1}(l_1) + \phi^{-1}(l_2))) ,$$

where $\phi : \mathbb{Z}_p \to \mathsf{X}$ is a bijective function.

Games with primitives. An *oracle* game G^{π} is one where both the adversary $\mathcal{A}$ and the game procedures are given access to an oracle π, from an understood set of possible functions π, which we refer to as *compatible* with the game G. We denote by $\mathsf{G}^{\pi}(\mathcal{A}^{\pi})$ both the experiment where $\mathcal{A}$ plays the game, and is given access to the same π as the game as well as the random variable denoting the output. We say that an (oracle) game G is *compatible* with an ideal primitive $\mathcal{I}$, if the range of $\mathcal{I}$ is a subset of the compatible oracles for G. We write specifically

$$\mathsf{Adv}^{\mathsf{G}}_{\mathcal{I}}(\mathcal{A}) = \mathsf{Pr}\left[\mathsf{G}^{\pi}(\mathcal{A}^{\pi})\right] \tag{11.1}$$

where $\pi \xleftarrow{\$} \mathcal{I}$. One could define a more general notion that permits other advantage formats (e.g., to model distinguishing notions). This is straightforward, and outside the scope of this chapter.

Defining offline-online attacks. With the above formalism, given an oracle game G, we introduce a new oracle game $\mathsf{pre\text{-}G}$, which enhances the original game to model offline-online attacks. Both games preserve compatibility with any oracle. In particular, an adversary $\mathcal{A}$ is split into two parts, the offline adversary $\mathcal{A}_1$ and the online adversary $\mathcal{A}_2$. Initially, in the *offline stage*, $\mathcal{A}_1$ is given access *solely* to the game oracle π. At the end of this stage, $\mathcal{A}_1$ outputs a state st. Then, in the *online stage*, adversary $\mathcal{A}_2$ is initialized with state st and run in the game G. Both $\mathcal{A}_2$ and G are given access to the oracle π. Crucially, the game G might give $\mathcal{A}_2$ additional oracles plus additional initialization values, etc., which are not available in the offline stage. (This is formalized in Figure 11.1 on the left). We colloquially refer to $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as an offline-online adversary. Further, we say that $\mathcal{A}$ is a (T_1, T_2)-adversary if $\mathcal{A}_1$ makes at most T_1 queries and $\mathcal{A}_2$ makes at most T_2 queries. (Note that $\mathcal{A}_2$ could make additional game-dependent queries, which we would specify separately if necessary.) We overload notation and define advantage in terms of (T_1, T_2) as follows.

$$\mathsf{Adv}^{\mathsf{G}}_{\mathcal{I}}(T_1, T_2) = \max_{(T_1,T_2)\text{-adversaries } \mathcal{A}} \mathsf{Adv}^{\mathsf{G}}_{\mathcal{I}}(\mathcal{A}) . \tag{11.2}$$

Some basic facts. The following elementary fact, while straightforward, establishes some important baselines for when offline-online attacks are interesting. It relies on the basic observation that one can consider $\mathcal{A}_1$ and $\mathcal{A}_2$ to be a single online adversary.

Game $\mathsf{pre\text{-}G}^{\pi}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2))$	Game $s\text{-}\mathsf{pre\text{-}G}^{\pi}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2))$
$\mathsf{st} \leftarrow \mathcal{A}_1^{\pi}$	$\mathsf{st} \leftarrow \mathcal{A}_1^{\pi}$
Return $\mathsf{G}^{\pi}(\mathcal{A}_2^{\pi}(\mathsf{st}))$	$a \leftarrow\!\!\$\ \{0,1\}^s$
	Return $\mathsf{G}^{\pi_a}(\mathcal{A}_2^{\pi}(\mathsf{st}, a))$

Figure 11.1: Offline-Online security games for an original game G. Left: the unsalted case. Right: the salted case. Here, π is meant to be sampled from a salted ideal primitive $\mathcal{I}_s$, and $\pi_a(\cdot) = \pi(a, \cdot)$, i.e., the primitive with salt fixed to a.

Lemma 11.2.1. *Let G be a game compatible with the ideal primitive $\mathcal{I}$. For any (T_1, T_2)-adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}_{\mathcal{I}}^{\mathsf{pre\text{-}G}}(\mathcal{A}) = \mathsf{Adv}_{\mathcal{I}}^{\mathsf{G}}(\mathcal{B}) \ .$$

Here, $\mathcal{B}$ makes $T_1 + T_2$ queries to the primitive, and its time-complexity is the sum of the time-complexities of $\mathcal{A}_1$ and $\mathcal{A}_2$. Further, for any game-dependent type of query, $\mathcal{B}$ makes the same number of queries as $\mathcal{A}_2$.

For an ideal primitive $\mathcal{I}$, we let $Q(\mathcal{I})$ be an upper bound on the number of queries needed by an adversary, given oracle access to $\pi \leftarrow\!\!\$\ \mathcal{I}$, to reconstruct π with probability 1. Then, the following also holds true and formalizes the fact that one can simulate an offline-online adversary by having the offline adversary first reconstruct π and then store it in st.

Lemma 11.2.2. *Let $\mathcal{I}$ be a primitive that is compatible with game G. For all offline-online adversaries $\mathcal{A}$, there exists a $(Q(\mathcal{I}), 0)$-adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}_{\mathcal{I}}^{\mathsf{pre\text{-}G}}(\mathcal{A}) = \mathsf{Adv}_{\mathcal{I}}^{\mathsf{pre\text{-}G}}(\mathcal{B})$$

Note that the adversary $\mathcal{B}$ could be much less efficient than $\mathcal{A}$; however, in many cases, we will study games that only target information-theoretic security, and time-complexity will not matter. It also naïvely follows that there always exists an *optimal* adversary that is a $(Q(\mathcal{I}), 0)$-adversary.

WHICH GAMES ARE INTERESTING? Some games are more interesting than others in the context of offline-online tradeoffs, and the above two lemmas already provide some guidance.

Consider the problem of inverting a random permutation $\pi : \{0,1\}^n \to \{0,1\}^n$. It is well known that the best adversary takes time $\Omega(N)$ queries, where $N = 2^n$, to invert with constant probability. Then, Lemma 11.2.1 implies that any (T_1, T_2)-offline adversary needs $T_1 + T_2 = \Omega(N)$ to invert with constant probability. Thus, to get $T_2 = o(N)$ and beat the naïve inversion attack in the online phase, we need

$T_1 = \Omega(N)$. Further, we already have an $(N, 0)$-adversary by Lemma 11.2.2, so we cannot really expect interesting tradeoffs. The question becomes interesting only if we limit the state size between $\mathcal{A}_1$ and $\mathcal{A}_2$, which is exactly what is considered by prior works on space-time tradeoffs.

This is in contrast to the setting of the discrete logarithm problem with preprocessing [CK18]. There, for a group of order p, by combining Lemma 11.2.1 with the well-known result by Shoup [Sho97], we get that $T_1 + T_2 = \Omega(\sqrt{p})$. Lemma 11.2.2 guarantees only a $(p, 0)$ attacker, so we can expect that $T_2 = o(\sqrt{p})$ while still having $T_1 = o(p)$. And indeed, one can achieve the tradeoff $T_1 \cdot T_2 \geqslant p$, as indicated in [CK18].

11.2.2 The power of salting

A special case of interest is that of *salting*, where the cryptographic primitive $\mathcal{I}$ permits an additional input–called a *salt*–that is chosen in the online phase of an attack.

GENERIC SALTING. Let $\mathcal{I}$ be an ideal primitive, whose range is a subset of $\mathsf{Fcs}(\mathsf{D}, \mathsf{R})$. We define its s-bit *salted* version, denoted $\mathcal{I}_s$, as the ideal primitive with range $\mathsf{Fcs}(\{0,1\}^s \times \mathsf{D}, \mathsf{R})$; sampling a function $\pi : \{0,1\}^s \times \mathsf{D} \to \mathsf{R}$ occurs by first sampling 2^s independent copies $\pi_a \xleftarrow{\$} \mathcal{I}$ for each $a \in \{0,1\}^s$ and then letting

$$\pi(a, x) = \pi_a(x) \ ,$$

for all $a \in \{0,1\}^s$ and $x \in \mathsf{D}$.

For any game G compatible with an ideal primitive $\mathcal{I}$, we can now define an s-bit salted version of the game, s-pre-G, that is compatible with $\mathcal{I}_s$. This is given on the right of Figure 11.1. Essentially, we now sample a primitive $\pi \xleftarrow{\$} \mathcal{I}_s$ to which the adversary $\mathcal{A}$ is given access. However, in the online phase, the games themselves have access only to π_a for a randomly sampled salt a, which is revealed to only the adversary $\mathcal{A}_2$.

FROM SALTED TO UNSALTED GAMES. The following theorem relates the advantage of an offline-online adversary for the salted game with that of an adversary for the original game. We provide an interpretation below (and use this lemma quantitatively in Section 11.3).

Theorem 11.2.3. *Let G be a game compatible with an ideal primitive $\mathcal{I}$. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a (T_1, T_2) offline-online adversary. Then, there exists an adversary $\mathcal{B}$ playing G such that*

$$\mathsf{Adv}^{s\text{-pre-}\mathsf{G}}_{\mathcal{I}_s}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{G}}_{\mathcal{I}}(\mathcal{B}) \ .$$

The adversary $\mathcal{B}$ makes a number of queries, expressed as a random variable T with expectation $\mathsf{E}[T] \leqslant T_1/2^s + T_2$.

Proof. Given access to π in the range of $\mathcal{I}$, the adversary $\mathcal{B}$ samples a random salt a from $\{0,1\}^s$ and then samples $\pi_{a'} \xleftarrow{\$} \mathcal{I}$ for $a' \neq a$. It then simulates an execution of $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in s-pre-G as follows: it answers

the ideal-primitive query of $\mathcal{A}$ for salt a using oracle queries to π, and those for salt $a' \neq a$ using the local evaluation of $\pi_{a'}$. It is immediate that $\mathcal{B}$ perfectly simulates the execution of s-pre-G to $\mathcal{A}$. Therefore, $\mathcal{A}$ wins if and only if $\mathcal{B}$ does and the claim about advantages follows.

Observe that $\mathcal{B}$ queries its ideal primitive, sampled from $\mathcal{I}$ only when it receives an ideal object query from $\mathcal{A}$ that is prefixed by the actual salt a. Since it is not given access to a when simulating $\mathcal{A}_1$, its queries are independent of a, and each of them is indeed on salt a with probability $1/2^s$. In contrast, queries $\mathcal{A}_2$ makes queries after learning a, and therefore, those queries are on salt a with probability upper bounded by one. By linearity of expectation, the total number of queries T to $\pi = \pi_a$ made by $\mathcal{B}$ satisfies $\mathsf{E}[T] \leqslant T_1/|\mathsf{S}| + T_2$, as we intended to show. □ ■

Salting generically defeats preprocessing attacks. We illustrate one first main application of Theorem 11.2.3, i.e., the fact that salting generically defeats preprocessing in a *qualitative* sense. Here, "qualitative" means that we only look at the power of attacks that achieve large advantage. Subsequent sections (Sections 11.3 and 11.4) take a more quantitative angle on this, studying the whole advantage curve.

We now say that a game G compatible with $\mathcal{I}$ is (T^*, ε)-hard if $\mathsf{Adv}^{\mathsf{G}}_{\mathcal{I}}(\mathcal{A}) \leqslant \varepsilon$ for all T^*-query $\mathcal{A}^*$. We say that game G is (T^*, ε)-expected-hard if the same holds for all adversaries running in *expected* time at most T^*. The following fact is helpful.

Lemma 11.2.4. *If* G *is* $(T^*, 0.4)$*-hard, then it is* $(T^*/2, 0.9)$*-expected-hard.*

Proof. By contradiction, let $\mathcal{A}$ run in expected time at most $T^*/2$, and $\mathsf{Adv}^{\mathsf{G}}_{\mathcal{I}}(\mathcal{A}) > 0.9$. Then, build $\mathcal{B}$ that runs $\mathcal{A}$ for T^* queries and then aborts with some default answer if $\mathcal{A}$ did not finish running. Let T be the running time of $\mathcal{A}$. Then, for $\pi \leftarrow\!\!\$\, \mathcal{I}$,

$$\begin{aligned}\mathsf{Pr}[\mathsf{G}^{\pi}(\mathcal{B}^{\pi})] &= \mathsf{Pr}[\mathsf{G}^{\pi}(\mathcal{A}^{\pi}) \ \wedge \ T \leqslant T^*] \\ &\geqslant \mathsf{Pr}[\mathsf{G}^{\pi}(\mathcal{A}^{\pi})] - \mathsf{Pr}[T > T^*] > 0.9 - 0.5 = 0.4\,,\end{aligned}$$

where we used Markov's inequality and the fact that $\mathsf{E}[T] \leqslant T^*/2$. □ ■

Now, say that s-pre-G is (T_1, T_2, ε)-hard if for all (T_1, T_2)-adversaries $\mathcal{A}$, we have that $\mathsf{Adv}^{s\text{-pre-}\mathsf{G}}_{\mathcal{I}}(\mathcal{A}) \leqslant \varepsilon$. Then, Lemma 11.2.4 and Theorem 11.2.3 yield the following corollary.

Corollary 11.2.5. *If* G *is* $(T^*, 0.4)$*-hard, then* s-pre-G *is* $(T_1, T_2, 0.9)$*-hard for any* T_1, T_2 *such that* $T_1/2^s + T_2 \leqslant T^*/2$*.*

This means that if a (T_1, T_2)-adversary is to achieve advantage larger than 0.9 in s-pre-G, then $T_1 \geqslant 2^s T^*/4$ or $T_2 \geqslant T^*/4$. In other words, in order to win s-pre-G with an advantage larger than 0.9, an attacker needs to either use online time which is (almost) as large as that of the best online attack achieving advantage 0.4 or run 2^s times that amount of time in the offline stage.

MOVING ON. We can easily revisit the remainder of this chapter using what we saw in this section. First of all, our conclusion about salting applies only to large advantage adversaries since otherwise we cannot prove an analogue of Lemma 11.2.4. Section 11.3 examines *tight* exact bounds on the advantage of (T_1, T_2)-adversaries that hold for each choice of T_1 and T_2. Second, this conclusion applies only to the case where G salts *every call* to the primitive. In Section 11.4, we characterize the pre-image- and the collision-resistance of the salted MD construction against offline-online attacks and show that salting, while still useful, has a more limited effect.

11.3 Offline-Online Security of Salted Random oracles

In this section, we study the security of salted monolithic random oracles against offline-online adversaries. Specifically, we consider the security properties of pre-image-resistance and collision-resistance. Our analysis begins by applying Theorem 11.2.3 in conjunction with [JT20, Theorem 1] to derive advantage upper bounds for offline-online adversaries against these properties. This approach already yields a tight bound for pre-image-resistance, but not for collision-resistance. We then use a non-generic technique to prove a tight bound for the latter.

11.3.1 Pre-image-resistance of a salted random oracle

Oracle game PR^{h} in Fig. 11.2 formalizes the preimage-resistance of oracle h, which has co-domain $\{0,1\}^n$. In the game the adversary is given as input y, which is randomly sampled from $\{0,1\}^n$. It has oracle access to h and wins if it manages to output x such that $\mathrm{h}(x) = y$.

We aim to upper bound the advantage of offline-online adversaries $\mathcal{A}$ against pre-image-resistance of salted random oracles. Let $H_{s,\ell,n}$ be the uniform distribution over $\mathsf{Fcs}(\{0,1\}^s \times \{0,1\}^\ell, \{0,1\}^n)$. The quantity of interest is $\mathsf{Adv}^{\mathsf{s\text{-}pre\text{-}PR}}_{H_{s,\ell,n}}(T_1, T_2)$. Using Theorem 11.2.3 and [JT20, Theorem 1], we get the following corollary.

Corollary 11.3.1. *Let $T_1, T_2, n, s, \ell \in \mathbb{N}_{>0}$. Then*

$$\mathsf{Adv}^{\mathsf{s\text{-}pre\text{-}PR}}_{H_{s,\ell,n}}(T_1, T_2) \leqslant 5\left(\frac{T_1}{2^{s+n}} + \frac{T_2}{2^n}\right) .$$

Proof. We fix the adversary (T_1, T_2)-adversary $\mathcal{A}$ that maximizes $\mathsf{Adv}^{\mathsf{s\text{-}pre\text{-}PR}}_{H_{s,\ell,n}}(T_1, T_2)$. From Theorem 11.2.3 we have that there exists an adversary $\mathcal{B}$ that makes at most T queries to its h oracle, where $\mathsf{E}[T] \leqslant T_1/2^s + T_2$ and

$$\mathsf{Adv}^{\mathsf{s\text{-}pre\text{-}PR}}_{H_{s,\ell,n}}(\mathcal{A}) = \mathsf{Adv}^{\mathsf{PR}}_{H_{s,\ell,n}}(\mathcal{B}) .$$

Using Theorem 1 in [JT20], we can show that $\mathsf{Adv}^{\mathsf{PR}}_{H_{s,\ell,n}}(\mathcal{B}) \leqslant \frac{5\mathsf{E}[T]}{2^n}$, which completes the proof. ∎

Game $\mathsf{PR}^{\mathrm{h}}(\mathcal{B})$	Game $\mathsf{CR}^{\mathrm{h}}(\mathcal{A})$
$x \leftarrow\!\!\$\ \{0,1\}^*$	$(M, M') \leftarrow \mathcal{A}^{\mathrm{h}}$
$y \leftarrow \mathrm{h}(x)$	If $M \neq M'$ and $\mathrm{h}(M) = \mathrm{h}(M')$
$x' \leftarrow \mathcal{B}^{\mathrm{h}}(y)$	Return true
If $\mathrm{h}(x') = y$:	Return false
Return true	
Return false	

Figure 11.2: Left: Oracle Game PR^{h} for preimage-resistance of oracle h. Right: Oracle Game CR^{h} for collision-resistance of oracle h.

TIGHTNESS. We remark that this bound is tight up to constant factors. To see the tightness of the term $T_2/2^n$, consider the online-only adversary that simply makes k distinct queries with the salt a. It fails only if all the queries have answer different from y, which happens with probability $(1-1/2^n)^{T_2} \leqslant e^{-T_2/2^n}$. Since $e^{-x} \leqslant 1 - x/2$ for $x \leqslant 1.5$, for $T_2 \leqslant 2^n$, $e^{-T_2/2^n} \leqslant 1 - T_2/2^{n+1}$. This means the adversary succeeds with probability at least $T_2/2^{n+1}$, meaning the second term in the bound is tight up to constant factors.

To see why the first term is tight, consider the adversary $\mathcal{A}_1$ which makes 2^n queries on different inputs for $k = T_1/2^n$ different salts (where T_1 is a multiple of 2^n). In the online phase, it simply checks whether it had made a query with salt a, that had output y; if so it returns the query input.

Let the set of k salts $\mathcal{A}_1$ had made queries on be S. For each salt in S, the probability that $\mathcal{A}_1$ had not made query with that salt that had answer y is at most $(1-1/2^n)^{2^n} \leqslant 1/e$. So, for each salt in S with probability at least $(1-1/e)$, $\mathcal{A}_1$ had made a query that had answer y. Now $\mathcal{A}$ wins if the a sampled is in S, and $\mathcal{A}_1$ has made a query with salt a that had answer y; this probability is at least $(1-1/e)k/2^s$ since a is sampled at random. Since $k = T_1/2^n$, it follows that the second term in the bound is tight as well.

11.3.2 Collision-resistance of a salted random oracle

Oracle game CR^{h} in Fig. 11.2 formalizes the collision-resistance of oracle h. In the game the adversary has oracle access to h and wins if it manages to output M, M' such that $M \neq M'$ and $\mathrm{h}(M) = \mathrm{h}(M')$.

We aim to upper bound the advantage of offline-online adversaries $\mathcal{A}$ against collision-resistance of salted random oracles. Let $H_{s,\ell,n}$ be the uniform distribution over $\mathsf{Fcs}(\{0,1\}^s \times \{0,1\}^\ell, \{0,1\}^n)$. We seek to tightly upper bound $\mathsf{Adv}^{\mathsf{s\text{-}pre\text{-}CR}}_{H_{s,\ell,n}}(T_1, T_2)$. Using Theorem 11.2.3 and [JT20, Theorem 1] we get the following corollary.

Corollary 11.3.2. *Let $T_1, T_2, n, s, \ell \in \mathbb{N}_{>0}$. Then*

$$\mathsf{Adv}^{s\text{-}\mathsf{pre}\text{-}\mathsf{CR}}_{H_{s,\ell,n}}(T_1, T_2) \leqslant 5\sqrt{2}\left(\frac{T_1}{2^{s+n/2}} + \frac{T_2}{2^{n/2}}\right) .$$

Proof. We fix the adversary (T_1, T_2)-adversary $\mathcal{A}$ that maximizes $\mathsf{Adv}^{s\text{-}\mathsf{pre}\text{-}\mathsf{CR}}_{H_{s,\ell,n}}(T_1, T_2)$. From Theorem 11.2.3 we have that there exists an adversary $\mathcal{B}$ such that it makes at most T queries to its h oracle, where $\mathsf{E}[T] \leqslant T_1/2^s + T_2$, and

$$\mathsf{Adv}^{s\text{-}\mathsf{pre}\text{-}\mathsf{PR}}_{H_{s,\ell,n}}(\mathcal{A}) \leqslant \mathsf{Adv}^{\mathsf{PR}}_{H_{s,\ell,n}}(\mathcal{B}) .$$

Using [JT20, Theorem 1], we can show that $\mathsf{Adv}^{\mathsf{CR}}_{H_{s,\ell,n}}(\mathcal{B}) \leqslant 5\sqrt{\frac{2\mathsf{E}[T]^2}{2^n}}$, which concludes the proof. ∎

TIGHT BOUND. The bound in corollary 11.3.2 is suboptimal. In Theorem 11.3.3, we obtain a better bound for $\mathsf{Adv}^{s\text{-}\mathsf{pre}\text{-}\mathsf{CR}}_{H_{s,\ell,n}}(T_1, T_2)$.

Theorem 11.3.3. *Let $n, s, \ell, T_1, T_2 \in \mathbb{N}_{>0}$. Let $H_{s,\ell,n}$ be the uniform distribution on $\mathsf{Fcs}(\{0,1\}^s \times \{0,1\}^\ell, \{0,1\}^n)$. Then, we have that*

$$\mathsf{Adv}^{s\text{-}\mathsf{pre}\text{-}\mathsf{CR}}_{H_{s,\ell,n}}(T_1, T_2) \leqslant \frac{\binom{T_2}{2}}{2^n} + \frac{T_2 T_1}{2^{s+n}} + \frac{eT_1}{2^{s+n/2}} + \frac{n}{2^{s+1}} + \frac{1}{2^n} .$$

TIGHTNESS. We argue that this bound is tight up to constant factors. Initially, observe that for $T_2 \geqslant 2^{n/2}$, the right side becomes greater than one, and the bound always holds. For $T_2 \leqslant 2^{n/2}$, we have that $\frac{T_1 T_2}{2^{n+s}} \leqslant \frac{eT_1}{2^{s+n/2}}$. Therefore, the term $\frac{T_1 T_2}{2^{n+s}}$ is never the dominant term in the bound, and it suffices to show attacks that achieve advantage of the order $\frac{\binom{T_2}{2}}{2^n}$ and $\frac{eT_1}{2^{s+n/2}}$ to show that this bound is tight. A birthday style attack with T_2 queries achieves advantage of the order $\frac{\binom{T_2}{2}}{2^n}$. Finally, we prove the following theorem to show that term $\frac{eT_1}{2^{s+n/2}}$ is tight up to constant factors.

Theorem 11.3.4. *Let $T_1, s, n, \ell \in \mathbb{N}_{>0}$ such that n is a multiple of 2 and T_1 is a multiple of $2^{n/2+1}$. Let $H_{s,\ell,n}$ be the uniform distribution over $\mathsf{Fcs}(\{0,1\}^s \times \{0,1\}^\ell, \{0,1\}^n)$. Then there exists a $(T_1, 0)$-adversary $\mathcal{A}$ such that*

$$\mathsf{Adv}^{s\text{-}\mathsf{pre}\text{-}\mathsf{CR}}_{H_{s,\ell,n}}(\mathcal{A}) \geqslant \frac{(1-1/e)T_1}{2^{s+n/2+1}} .$$

We defer the formal proof of this theorem to Section 11.3.3. We next prove Theorem 11.3.3

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be the (T_1, T_2)-offline-online adversary that maximizes $\mathsf{Adv}^{s\text{-}\mathsf{pre}\text{-}\mathsf{CR}}_{H_{s,\ell,n}}(T_1, T_2)$. We can treat $\mathcal{A}$ as deterministic by fixing its randomness that maximizes its advantage.

We seek to upper bound the probability that the adversary $\mathcal{A}$ finds a one-block collision for the randomly chosen salt a that it gets as input in its online phase. We can assume without loss of generality that if $\mathcal{A}$ outputs a collision, it must have made the relevant queries either in the offline or the online phase. This is

without loss of generality because if $\mathcal{A}$ does not make one of these queries, we can construct a (T_1, T_2+2)-offline-online adversary $\mathcal{A}'$ that does whatever $\mathcal{A}$ does, and at the end of its online phase, makes the two relevant queries if not made earlier after $\mathcal{A}$ outputs M, M'. The term T_2 would then be replaced by T_2+2 in our bounds; for ease of readability, we omit this.

Also, without loss of generality we can assume that no query across the offline and online phases is repeated because the adversary can simply remember the query answer since we do not restrict its memory or the amount of advice it can pass on from offline to the online phase.

We define the following three events.

1. $\mathsf{onecoll}_{\mathsf{on}}$: $\mathcal{A}_2$ makes two queries $\mathrm{h}(a, M) = z$, $\mathrm{h}(a, M') = z$ for some $M \neq M'$

2. $\mathsf{onecoll}_{\mathsf{offon}}$: $\mathcal{A}_1$ makes a query $\mathrm{h}(a, M) = z$, and $\mathcal{A}_2$ makes a query whose answer is z

3. $\mathsf{onecoll}_{\mathsf{off}}$: $\mathcal{A}_1$ makes two queries $\mathrm{h}(a, M) = z$, $\mathrm{h}(a, M') = z$ for some $M \neq M'$

Observe that if none of $\mathsf{onecoll}_{\mathsf{on}}, \mathsf{onecoll}_{\mathsf{offon}}, \mathsf{onecoll}_{\mathsf{off}}$ happen, $\mathcal{A}$ cannot find a collision. We have that

$$\mathsf{Pr}\left[s\text{-}\mathsf{pre}\text{-}\mathsf{CR}^{\mathrm{h}}_{H_{s,\ell,n}}(\mathcal{A})\right] \leqslant \mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{on}}\right] + \mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{offon}}\right] + \mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{on}}\right] \ . \tag{11.3}$$

We upper bound the probability of these three events one by one.

Upper bounding $\mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{on}}\right]$. This event happens only if $\mathcal{A}_2$ makes two queries that collide. The probability of any two queries of $\mathcal{A}_2$ colliding is $1/2^n$. Using a union bound over all pairs of queries of $\mathcal{A}_2$, we have

$$\mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{on}}\right] \leqslant \frac{\binom{T_2}{2}}{2^n} \ . \tag{11.4}$$

Upper bounding $\mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{offon}}\right]$. Observe that $\mathsf{onecoll}_{\mathsf{offon}}$ happens only if there is an online query that has the same answer as one of the offline queries that had input salt a. There are a total of T_1 offline queries, and a is random. Therefore, the expected number of offline queries with salt a is $T_1/2^s$. We have that

$$\begin{aligned}
\mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{offon}}\right] &\leqslant \sum_{k=0}^{T_1} \mathsf{Pr}\left[\text{There are } k \text{ offline queries with salt } a\right] \frac{kT_2}{2^n} \\
&= \mathsf{E}\left[\text{Number of offline queries with salt } a\right] \frac{T_2}{2^n} \qquad (11.5) \\
&= \frac{T_1 T_2}{2^{s+n}} \ . \qquad (11.6)
\end{aligned}$$

UPPER BOUNDING $\mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{off}}\right]$. The main challenge of this proof is proving an upper bound on $\mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{off}}\right]$. We do it as follows: we define an event $\mathsf{off\text{-}oneblk}\text{-}k$ since there are k different salts for which a one-block collision has been found in the offline phase. We have that for any k,

$$\mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{off}}\right] \leqslant \mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}oneblk}\text{-}k\right] + \mathsf{Pr}\left[\mathsf{off\text{-}oneblk}\text{-}k\right] \ . \tag{11.7}$$

Since $\mathsf{onecoll}_{\mathsf{off}}$ happens if for the salt a that is chosen uniformly at random from $\{0,1\}^s$, $\mathcal{A}_1$ had queried $\mathrm{h}(a,M), \mathrm{h}(a,M')$ that have the same answer, we have that $\mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}oneblk}\text{-}k\right] \leqslant k/2^n$. We upper bound $\mathsf{Pr}\left[\mathsf{off\text{-}oneblk}\text{-}k\right]$ using a compression argument.

The encoding procedure encodes the random oracle h as follows.

1. It runs $\mathcal{A}_1^{\mathrm{h}}$ and initializes a list L to the empty list and a set S to the empty set.

2. For every query $\mathrm{h}(a,M)$ made by $\mathcal{A}_1$, it does the following:

   (a) Let $z = \mathrm{h}(a,M)$. If there was *exactly one* earlier query by $\mathcal{A}_1$ of the form (a,M') for some $M' \neq M$ that had answer z, and $|\mathsf{S}| < 2k$, it adds the index of the query $\mathrm{h}(a,M')$ and the current query to S.

   (b) Otherwise, it adds $\mathrm{h}(a,M)$ to L.

3. It appends the evaluation of h on the points not queried by $\mathcal{A}_1$ to L in the lexicographical order of the inputs.

4. If $|\mathsf{S}| < 2k$ it outputs $\varnothing$; otherwise, it outputs L, S.

The decoding procedure works as follows.

1. If the encoding is $\varnothing$, it aborts.

2. It runs $\mathcal{A}_1^{\mathrm{h}}$.

3. For every query $\mathrm{h}(a,M)$ made by $\mathcal{A}_1$, it does the following:

   (a) If the index of the query is in S, and there is an earlier query $\mathrm{h}(a,M')$ for some M' by $\mathcal{A}_1$ such that its index is in S, answer this query with $\mathrm{h}(a,M')$.

   (b) Otherwise, it removes the element in front of L and answers with that.

4. It populates h on the points not queried by $\mathcal{A}_1$ in the lexicographical order by the remaining entries of L

Correctness of decoding: For adversary $\mathcal{A}_1$ that causes the event $\mathsf{off\text{-}oneblk}\text{-}k$ to happen, the encoding algorithm will never return $\varnothing$ because by the definition of $\mathsf{off\text{-}oneblk}\text{-}k$ there will be at least k different salts a_i for which $\mathcal{A}_1$ queries $\mathrm{h}(a_i, M_i) = z_i$, $\mathrm{h}(a_i, M_i') = z_i$; meaning the size of S will be $2k$. For such an adversary $\mathcal{A}$ it is easy to verify the decoding algorithm will always produce the correct output because for the answers of h that were not added to L, the decoding algorithm recovers them using the set of indices S. Therefore, we have that

$$\mathsf{Pr}\left[\text{ Decoding is correct }\right] \geqslant \mathsf{Pr}\left[\mathsf{off\text{-}oneblk}\text{-}k\right] \ .$$

The size of the output space of the encoding algorithm is upper bounded by $\binom{T_1}{2k} \cdot (2^n)^{2^{s+\ell}-k}$. The size of the input space is $(2^n)^{2^{s+\ell}}$. From the compression lemma (Proposition 11.1.1), we have that

$$\mathsf{Pr}\left[\mathsf{off\text{-}oneblk}\text{-}k\right] \leqslant \mathsf{Pr}\left[\text{ Decoding is correct }\right] \leqslant \frac{\binom{T_1}{2k}}{2^{kn}} \ .$$

We $k = \max\left(\frac{eT_1}{2^{n/2}}, n/2\right)$. If $\max\left(\frac{eT_1}{2^{n/2}}, n/2\right) = \frac{eT_1}{2^{n/2}}$, then $k \geqslant n/2$. Therefore,

$$\frac{\binom{T_1}{2k}}{2^{nk}} \leqslant \left(\frac{eT_1}{2^{n/2}(2k)}\right)^{2k} \leqslant \left(\frac{1}{2}\right)^n \ .$$

If $\max\left(\frac{eT_1}{2^{n/2}}, n/2\right) = n/2$, then $k \geqslant \frac{eT_1}{2^{n/2}}$. Therefore,

$$\frac{\binom{T_1}{2k}}{2^{nk}} \leqslant \left(\frac{eT_1}{2^{n/2}(2k)}\right)^{2k} \leqslant \left(\frac{1}{2}\right)^n \ .$$

Therefore, for $k = \max\left(\frac{eT_1}{2^{n/2}}, n/2\right)$, $\mathsf{Pr}\left[\mathsf{off\text{-}oneblk}\text{-}k\right] \leqslant 1/2^n$. Therefore, from (11.8) we have

$$\mathsf{Pr}\left[\mathsf{onecoll}_{\mathsf{off}}\right] \leqslant \frac{eT_1}{2^{s+n/2}} + \frac{n}{2^{s+1}} + \frac{1}{2^n} \ . \tag{11.8}$$

Plugging (11.4), (11.6) and (11.7) into (11.3) gives us that

$$\mathsf{Adv}^{s\text{-}\mathsf{pre}\text{-}\mathsf{CR}}_{H_{s,\ell,n}}(\mathcal{A}) \leqslant \frac{\binom{T_2}{2}}{2^n} + \frac{T_1T_2}{2^{n+s}} + \frac{eT_1}{2^{s+n/2}} + \frac{n}{2^{s+1}} + \frac{1}{2^n} \ .$$

□ ■

11.3.3 Attacks

In this section we add the proof of the various theorems that show the tightness of the bounds we obtained for security against offline-online adversaries.

Attacks for pre-2-CR-$\mathsf{MD}_n^{\mathrm{h}}$ *(Proofs of Theorems 11.4.5 and 11.4.6)*

We first prove Theorem 11.4.6.

Proof. [Theorem 11.4.6] Let $T_1/2^{n/2+2} = k$. We start by describing $\mathcal{A}$. Its offline phase $\mathcal{A}_1$ does the following.

1. Let S be a set of k distinct salts.
2. For each salt a in S, it makes $2^{n/2+2}$ distinct queries with the salt
3. For each salt in $\{0,1\}^n \backslash \mathsf{S}$, it makes $T_1/2^{n+1}$ distinct queries

Its online phase $\mathcal{A}_2$ does the following: on getting input salt a, it checks if $\mathcal{A}_1$ had made queries that led to a two-block MD-collision for a. If so it outputs the collision.

Clearly $\mathcal{A}_1$ makes $T_1/2^{n/2+2} \cdot 2^{n/2+1} + T_1/2^{n+1} \cdot (2^n - T_1/2n/2 + 2) < T_1$ queries, and $\mathcal{A}_2$ makes no queries. So $\mathcal{A}$ is a $(T_1, 0)$-adversary.

Now, we analyze the advantage of $\mathcal{A}$. We say that a salt a' in S is *good* if the $\mathcal{A}_1$ made two queries for a' that collided. For each $a' \in \mathsf{S}$, we have that

$$\mathsf{Pr}\left[a' \text{ is good}\right] = 1 - \mathsf{Pr}\left[a' \text{ is not good}\right] .$$

Observe that a' is not good if and only if all the $2^{n/2+1}$ queries made on it produce distinct answers

$$\begin{aligned}
\mathsf{Pr}\left[a' \text{ is not good}\right] &= \prod_{i=1}^{2^{n/2+1}} \left(1 - \frac{i-1}{2^n}\right) \\
&\leqslant \prod_{i=1}^{2^{n/2+1}} e^{-\frac{(i-1)}{2^n}} \\
&= e^{-\frac{(2^{n/2+1})(2^{n/2+1}-1)}{2^n}} \leqslant 1/e .
\end{aligned}$$

Let X be the random variable that denotes the number of salts in S that are not good. We have that $\mathsf{E}\left[X\right] \leqslant k/e$. Using Markov's equality we have that

$$\mathsf{Pr}\left[X \geqslant 2k/e\right] \leqslant \frac{\mathsf{E}\left[X\right]}{2k/e} \leqslant \frac{1}{2} .$$

Therefore,

$$\mathsf{Pr}\left[X \leqslant 2k/e\right] \geqslant \frac{1}{2} .$$

Let Q_a be the event that for a salt a, $\mathcal{A}_1$ made a query on a in line 3 such that one of the queries had an answer a' such that a' is a good salt in S. Let $\mathsf{win}_{\mathcal{A}}$ be the event that the a that was sampled before the online phase was such that

1. $a \notin \mathsf{S}$

2. Q_a happens

We have that

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}}\right] &= \mathsf{Pr}\left[a \notin \mathsf{S} \wedge Q_a\right] \\ &\geqslant \mathsf{Pr}\left[a \notin \mathsf{S} \wedge Q_a \wedge X \leqslant 2k/e\right] \\ &\quad \mathsf{Pr}\left[a \notin \mathsf{S} \wedge X \leqslant 2k/e\right] \mathsf{Pr}\left[Q_a \,\middle|\, a \notin \mathsf{S} \wedge X \leqslant 2k/e\right] .\end{aligned}$$

Since a is sampled independent of $\mathcal{A}_1$ we have that

$$\mathsf{Pr}\left[a \notin \mathsf{S} \wedge X \leqslant 2k/e\right] = \mathsf{Pr}\left[a \notin \mathsf{S}\right] \cdot \mathsf{Pr}\left[X \leqslant 2k/e\right] .$$

$\mathsf{Pr}\left[a \notin \mathsf{S}\right]$ $1 - k/2^n$ since a is sampled uniformly at random. Since $T_1 \leqslant 2^{5n/4}$, and $n \geqslant 2$, $\mathsf{Pr}\left[a \notin \mathsf{S}\right] \geqslant 1/2$. Further we have $\mathsf{Pr}\left[X \leqslant 2k/e\right] \geqslant 1/2$. Therefore,

$$\mathsf{Pr}\left[a \notin \mathsf{S} \wedge X \leqslant 2k/e\right] \geqslant 1/4 .$$

If $a \notin \mathsf{S}$, Q_a does not happen only if none queries made on $\mathcal{A}_1$ have answers that are not among the good salts in S. Therefore, we have that

$$\mathsf{Pr}\left[Q_a \,\middle|\, a \notin \mathsf{S} \wedge X \leqslant 2k/e\right] \geqslant 1 - \left(\frac{2^n - k(1-2/e)}{2^n}\right)^{T_1/2^{n+1}} .$$

We have that

$$\begin{aligned}\left(\frac{2^n - k(1-2/e)}{2^n}\right)^{T_1/2^{n+1}} &= \left(1 - \frac{k(1-2/e)}{2^n}\right)^{T_1/2^{n+1}} \\ &\leqslant e^{-\frac{k(1-2/e)}{2^n} \cdot \frac{T_1}{2^{n+1}}} .\end{aligned}$$

Note that

$$\frac{k(1-2/e)}{2^n} \cdot \frac{T_1}{2^{n+1}} = \frac{(1-2/e)T_1^2}{2^{5n/2+3}} \leqslant 1 .$$

Above we used the fact that $T_1 \leqslant 2^{5n/4}$. We have that $e^{-x} \leqslant 1 - x/2$ for $0 \leqslant x \leqslant 1.5$, therefore,

$$\left(\frac{2^n - k(1-2/e)}{2^n}\right)^{T_1/2^{n+1}} \leqslant 1 - \frac{(1-2/e)T_1^2}{2^{5n/2+4}} .$$

This implies

$$\mathsf{Pr}\left[Q_a \,\middle|\, a \notin \mathsf{S} \wedge X \leqslant 2k/e\right] \geqslant \frac{(1-2/e)T_1^2}{2^{5n/2+4}} .$$

Therefore,

$$\mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}}\right] \geqslant (1/4) \cdot \frac{(1-2/e)T_1^2}{2^{5n/2+4}} = \frac{(1-2/e)T_1^2}{2^{5n/2+6}} \ .$$

Clearly if $\mathsf{win}_{\mathcal{A}}$ happens, $\mathcal{A}$ has won. Hence

$$\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}CR\text{-}MD}}_{H_{n,\ell,n}} \geqslant \frac{(1-2/e)T_1^2}{2^{5n/2+6}} \ .$$

□ ■

We next prove Theorem 11.4.5.

Proof. [Theorem 11.4.5] Let $T_1/2^{n/2+1} = k$. We start by describing $\mathcal{A}$. Its offline phase $\mathcal{A}_1$ does the following.

1. Let S be a set of k distinct salts.

2. For each salt a in S, it makes $2^{n/2+1}$ distinct queries with the salt

Its online phase $\mathcal{A}_2$ does the following:

1. Let T be the set of salts for which $\mathcal{A}_1$ found a collision.

2. On getting salt a input it makes T_2 distinct queries with salt a. If any of the answers are in T – say the query $\mathrm{h}(a, M)$ produced a' such that $\mathcal{A}_1$ had made queries $\mathrm{h}(a', M') = z$ and $\mathrm{h}(a', M'') = z$ for $M' \neq M''$. It outputs (M, M') and (M, M'') as the two-block collision.

Clearly $\mathcal{A}_1$ makes $T_1/2^{n/2+1} \cdot 2^{n/2+1} = T_1$ queries, and $\mathcal{A}_2$ makes T_2 queries. So $\mathcal{A}$ is a (T_1, T_2)-adversary.

Now, we analyze the advantage of $\mathcal{A}$. We say that a salt a' in S is *good* if the $\mathcal{A}_1$ made two queries for a' that collided. Let X be the random variable that denotes the number of salts in S that are not good.

In our analysis in 11.4.6, we showed that

$$\mathsf{Pr}\left[X \leqslant 2k/e\right] \geqslant \frac{1}{2} \ .$$

Let $\mathsf{win}_{\mathcal{A}}$ be the event that for a salt a, $\mathcal{A}_2$ made a query on a such that one of the queries had an answer a' such that a' is a good salt in S. We have that

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}}\right] &\geqslant \mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}} \wedge X \leqslant (2k/e\right] \\ &\quad \mathsf{Pr}\left[X \leqslant 2k/e\right] \mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}} \,\middle|\, X \leqslant 2k/e\right] \ .\end{aligned}$$

Note that $\mathsf{win}_{\mathcal{A}}$ does not happen only if none queries made on $\mathcal{A}_2$ have answers that are not among the good salts in S. Therefore, we have that

$$\mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}} \,\middle|\, X \leqslant 2k/e\right] \geqslant 1 - \left(\frac{2^n - k(1-2/e)}{2^n}\right)^{T_2} .$$

We have that

$$\begin{aligned}\left(\frac{2^n - k(1-2/e)}{2^n}\right)^{T_2} &= \left(1 - \frac{k(1-2/e)}{2^n}\right)^{T_2} \\ &\leqslant e^{-\frac{k(1-2/e)}{2^n}\cdot T_2} .\end{aligned}$$

Note that

$$\frac{k(1-2/e)}{2^n} \cdot T_2 = \frac{(1-2/e)T_1T_2}{2^{3n/2+1}} \leqslant 1 .$$

Above we used the fact that $T_1T_2 \leqslant 2^{3n/2}$. We have that $e^{-x} \leqslant 1 - x/2$ for $0 \leqslant x \leqslant 1.5$. Therefore,

$$\left(\frac{2^n - k(1-2/e)}{2^n}\right)^{T_2} \leqslant 1 - \frac{(1-2/e)T_1T_2}{2^{3n/2+2}} .$$

This implies

$$\mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}} \,\middle|\, X \leqslant 2k/e\right] \geqslant \frac{(1-2/e)T_1T_2}{2^{3n/2+2}} .$$

Therefore,

$$\mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}}\right] \geqslant (1/2) \cdot \frac{(1-2/e)T_1T_2}{2^{3n/2+2}} = \frac{(1-2/e)T_1T_2}{2^{3n/2+3}} .$$

Clearly if $\mathsf{win}_{\mathcal{A}}$ happens, $\mathcal{A}$ has won. Hence

$$\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}CR\text{-}MD}}_{H_{n,\ell,n}} \geqslant \frac{(1-2/e)T_1T_2}{2^{3n/2+3}} .$$

□ ■

Attacks on $\mathsf{pre\text{-}2\text{-}PR\text{-}MD}^{\mathrm{h}}_n$ *(Proofs of Theorems 11.4.2 and 11.4.3)*

We first prove Theorem 11.4.2.

Proof. [Theorem 11.4.2] Let $k = T_1/2^n$. The offline phase of the adversary $\mathcal{A}$ does the following:

1. It chooses a set of k distinct salts S

2. For each salt $a \in \mathsf{S}$, it makes 2^{n+2} distinct queries

Its online phase $\mathcal{A}_2$ does the following: on getting input salt a, it makes T_2 distinct queries with salt a. If some query $\mathrm{h}(a, M') = a'$ such that $a' \in \mathsf{S}$ and for $\mathcal{A}_1$ had made a query $\mathrm{h}(a', M'') = y$ then $\mathcal{A}_2$ outputs the (M', M'').

Clearly $\mathcal{A}_1$ makes $T_1/2^n \cdot 2^n = T_1$ queries, and $\mathcal{A}_2$ makes T_2 queries. So $\mathcal{A}$ is a (T_1, T_2)-adversary.

Now, we analyze the advantage of $\mathcal{A}$. We say that a salt a' in S is *good* if the $\mathcal{A}_1$ made a query for a' that had answer y. For each $a' \in \mathsf{S}$, we have that

$$\mathsf{Pr}\left[a' \text{ is good}\right] = 1 - \mathsf{Pr}\left[a' \text{ is not good}\right] .$$

Observe that a' is not good if and only if all the 2^n queries made on it produce answers other than y

$$\begin{aligned}\mathsf{Pr}\left[a' \text{ is not good}\right] &= \left(1 - \frac{1}{2^n}\right)^{2^n} \\ &\leqslant 1/e .\end{aligned}$$

Let X be the random variable that denotes the number of salts in S that are not good. We have that $\mathsf{E}[X] \leqslant k/e$. Using Markov's equality we have that

$$\mathsf{Pr}\left[X \geqslant 2k/e\right] \leqslant \frac{\mathsf{E}[X]}{2k/e} \leqslant \frac{1}{2} .$$

Therefore,

$$\mathsf{Pr}\left[X \leqslant 2k/e\right] \geqslant \frac{1}{2} .$$

Let $\mathsf{win}_{\mathcal{A}}$ be the event that for a salt a, $\mathcal{A}_2$ made a query on a such that one of the query had an answer a' such that a' is a good salt in S. We have that

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}}\right] \geqslant& \mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}} \wedge X \leqslant 2k/e\right] \\ =& \mathsf{Pr}\left[X \leqslant 2k/e\right] \mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}} \,\middle|\, X \leqslant 2k/e\right] .\end{aligned}$$

Note that $\mathsf{win}_{\mathcal{A}}$ does not happen only if none queries made on $\mathcal{A}_1$ have answers that are not among the good salts in S. Therefore, we have that

$$\mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}} \,\middle|\, X \leqslant 2k/e\right] \geqslant 1 - \left(\frac{2^n - k(1-2/e)}{2^n}\right)^{T_2} .$$

We have that

$$\begin{aligned}\left(\frac{2^n - k(1-2/e)}{2^n}\right)^{T_1/2^{n+1}} &= \left(1 - \frac{k(1-2/e)}{2^n}\right)^{T_2} \\ &\leqslant e^{-\frac{k(1-2/e)}{2^n} \cdot T_2} .\end{aligned}$$

Note that

$$\frac{k(1-2/e)}{2^n} \cdot T_2 = \frac{(1-2/e)T_1T_2}{2^{2n}} \leqslant 1 \ .$$

Above we used the fact that $T_1T_2 \leqslant 2^{2n}$. We have that $e^{-x} \leqslant 1 - x/2$ for $0 \leqslant x \leqslant 1.5$, therefore,

$$\left(\frac{2^n - k(1-2/e)}{2^n}\right)^{T_2} \leqslant 1 - \frac{(1-2/e)T_1T_2}{2^{2n}} \ .$$

This implies

$$\mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}} \,\middle|\, X \leqslant 2k/e\right] \geqslant \frac{(1-2/e)T_1T_2}{2^{2n}} \ .$$

Therefore,

$$\mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}}\right] \geqslant (1/2) \cdot \frac{(1-2/e)T_1T_2}{2^{2n}} = \frac{(1-2/e)T_1T_2}{2^{2n+1}} \ .$$

Clearly if $\mathsf{win}_{\mathcal{A}}$ happens, $\mathcal{A}$ has won. Hence

$$\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}PR\text{-}MD}}_{H_{n,\ell,n}} \geqslant \frac{(1-2/e)T_1T_2}{2^{2n+1}} \ .$$

□ ■

We next prove Theorem 11.4.3.

Proof. [Theorem 11.4.3] Let $k = T_1/2^{n+1}$. The offline phase of the adversary $\mathcal{A}$ does the following:

1. It chooses a set of k distinct salts S
2. For each salt $a \in \mathsf{S}$, it makes 2^{n+2} distinct queries
3. For every salt in $\{0,1\}^n \backslash \mathsf{S}$ it makes $T_1/2^{n+1}$ distinct queries

Its online phase $\mathcal{A}_2$ does the following: on getting input salt a, it checks the queries made by $\mathcal{A}_1$ to find a message M, at most two blocks long such that the $\mathsf{MD}^{\mathrm{h}}(a, M) = y$. If so it outputs M.

Clearly $\mathcal{A}_1$ makes $T_1/2^{n+1} \cdot 2^n + T_1/2^{n+1} \cdot (2^n - T_1/2n/2 + 2) < T_1$ queries, and $\mathcal{A}_2$ makes no queries. So $\mathcal{A}$ is a $(T_1, 0)$-adversary.

Now, we analyze the advantage of $\mathcal{A}$. We say that a salt a' in S is *good* if the $\mathcal{A}_1$ made a query for a' that had answer y. For each $a' \in \mathsf{S}$, we have that

$$\mathsf{Pr}\left[a' \text{ is good}\right] = 1 - \mathsf{Pr}\left[a' \text{ is not good}\right] \ .$$

Observe that a' is not good if and only if all the 2^n queries made on it produce answers other than y

$$\begin{aligned}\mathsf{Pr}\left[a' \text{ is not good}\right] &= \left(1 - \frac{1}{2^n}\right)^{2^n} \\ &\leqslant 1/e \ .\end{aligned}$$

Let X be the random variable that denotes the number of salts in S that are not good. We have that $\mathsf{E}\left[X\right] \leqslant k/e$. Using Markov's equality we have that

$$\mathsf{Pr}\left[X \geqslant 2k/e\right] \leqslant \frac{\mathsf{E}\left[X\right]}{2k/e} \leqslant \frac{1}{2} \ .$$

Therefore,

$$\mathsf{Pr}\left[X \leqslant 2k/e\right] \geqslant \frac{1}{2} \ .$$

Let Q_a be the event that for a salt a, $\mathcal{A}_1$ made a query on a in line 3 such that one of the queries had an answer a' such that a' is a good salt in S. Let $\mathsf{win}_{\mathcal{A}}$ be the event that the a that was sampled before the online phase was such that

1. $a \notin \mathsf{S}$
2. Q_a happens

We have that

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}}\right] &= \mathsf{Pr}\left[a \notin \mathsf{S} \wedge Q_a\right] \\ &\geqslant \mathsf{Pr}\left[a \notin \mathsf{S} \wedge Q_a \wedge X \leqslant 2k/e\right] \\ &\quad \mathsf{Pr}\left[a \notin \mathsf{S} \wedge X \leqslant 2k/e\right] \mathsf{Pr}\left[Q_a \,\middle|\, a \notin \mathsf{S} \wedge X \leqslant 2k/e\right] \ .\end{aligned}$$

Since a is sampled independent of $\mathcal{A}_1$ we have that

$$\mathsf{Pr}\left[a \notin \mathsf{S} \wedge X \leqslant 2k/e\right] = \mathsf{Pr}\left[a \notin \mathsf{S}\right] \cdot \mathsf{Pr}\left[X \leqslant 2k/e\right] \ .$$

We have that $\mathsf{Pr}\left[a \notin \mathsf{S}\right] = 1 - k/2^n$ since a is sampled uniformly at random. Since $T_1 \leqslant 2^{3n/2}$, and $n \geqslant 2$, $\mathsf{Pr}\left[a \notin \mathsf{S}\right] \geqslant 1/2$. Further we have $\mathsf{Pr}\left[X \leqslant 2k/e\right] \geqslant 1/2$. Therefore,

$$\mathsf{Pr}\left[a \notin \mathsf{S} \wedge X \leqslant 2k/e\right] \geqslant 1/4 \ .$$

If $a \notin \mathsf{S}$, Q_a does not happen only if none queries made on $\mathcal{A}_1$ have answers that are not among the good salts in S. Therefore, we have that

$$\mathsf{Pr}\left[Q_a \,\middle|\, a \notin \mathsf{S} \wedge X \leqslant 2k/e\right] \geqslant 1 - \left(\frac{2^n - k(1 - 2/e)}{2^n}\right)^{T_1/2^{n+1}} \ .$$

We have that

$$\left(\frac{2^n - k(1-2/e)}{2^n}\right)^{T_1/2^{n+1}} = \left(1 - \frac{k(1-2/e)}{2^n}\right)^{T_1/2^{n+1}} \leqslant e^{-\frac{k(1-2/e)}{2^n}\cdot\frac{T_1}{2^{n+1}}}$$

Note that

$$\frac{k(1-2/e)}{2^n}\cdot\frac{T_1}{2^{n+1}} = \frac{(1-2/e)T_1^2}{2^{3n+2}} \leqslant 1 \ .$$

Above we used the fact that $T_1 \leqslant 2^{3n}$. We have that $e^{-x} \leqslant 1 - x/2$ for $0 \leqslant x \leqslant 1.5$, therefore,

$$\left(\frac{2^n - k(1-2/e)}{2^n}\right)^{T_1/2^{n+1}} \leqslant 1 - \frac{(1-2/e)T_1^2}{2^{3n+2}} \ .$$

This implies

$$\mathsf{Pr}\left[Q_a \,\middle|\, a \notin \mathsf{S} \wedge X \leqslant 2k/e\right] \geqslant \frac{(1-2/e)T_1^2}{2^{3n+2}} \ .$$

Therefore,

$$\mathsf{Pr}\left[\mathsf{win}_{\mathcal{A}}\right] \geqslant (1/4)\cdot\frac{(1-2/e)T_1^2}{2^{3n+2}} = \frac{(1-2/e)T_1^2}{2^{3n+4}} \ .$$

Clearly if $\mathsf{win}_{\mathcal{A}}$ happens, $\mathcal{A}$ has won. Hence

$$\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}PR\text{-}MD}}_{H_{n,\ell,n}} \geqslant \frac{(1-2/e)T_1^2}{2^{3n+4}} \ .$$

□ ■

Offline-only attack for s-pre-CR$^{\mathrm{h}}$ (Proof of Theorem 11.3.4)

In this section we prove Theorem 11.3.4.

Proof. [Theorem 11.3.4] Let $k = T_1/2^{n/2+1}$. We describe the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. Its offline phase $\mathcal{A}_1$ works as follows:

1. It picks a set S of k distinct salts
2. For each salt in S it makes $2^{n/2+1}$ distinct queries

The online phase on getting salt a as input simply checks whether $\mathcal{A}_1$ queried a collision for $\mathcal{A}$. If so it outputs the collision.

We say that a salt a' in S is *good* if the $\mathcal{A}_1$ made two queries for a' that collided. For each $a' \in \mathsf{S}$, we have that

$$\mathsf{Pr}\left[a' \text{ is good}\right] = 1 - \mathsf{Pr}\left[a' \text{ is not good}\right] .$$

Observe that a' is not good if and only if all the $2^{n/2+1}$ queries made on it produce distinct answers

$$\begin{aligned}\mathsf{Pr}\left[a' \text{ is not good}\right] &= \prod_{i=1}^{2^{n/2+1}} \left(1 - \frac{i-1}{2^n}\right) \leqslant \prod_{i=1}^{2^{n/2+1}} e^{-\frac{(i-1)}{2^n}} \\ &= e^{-\frac{(2^{n/2+1})(2^{n/2+1}-1)}{2^n}} \leqslant 1/e .\end{aligned}$$

Now, notice that $\mathcal{A}$ wins whenever a is sampled from the set of good salts in S. Therefore

$$\begin{aligned}\mathsf{Adv}^{\mathsf{CR}}_{H_{s,m,n}}(\mathcal{A}) &\geqslant \mathsf{Pr}\left[a \in \mathsf{S} \wedge a \text{ is a good salt in } \mathsf{S}\right] \\ &= \mathsf{Pr}\left[a \in \mathsf{S}\right] \cdot \mathsf{Pr}\left[a \text{ is a good salt in } \mid a \in \mathsf{S}\right] \\ &\geqslant (1 - 1/e)k/2^n = \frac{(1-1/e)T_1}{2^{s+n/2+1}} .\end{aligned}$$

The second inequality above follows since a is sampled at random and for all $a' \in \mathsf{S}$, $\mathsf{Pr}\left[a' \text{ is good}\right] \geqslant 1 - 1/e$. □ ■

We note that there is a dual of this strategy for an offline-only adversary compared to the one we presented in this proof – instead of making $2^{n/2}$ queries for $T_1/2^{n/2}$ salts, it could make $T_1/2^s$ for all 2^s salts – in this case its advantage of be of the order $T_1^2/2^{2s+n}$. Note that if we care about constant advantage, then both the attack strategies need the same amount of queries. However, this latter attack is worse than the one we give in Theorem 11.3.4 since $T_1^2/2^{2s+n} \leqslant T_1/2^{s+n/2}$ when both the terms are at most one. In Section 11.3.4, we formalize this difference between the two attack strategies for a general class of games.

11.3.4 Strategies for offline-only attacks

In this section, we present the qualitative difference between two strategies behind offline only attacks.

Suppose game G is compatible with ideal distribution $\mathcal{I}$. Suppose there exists $d, N, T_0 \in \mathbb{N}$, $c \in [0, 1]$ that depend on $\mathsf{G}, \mathcal{I}$ and satisfy the following.

- $T_0 \geqslant N^{1/d}$
- for all $T \leqslant T_0$, there exists a T-query adversary $\mathcal{A}_T$ such that $\mathsf{Adv}^{\mathsf{G}}_{\mathcal{I}}(\mathcal{A}_T) \geqslant c \cdot \frac{T^d}{N}$

$$\mathsf{Adv}^{\mathsf{G}}_{\mathcal{I}}(\mathcal{A}) \geqslant c \cdot \frac{T^d}{N} .$$

This property hold for several random oracle based games, e.g., for collision-resistance of random oracles with n bit outputs, $c = 1/2, d = 2, T_0 = 2^{n/2}$.

Let $T_1, s \in \mathbb{N}_{>0}$. Assume T_1 is a multiple of 2^s, and $N^{1/d}$, and $T_1 \leqslant N^{1/d} \cdot 2^s$. Consider the two following T_1-query offline-only adversaries $\mathcal{B}, \mathcal{C}$ against s-pre-G^π.

- $\mathcal{B}_1$: It chooses a set S of $T_1/N^{1/d}$ salts. For each salt a in the set, it runs $\mathcal{A}_T$ for $T = N^{1/d}$ and simulates G^{π_a} to it.

- $\mathcal{C}_1$: For each salt $a \in \{0,1\}^s$, it runs $\mathcal{A}_{T'}$ for $T' = T_1/2^s$, simulating G^{π_a} to it.

The online phases of both adversaries do not make any queries themselves, and decide their output based on the queries made by their offline phase.

First we analyze the success probability of $\mathcal{B}$. We have that $\mathcal{B}$ wins if the sampled salt a was in S and $\mathsf{G}^{\pi_a}(\mathcal{A}_T)$ (where $T = N^{1/d}$) returns true. Therefore,

$$\begin{aligned}\mathsf{Adv}^{s\text{-pre-}\mathsf{G}}_{\mathcal{I}_s}(\mathcal{B}) &= \mathsf{Pr}\left[\{\right] a \in \mathsf{S}\}\mathsf{Pr}\left[\mathsf{G}^{\pi_a}(\mathcal{A}_T) \,\middle|\, a \in \mathsf{S}\right] \\ &\geqslant \frac{T_1}{N^{1/d}2^s} \cdot c = \frac{cT_1}{N^{1/d}2^s} \,.\end{aligned}$$

The inequality follows because a is sampled independently of $\mathcal{A}$, and for any salt a, $\mathsf{Adv}^{\mathsf{G}}_{\mathcal{I}}(\mathcal{A}_T) \geqslant c \cdot \frac{T^d}{N} = c$.

Next we analyze the success probability of $\mathcal{C}$. We have that $\mathcal{C}$ wins if $\mathsf{G}^{\pi_a}(\mathcal{A}_{T'})$ (where $T' = T_1/2^s$) returns true. Therefore,

$$\begin{aligned}\mathsf{Adv}^{s\text{-pre-}\mathsf{G}}_{\mathcal{I}_s}(\mathcal{B}) &= \mathsf{Pr}\left[\mathsf{G}^{\pi_a}(\mathcal{A}_{T'})\right] \\ &\geqslant c \cdot \frac{T_1^d}{2^{sd}N} \,.\end{aligned}$$

Note that since $T_1 \leqslant N^{1/d} \cdot 2^s$, $\frac{cT_1}{N^{1/d}2^s} \geqslant \frac{T_1^d}{2^{sd}N}$. So for $d \geqslant 1$, the better offline only strategy is to trying to win on $T_1/N^{1/d}$ salts instead of distributing queries across all 2^s salts.

11.4 Offline-Online Security of Two-Block Merkle-Damgård

In previous sections, we focused on proving security guarantees against offline-online attacks on constructions where every query to the ideal primitive is salted. Here, we will see an example of a construction (Merkle Damgård) where only the first query to the underlying primitive (random oracle) is salted. Specifically, we will study the pre-image-resistance and collision-resistance for two-block Merkle-Damgård.

The main takeaway from this section is that for primitives that are not salted for every call, we can have parameter regimes where a term of the form T_1T_2 dominates the bound, meaning that in these regimes there are tradeoffs between the number of offline and online queries for the advantage to be close to one. This

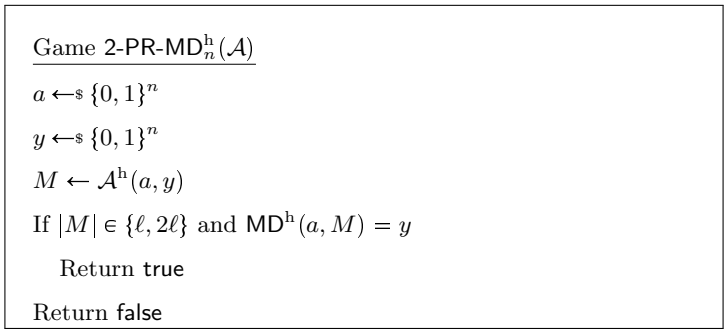


Figure 11.3: Oracle game 2-PR-MD$_n^{\mathrm{h}}$ formalizing the pre-image-resistance of the two-block MD construction.

contrasts with what we saw earlier in Section 11.2.2. There we demonstrated that for constructions that salt every query to the ideal primitive, an adversary must have either online time close to the online time of the best online-only attack that attains an advantage close to one, or offline time close to the best offline-only attack that attains an advantage close to one to achieve an advantage close to one.

11.4.1 Pre-image-resistance of two-block Merkle-Damgård

In this section we study the offline-online attacks against the pre-image-resistance of the two-block Merkle-Damgaard (MD) construction. Pre-image-resistance of two-block Merkle-Damgård is formalized in the game 2-PR-MD$_n^{\mathrm{h}}$ in Figure 11.3. A salt a and a value y are sampled uniformly at random from $\{0,1\}^n$ and given as input to $\mathcal{A}$. $\mathcal{A}$ can make queries to h and wins if it outputs a message M that is one or two blocks long whose MD evaluation with a is y.

Let $H_{n,\ell,n}$ be the uniform distribution over $\mathsf{Fcs}(\{0,1\}^n \times \{0,1\}^\ell, \{0,1\}^n)$. We are interested in proving an upper bound on $\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}PR\text{-}MD}}_{H_{n,\ell,n}}(T_1,T_2)$. We prove the following theorem.

Theorem 11.4.1. *Let $T_1, T_2, n, \ell \in \mathbb{N}_{>0}$. Let $H_{n,\ell,n}$ be the uniform distribution over $\mathsf{Fcs}(\{0,1\}^n \times \{0,1\}^\ell, \{0,1\}^n)$. Then*

$$\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}PR\text{-}MD}}_{H_{n,\ell,n}}(T_1,T_2) \leqslant \frac{2T_2+1}{2^n} + \frac{T_1T_2 + nT_1 + T_1}{2^{2n}} + \frac{4eT_1^2}{2^{3n}} \ .$$

Offline-online tradeoffs. Note that in the regime $T_1 = 2^{n(1+\varepsilon)}, T_2 = 2^{n(1-\varepsilon)}$ for $0 < \varepsilon < 1/2$, the term $T_1T_2/2^{2n}$ dominates the bound, meaning there is an offline-online query tradeoff in that regime.

Tightness. We show that this bound is tight up to factors of n. Observe that the dominant terms are of the order $T_2/2^n$, $T_1T_2/2^{2n}$, and $T_1^2/2^{3n}$. We briefly describe how we show this.

The tightness of $T_2/2^n$ follows easily – the online only attack simply makes T_2 queries with the given salt. The advantage of this attack is at least $(1-1/e)T_2/2^n$ for $T_2 \leqslant 2^n$, as argued in the tightness discussion for pre-image-resistance of the salted random oracle.

The following theorem proves that the term $T_1T_2/2^{2n}$ is tight.

Theorem 11.4.2. *Let $T_1, T_2, n, \ell \in \mathbb{N}_{>0}$ such that T_1 is a multiple of 2^n, $T_1T_2 \leqslant 2^{2n}$. Let $H_{n,\ell,n}$ be the uniform distribution over $\mathsf{Fcs}(\{0,1\}^n \times \{0,1\}^\ell, \{0,1\}^n)$. Then there exists a (T_1, T_2)-adversary such that*

$$\mathsf{Adv}^{\text{2-PR-MD}}_{H_{n,\ell,n}}(\mathcal{A}) \geqslant \frac{T_1T_2(1-2/e)}{2^{2n+1}} .$$

We defer the proof of this theorem to Section 11.3.3.

The following theorem proves that the term $T_1^2/2^{3n}$ is tight.

Theorem 11.4.3. *Let $T_1, n, \ell \in \mathbb{N}_{>0}$ such that T_1 is a multiple of 2^{n+1}, $T_1 \leqslant 2^{3n/2}$. Let $H_{n,\ell,n}$ be the uniform distribution over $\mathsf{Fcs}(\{0,1\}^n \times \{0,1\}^\ell, \{0,1\}^n)$. Then there exists a $(T_1, 0)$-adversary such that*

$$\mathsf{Adv}^{\text{2-PR-MD}}_{H_{n,\ell,n}}(\mathcal{A}) \geqslant \frac{T_1^2(1-2/e)}{2^{3n+4}} .$$

We defer the proof of this theorem to Section 11.3.3.

Proof. [Theorem 11.4.1] Let $\mathcal{A}$ be the (T_1, T_2)-adversary that maximizes $\mathsf{Adv}^{\text{2-PR-MD}}_{H_{n,\ell,n}}(T_1, T_2)$. Without loss of generality $\mathcal{A}$ is deterministic and does not repeat any queries. We also assume without loss of generality that $\mathcal{A}$ makes all the queries needed to compute the MD evaluation of the messages it outputs.

We can formulate an alternate version of the game for pre-image-resistance of MD in game H in Fig. 11.4. Note that whenever $\mathcal{A}$ wins $\text{2-PR-MD}^{\text{h}}_n$, it has to win H because from our assumption that $\mathcal{A}$ makes all the queries needed to compute the MD evaluation of the messages it outputs, it follows that at least one of the following happens.

- $\mathcal{A}$ makes a query $\text{h}(a, M') = y$ – meaning it has found a one-block message M' whose MD evaluation with salt a is y.

- $\mathcal{A}$ queries $\text{h}(a, M') = z$ and $\text{h}(z, M'') = y$ – meaning it has found a two-block message (M', M'') whose MD evaluation with salt a is y.

In either of these cases the flag win is set in H, meaning $\mathcal{A}$ wins the game. Therefore,

$$\mathsf{Adv}^{\text{2-PR-MD}}_{H_{n,\ell,n}}(\mathcal{A}) \leqslant \mathsf{Pr}\left[\mathsf{H}(\mathcal{A})\right] .$$

Note that win is set to true H only if one of $\mathsf{oneblkinv}, \mathsf{twoblkinv}$ is set to true. It follows that

$$\mathsf{Pr}\left[\mathsf{H}(\mathcal{A})\right] = \mathsf{Pr}\left[\mathcal{A} \text{ sets } \mathsf{win}\right] \leqslant \mathsf{Pr}\left[\mathsf{oneblkinv}\right] + \mathsf{Pr}\left[\mathsf{twoblkinv}\right] .$$

Game $\mathsf{H}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2))$

$\mathrm{h} \xleftarrow{\$} \mathsf{Fcs}(\{0,1\}^n \times \{0,1\}^\ell, \{0,1\}^n)$

win $\leftarrow$ false

oneblkinv, twoblkinv $\leftarrow$ false

$\tau \leftarrow []$, $a, y \leftarrow \perp$

st $\leftarrow \mathcal{A}_1^{\mathrm{H}}$

$a \xleftarrow{\$} \{0,1\}^n$

$y \xleftarrow{\$} \{0,1\}^n$

If $\exists M : ((a, M), y) \in \tau$:

  win $\leftarrow$ true

  oneblkinv $\leftarrow$ true

If $\exists M, M', z$:

  $((a, M), z), ((z, M'), y) \in \tau$:

  win $\leftarrow$ true

  twoblkinv $\leftarrow$ true

$\mathcal{A}_2^{\mathrm{H}}(\mathsf{st}, a, y)$

Return win

Oracle $\mathrm{H}(a', M')$

$\tau \leftarrow \tau \cup \{((a', M'), \mathrm{h}(a', M'))\}$

If $\exists M' : ((a, M'), y) \in \tau$:

  win $\leftarrow$ true

  oneblkinv $\leftarrow$ true

If $\exists M', M'', z$:

  $((a, M'), z), ((z, M''), y) \in \tau$:

  win $\leftarrow$ true

  twoblkinv $\leftarrow$ true

Return $\mathrm{h}(a', M')$

Figure 11.4: Game H used in the analysis of pre-image-resistance of two-block MD. The events introduced in this game are marked in red.

We show that

$$\mathsf{Pr}\left[\mathsf{oneblkinv}\right] \leqslant \frac{T_2}{2^n} + \frac{T_1}{2^{2n}} \ ,$$

and

$$\mathsf{Pr}\left[\mathsf{twoblkinv}\right] \leqslant \frac{T_2+1}{2^n} + \frac{T_1T_2+nT_1}{2^{2n}} + \frac{4eT_1^2}{2^{3n}} \ .$$

Putting it all together would give us the theorem.

We next upper bound $\mathsf{Pr}\left[\mathsf{oneblkinv}\right], \mathsf{Pr}\left[\mathsf{twoblkinv}\right]$.

Towards upper bounding $\mathsf{Pr}\left[\mathsf{oneblkinv}\right]$, we define the two following events:

1. $\mathsf{oneblkinv}_{\mathsf{off}}$: $\mathcal{A}_1$ makes a query with input salt a and output y

2. $\mathsf{oneblkinv}_{\mathsf{on}}$: $\mathcal{A}_2$ makes a query which has output y

It is easy to see that if $\mathsf{oneblkinv}$ happens, then at least one of $\mathsf{oneblkinv}_{\mathsf{off}}$, $\mathsf{oneblkinv}_{\mathsf{on}}$ has to happen. Therefore

$$\mathsf{Pr}\left[\mathsf{oneblkinv}\right] \leqslant \mathsf{Pr}\left[\mathsf{oneblkinv}_{\mathsf{off}}\right] + \mathsf{Pr}\left[\mathsf{oneblkinv}_{\mathsf{on}}\right] \ .$$

We first upper bound $\mathsf{Pr}\left[\mathsf{oneblkinv}_{\mathsf{on}}\right]$. Each query by $\mathcal{A}_2$ has answer y with probability $1/2^n$. Using a union bound over all queries of $\mathcal{A}_2$, we have that

$$\mathsf{Pr}\left[\mathsf{oneblkinv}_{\mathsf{on}}\right] \leqslant T_2/2^n \ .$$

We next upper bound $\mathsf{Pr}\left[\mathsf{oneblkinv}_{\mathsf{off}}\right]$. Consider the set of (s, y) pairs such that there is a query by $\mathcal{A}_1$ with input salt s and answer y. There are at most T_1 such pairs. Note that, $\mathsf{oneblkinv}_{\mathsf{off}}$ happens only if (a, y) which is sampled uniformly at random, is among those at most T_1 pairs. Hence,

$$\mathsf{Pr}\left[\mathsf{oneblkinv}_{\mathsf{off}}\right] \leqslant T_1/2^{2n} \ .$$

Putting this together, we have the required bound on $\mathsf{Pr}\left[\mathsf{oneblkinv}\right]$.

We next upper bound $\mathsf{Pr}\left[\mathsf{twoblkinv}\right]$. We define the three following events.

1. $\mathsf{twoblkinv}_{\mathsf{off}}$: $\mathcal{A}_1$ makes queries $\mathrm{h}(a, M) = z$ and $\mathrm{h}(z, M') = y$ for some M, M', z

2. $\mathsf{twoblkinv}_{\mathsf{offon}}$: $\mathcal{A}_1$ makes a query $\mathrm{h}(z, M) = y$ and $\mathcal{A}_2$ makes a query $\mathrm{h}(a, M') = z$ for some M, M', z

3. $\mathsf{twoblkinv}_{\mathsf{on}}$: $\mathcal{A}_2$ makes a query with answer y

It is easy to see that if $\mathsf{twoblkinv}$ happens then at least one of $\mathsf{twoblkinv_{off}}$, $\mathsf{twoblkinv_{offon}}$, $\mathsf{twoblkinv_{on}}$ has to happen. Therefore,

$$\mathsf{Pr}\left[\mathsf{twoblkinv}\right] \leqslant \mathsf{Pr}\left[\mathsf{twoblkinv_{off}}\right] + \mathsf{Pr}\left[\mathsf{twoblkinv_{offon}}\right] + \mathsf{Pr}\left[\mathsf{twoblkinv_{on}}\right] \ . \tag{11.9}$$

We upper bound these probabilities one by one starting with $\mathsf{Pr}\left[\mathsf{twoblkinv_{on}}\right]$. Observe that every query by $\mathcal{A}_2$ has probability $1/2^n$ of having answer y. Using a union bound over all queries of $\mathcal{A}_2$, it follows that

$$\mathsf{Pr}\left[\mathsf{twoblkinv_{on}}\right] \leqslant \frac{T_2}{2^n} \ . \tag{11.10}$$

We next upper bound $\mathsf{Pr}\left[\mathsf{twoblkinv_{offon}}\right]$. Observe that this event happens only if $\mathcal{A}_2$ makes a query that has answer z such that $\mathcal{A}_1$ made a query with salt z that had answer y. Therefore, using total probability

$$\begin{aligned}
\mathsf{Pr}\left[\mathsf{twoblkinv_{offon}}\right] &\leqslant \sum_{k=1}^{T_1} \mathsf{Pr}\left[\mathcal{A}_1 \text{ made } k \text{ queries with answer } y\right] \frac{kT_2}{2^n} \\
&= \frac{T_2}{2^n} \sum_{k=1}^{T_1} \mathsf{E}\left[\text{Number of queries of } \mathcal{A}_1 \text{ with answer } y\right] \\
&= \frac{T_2 T_1}{2^n} \ .
\end{aligned} \tag{11.11}$$

The last equality follows since each query of $\mathcal{A}_1$ has answer y with probability $1/2^n$.

Finally, we upper bound $\mathsf{Pr}\left[\mathsf{twoblkinv_{off}}\right]$. For this we initially take a short detour and define the event $(m+1)$-col as the event that $\mathcal{A}_1$ has made $m+1$ distinct random oracle queries, all of which have the same answer. We claim that

$$\mathsf{Pr}\left[\mathsf{twoblkinv_{off}} \,\middle|\, \neg(m+1)\text{-}\mathsf{col}\right] \leqslant \frac{mT_1}{2^{2n}} \ .$$

This is because if $(m+1)$-col does not happen, there can be at most mT_1 pairs of queries such that the answer of the one query of the pair is the input of the other query (as otherwise there would be a $m+1$-multi-collision since there are T_1 queries made by $\mathcal{A}_1$). Now, $\mathsf{twoblkinv_{off}}$ happens only if (a, y) that is sampled uniformly at random is such that one of these at most mT_1 pairs have a as the salt for one query and y as the answer of the other query. This happens with probability at most $\frac{mT_1}{2^{2n}}$.

Finally, we upper bound $\mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right]$. For any subset of $m+1$ queries made by $\mathcal{A}_1$, the probability that they have the same answer is $1/2^{nm}$. Using a union bound over all possible $m+1$ sized subsets of the queries of $\mathcal{A}_1$, we have that

$$\mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \leqslant \frac{\binom{T_1}{m+1}}{2^{nm}} \ .$$

We let $m = \max\left(n, \frac{4eT_1}{2^n}\right)$. If $n \leqslant \frac{4eT_1}{2^n}$, we have that $m = \frac{4eT_1}{2^n} \geqslant n$. Therefore,

$$\mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \leqslant \frac{\binom{T_1}{m+1}}{2^{mn}} \leqslant \left(\frac{eT_1}{(m+1)2^n}\right)^{m+1} 2^n \leqslant \left(\frac{1}{4}\right)^m 2^n \leqslant \left(\frac{1}{2}\right)^n \ .$$

Otherwise, if $n > \frac{4eT_1}{2^n}$, we have that $m = n > \frac{4eT_1}{2^n}$. Therefore,

$$\mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \leqslant \frac{\binom{T_1}{m+1}}{2^{nm}} \leqslant \left(\frac{eT_1}{(m+1)2^n}\right)^{m+1} 2^n \leqslant \left(\frac{1}{4}\right)^{m+1} 2^n \leqslant \left(\frac{1}{2}\right)^n .$$

Therefore, we have that for $m = \max\left(n, \frac{4eT_1}{2^n}\right)$, $\mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \leqslant 1/2^n$. Hence

$$\mathsf{Pr}\left[\mathsf{twoblkinv}_{\mathsf{off}}\right] \leqslant \frac{nT_1}{2^{2n}} + \frac{4eT_1^2}{2^{3n}} + \frac{1}{2^n} . \tag{11.12}$$

Plugging (11.10) to (11.12) into (11.9) gives us the required bound for $\mathsf{Pr}\left[\mathsf{twoblkinv}\right]$ and concludes the proof. □ ■

11.4.2 Collision-resistance of two-block Merkle-Damgård

In this section, we study the collision-resistance of two-block Merkle-Damgård (MD) against offline-online adversaries. Collision-resistance of two-block MD is formalized by the oracle game $\mathsf{2\text{-}CR\text{-}MD}^{\mathrm{h}}_n$ in Fig. 11.5. In this game a salt a is picked at random from $\{0,1\}^n$ that is given to the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ has oracle access to h, and wins if it can output two messages M, M' that are distinct; both at most 2 blocks long, and satisfy $\mathsf{MD}^{\mathrm{h}}(a, M) = \mathsf{MD}^{\mathrm{h}}(a, M')$.

The game $\mathsf{pre\text{-}2\text{-}CR\text{-}MD}^{\mathrm{h}}_n$ captures the collision-resistance of 2-block MD against offline-online adversaries. We prove the following upper bound on $\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}CR\text{-}MD},n}_{H_{n,\ell,n}}$.

Theorem 11.4.4. *Let $T_1, T_2, s, \ell, n \in \mathbb{N}_{>0}$. Let $H_{n,\ell,n}$ be the uniform distribution on $\mathsf{Fcs}(\{0,1\}^n \times \{0,1\}^\ell, \{0,1\}^n)$.*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}CR\text{-}MD}}_{H_{s,\ell,n}}(\mathcal{A}) \leqslant\ & \frac{2T_2^2 + nT_2/2 + 3n^2/2 + 99n/2 + 33}{2^n} \\ & + \left(\frac{T_1T_2}{2^{3n/2}}\right)(n^2 + 5n + 83) \\ & + \left(\frac{T_1}{2^{5n/4}}\right)(53n + 14n^{1/2} + 56n^{1/3} + 342) + 468\left(\frac{T_1^2}{2^{7n/3}}\right) .\end{aligned}$$

Offline-online tradeoffs. Note that, in the regime of parameters $T_1 = 2^{n(1+\varepsilon)}, T_2 = 2^{n(1/2-\varepsilon)}$ for $0 < \varepsilon < 1/6$, the term $T_1T_2/2^{3n/2}$ dominates the bound, i.e., there is a tradeoff between the number of offline and online queries in that regime.

Tightness of the bound. We show that the first three terms in the above bound are tight by giving matching attacks. We could not find an attack matching the last term in the bound and leave improving it or showing it tight to be future research.

We briefly describe how we show the other terms to be tight. The first term is dominated by $T_2^2/2^n$ – we can show that this is tight up to constant factors using the birthday attack, which achieves advantage of the order $T_2^2/2^n$.

In the second term, ignoring constants and powers of n, the dominant factor is $\frac{T_1T_2}{2^{3n/2}}$. We prove this theorem to show that it is tight.

Game 2-CR-MD$^{\mathrm{h}}_n(\mathcal{A})$

$a \xleftarrow{\$} \{0,1\}^n$

$(M, M') \leftarrow \mathcal{A}^{\mathrm{h}}(a)$

If $|M|, |M'| \in \{\ell, 2\ell\}$ and $M \neq M'$ and $\mathsf{MD}^{\mathrm{h}}(a, M) = \mathsf{MD}^{\mathrm{h}}(a, M')$

  Return true

Return false

Figure 11.5: Oracle game 2-CR-MD$^{\mathrm{h}}_n$ formalizing collision-resistance of two-block MD.

Theorem 11.4.5. *Let $T_1, T_2, n, \ell \in \mathbb{N}_{>0}$ such that n is a multiple of 2, T_1 is a multiple of $2^{n/2+1}$, and $T_1 T_2 \leqslant 2^{3n/2}$. Let $H_{n,\ell,n}$ be the uniform distribution over $\mathsf{Fcs}(\{0,1\}^n \times \{0,1\}^\ell, \{0,1\}^n)$. There exists a (T_1, T_2) adversary $\mathcal{A}$ such that*

$$\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}CR\text{-}MD}}_{H_{n,\ell,n}}(\mathcal{A}) \geqslant \frac{(1-2/e)T_1T_2}{2^{3n/2+3}} .$$

The proof of this theorem is in Section 11.3.3.

In the third term, ignoring constants and powers of n, the dominant factor is $\frac{T_1}{2^{5n/4}}$. We give an attack that achieves advantage of the order $\frac{T_1^2}{2^{5n/2}}$. While $\frac{T_1^2}{2^{5n/2}} \leqslant \frac{T_1}{2^{5n/4}}$ for $T_1 \leqslant 2^{5n/4}$, observe that both of them become one at $T_1 = 2^{5n/4}$. Formally, we prove the following theorem.

Theorem 11.4.6. *Let $T_1, T_2, n, \ell \in \mathbb{N}_{>0}$ such that n is a multiple of 2, and T_1 is a multiple of $2^{n/2+1}$. Let $H_{n,\ell,n}$ be the uniform distribution over $\mathsf{Fcs}(\{0,1\}^n \times \{0,1\}^\ell, \{0,1\}^n)$. There exists a (T_1, T_2) adversary $\mathcal{A}$ such that*

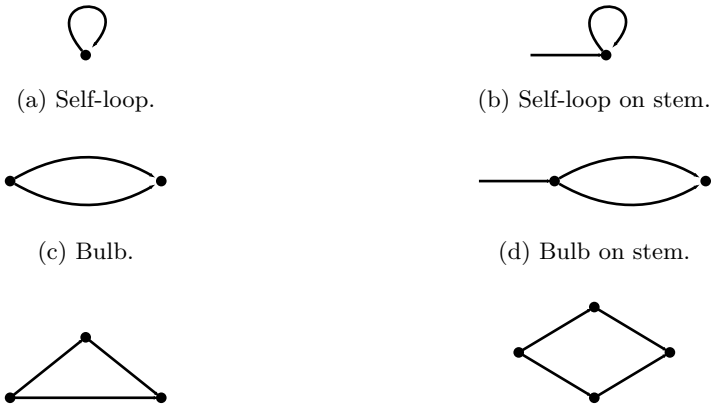
$$\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}CR\text{-}MD}}_{H_{n,\ell,n}}(\mathcal{A}) \geqslant \frac{(1-2/e)T_1^2}{2^{5n/2+6}} .$$

The proof of this theorem is in Section 11.3.3.

We now proceed to prove Theorem 11.4.4.

Proof. The proof of this theorem fixes the (T_1, T_2)-offline-online adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that maximizes $\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}CR\text{-}MD}}_{H_{n,\ell,n}}(T_1, T_2)$. We can treat $\mathcal{A}$ as deterministic by fixing its randomness that maximizes its advantage. Without loss of generality we can assume that $\mathcal{A}$ does not repeat any query across the offline and online phases because we have no restrictions on the memory of the adversary.

We rewrite the collision-resistance game for two-block MD in game H in Figure 11.7. Note that whenever $\mathcal{A}$ wins $\mathsf{G}^{\mathsf{2\text{-}PR\text{-}MD}}_{n,\ell}$, from our assumption that $\mathcal{A}$ makes all the queries needed to compute the MD evaluation of the messages it outputs, at least one of the following happens.



(a) Self-loop.

(b) Self-loop on stem.

(c) Bulb.

(d) Bulb on stem.

(e) Triangle.

(f) Diamond.

Figure 11.6: The structure of the six different types of two-block MD collisions in the query graph. The nodes in the query graph are labelled with values in $\{0,1\}^n$, and there is an edge (a, a') labelled with M if the adversary made a query $\mathrm{h}(a, M) = a'$. We omit the node and edge labels for simplicity.

- $\mathcal{A}$ makes a query $\mathrm{h}(a, M') = a$, meaning it has found a one-block message M' whose MD evaluation with salt a is a. This is sufficient for a two-block collision because for any $M'' \in \{0,1\}^\ell$, (M', M'') and M'' have the same MD evaluation with salt a.

- $\mathcal{A}$ makes queries $\mathrm{h}(a, M') = z$ and $\mathrm{h}(z, M'') = z$; this is a two-block collision because (M', M'') and M' have the same MD evaluation with salt a.

- $\mathcal{A}$ makes queries $\mathrm{h}(a, M') = z$ and $\mathrm{h}(a, M'') = z$ for $M' \neq M''$; this is a two-block collision because M', and M'' have the same MD evaluation with salt a.

- $\mathcal{A}$ makes queries $\mathrm{h}(a, M') = z$, $\mathrm{h}(y, M'') = z$ and $\mathrm{h}(y, M''') = z$ for $M'' \neq M'''$; this is a two-block collision because (M', M'') and (M', M'') have the same MD evaluation with salt a.

- $\mathcal{A}$ makes queries $\mathrm{h}(a, M') = y$, $\mathrm{h}(y, M'') = z$ and $\mathrm{h}(a, M''') = z$; this is a two-block collision because (M', M'') and M''' have the same MD evaluation with salt a.

- $\mathcal{A}$ makes queries $\mathrm{h}(a, M') = y$, $\mathrm{h}(y, M'') = z$, $\mathrm{h}(a, M''') = y'$, and $\mathrm{h}(y', M''') = z$ for $M' \neq M'''$ and $M'' \neq M''''$; this is a two-block collision because (M', M'') and (M''', M'''') have the same MD evaluation with salt a.

If any of these occur, win is set in H, meaning $\mathcal{A}$ wins the game. Therefore,

$$\mathsf{Adv}^{\text{pre-2-CR-MD}}_{H_{n,\ell,n}}(\mathcal{A}) \leqslant \Pr\left[\mathsf{H}(\mathcal{A})\right] .$$

The game H defines events sl, sos, bulb, bos, tri, dia. We name the events this way because of the following alternative view of MD collisions via the query graph of $\mathcal{A}$: the nodes of the query graph are labelled with strings from $\{0,1\}^n$, and whenever $\mathcal{A}$ makes a query $\mathrm{h}(a, M) = a'$, an edge (a, a') labelled M is added to the graph. Finding a two block collision can be viewed as finding one of the following structures in the query graph: self-loop, self-loop on stem, bulb, bulb-on-stem, triangle, and diamond; Fig. 11.6 shows these structures.

It follows from inspection that in game H, win is set only if one of these event among {sl, sos, bulb, bos, tri, dia} happen. Therefore, using the union bound we have that

$$\begin{aligned}\Pr\left[\mathsf{H}(\mathcal{A})\right] &= \Pr\left[\mathcal{A} \text{ sets win}\right] \\ &\leqslant \sum_{\text{event}\in\{\mathsf{sl},\mathsf{sos},\mathsf{bulb},\mathsf{bos},\mathsf{tri},\mathsf{dia}\}} \Pr\left[\text{event}\right] \qquad (11.13)\end{aligned}$$

Our proof is divided into these following lemmas each of which upper bound the probability of these events.

Lemma 11.4.7.

$$\Pr\left[\mathsf{sl}\right] \leqslant \frac{1}{2^n} + \frac{n}{2^n} + \frac{2eT_1}{2^{2n}} + \frac{T_2}{2^n} .$$

Lemma 11.4.8.

$$\Pr\left[\mathsf{sos}\right] \leqslant \frac{T_2 + 3 + nT_2 + n^2}{2^n} + \frac{2eT_1T_2 + 6enT_1}{2^{2n}} + \frac{8e^2T_1^2}{2^{3n}} .$$

Lemma 11.4.9.

$$\Pr\left[\mathsf{bulb}\right] \leqslant \frac{\binom{T_2}{2}}{2^n} + \frac{T_2T_1}{2^{2n}} + \frac{eT_1}{2^{3n/2}} + \frac{n}{2^{n+1}} + \frac{1}{2^n} .$$

Lemma 11.4.10.

$$\begin{aligned}\Pr\left[\mathsf{bos}\right] \leqslant &\frac{\binom{T_2}{2} + nT_2/2 + n^2/2 + 4}{2^n} + \frac{eT_1T_2 + enT_1}{2^{3n/2}} + \frac{nT_1T_2 + T_1T_2^2 + 2enT_1}{2^{2n}} \\ &+ \frac{4e^2T_1^2}{2^{5n/2}} + \frac{4eT_1^2T_2}{2^{3n}} .\end{aligned}$$

Lemma 11.4.11.

$$\begin{aligned}\Pr\left[\mathsf{tri}\right] \leqslant &\frac{\binom{T_2}{2} + 18n + 8}{2^n} + \frac{3(24e)^{1/2}T_1 + 3(8en)^{1/2}T_1}{2^{3n/2}} + \frac{9(2)^{1/3}eT_1^{4/3}}{2^{5n/3}} \\ &+ \frac{2nT_1T_2 + T_1T_2 + T_2^2T_1 + 12e(2)^{1/2}T_1^{3/2}}{2^{2n}} + \frac{8eT_1^2T_2 + 2T_1T_2}{2^{3n}} .\end{aligned}$$

Game $\mathsf{H}(\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2))$

$\mathrm{h} \xleftarrow{\$} \mathsf{Fcs}(\{0,1\}^n \times \{0,1\}^m, \{0,1\}^n)$

win $\leftarrow$ false, $\tau \leftarrow []$, $a \leftarrow \perp$

sl, sos, bulb, bos, tri, dia $\leftarrow$ false

st $\leftarrow \mathcal{A}_1^{\mathrm{H}}$

$a \xleftarrow{\$} \{0,1\}^n$

If $\exists M' : ((a, M'), a) \in \tau$:

  win $\leftarrow$ true, sl $\leftarrow$ true

If $\exists M', M'', z :$

  $((a, M'), z), ((z, M''), z) \in \tau$:

  win $\leftarrow$ true, sos $\leftarrow$ true

If $\exists M' \neq M'', z :$

  $((a, M'), z), ((a, M''), z) \in \tau$:

  win $\leftarrow$ true, bulb $\leftarrow$ true

If $\exists M', M'' \neq M''', y, z :$

  $((a, M'), y), ((y, M''), z) \in \tau,$

  $((y, M'''), z) \in \tau$:

  win $\leftarrow$ true, bos $\leftarrow$ true

If $\exists M', M'', M''', y, z :$

  $((a, M'), y), ((y, M''), z) \in \tau,$

  $((a, M'''), z) \in \tau$:

  win $\leftarrow$ true, tri $\leftarrow$ true

If $\exists M' \neq M''', M'' \neq M'''', x, y, z :$

  $((a, M'), x), ((a, M''), y) \in \tau,$

  $((x, M'''), z), ((y, M''''), z) \in \tau$:

  win $\leftarrow$ true, dia $\leftarrow$ true

$\mathcal{A}_2^{\mathrm{H}}(\mathsf{st}, a)$

Return win

Oracle $\mathrm{H}(a, M)$

$\tau \leftarrow \tau \cup \{((a, M), \mathrm{h}(a, M))\}$

If $\exists M' : ((a, M'), a) \in \tau$:

  win $\leftarrow$ true, sl $\leftarrow$ true

If $\exists M', M'', z :$

  $((a, M'), z), ((z, M''), z) \in \tau$:

  win $\leftarrow$ true, sos $\leftarrow$ true

If $\exists M' \neq M'', z :$

  $((a, M'), z), ((a, M''), z) \in \tau$:

  win $\leftarrow$ true, bulb $\leftarrow$ true

If $\exists M', M'' \neq M''', y, z :$

  $((a, M'), y), ((y, M''), z) \in \tau,$

  $((y, M'''), z) \in \tau$:

  win $\leftarrow$ true, bos $\leftarrow$ true

If $\exists M', M'', M''', y, z :$

  $((a, M'), y), ((y, M''), z) \in \tau,$

  $((a, M'''), z) \in \tau$:

  win $\leftarrow$ true, tri $\leftarrow$ true

If $\exists M' \neq M''', M'' \neq M'''', x, y, z :$

  $((a, M'), x), ((a, M''), y) \in \tau,$

  $((x, M'''), z), ((y, M''''), z) \in \tau$:

  win $\leftarrow$ true, dia $\leftarrow$ true

$\mathcal{A}_2^{\mathrm{H}}(\mathsf{st}, a)$

Return $\mathrm{h}(a, M)$

Figure 11.7: Game H used in the analysis of collision-resistance of two-block MD against offline-online adversaries. The events introduced in this game are marked in red.

Lemma 11.4.12.

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{dia}\right] \leqslant & \frac{\binom{T_2}{2} + 30n + 16}{2^n} + \frac{4eT_1^2T_2 + 2T_1T_2 + 4eT_1^2T_2^2}{2^{3n}} + \\ & \frac{nT_1T_2 + T_2^2T_1 + n^2T_1T_2 + nT_1T_2^2}{2^{2n}} + \frac{4eT_1^3T_2}{2^{4n}} \\ & + \frac{40(en)^{1/3}T_1}{2^{4n/3}} + \frac{(8e)^{1/2}nT_1 + 10(2e)^{1/2}nT_1}{2^{3n/2}} \\ & + \frac{240e^{2/3}T_1^2}{2^{7n/3}} + \frac{8(2)^{1/2}e^{3/2}T_1^2 + 40(2e)^{1/2}T_1^2}{2^{5n/2}} \ .\end{aligned}$$

Plugging in Lemmas 11.4.7 to 11.4.12 into (11.13) and grouping the terms, we get that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}CR\text{-}MD}}_{H_{s,\ell,n}}(\mathcal{A}) \leqslant & \frac{33 + 99n/2 + 2T_2 + 3nT_2/2 + 3n^2/2 + 4\binom{T_2}{2}}{2^n} \\ & + \frac{(2e + 8ne)T_1 + (3+n)T_1T_2^2 + 12e(2)^{1/2}T_1^{3/2}}{2^{2n}} \\ & + \frac{(2 + 2e + 4n + n^2)T_1T_2}{2^{2n}} + \frac{eT_1T_2}{2^{3n/2}} \\ & + \frac{(e + en + 3(24e)^{1/2} + 3(8en)^{1/2} + (8e)^{1/2}n + 10(2e)^{1/2}n)T_1}{2^{3n/2}} \\ & + \frac{8e^2T_1^2 + 16eT_1^2T_2 + 4T_1T_2 + 4eT_1^2T_2^2}{2^{3n}} \\ & + \frac{(4e^2 + 8(2)^{1/2}e^{3/2} + 40(2e)^{1/2})T_1^2}{2^{5n/2}} \\ & + \frac{9(2)^{1/3}eT_1^{4/3}}{2^{5n/3}} + \frac{4eT_1^3T_2}{2^{4n}} + \frac{40(en)^{1/3}T_1}{2^{4n/3}} + \frac{240e^{2/3}T_1^2}{2^{7n/3}} \ . \end{aligned} \tag{11.14}$$

Note that this bound trivially holds when $T_2 > 2^{n/2}$ because in that case $T_2^2/2^n > 1$ and the left hand side is a probability which is at most 1. Similarly, it trivially holds when $T_2T_1 > 2^{3n/2}$ or $T_1 > 2^{5n/4}$. Therefore, we can assume $T_2 \leqslant 2^{n/2}$, $T_1 \leqslant 2^{5n/4}$, and $T_1T_2 \leqslant 2^{3n/2}$. For $T_1 \leqslant 2^{5n/4}$, we have the following inequalities.

$$\frac{T_1^{3/2}}{2^{2n}}, \frac{T_1^2}{2^{3n}}, \frac{T_1^2}{2^{5n/2}}, \frac{T_1^{4/3}}{2^{5n/3}}, \frac{T_1^2T_2}{2^{3n}}, \frac{T_1^3T_2}{2^{4n}} \leqslant \frac{T_1}{2^{5n/4}}$$

For $T_2 \leqslant 2^{n/2}$, we have the following inequalities.

$$\frac{T_1T_2^2}{2^{2n}}, \frac{T_1T_2^2}{2^{2n}} \leqslant \frac{T_1T_2}{2^{3n/2}}$$

For $T_1T_2 \leqslant 2^{3n/2}$ we have that $T_1^2T_2^2/2^{3n} \leqslant T_1T_2/2^{3n/2}$. Further for any $T_1, T_2 \geqslant 0$, we have the following inequalities.

$$\begin{aligned}&\frac{T_1}{2^{2n}}, \frac{T_1}{2^{3n/2}}, \frac{T_1}{2^{4n/3}} \leqslant \frac{T_1}{2^{5n/4}} \\ &\frac{T_1T_2}{2^{2n}}, \frac{T_1T_2}{2^{3n}} \leqslant \frac{T_1T_2}{2^{3n/2}}\end{aligned}$$

Using all of these inequalities in (11.14), we have that

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}CR\text{-}MD}}_{H_{s,\ell,n}}(\mathcal{A}) \leqslant & \frac{33 + 99n/2 + 2T_2 + 3nT_2/2 + 3n^2/2 + 4\binom{T_2}{2}}{2^n} \\
& + \frac{((2e + 8ne) + 12e(2)^{1/2})T_1}{2^{5n/4}} + \frac{(3+n)T_1T_2}{2^{3n/2}} \\
& + \frac{(2 + 2e + 4n + n^2)T_1T_2}{2^{3n/2}} + \frac{eT_1T_2}{2^{3n/2}} \\
& + \frac{(e + en + 3(24e)^{1/2} + 3(8en)^{1/2} + (8e)^{1/2}n + 10(2e)^{1/2}n)T_1}{2^{5n/4}} \\
& + \frac{8e^2T_1}{2^{5n/4}} + \frac{(16e + 4 + 4e)T_1T_2}{2^{3n/2}} \\
& + \frac{(4e^2 + 8(2)^{1/2}e^{3/2} + 40(2e)^{1/2})T_1}{2^{5n/4}} \\
& + \frac{9(2)^{1/3}eT_1}{2^{5n/4}} + \frac{4eT_1T_2}{2^{3n/2}} + \frac{40(en)^{1/3}T_1}{2^{5n/4}} + \frac{240e^{2/3}T_1^2}{2^{7n/3}} \,.
\end{aligned}$$

Consolidating terms we get that

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}CR\text{-}MD}}_{H_{s,\ell,n}}(\mathcal{A}) \leqslant & \frac{2T_2^2 + nT_2/2 + 3n^2/2 + 99n/2 + 33}{2^n} \\
& + \left(\frac{T_1T_2}{2^{3n/2}}\right)(n^2 + 5n + 27e + 9) \\
& + \left(\frac{T_1}{2^{5n/4}}\right)(n(9e + (8e)^{1/2} + 10(2e)^{1/2}) + n^{1/2}(3(8e)^{1/2}) \\
& + n^{1/3}(40e^{1/3}) + (3e + 12e(2)^{1/2} + 12e^2 + 8(2)^{1/2}e^{3/2} + 40(2e)^{1/2} \\
& + 9(2)^{1/3}e + 3(24e)^{1/2}) + \frac{240e^{2/3}T_1^2}{2^{7n/3}} \,.
\end{aligned}$$

We observe that the following inequalities hold

$$\begin{aligned}
& 27e + 9 \leqslant 83 \\
& 9e + (8e)^{1/2} + 10(2e)^{1/2} \leqslant 53 \\
& 3(8e)^{1/2} \leqslant 14 \\
& 40e^{1/3} \leqslant 56 \\
& 3e + 12e(2)^{1/2} + 12e^2 + 8(2)^{1/2}e^{3/2} + 40(2e)^{1/2} + 9(2)^{1/3}e + 3(24e)^{1/2} \leqslant 342 \\
& 240e^{2/3} \leqslant 468
\end{aligned}$$

Using this we have that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{pre\text{-}2\text{-}CR\text{-}MD}}_{H_{s,\ell,n}}(\mathcal{A}) \leqslant & \frac{2T_2^2 + nT_2/2 + 3n^2/2 + 99n/2 + 33}{2^n} \\ & + \left(\frac{T_1 T_2}{2^{3n/2}}\right)(n^2 + 5n + 83) \\ & + \left(\frac{T_1}{2^{5n/4}}\right)(53n + 14n^{1/2} + 56n^{1/3} + 342) + 468\left(\frac{T_1^2}{2^{7n/3}}\right) .\end{aligned}$$

This proves the required bound.

We prove Lemmas 11.4.7 to 11.4.12 in Sections 11.4.3 to 11.4.8, respectively. □ ■

11.4.3 Proof of Lemma 11.4.7

Proof. [Lemma 11.4.7] We define the two following events.

1. $\mathsf{sl}_{\mathsf{off}}$: $\mathcal{A}_1$ made a query $\mathrm{h}(a, M) = a$
2. $\mathsf{sl}_{\mathsf{on}}$: $\mathcal{A}_2$ made a query $\mathrm{h}(a, M) = a$

Notice that sl happens only if at least one of $\mathsf{sl}_{\mathsf{off}}$ or $\mathsf{sl}_{\mathsf{on}}$ happens. Therefore, we have that

$$\mathsf{Pr}\left[\mathsf{sl}\right] \leqslant \mathsf{Pr}\left[\mathsf{sl}_{\mathsf{off}}\right] + \mathsf{Pr}\left[\mathsf{sl}_{\mathsf{on}}\right] . \tag{11.15}$$

We first prove an upper bound on $\mathsf{Pr}\left[\mathsf{sl}_{\mathsf{on}}\right]$. Note that, for every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_2$, the probability that its answer is a is $1/2^n$. Therefore, using a union bound over all the queries of $\mathcal{A}_2$, we have that

$$\mathsf{Pr}\left[\mathsf{sl}_{\mathsf{on}}\right] \leqslant T_2/2^n .$$

To prove an upper bound on $\mathsf{Pr}\left[\mathsf{sl}_{\mathsf{off}}\right]$, we define the following event $\mathsf{off\text{-}sl}\text{-}k$: $\mathcal{A}_1$ makes at least k different queries such that the input salt of the query is the answer of the query. Using total probability, we have that for any k

$$\mathsf{Pr}\left[\mathsf{sl}_{\mathsf{off}}\right] \leqslant \mathsf{Pr}\left[\mathsf{sl}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}sl}\text{-}k\right] + \mathsf{Pr}\left[\mathsf{off\text{-}sl}\text{-}k\right] + \frac{k}{2^n} . \tag{11.16}$$

Note that, if the adversary $\mathcal{A}_1$ makes at most k different queries, such that the input salt of the query is the answer, $\mathsf{sl}_{\mathsf{off}}$ happens only if the salt a that is sampled uniformly at random is same as the salt for one of those at most k queries. Therefore, $\mathsf{Pr}\left[\mathsf{sl}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}sl}\text{-}k\right] \leqslant k/2^n$.

We upper bound $\mathsf{Pr}\left[\mathsf{off\text{-}sl}\text{-}k\right]$ as follows. Let B_j be the indicator random variable that indicates whether the j-th query of $\mathcal{A}_1$ is such that its answer is same as its input salt. Since $\mathcal{A}_1$ does not repeat queries, all the B_j's are independent, and $\mathsf{Pr}\left[B_j\right] = 1/2^n$. From the definition of B_j's, it follows that

$$\mathsf{Pr}\left[\mathsf{off\text{-}sl}\text{-}k\right] = \mathsf{Pr}\left[\sum_{j=1}^{T_1} B_j \geqslant k\right] .$$

We rewrite the term on the right as

$$\mathsf{Pr}\left[\sum_{j=1}^{T_1} B_j \geqslant k\right] = \mathsf{Pr}\left[\exists S \subseteq [T_1], |S| = k : \forall j \in S, B_j = 1\right] .$$

Using a union bound over all subsets of T_1 of size k, we have

$$\mathsf{Pr}\left[\exists S \subseteq [T_1], |S| = k : \forall j \in S, B_j = 1\right] \leqslant \sum_{S \subseteq [T_1], |S|=k} \mathsf{Pr}\left[\forall j \in S, B_j = 1\right] .$$

Since there are $\binom{T_1}{k}$ subsets of $[T_1]$ of size k, and all the B_j's are independent and $\mathsf{Pr}\left[B_j = 1\right] = 1/2^n$, we have that

$$\mathsf{Pr}\left[\exists S \subseteq [T_1], |S| = k : \forall j \in S, B_j = 1\right] \leqslant \frac{\binom{T_1}{k}}{2^{nk}} .$$

Therefore

$$\mathsf{Pr}\left[\mathsf{off\text{-}sl}\text{-}k\right] \leqslant \frac{\binom{T_1}{k}}{2^{nk}} \leqslant \left(\frac{eT_1}{k2^n}\right)^k .$$

Plugging this in (11.16),we have that for any k,

$$\mathsf{Pr}\left[\mathsf{sl}_{\mathsf{off}}\right] \leqslant \left(\frac{eT_1}{k2^n}\right)^k + \frac{k}{2^n} .$$

We let $k = \max\left(n, \frac{2eT_1}{2^n}\right)$. If $n \leqslant \frac{2eT_1}{2^n}$, we have that $k = \frac{2eT_1}{2^n} \geqslant n$. Therefore,

$$\frac{\binom{T_1}{k}}{2^{nk}} \leqslant \left(\frac{eT_1}{k2^n}\right)^k \leqslant \left(\frac{1}{2}\right)^k \leqslant \frac{1}{2^n} .$$

Otherwise if $n > \frac{2eT_1}{2^n}$, we have that $k = n > \frac{2eT_1}{2^n}$. Therefore,

$$\frac{\binom{T_1}{k}}{2^{nk}} \leqslant \left(\frac{eT_1}{k2^n}\right)^k \leqslant \left(\frac{1}{2}\right)^k = \frac{1}{2^n} .$$

Hence,

$$\mathsf{Pr}\left[\mathsf{sl}_{\mathsf{off}}\right] \leqslant \frac{1}{2^n} + \frac{n}{2^n} + \frac{2eT_1}{2^{2n}} .$$

Plugging this back into (11.15) gives us

$$\mathsf{Pr}\left[\mathsf{sl}\right] \leqslant \frac{1}{2^n} + \frac{n}{2^n} + \frac{2eT_1}{2^{2n}} + \frac{T_2}{2^n} .$$

□ ■

11.4.4 Proof of Lemma 11.4.8

Proof. [Lemma 11.4.8] We first define the three following events.

1. $\mathsf{sos}_{\mathsf{off}}$: $\mathcal{A}_1$ made queries $\mathrm{h}(a, M') = z$ and $\mathrm{h}(z, M'') = z$
2. $\mathsf{sos}_{\mathsf{offon}}$: $\mathcal{A}_1$ made a query $\mathrm{h}(y, M'') = y$ and $\mathcal{A}_2$ made a query $\mathrm{h}(a, M') = y$
3. $\mathsf{sos}_{\mathsf{on}}$: $\mathcal{A}_2$ made a query $\mathrm{h}(z, M') = z$, i.e., a query whose answer is the same as its input salt

From inspection one can verify that sos happens only if at least one of $\mathsf{sos}_{\mathsf{off}}$, $\mathsf{sos}_{\mathsf{on}}$, $\mathsf{sos}_{\mathsf{offon}}$ happen. It follows that

$$\mathsf{Pr}\left[\mathsf{sos}\right] \leqslant \mathsf{Pr}\left[\mathsf{sos}_{\mathsf{off}}\right] + \mathsf{Pr}\left[\mathsf{sos}_{\mathsf{on}}\right] + \mathsf{Pr}\left[\mathsf{sos}_{\mathsf{offon}}\right] \ . \tag{11.17}$$

We first upper bound $\mathsf{Pr}\left[\mathsf{sos}_{\mathsf{on}}\right]$. Note that for every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_2$, the probability that its answer is a is $1/2^n$. Therefore, using a union bound over all the queries of $\mathcal{A}_2$, we have that

$$\mathsf{Pr}\left[\mathsf{sos}_{\mathsf{on}}\right] \leqslant T_2/2^n \ . \tag{11.18}$$

We next upper bound $\mathsf{Pr}\left[\mathsf{sos}_{\mathsf{offon}}\right]$. Recall that the event $\mathsf{off\text{-}sl}\text{-}k$ defined in the proof of Lemma 11.4.7: $\mathcal{A}_1$ makes at least k different queries such that the input salt of the query is the answer. We have that

$$\mathsf{Pr}\left[\mathsf{sos}_{\mathsf{offon}}\right] \leqslant \mathsf{Pr}\left[\mathsf{sos}_{\mathsf{offon}} \,\middle|\, \neg\mathsf{off\text{-}sl}\text{-}k\right] + \mathsf{Pr}\left[\mathsf{off\text{-}sl}\text{-}k\right] \ .$$

In this case, $\mathsf{sos}_{\mathsf{offon}}$ happens only if $\mathcal{A}_2$ makes a query whose answer is the input salt of one of at most k such queries. Therefore, we have that for any k,

$$\mathsf{Pr}\left[\mathsf{sos}_{\mathsf{offon}}\right] \leqslant \mathsf{Pr}\left[\mathsf{off\text{-}sl}\text{-}k\right] + \frac{kT_2}{2^n} \ .$$

As seen in the proof of Lemma 11.4.7, setting $k = \max\left(n, \frac{2eT_1}{2^n}\right)$ makes $\mathsf{Pr}\left[\mathsf{off\text{-}sl}\text{-}k\right] \leqslant 1/2^n$. Therefore, by setting this value of k, we have that

$$\mathsf{Pr}\left[\mathsf{sos}_{\mathsf{offon}}\right] \leqslant \frac{nT_2}{2^n} + \frac{2eT_1T_2}{2^{2n}} + \frac{1}{2^n} \ . \tag{11.19}$$

We finally upper bound $\mathsf{Pr}\left[\mathsf{sos}_{\mathsf{off}}\right]$. For this we recall $(m+1)\text{-}\mathsf{col}$ as the event that we defined in the proof of Theorem 11.4.1. We say that $(m+1)\text{-}\mathsf{col}$ happens if the $\mathcal{A}_1$ has made $m+1$ distinct random oracle queries that all have the same answer. Using total probability, we have that for any k, m

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{sos}_{\mathsf{off}}\right] &\leqslant \mathsf{Pr}\left[\mathsf{sos}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}sl}\text{-}k \wedge \neg(m+1)\text{-}\mathsf{col}\right] + \mathsf{Pr}\left[\mathsf{off\text{-}sl}\text{-}k \vee (m+1)\text{-}\mathsf{col}\right] \\ &\leqslant \mathsf{Pr}\left[\mathsf{sos}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}sl}\text{-}k \wedge \neg(m+1)\text{-}\mathsf{col}\right] + \mathsf{Pr}\left[\mathsf{off\text{-}sl}\text{-}k\right] + \mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \ . \end{aligned} \tag{11.20}$$

We claim that

$$\mathsf{Pr}\left[\mathsf{sos}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}sl}\text{-}k \wedge \neg(m+1)\text{-}\mathsf{col}\right] \leqslant \frac{mk}{2^n} \ .$$

This is because if $\mathsf{off\text{-}sl}\text{-}k$ and $(m+1)\text{-}\mathsf{col}$ do not happen, there can be at most $k \cdot m$ salts a that satisfy that $\mathcal{A}_1$ makes a query $\mathrm{h}(a, M') = z$ and $\mathrm{h}(z, M'') = z$. The probability that the salt a that is sampled uniformly at random is among one of those at most $k \cdot m$ salts is at most $\frac{mk}{2^n}$.

From our calculations in the proof of Theorem 11.4.1, we have that for $m = \max\left(n, \frac{4eT_1}{2^n}\right)$, $\mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \leqslant 1/2^n$. We also know that for $k = \max\left(n, \frac{2eT_1}{2^n}\right)$, $\mathsf{Pr}\left[\mathsf{off\text{-}sl}\text{-}k\right] \leqslant 1/2^n$. We set m, k to these values and obtain from (11.20) that

$$\mathsf{Pr}\left[\mathsf{sos}_{\mathsf{off}}\right] \leqslant \frac{2}{2^n} + \frac{n^2}{2^n} + \frac{6enT_1}{2^{2n}} + \frac{8e^2T_1^2}{2^{3n}} \ . \tag{11.21}$$

This is because for $m = \max\left(n, \frac{4eT_1}{2^n}\right)$, $k = \max\left(n, \frac{2eT_1}{2^n}\right)$,

$$k \cdot m \leqslant n^2 + 6enT_1/2^n + 8e^2T_1^2/2^{2n} \ .$$

Plugging (11.18), (11.19) and (11.21) into (11.17), we get that

$$\mathsf{Pr}\left[\mathsf{sos}\right] \leqslant \frac{T_2 + 3 + nT_2 + n^2}{2^n} + \frac{2eT_1T_2 + 6enT_1}{2^{2n}} + \frac{8e^2T_1^2}{2^{3n}} \ .$$

□ ■

11.4.5 Proof of Lemma 11.4.9

Proof. We define the three following events.

1. $\mathsf{bulb}_{\mathsf{off}}$: $\mathcal{A}_1$ made queries $\mathrm{h}(a, M) = y$, $\mathrm{h}(a, M') = y$ for some $M \neq M'$ and y
2. $\mathsf{bulb}_{\mathsf{offon}}$: $\mathcal{A}_1$ made queries $\mathrm{h}(a, M) = y$, and $\mathcal{A}_2$ made a query with answer y for some M, y
3. $\mathsf{bulb}_{\mathsf{on}}$: $\mathcal{A}_2$ made queries $\mathrm{h}(a, M) = y$, $\mathrm{h}(a, M') = y$ for some $M \neq M'$ and y

Observe that bulb happens only if at least one of $\mathsf{bulb}_{\mathsf{off}}$, $\mathsf{bulb}_{\mathsf{offon}}$, $\mathsf{bulb}_{\mathsf{on}}$ happen. Therefore

$$\mathsf{Pr}\left[\mathsf{bulb}\right] \leqslant \mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{off}}\right] + \mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{offon}}\right] + \mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{on}}\right] \ . \tag{11.22}$$

The rest of the proof consists of upper bounding these probabilities one by one. We begin with $\mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{on}}\right]$. Observe that $\mathsf{bulb}_{\mathsf{on}}$ happens only if $\mathcal{A}_2$ makes two queries that have the same answer. The probability of

any two queries of $\mathcal{A}$ having the same answer is $1/2^n$. Using a union bound over all pairs of queries of $\mathcal{A}_2$, we have that

$$\mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{on}}\right] \leqslant \frac{\binom{T_2}{2}}{2^n} \ . \tag{11.23}$$

We next upper bound $\mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{offon}}\right]$. Let Q_a be the random variable denoting the number of queries $\mathcal{A}_1$ makes with salt a. Using total probability

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{offon}}\right] &= \sum_{i=1}^{T_1} \mathsf{Pr}\left[Q_a = k\right] \mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{offon}} \,\middle|\, Q_a = k\right] \\ &= \sum_{i=1}^{T_1} \mathsf{Pr}\left[Q_a = k\right] \cdot \frac{kT_2}{2^n} \\ &= \frac{T_2}{2^n} \cdot \mathsf{E}\left[Q_a\right] = \frac{T_1 T_2}{2^n} \ . \end{aligned} \tag{11.24}$$

The second equality above follows because if $\mathcal{A}_1$ makes k queries with salt a, the probability that $\mathsf{bulb}_{\mathsf{offon}}$ happens is at most $kT_2/2^n$ using a union bound over all queries of $\mathcal{A}_2$. The final equality uses the fact that $\mathsf{E}\left[Q_a\right] = T_1/2^n$, because $\mathcal{A}_1$ makes T_1 queries and a is sampled uniformly at random.

Finally, we upper bound $\mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{off}}\right]$. We define an event $\mathsf{off\text{-}bulbs}\text{-}k$ as follows: there is a set of at least k distinct salts $a_1, \ldots, a_k$ such that for each a_i, $\mathcal{A}_1$ has made a pair of queries $\mathrm{h}(a_i, M_i) = z$ and $\mathrm{h}(a_i, M_i') = z$ for $M_i \neq M_i'$.

We have that for any k,

$$\mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{off}}\right] \leqslant \mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}bulbs}\text{-}k\right] + \mathsf{Pr}\left[\mathsf{off\text{-}bulbs}\text{-}k\right] \ . \tag{11.25}$$

Since $\mathsf{bulb}_{\mathsf{off}}$ happens if for the salt a that is chosen uniformly at random from $\{0,1\}^n$, $\mathcal{A}_1$ had queried $\mathrm{h}(a, M), \mathrm{h}(a, M')$ that have the same answer, we have that $\mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}bulbs}\text{-}k\right] \leqslant k/2^n$. We upper bound $\mathsf{Pr}\left[\mathsf{off\text{-}bulbs}\text{-}k\right]$ using a compression argument.

Note that the event $\mathsf{off\text{-}bulbs}\text{-}k$ is similar to the event $\mathsf{off\text{-}oneblk}\text{-}k$ we defined in the proof of Theorem 11.4.1, the only difference being the salt length was s there and is n here. However, $\mathsf{Pr}\left[\mathsf{off\text{-}oneblk}\text{-}k\right]$ did not depend on s, hence we can prove the same bound for $\mathsf{Pr}\left[\mathsf{off\text{-}bulbs}\text{-}k\right]$. We showed in that proof that for $k = \max\left(\frac{eT_1}{2^{n/2}}, n/2\right)$, $\mathsf{Pr}\left[\mathsf{off\text{-}oneblk}\text{-}k\right] \leqslant 1/2^n$.

Hence from (11.25) we have

$$\mathsf{Pr}\left[\mathsf{bulb}_{\mathsf{off}}\right] \leqslant \frac{eT_1}{2^{3n/2}} + \frac{n}{2^{n+1}} + \frac{1}{2^n} \ . \tag{11.26}$$

Plugging (11.23) to (11.25) into (11.22) gives us that

$$\mathsf{Pr}\left[\mathsf{bulb}\right] \leqslant \frac{\binom{T_2}{2}}{2^n} + \frac{T_2 T_1}{2^{2n}} + \frac{eT_1}{2^{3n/2}} + \frac{n}{2^{n+1}} + \frac{1}{2^n} \ .$$

□ ■

11.4.6 Proof of Lemma 11.4.10

Proof. We define the five following events.

1. $\mathsf{bos}_{\mathsf{off}}$: $\mathcal{A}_1$ makes queries $\mathrm{h}(a, M') = y$, $\mathrm{h}(y, M'') = z$, $\mathrm{h}(y, M''') = z$ for some $M', M'' \neq M'''$ and y

2. $\mathsf{bos}_{\mathsf{offon},1}$: $\mathcal{A}_1$ makes queries $\mathrm{h}(a, M') = y$, $\mathrm{h}(y, M'') = z$ and $\mathcal{A}_2$ makes a query with answer z for some z.

3. $\mathsf{bos}_{\mathsf{offon},2}$: $\mathcal{A}_1$ makes a query $\mathrm{h}(y, M') = z$, and $\mathcal{A}_2$ makes a query with answer z and another with answer y, for some y, z, M'.

4. $\mathsf{bos}_{\mathsf{offon},3}$: $\mathcal{A}_1$ makes queries $\mathrm{h}(y, M') = z$, and $\mathrm{h}(y, M'') = z$ and $\mathcal{A}_2$ makes query with answer z for some $M' \neq M'', y, z$.

5. $\mathsf{bos}_{\mathsf{on}}$: $\mathcal{A}_2$ makes queries two queries that have the same answer.

Observe that bulb happens only if at least one of $\mathsf{bos}_{\mathsf{off}}$, $\mathsf{bos}_{\mathsf{offon},1}$, $\mathsf{bos}_{\mathsf{offon},2}$, $\mathsf{bos}_{\mathsf{offon},3}$, $\mathsf{bos}_{\mathsf{on}}$ happens. To see why this is true, observe that for bos to happen $\mathcal{A}$ has to make queries $q_1 = \mathrm{h}(a, M') = y$, $q_2 = \mathrm{h}(y, M'') = z$, $q_3 = \mathrm{h}(y, M''') = z$ for $M'' \neq M'''$, and some y, z. We show that all the possibilities are covered.

- If q_2, q_3 are both online, then $\mathsf{bos}_{\mathsf{on}}$ happens.

- If one of q_2, q_3 is online, the other offline and

  – if q_1 is offline then $\mathsf{bos}_{\mathsf{offon},1}$ happens

  – if q_1 is online, then $\mathsf{bos}_{\mathsf{offon},1}$ happens

- If both q_2, q_3 are offline and

  – q_1 is online, then $\mathsf{bos}_{\mathsf{offon},3}$ happens

  – q_1 is offline, then $\mathsf{bos}_{\mathsf{off}}$ happens

Therefore

$$\mathsf{Pr}\left[\mathsf{bos}\right] \leqslant \mathsf{Pr}\left[\mathsf{bos}_{\mathsf{off}}\right] + \mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},1}\right] + \mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},2}\right] + \mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},3}\right] + \mathsf{Pr}\left[\mathsf{bos}_{\mathsf{on}}\right] \ . \tag{11.27}$$

We upper bound these probabilities one-by-one. First off, we upper bound $\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{on}}\right]$. For every pair of queries made by $\mathcal{A}_2$, the probability that they have the same answer is $1/2^n$. From a union bound over all pairs of queries by $\mathcal{A}_1$ it follows that

$$\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{on}}\right] \leqslant \frac{\binom{T_2}{2}}{2^n} \ . \tag{11.28}$$

Next, we upper bound $\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},2}\right]$. Fix a query $q = \mathrm{h}(y, M) = z$ made by $\mathcal{A}_1$ and a pair of queries q', q'' by $\mathcal{A}_2$. The probability that the answer of q' is y and that of q'' is z, is $1/2^{2n}$. Taking a union bound over all possible q, q', q'' we have that

$$\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},2}\right] \leqslant \frac{T_2^2 T_1}{2^{2n}} \ . \tag{11.29}$$

Next, we upper bound $\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},3}\right]$. Recall the event $\mathsf{off\text{-}bulbs}\text{-}k$ defined in the proof of Lemma 11.4.9: there is a set of at least k distinct salts $a_1, \ldots, a_k$ such that for each a_i, $\mathcal{A}_1$ has made a pair of queries $\mathrm{h}(a_i, M_i) = z$ and $\mathrm{h}(a_i, M_i') = z$. For a salt a_i, we refer to the adversary querying $\mathrm{h}(a_i, M_i) = z$ and $\mathrm{h}(a_i, M_i') = z$ for some $M_i \neq M_i'$ as a bulb query for the salt. In other words $\mathsf{off\text{-}bulbs}\text{-}k$ is the event that the adversary makes bulb queries for at least k salts.

We have that

$$\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},3}\right] \leqslant \mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},3} \,\middle|\, \neg\mathsf{off\text{-}bulbs}\text{-}k\right] + \mathsf{Pr}\left[\mathsf{off\text{-}bulbs}\text{-}k\right] \ .$$

If $\mathcal{A}_1$ makes bulb queries for at most k salts, $\mathsf{bos}_{\mathsf{offon},3}$ happens only if $\mathcal{A}_1$ makes a query whose answer is among these at most k salts. Hence $\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},3} \,\middle|\, \neg\mathsf{off\text{-}bulbs}\text{-}k\right] \leqslant kT_1/2^n$. We know from the analysis in Lemma 11.4.9, that for $k = \max\left(eT_1/2^{n/2}, n/2\right)$, $\mathsf{Pr}\left[\mathsf{off\text{-}bulbs}\text{-}k\right] \leqslant 1/2^n$. Therefore, it follows that

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},3}\right] &\leqslant \frac{1}{2^n} + \frac{T_2}{2^n} \cdot \left(\frac{n}{2} + \frac{eT_1}{2^{n/2}}\right) \\ &= \frac{nT_2/2 + 1}{2^n} + \frac{eT_1T_2}{2^{3n/2}} \ .\end{aligned} \tag{11.30}$$

Next, we upper bound $\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},1}\right]$. We define the notion of a query being $\mathcal{A}_1$ being a *successor* query for a particular salt a. We say that a query $q = \mathrm{h}(y, M) = z$ made by $\mathcal{A}_1$ is a successor query for a salt a if

- $y \neq a, z \neq a, y \neq z$

- there is a query $q' = \mathrm{h}(a, M) = y$ made by $\mathcal{A}_1$ for some M

In Fig. 11.8, we illustrate how successor queries to a salt look like in the query graph of $\mathcal{A}$.

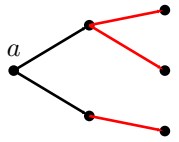


Figure 11.8: Illustrating successor queries of a salt in the query graph. The queries marked in red to a salt a are successor queries for a.

Let Q_a denote the number of successor queries for a salt a. Note that $\mathsf{bos}_{\mathsf{offon},1}$ happens only if $\mathcal{A}_2$ makes a query whose answer is same as the answer of one of the successor queries made of a. We have that

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},1}\right] &= \sum_{k=1}^{T_1} \mathsf{Pr}\left[Q_a = k\right] \cdot \frac{kT_1}{2^n} \\ &= \mathsf{E}\left[Q_a\right] \frac{T_1}{2^n} \ .\end{aligned}$$

We prove an upper bound on Q_a. Recall the event $m+1$-col: $(m+1)$-col happens if the $\mathcal{A}_1$ has made $m+1$ distinct h all of which have the same answer. We have using total expectation

$$\mathsf{E}\left[Q_a\right] \leqslant \mathsf{E}\left[Q_a \,|\, \neg(m+1)\text{-}\mathsf{col}\right] + \mathsf{E}\left[Q_a \,|\, (m+1)\text{-}\mathsf{col}\right] \mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \ . \tag{11.31}$$

We claim that $\mathsf{E}\left[Q_a \,|\, \neg(m+1)\text{-}\mathsf{col}\right] \leqslant mT_1/2^n$. This is because we have that $\mathsf{E}\left[\sum_{a\in\{0,1\}^n} Q_a \,|\, \neg m+1\text{-}\mathsf{col}\right] \leqslant mT_1$- since if there is no $(m+1)$-multicollision, a query can be a successor query to at most m different salts. Since a is sampled uniformly at random from $\{0,1\}^n$, the claim follows.

We have from earlier analysis in Theorem 11.4.1 that for $m = \max\left(n, \frac{4eT_1}{2^n}\right)$

$$\mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \leqslant 1/2^n \ .$$

Moreover $\mathsf{E}\left[Q_a \,|\, (m+1)\text{-}\mathsf{col}\right] \leqslant T_1$ since a salt cannot have more than T_1 successor queries. Therefore, plugging this into (11.31)

$$\mathsf{E}\left[Q_a\right] \leqslant \frac{4eT_1^2}{2^{2n}} + \frac{nT_1}{2^n} + \frac{T_1}{2^{2n}} \ .$$

So, we have

$$\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{offon},1}\right] = \mathsf{E}\left[Q_a\right] \frac{T_2}{2^n} \leqslant \frac{4eT_1^2T_2}{2^{3n}} + \frac{nT_1T_2}{2^{2n}} + \frac{T_1T_2}{2^{3n}} \ . \tag{11.32}$$

Finally, we upper bound $\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{off}}\right]$. We have that for any k, m

$$\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{off}}\right] \leqslant \mathsf{Pr}\left[\mathsf{bos}_{\mathsf{off}} \,\middle|\, \neg(m+1)\text{-}\mathsf{col} \wedge \neq \mathsf{off}\text{-}\mathsf{bulbs}\text{-}k\right] + \mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] + \mathsf{Pr}\left[\mathsf{off}\text{-}\mathsf{bulbs}\text{-}k\right] \ .$$

We claim that $\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{off}} \,\middle|\, \neg(m+1)\text{-}\mathsf{col} \wedge \neq \mathsf{off}\text{-}\mathsf{bulbs}\text{-}k\right] \leqslant k \cdot m/2^n$, because if there are no $m+1$-multi-collisions and the adversary finds bulbs for at most k salts, there are at most $k \cdot m$ salts such that these when sampled cause $\mathsf{bos}_{\mathsf{off}}$. Since a is sampled uniformly at random, it follows that this probability is at most $k \cdot m/2^n$.

We let $k = \max\left(\frac{eT_1}{2^{n/2}}, n/2\right)$, $m = \max\left(n, \frac{4eT_1}{2^n}\right)$, we have that

$$k \cdot m \leqslant n^2/2 + \frac{eT_1 n}{2^{n/2}} + \frac{2eT_1 n}{2^n} + \frac{4e^2T_1^2}{2^{3n/2}};.$$

This implies

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{bos}_{\mathsf{off}}\right] \leqslant & \frac{1}{2^n} + \frac{1}{2^n} + \frac{1}{2^n} \cdot \left(n^2/2 + \frac{eT_1 n}{2^{n/2}} + \frac{2eT_1 n}{2^n} + \frac{4e^2T_1^2}{2^{3n/2}}\right) \\ = & \frac{n^2/2+2}{2^n} + \frac{enT_1}{2^{3n/2}} + \frac{2enT_1}{2^{2n}} + \frac{4e^2T_1^2}{2^{5n/2}} \,. \end{aligned} \tag{11.33}$$

Plugging (11.28) to (11.30), (11.32) and (11.33) into (11.27) we get

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{bos}\right] \leqslant & \frac{\binom{T_2}{2} + nT_2/2 + n^2/2 + 4}{2^n} + \frac{eT_1T_2 + enT_1}{2^{3n/2}} + \frac{nT_1T_2 + T_1T_2^2 + 2enT_1}{2^{2n}} \\ & + \frac{4e^2T_1^2}{2^{5n/2}} + \frac{4eT_1^2T_2}{2^{3n}} \,. \end{aligned}$$

□ ■

11.4.7 Proof of Lemma 11.4.11

Proof. [Lemma 11.4.11] We define the six following events.

1. $\mathsf{tri}_{\mathsf{off}}$: $\mathcal{A}_1$ makes queries $\mathrm{h}(a, M') = y$, $\mathrm{h}(y, M'') = z$, $\mathrm{h}(a, M''') = z$ for some M', M'', M''' and $y \neq a$

2. $\mathsf{tri}_{\mathsf{offon},1}$: $\mathcal{A}_1$ makes queries $\mathrm{h}(a, M') = y$, $\mathrm{h}(y, M'') = z$ and $\mathcal{A}_2$ makes a query with answer z for some z.

3. $\mathsf{tri}_{\mathsf{offon},2}$: $\mathcal{A}_1$ makes a query $\mathrm{h}(y, M') = z$, and $\mathcal{A}_2$ makes a query with answer z and another with answer y, for some y, z, M'.

4. $\mathsf{tri}_{\mathsf{offon},3}$: $\mathcal{A}_1$ makes queries $\mathrm{h}(a, M') = z$, and $\mathcal{A}_2$ makes query with answer z for some M', z.

5. $\mathsf{tri}_{\mathsf{offon},4}$: $\mathcal{A}_1$ makes queries $\mathrm{h}(a, M') = z$, $\mathrm{h}(y, M'') = z$ and $\mathcal{A}_2$ makes a query with answer y for some y, z, M', M''

6. $\mathsf{tri}_{\mathsf{on}}$: $\mathcal{A}_2$ makes queries two queries that have the same answer.

Observe that tri happens only if at least one of $\mathsf{tri}_{\mathsf{off}}$, $\mathsf{tri}_{\mathsf{offon},1}$, $\mathsf{tri}_{\mathsf{offon},2}$, $\mathsf{tri}_{\mathsf{offon},3}$, $\mathsf{tri}_{\mathsf{offon},4}$, $\mathsf{tri}_{\mathsf{on}}$ happens. To see why this is true, observe that for tri to happen $\mathcal{A}$ has to make queries $q_1 = \mathrm{h}(a, M') = y$, $q_2 = \mathrm{h}(y, M'') = z$, $q_3 = \mathrm{h}(a, M''') = z$ for M', M'', M''', y, z. We show that all the possibilities are covered.

- If q_2, q_3 are both online, then $\mathsf{tri}_{\mathsf{on}}$ happens.
- If q_2 is online, q_3 is offline then $\mathsf{tri}_{\mathsf{offon},3}$ happens
- If q_3 is online, q_2 is offline and
  - q_1 is online, then $\mathsf{tri}_{\mathsf{offon},2}$ happens
  - q_1 is offline, then $\mathsf{tri}_{\mathsf{offon},1}$ happens
- If both q_2, q_3 are offline and
  - q_1 is online, then $\mathsf{tri}_{\mathsf{offon},4}$ happens
  - q_1 is offline, then $\mathsf{tri}_{\mathsf{off}}$ happens

Therefore

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{tri}\right] \leqslant & \mathsf{Pr}\left[\mathsf{tri}_{\mathsf{off}}\right] + \mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},1}\right] + \mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},2}\right] + \mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},3}\right] \\ & + \mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},4}\right] + \mathsf{Pr}\left[\mathsf{tri}_{\mathsf{on}}\right] \ . \end{aligned} \tag{11.34}$$

We upper bound these probabilities one-by-one.

We start with upper bounding $\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{on}}\right]$. Observe that $\mathsf{tri}_{\mathsf{on}}$ is the same event as $\mathsf{bos}_o n$ in the proof of Lemma 11.4.10. Therefore, from (11.28), it follows that

$$\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{on}}\right] \leqslant \frac{\binom{T_2}{2}}{2^n} \ . \tag{11.35}$$

Next we upper bound $\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},1}\right]$. Observe that $\mathsf{tri}_{\mathsf{offon},1}$ is the same event as $\mathsf{bos}_{\mathsf{offon},1}$ in the proof of Lemma 11.4.10. Therefore, from (11.32), it follows that

$$\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},1}\right] \leqslant \frac{4eT_1^2T_2}{2^{3n}} + \frac{nT_1T_2}{2^{2n}} + \frac{T_1T_2}{2^{3n}} \ . \tag{11.36}$$

Next we upper bound $\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},2}\right]$. Observe that $\mathsf{tri}_{\mathsf{offon},2}$ is the same event as $\mathsf{bos}_{\mathsf{offon},2}$ in the proof of Lemma 11.4.10. Therefore, from (11.29), it follows that

$$\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},2}\right] \leqslant \frac{T_2^2T_1}{2^{2n}} \ . \tag{11.37}$$

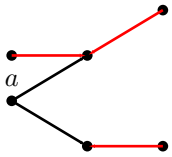


Figure 11.9: Illustrating meeting queries of a salt in the query graph. The queries marked in red to a salt a are meeting queries for a.

Next we upper bound $\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},3}\right]$. Observe that $\mathsf{tri}_{\mathsf{offon},3}$ is the same event as $\mathsf{bulb}_{\mathsf{offon}}$ in the proof of Lemma 11.4.9. Therefore, from (11.24), it follows that

$$\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},3}\right] \leqslant \frac{T_1 T_2}{2^{2n}} \ . \tag{11.38}$$

Next we upper bound $\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},4}\right]$. We define the notion of a query being $\mathcal{A}_1$ being a *meeting* query for a particular salt a. We say that a query $q = \mathrm{h}(y, M) = z$ $(y \neq a)$ made by $\mathcal{A}_1$ is a meeting query for a salt a if:

- $y \neq a, y \neq z, a \neq z$

- there is a query $q' = \mathrm{h}(a, M) = z$ made by $\mathcal{A}_1$

In Fig. 11.9, we illustrate how meeting queries to a salt look like in the query graph of $\mathcal{A}$.

Let Q_a denote the number of meeting queries for a salt a. Note that $\mathsf{tri}_{\mathsf{offon},4}$ happens only if $\mathcal{A}_2$ makes a query whose answer is same as the answer of one of the meeting queries made of a. We have that

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},4}\right] &= \sum_{k=1}^{T_1} \mathsf{Pr}\left[Q_a = k\right] \cdot \frac{kT_1}{2^n} \\ &= \mathsf{E}\left[Q_a\right] \frac{T_1}{2^n} \ .\end{aligned}$$

We prove an upper bound on $\mathsf{E}\left[Q_a\right]$. Recall the event $m+1$-col: $(m+1)$-col happens if the $\mathcal{A}_1$ has made $m+1$ distinct h all of which have the same answer. We have using total expectation

$$\mathsf{E}\left[Q_a\right] \leqslant \mathsf{E}\left[Q_a \,|\, \neg(m+1)\text{-}\mathsf{col}\right] + \mathsf{E}\left[Q_a \,|\, (m+1)\text{-}\mathsf{col}\right] \mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \ . \tag{11.39}$$

We claim that $\mathsf{E}\left[Q_a \,|\, \neg(m+1)\text{-}\mathsf{col}\right] \leqslant mT_1/2^n$. This is because we have that $\mathsf{E}\left[\sum_{a\in\{0,1\}^n} Q_a \,|\, \neg m+1\text{-}\mathsf{col}\right] \leqslant mT_1$ – since if there is no $(m+1)$-multicollision, a query can be a meeting query to at most m different salts. Since a is sampled uniformly at random from $\{0,1\}^n$, the claim follows.

We have from earlier analysis in Theorem 11.4.1 that for $m = \max\left(n, \frac{4eT_1}{2^n}\right)$

$$\mathsf{Pr}\left[(m+1)\text{-col}\right] \leqslant 1/2^n \ .$$

Moreover $\mathsf{E}\left[Q_a \,|\, (m+1)\text{-col}\right] \leqslant T_1$ since a salt cannot have more than T_1 meeting queries. Therefore, plugging this into (11.39)

$$\mathsf{E}\left[Q_a\right] \leqslant \frac{4eT_1^2}{2^{2n}} + \frac{nT_1}{2^n} + \frac{T_1}{2^{2n}} \ .$$

So, we have

$$\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{offon},4}\right] = \mathsf{E}\left[Q_a\right]\frac{T_2}{2^n} \leqslant \frac{4eT_1^2T_2}{2^{3n}} + \frac{nT_1T_2}{2^{2n}} + \frac{T_1T_2}{2^{3n}} \ . \tag{11.40}$$

We finally upper bound $\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{off}}\right]$ in Lemma 11.4.13.

Lemma 11.4.13.

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{off}}\right] \leqslant & \frac{12e(2)^{1/2}T_1^{3/2}}{2^{2n}} + \frac{3(8en)^{1/2}T_1}{2^{3n/2}} + \frac{3(24e)^{1/2}T_1}{2^{3n/2}} + \frac{9(2)^{1/3}eT_1^{4/3}}{2^{5n/3}} \\ & + \frac{18n}{2^n} + \frac{8}{2^n} \ .\end{aligned} \tag{11.41}$$

Plugging (11.35) to (11.38), (11.40) and (11.41) in (11.34) we have that

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{tri}\right] \leqslant & \frac{\binom{T_2}{2}}{2^n} + \frac{4eT_1^2T_2}{2^{3n}} + \frac{nT_1T_2}{2^{2n}} + \frac{T_1T_2}{2^{3n}} + \frac{T_2^2T_1}{2^{2n}} + \frac{T_1T_2}{2^{2n}} + \frac{4eT_1^2T_2}{2^{3n}} + \frac{nT_1T_2}{2^{2n}} \\ & + \frac{T_1T_2}{2^{3n}} + \frac{12e(2)^{1/2}T_1^{3/2}}{2^{2n}} + \frac{3(8en)^{1/2}T_1}{2^{3n/2}} + \frac{3(24e)^{1/2}T_1}{2^{3n/2}} + \frac{9(2)^{1/3}eT_1^{4/3}}{2^{5n/3}} \\ & + \frac{18n}{2^n} + \frac{8}{2^n} \\ \leqslant & \frac{\binom{T_2}{2} + 18n + 8}{2^n} + \frac{3(24e)^{1/2}T_1 + 3(8en)^{1/2}T_1}{2^{3n/2}} + \frac{9(2)^{1/3}eT_1^{4/3}}{2^{5n/3}} \\ & + \frac{2nT_1T_2 + T_1T_2 + T_2^2T_1 + 12e(2)^{1/2}T_1^{3/2}}{2^{2n}} + \frac{8eT_1^2T_2 + 2T_1T_2}{2^{3n}}\end{aligned}$$

□ ■

We defer the proof of Lemma 11.4.13 to Section 11.4.9.

11.4.8 Proof of Lemma 11.4.12

Proof. [Lemma 11.4.12] We define the six following events.

1. $\mathsf{dia}_{\mathsf{off}}$: $\mathcal{A}_1$ makes queries $\mathrm{h}(a, M') = y$, $\mathrm{h}(y, M'') = z$, $\mathrm{h}(a, M''') = y'$, $\mathrm{h}(a, M'''') = z$ for some $M' \neq M''', M'' \neq M''''$ and $y \neq y'$

2. $\mathsf{dia}_{\mathsf{offon},1}$: $\mathcal{A}_1$ makes queries $\mathrm{h}(a, M') = y$, $\mathrm{h}(y, M'') = z$ and $\mathcal{A}_2$ makes a query with answer z for some z.

3. $\mathsf{dia}_{\mathsf{offon},2}$: $\mathcal{A}_1$ makes a query $\mathrm{h}(y, M') = z$, and $\mathcal{A}_2$ makes a query with answer z and another with answer y, for some y, z, M'.

4. $\mathsf{dia}_{\mathsf{offon},3}$: $\mathcal{A}_1$ makes queries $\mathrm{h}(a, M') = y$, $\mathrm{h}(y, M'') = z$, $\mathrm{h}(y', M''') = z$, $\mathcal{A}_2$ makes a query with answer y' for some $y \neq y'$, M', M'', M'''

5. $\mathsf{dia}_{\mathsf{offon},4}$: $\mathcal{A}_1$ makes queries $\mathrm{h}(y, M') = z$, $\mathrm{h}(y', M'') = z$, $\mathcal{A}_2$ makes queries with answer y', y for some $y \neq y'$, M', M''

6. $\mathsf{dia}_{\mathsf{on}}$: $\mathcal{A}_2$ makes queries two queries that have the same answer.

Observe that dia happens only if at least one of $\mathsf{dia}_{\mathsf{off}}$, $\mathsf{dia}_{\mathsf{offon},1}$, $\mathsf{dia}_{\mathsf{offon},2}$, $\mathsf{dia}_{\mathsf{offon},3}$, $\mathsf{dia}_{\mathsf{offon},4}$, $\mathsf{dia}_{\mathsf{on}}$ happens. To see why this is true, observe that for dia to happen $\mathcal{A}$ has to make queries $q_1 = \mathrm{h}(a, M') = y$, $q_2 = \mathrm{h}(y, M'') = z$, $q_3 = \mathrm{h}(a, M''') = y'$, $q_4 = \mathrm{h}(y', M'''') = z$ for $M', M'', M''', M'''', y, y', z$ such that $y \neq y'$, $M' \neq M'''$, $M'' \neq M''''$. We show that all the possibilities are covered.

- If q_2, q_4 are both online, then $\mathsf{dia}_{\mathsf{on}}$ happens.

- If q_2 is online, q_4 is offline and

  - q_3 is online, then $\mathsf{dia}_{\mathsf{offon},2}$ happens
  - q_3 is offline, then $\mathsf{dia}_{\mathsf{offon},1}$ happens

- If q_2 is offline, q_4 is online and

  - q_1 is online, then $\mathsf{dia}_{\mathsf{offon},2}$ happens
  - q_1 is offline, then $\mathsf{dia}_{\mathsf{offon},1}$ happens

- If q_2, q_4 are both offline

  - q_1, q_3 are both online, then $\mathsf{tri}_{\mathsf{offon},4}$ happens
  - one of q_1, q_3 is offline, the other offline, then $\mathsf{tri}_{\mathsf{offon},3}$ happens
  - q_1, q_3 are both offline, then $\mathsf{tri}_{\mathsf{off}}$ happens

Therefore

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{dia}\right] \leqslant & \mathsf{Pr}\left[\mathsf{dia}_{\mathsf{off}}\right] + \mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},1}\right] + \mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},2}\right] + \mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},3}\right] \\ & + \mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},4}\right] + \mathsf{Pr}\left[\mathsf{dia}_{\mathsf{on}}\right] \ . \end{aligned} \tag{11.42}$$

We upper bound these probabilities one-by-one.

We start with upper bounding $\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{on}}\right]$. Observe that $\mathsf{dia}_{\mathsf{on}}$ is the same event as $\mathsf{bos}_o n$ in the proof of Lemma 11.4.10. Therefore, from (11.28), it follows that

$$\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{on}}\right] \leqslant \frac{\binom{T_2}{2}}{2^n} \ . \tag{11.43}$$

Next we upper bound $\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},1}\right]$. Observe that $\mathsf{dia}_{\mathsf{offon},1}$ is the same event as $\mathsf{bos}_{\mathsf{offon},1}$ in the proof of Lemma 11.4.10. Therefore, from (11.32), it follows that

$$\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},1}\right] \leqslant \frac{4eT_1^2T_2}{2^{3n}} + \frac{nT_1T_2}{2^{2n}} + \frac{T_1T_2}{2^{3n}} \ . \tag{11.44}$$

Next we upper bound $\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},2}\right]$. Observe that $\mathsf{dia}_{\mathsf{offon},2}$ is the same event as $\mathsf{bos}_{\mathsf{offon},2}$ in the proof of Lemma 11.4.10. Therefore, from (11.29), it follows that

$$\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},2}\right] \leqslant \frac{T_2^2T_1}{2^{2n}} \ . \tag{11.45}$$

Next we upper bound $\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},3}\right]$. We first define the notion of a *successor-hitting* query for a salt. We say that a query $q = \mathrm{h}(y, M) = z$ made by $\mathcal{A}_1$ is a successor-hitting query for a salt a if

- $z \neq a, y \neq a, y \neq z$

- there are queries $q' = \mathrm{h}(a, M') = b$, $q'' = \mathrm{h}(b, M'') = z$ with $b \neq a, z \neq b$ made by $\mathcal{A}_1$ for some M, M'

Informally, a query q is a successor-hitting query to salt a if there query $q' = \mathrm{h}(y', M') = z$ such that q' is a successor query to a and $q' \neq q$. In Fig. 11.10, we illustrate how successor-hitting queries to a salt look like in the query graph of $\mathcal{A}$.

Let Q_a denote the number of successor-hitting queries for a salt a. Note that $\mathsf{dia}_{\mathsf{offon},3}$ happens only if $\mathcal{A}_2$ makes a query whose answer is same as the answer of one of the successor-hitting queries made of a. We have that

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},3}\right] &= \sum_{k=1}^{T_1} \mathsf{Pr}\left[Q_a = k\right] \cdot \frac{kT_1}{2^n} \\ &= \mathsf{E}\left[Q_a\right] \frac{T_1}{2^n} \ . \end{aligned}$$

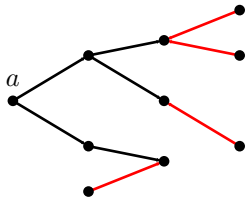


Figure 11.10: Illustrating successor-hitting queries of a salt in the query graph. The queries marked in red to a salt a are successor-hitting queries for a.

We prove an upper bound on $\mathsf{E}[Q_a]$. Recall the event $m+1$-col: $(m+1)$-col happens if the $\mathcal{A}_1$ has made $m+1$ distinct h all of which have the same answer. We have using total expectation

$$\mathsf{E}[Q_a] \leqslant \mathsf{E}[Q_a \mid \neg(m+1)\text{-col}] + \mathsf{E}[Q_a \mid (m+1)\text{-col}]\,\mathsf{Pr}[(m+1)\text{-col}] \ . \tag{11.46}$$

We claim that $\mathsf{E}[Q_a \mid \neg(m+1)\text{-col}] \leqslant m^2T_1/2^n$. This is because we have that $\mathsf{E}\left[\sum_{a\in\{0,1\}^n} Q_a \mid \neg m+1\text{-col}\right] \leqslant m^2T_1$ – since if there is no $(m+1)$-multicollision, a query can be a sucessor-hitting query to at most m^2 different salts. Since a is sampled uniformly at random from $\{0,1\}^n$, the claim follows.

We have from earlier analysis in Theorem 11.4.1 that for $m = \max\left(n, \frac{4eT_1}{2^n}\right)$

$$\mathsf{Pr}[(m+1)\text{-col}] \leqslant 1/2^n \ .$$

Moreover $\mathsf{E}[Q_a \mid (m+1)\text{-col}] \leqslant T_1$ since a salt cannot have more than T_1 successor-hitting queries. Therefore, plugging this into (11.46)

$$\mathsf{E}[Q_a] \leqslant \frac{4eT_1^3}{2^{3n}} + \frac{n^2T_1}{2^n} + \frac{T_1}{2^{2n}} \ .$$

So, we have

$$\mathsf{Pr}[\mathsf{dia}_{\mathsf{offon},3}] = \mathsf{E}[Q_a]\,\frac{T_2}{2^n} \leqslant \frac{4eT_1^3T_2}{2^{4n}} + \frac{n^2T_1T_2}{2^{2n}} + \frac{T_1T_2}{2^{3n}} \ . \tag{11.47}$$

Next, we upper bound $\mathsf{Pr}[\mathsf{dia}_{\mathsf{offon},4}]$. We have that

$$\mathsf{Pr}[\mathsf{dia}_{\mathsf{offon},4}] \leqslant \mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},4} \mid \neg(m+1)\text{-col}\right] + \mathsf{Pr}[(m+1)\text{-col}] \ .$$

We claim that $\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},4} \mid \neg(m+1)\text{-col}\right] \leqslant mT_1T_2^2/2^{2n}$. This is because, given there are no $(m+1)$-multicollision there can be at most mT_1 pairs of queries that collide. For $\mathsf{dia}_{\mathsf{offon},4}$ to happen, $\mathcal{A}_2$ needs to make two queries such that the answer of the queries are the input salts of a pair of colliding offline queries. Using a union bound over all colliding offline queries and pairs of queries by $\mathcal{A}_2$, the claim follows. Moreover, we have from earlier analysis in Theorem 11.4.1 that for $m = \max\left(n, \frac{4eT_1}{2^n}\right)$

$$\mathsf{Pr}[(m+1)\text{-col}] \leqslant 1/2^n \ .$$

Therefore, we have that

$$\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{offon},4}\right] \leqslant \frac{4eT_1^2T_2^2}{2^{3n}} + \frac{nT_1T_2^2}{2^{2n}} + \frac{1}{2^n} \ . \tag{11.48}$$

We finally upper bound $\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{off}}\right]$ in Lemma 11.4.14.

Lemma 11.4.14.

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{off}}\right] \leqslant & \frac{30n+15}{2^n} + \frac{40(en)^{1/3}T_1}{2^{4n/3}} + \frac{(8e)^{1/2}nT_1 + 10(2e)^{1/2}nT_1}{2^{3n/2}} \\ & + \frac{240e^{2/3}T_1^2}{2^{7n/3}} + \frac{8(2)^{1/2}e^{3/2}T_1^2 + 40(2e)^{1/2}T_1^2}{2^{5n/2}} \ .\end{aligned} \tag{11.49}$$

Plugging (11.43) to (11.45) and (11.47) to (11.49) in (11.42) we have that

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{dia}\right] \leqslant & \frac{\binom{T_2}{2}}{2^n} + \frac{4eT_1^2T_2}{2^{3n}} + \frac{nT_1T_2}{2^{2n}} + \frac{T_1T_2}{2^{3n}} + \frac{T_2^2T_1}{2^{2n}} + \frac{4eT_1^3T_2}{2^{4n}} + \frac{n^2T_1T_2}{2^{2n}} \\ & + \frac{T_1T_2}{2^{3n}} + \frac{4eT_1^2T_2^2}{2^{3n}} + \frac{nT_1T_2^2}{2^{2n}} + \frac{1}{2^n} + \frac{30n+15}{2^n} \\ & + \frac{40(en)^{1/3}T_1}{2^{4n/3}} + \frac{(8e)^{1/2}nT_1 + 10(2e)^{1/2}nT_1}{2^{3n/2}} \\ & + \frac{240e^{2/3}T_1^2}{2^{7n/3}} + \frac{8(2)^{1/2}e^{3/2}T_1^2 + 40(2e)^{1/2}T_1^2}{2^{5n/2}} \\ = & \frac{\binom{T_2}{2} + 30n + 16}{2^n} + \frac{4eT_1^2T_2 + 2T_1T_2 + 4eT_1^2T_2^2}{2^{3n}} \\ & + \frac{nT_1T_2 + T_2^2T_1 + n^2T_1T_2 + nT_1T_2^2}{2^{2n}} + \frac{4eT_1^3T_2}{2^{4n}} \\ & + \frac{40(en)^{1/3}T_1}{2^{4n/3}} + \frac{(8e)^{1/2}nT_1 + 10(2e)^{1/2}nT_1}{2^{3n/2}} \\ & + \frac{240e^{2/3}T_1^2}{2^{7n/3}} + \frac{8(2)^{1/2}e^{3/2}T_1^2 + 40(2e)^{1/2}T_1^2}{2^{5n/2}} \ .\end{aligned}$$

■

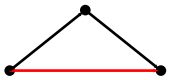
We defer the proof of Lemma 11.4.14 to Section 11.4.10.

11.4.9 Proof of Lemma 11.4.13

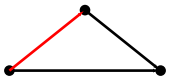
Proof. [Lemma 11.4.13] We say that $\mathcal{A}_1$ has found a triangle for a salt a if it has made queries $\mathrm{h}(a, M) = y$, $\mathrm{h}(y, M') = z$, $\mathrm{h}(a, M'') = z$ for some M, M', M'', y, z. We define the event $\mathsf{off\text{-}tri}\text{-}k$ as $\mathcal{A}_1$ finding triangles for at least k salts. We have that for any k,

$$\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{off}}\right] \leqslant \mathsf{Pr}\left[\mathsf{tri}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}tri}\text{-}k\right] + \mathsf{Pr}\left[\mathsf{off\text{-}tri}\text{-}k\right] \ . \tag{11.50}$$

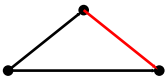
We claim that $\mathsf{Pr}\left[\mathsf{tri}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}tri}\text{-}k\right] \leqslant k/2^n$ – this is because if $\mathcal{A}_1$ finds triangles for at most k different salts, $\mathsf{tri}_{\mathsf{off}}$ happens with probability at most $k/2^n$.



(a) Category 1 Triangle.



(b) Category 2 Triangle.



(c) Category 3 Triangle.

Figure 11.11: The different categories of triangles. The edge colored red is the last of the three queries made.

Towards upper bounding $\mathsf{off\text{-}tri}\text{-}k$, we define a categorization of the triangles. For a triangle on salt a, let the three queries forming the triangle be $q_1 = \mathrm{h}(a, M) = y$, $q_2 = \mathrm{h}(y, M') = z$, $q_3 = \mathrm{h}(a, M'') = z$.

- We say that a triangle is of category 1, if the query q_3 was the last among the three

- We say that a triangle is of category 2, if the query q_1 was the last among the three

- We say that a triangle is of category 3, if the query q_2 was the last among the three

We illustrate the categories of triangles in Fig. 11.11.

We define the event $\mathsf{off\text{-}1\text{-}tri}\text{-}k_1$ as $\mathcal{A}_1$ find triangles of category 1 for k_1 different salts. We define the event $\mathsf{off\text{-}2\text{-}tri}\text{-}k_2$ as $\mathcal{A}_1$ find triangles of category 2 for k_2 different salts. We define the event $\mathsf{off\text{-}3\text{-}tri}\text{-}k_3$ as $\mathcal{A}_1$ find triangles of category 3 for k_3 different salts. We would have that for any k_1, k_2, k_3,

$$\mathsf{Pr}\left[\mathsf{off\text{-}tri\text{-}3}\max(k_1, k_2, k_3)\right] \leqslant \mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}tri}\text{-}k_1\right] + \mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}tri}\text{-}k_2\right] + \mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}tri}\text{-}k_3\right] \ . \tag{11.51}$$

We prove the three following claims.

Claim 11.4.15. *For any $k_1 \in \mathbb{N}_{>0}$, such that k_1 is a multiple of 2*

$$\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}tri}\text{-}k_1\right] \leqslant \left(\frac{16e^2T_1^3}{k_1^2 2^{2n}}\right)^{k_1/2} + \left(\frac{4enT_1^2}{k_1^2 2^n}\right)^{k_1/2} + \frac{1}{2^n} \ .$$

Claim 11.4.16. *For any $k_2 \in \mathbb{N}_{>0}$, such that k_2 is a multiple of 2*

$$\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}tri}\text{-}k_2\right] \leqslant \left(\frac{16e^2T_1^3}{k_2^2 2^{2n}}\right)^{k_2/2} + \left(\frac{4enT_1^2}{k_2^2 2^n}\right)^{k_2/2} + \frac{1}{2^n} \ .$$

Claim 11.4.17. *For any $k_3 \in \mathbb{N}_{>0}$, such that k_3 is a multiple of 6*

$$\mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}tri}\text{-}k_3\right] \leqslant \left(\frac{12eT_1^2}{k_3^2 2^n}\right)^{k_3/6} + \left(\frac{27e^3T_1^4}{k_3^3 2^{2n}}\right)^{k_3/3} \ .$$

We let $k_1 = \max\left(\left(\frac{32e^2T_1^3}{2^{2n}}\right)^{1/2}, \left(\frac{8enT_1^2}{2^n}\right)^{1/2}, 2n\right)$. We then have that

$$\mathsf{Pr}\left[\text{off-1-tri-}k_1\right] \leqslant \frac{3}{2^n} .$$

Similarly we let $k_2 = \max\left(\left(\frac{32e^2T_1^3}{2^{2n}}\right)^{1/2}, \left(\frac{8enT_1^2}{2^n}\right)^{1/2}, 2n\right)$, and have that

$$\mathsf{Pr}\left[\text{off-2-tri-}k_2\right] \leqslant \frac{3}{2^n} .$$

We let $k_3 = \max\left(\left(\frac{24eT_1^2}{2^n}\right)^{1/2}, \left(\frac{54e^3T_1^4}{2^{2n}}\right)^{1/3}, 6n\right)$, and have that

$$\mathsf{Pr}\left[\text{off-3-tri-}k_3\right] \leqslant \frac{2}{2^n} .$$

We have that

$$\begin{aligned}\max(k_1, k_2, k_3) &\leqslant \left(\frac{32e^2T_1^3}{2^{2n}}\right)^{1/2} + \left(\frac{8enT_1^2}{2^n}\right)^{1/2} + \left(\frac{24eT_1^2}{2^n}\right)^{1/2}, \left(\frac{54e^3T_1^4}{2^{2n}}\right)^{1/3} + 6n \\ &= \frac{4e(2)^{1/2}T_1^{3/2}}{2^n} + \frac{(8en)^{1/2}T_1}{2^{n/2}} + \frac{(24e)^{1/2}T_1}{2^{n/2}} + \frac{3(2)^{1/3}eT_1^{4/3}}{2^{2n/3}} + 6n .\end{aligned}$$

Let $k = 3 \cdot \left(\frac{4e(2)^{1/2}T_1^{3/2}}{2^n} + \frac{(8en)^{1/2}T_1}{2^{n/2}} + \frac{(24e)^{1/2}T_1}{2^{n/2}} + \frac{3(2)^{1/3}eT_1^{4/3}}{2^{2n/3}} + 6n\right)$. Using (11.51) we have that

$$\mathsf{Pr}\left[\text{off-tri-}k\right] \leqslant \frac{8}{2^n} . \tag{11.52}$$

Using the fact that we showed earlier that $\mathsf{Pr}\left[\text{tri} \,\middle|\, \neg\text{off-tri-}k\right] \leqslant k/2^n$, setting k as above, using (11.50) we have that

$$\mathsf{Pr}\left[\text{tri}_{\text{off}}\right] \leqslant \frac{12e(2)^{1/2}T_1^{3/2}}{2^{2n}} + \frac{3(8en)^{1/2}T_1}{2^{3n/2}} + \frac{3(24e)^{1/2}T_1}{2^{3n/2}} + \frac{9(2)^{1/3}eT_1^{4/3}}{2^{5n/3}} + \frac{18n}{2^n} + \frac{8}{2^n} .$$

We now prove these claims one by one.

Upper bounding $\mathsf{Pr}\left[\text{off-1-tri-}k_1\right]$. First off, we upper bound $\mathsf{Pr}\left[\text{off-1-tri-}k_1\right]$. We have that for any m,

$$\mathsf{Pr}\left[\text{off-1-tri-}k_1\right] \leqslant \mathsf{Pr}\left[\text{off-1-tri-}k_1 \wedge \neg(m+1)\text{-col}\right] + \mathsf{Pr}\left[m+1\text{-col}\right] . \tag{11.53}$$

We upper bound $\mathsf{Pr}\left[\text{off-1-tri-}k_1 \wedge \neg(m+1)\text{-col}\right]$ using a compression argument. Before giving the compression argument, we recall the notion of successor queries we introduced in the proof of Lemma 11.4.10: we say that a query $q = \mathrm{h}(y, M)$ $(y \neq a)$ made by $\mathcal{A}_1$ is a successor query for a salt a if there is a query $q' = \mathrm{h}(a, M) = y$ made by $\mathcal{A}_1$.

The encoding procedure encodes the random oracle h as follows.

1. It first runs $\mathcal{A}_1^{\mathrm{h}}$, answering all its queries using h.

2. If $m+1$-col happens or off-1-tri-k_1 does not happen, it outputs $\varnothing$
3. It finds a set of salts S of size $k_1/2$ such that for each $a \in S$
   - $\mathcal{A}_1$ found a category 1 triangle for a
   - The salt a has no more than $2mT_1/k_1$ successor queries
4. If no such S is found, return $\varnothing$
5. It marks the q_2, q_3 queries corresponding to the triangle for each salt in S
6. It initializes lists $\mathsf{L}_1, \mathsf{L}_2$ to empty lists, T to empty set
7. It starts running $\mathcal{A}_1$ again
8. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:
   (a) If the query is marked as a q_3 query for a salt $a \in S$, then it adds the index of this query in T, and it adds the lexicographical index of q_2 among all *successor* queries for the salt a in the list L_2
   (b) Otherwise it adds $\mathrm{h}(a, M)$ to L_1.
9. It appends the evaluation of h on the points not queried by $\mathcal{A}_1$ to L_1 in the lexicographical order of the inputs.
10. It outputs $\mathsf{L}_1, \mathsf{L}_2, \mathsf{T}$.

The decoding procedure works as follows.

1. If the encoding is $\varnothing$ it aborts.
2. It runs $\mathcal{A}_1^{\mathrm{h}}$.
3. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:
   (a) If the index of the query is in T, it removes the element i on the front of list L_2, and locates the query q which has lexicographic order i among all the successor queries of the salt a. It answers with the answer of q

(b) Otherwise it removes the element in front of L_1 and answers with that.

4. It populates h on the points not queried by $\mathcal{A}_1$ in the lexicographical order by the remaining entries of L_1

Correctness of decoding: First off, we argue that for adversary $\mathcal{A}_1$ that causes the event $\mathsf{off\text{-}1\text{-}tri}\text{-}k_1 \wedge \neg(m+1)\text{-}\mathsf{col}$ to happen, the encoding algorithm will never return $\varnothing$. To argue this it suffices to show that the encoding algorithm never returns $\varnothing$ from line 4. Note that the encoding algorithm reaches this line only if the event $\mathsf{off\text{-}1\text{-}tri}\text{-}k_1 \wedge \neg(m+1)\text{-}\mathsf{col}$ was caused by running $\mathcal{A}_1$, which means there is a set of k_1 salts S' such that $\mathcal{A}_1$ found category one triangles for them. Now, since $\mathcal{A}_1$ makes at most T_1 queries, and $\neq (m+1)\text{-}\mathsf{col}$ happens, there are at most mT_1 salt-successor query pairs. Therefore, for at least $k_1/2$ salts (note that from our assumption k_1 is a multiple of 2, so $k_1/2$ is an integer) in S' such that it has at most $2T_1m/k_1$ successor queries (because otherwise there would be more than mT_1 salt, successor query pairs). Therefore, such a set $S \subseteq S'$ of size $k_1/2$ always exists.

If the encoding algorithm does not return $\varnothing$, it is easy to see the decoding algorithm decodes correctly, because the query answers that the encoding algorithm does not add in the list L_1 can be recovered by the decoding algorithm correctly using S, L_2. Therefore, we have that

$$\mathsf{Pr}\left[\text{ Decoding is correct }\right] \geqslant \mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}tri}\text{-}k_1 \wedge \neg(m+1)\text{-}\mathsf{col}\right] .$$

Observe that the encoding algorithm removes $k_1/2$ answers of the random oracle from the encoding, and all the removed answers are distinct because those were q_3 queries for different salts. It instead adds an unordered set T of $k_1/2$ values in $[T_1]$, and an ordered list of $k_1/2$ elements where each value is at most $2mT_1/k_1$. Using the compression lemma, we have that

$$\mathsf{Pr}\left[\text{ Decoding is correct }\right] \leqslant \frac{(2mT_1/k_1)^{k_1/2}\binom{T_1}{k_1/2}}{2^{nk_1/2}} \leqslant \left(\frac{4emT_1^2}{k_1^2 2^n}\right)^{k_1/2} .$$

Therefore,

$$\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}tri}\text{-}k_1 \wedge \neg(m+1)\text{-}\mathsf{col}\right] \leqslant \left(\frac{4emT_1^2}{k_1^2 2^n}\right)^{k_1/2} .$$

We know from previous analyses that $\mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \leqslant 1/2^n$ for $m = \max\left(n, \frac{4eT_1}{2^n}\right)$. Plugging this value of m we have

$$\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}tri}\text{-}k_1 \wedge \neg(m+1)\text{-}\mathsf{col}\right] \leqslant \left(\frac{16e^2T_1^3}{k_1^2 2^{2n}}\right)^{k_1/2} + \left(\frac{4enT_1^2}{k_1^2 2^n}\right)^{k_1/2} .$$

Plugging this into (11.53), we get that

$$\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}tri}\text{-}k_1\right] \leqslant \left(\frac{16e^2T_1^3}{k_1^2 2^{2n}}\right)^{k_1/2} + \left(\frac{4enT_1^2}{k_1^2 2^n}\right)^{k_1/2} + \frac{1}{2^n} . \tag{11.54}$$

UPPER BOUNDING $\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}tri}\text{-}k_2\right]$. We next upper bound $\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}tri}\text{-}k_2\right]$. We have that for any m,

$$\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}tri}\text{-}k_2\right] \leqslant \mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}tri}\text{-}k_2 \wedge \neg(m+1)\text{-}\mathsf{col}\right] + \mathsf{Pr}\left[m+1\text{-}\mathsf{col}\right] \ . \tag{11.55}$$

We upper bound $\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}tri}\text{-}k_2 \wedge \neg(m+1)\text{-}\mathsf{col}\right]$ using a compression argument. Before giving the compression argument, we recall the notion of meeting queries we introduced in the proof earlier: we say that a query $q = \mathrm{h}(y, M) = z$ $(y \neq a)$ made by $\mathcal{A}_1$ is a meeting query for a salt a if:

- $y \neq a, y \neq z, a \neq z$
- there is a query $q' = \mathrm{h}(a, M) = z$ made by $\mathcal{A}_1$

The encoding procedure encodes the random oracle h as follows.

1. It first runs $\mathcal{A}_1^{\mathrm{h}}$, answering all its queries using h.
2. If $m+1$-col happens or $\mathsf{off\text{-}2\text{-}tri}\text{-}k_2$ does not happen, it outputs $\varnothing$
3. It finds a set of salts S of size $k_2/2$ such that for each $a \in S$
   - $\mathcal{A}_1$ found a category 2 triangle for a
   - The salt a has no more than $2mT_1/k_2$ meeting queries
4. If no such S is found, return $\varnothing$
5. It marks the q_1, q_2 queries corresponding to the triangle for each salt in S
6. It initializes lists $\mathsf{L}_1, \mathsf{L}_2$ to empty lists, T to empty set
7. It starts running $\mathcal{A}_1$ again
8. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:
   (a) If the query is marked as a q_1 query for a salt $a \in S$, then it adds the index of this query in T, and adds the lexicographical index of q_2 among all *successor* queries for the salt a in the list L_2
   (b) Otherwise it adds $\mathrm{h}(a, M)$ to L_1.
9. It appends the evaluation of h on the points not queried by $\mathcal{A}_1$ to L_1 in the lexicographical order of the inputs.

10. It outputs $\mathsf{L}_1, \mathsf{L}_2, \mathsf{T}$.

The decoding procedure works as follows.

1. If the encoding is $\varnothing$ it aborts.
2. It runs $\mathcal{A}_1^{\mathrm{h}}$.
3. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:
   (a) If the index of the query is in T, it removes the element i on the front of list L_2, and locates the query q which has lexicographic order i among all the successor queries of the salt a. It answers with the answer of q
   (b) Otherwise it removes the element in front of L_1 and answers with that.
4. It populates h on the points not queried by $\mathcal{A}_1$ in the lexicographical order by the remaining entries of L_1

Correctness of decoding: First off, we argue that for adversary $\mathcal{A}_1$ that causes the event $\mathsf{off\text{-}2\text{-}tri}\text{-}k_2 \wedge \neg(m+1)\text{-}\mathsf{col}$ to happen, the encoding algorithm will never return $\varnothing$. To argue this it suffices to show that the encoding algorithm never returns $\varnothing$ from line 4. Note that the encoding algorithm reaches this line only if the event $\mathsf{off\text{-}2\text{-}tri}\text{-}k_2 \wedge \neg(m+1)\text{-}\mathsf{col}$ was caused by running $\mathcal{A}_1$, which means there is a set of k_2 salts S' such that $\mathcal{A}_1$ found category two triangles for them. Now, since $\mathcal{A}_1$ makes at most T_1 queries, and $\neg(m+1)\text{-}\mathsf{col}$ happens, there are at most mT_1 salt-meeting query pairs. Because otherwise there exists a query which is a meeting query to at least $m+1$ different salts- which means $(m+1)\text{-}\mathsf{col}$ has happened. Therefore, for at least $k_2/2$ salts in S' such that it has at most $2T_1 m/k_2$ meeting queries (because otherwise there would be more than mT_1 salt, meeting query pairs). Therefore, such a set $S \subseteq S'$ of size $k_2/2$ always exists.

If the encoding algorithm does not return $\varnothing$, it is easy to see the decoding algorithm decodes correctly, because the query answers that the encoding algorithm does not add in the list L_1 can be recovered by the decoding algorithm correctly using S, L_2. Therefore, we have that

$$\mathsf{Pr}\left[\text{ Decoding is correct }\right] \geqslant \mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}tri}\text{-}k_2 \wedge \neg(m+1)\text{-}\mathsf{col}\right] .$$

Observe that the encoding algorithm removes $k_2/2$ answers of the random oracle from the encoding, and all the removed answers are distinct because those were q_1 queries for different salts. It instead adds an

unordered set T of $k_2/2$ values in $[T_1]$, and an ordered list of $k_2/2$ elements where each value is at most $2mT_1/k_2$. Using the compression lemma, we have that

$$\mathsf{Pr}\left[\text{ Decoding is correct }\right] \leqslant \frac{(2mT_1/k_2)^{k_2/2}\binom{T_1}{k_2/2}}{2^{nk_2/2}} \leqslant \left(\frac{4emT_1^2}{k_2^2 2^n}\right)^{k_2/2} .$$

Therefore,

$$\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}tri}\text{-}k_2 \wedge \neg(m+1)\text{-}\mathsf{col}\right] \leqslant \left(\frac{4emT_1^2}{k_2^2 2^n}\right)^{k_2/2} .$$

We know from previous analyses that $\mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \leqslant 1/2^n$ for $m = \max\left(n, \frac{4eT_1}{2^n}\right)$. Plugging this value of m we have

$$\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}tri}\text{-}k_2 \wedge \neg(m+1)\text{-}\mathsf{col}\right] \leqslant \left(\frac{16e^2T_1^3}{k_2^2 2^{2n}}\right)^{k_2/2} + \left(\frac{4enT_1^2}{k_2^2 2^n}\right)^{k_2/2} .$$

Plugging this into (11.55), we get that

$$\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}tri}\text{-}k_2\right] \leqslant \left(\frac{16e^2T_1^3}{k_2^2 2^{2n}}\right)^{k_2/2} + \left(\frac{4enT_1^2}{k_2^2 2^n}\right)^{k_2/2} + \frac{1}{2^n} . \tag{11.56}$$

Upper bounding $\mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}tri}\text{-}k_3\right]$. Finally, we upper bound $\mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}tri}\text{-}k_3\right]$. One could hope for an analysis similar to what we did for $\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}tri}\text{-}k_1\right]$ and $\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}tri}\text{-}k_2\right]$– with the difference that in the compression argument, the encoding algorithm omits the answer of the q_2 queries and instead stores information like the set S, L_2 to recover its answers. However, this approach does not work! The main reason for this is if the adversary finds triangles of category three for k_3 different salts, it is not necessary that the q_2 queries for the triangle are unique for each of the salts.

However, notice that if $\mathcal{A}_1$ finds triangles of category three for two salts a, a' that share the q_2 query, the following must have happened: It must have made queries $\mathrm{h}(a, M) = y$, $\mathrm{h}(a, M') = y'$, $\mathrm{h}(a', M'') = y$, $\mathrm{h}(a', M''') = y'$ for some y, y', M, M', M'', M''' such that $y \neq a, y' \neq a, y \neq y'$. If this happens, we say $\mathcal{A}_1$ has found a *zig-zag* for the salt pair (a, a'). We illustrate the structure of a zigzag in the query graph in Fig. 11.12. We define the event $\mathsf{off\text{-}zz}\text{-}k$ as $\mathcal{A}_1$ finding a *zig-zag* for k different salt pairs $\{\{a_{1,i}, a_{2,i}\}_{i\in k}\}$ where all the salts $a_{j,i}$'s are distinct. We have that for any k_4

$$\mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}tri}\text{-}k_3\right] \leqslant \mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}tri}\text{-}k_3 \wedge \neg\mathsf{off\text{-}zz}\text{-}k_4\right] + \mathsf{Pr}\left[\mathsf{off\text{-}zz}\text{-}k_4\right] . \tag{11.57}$$

We prove the following lemmas.

Lemma 11.4.18. *For $k \in \mathbb{N}_{>0}$,*

$$\mathsf{Pr}\left[\mathsf{off\text{-}zz}\text{-}k\right] \leqslant \left(\frac{e^3T_1^4}{k^3 2^{2n}}\right)^k .$$

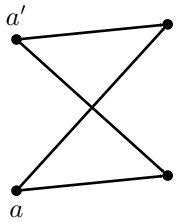


Figure 11.12: The zigzag structure for a salt pair (a, a') in the query graph.

Lemma 11.4.19. *For any $k, k' \in \mathbb{N}_{>0}$ such that $k \geqslant 2k'$, k is a multiple of 2,*

$$\mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}tri}\text{-}k \wedge \neg\mathsf{off\text{-}zz}\text{-}k'\right] \leqslant \left(\frac{4eT_1^2}{(k-2k')^2 2^n}\right)^{(k-2k')/2} .$$

Since k_3 is a multiple of 6, $k_3/3$ is an integer. By setting $k_4 = k_3/3$, putting together (11.58) with Lemmas 11.4.18 and 11.4.19, we have

$$\mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}tri}\text{-}k_3\right] \leqslant \left(\frac{12eT_1^2}{k_3^2 2^n}\right)^{k_3/6} + \left(\frac{27e^3T_1^4}{k_3^3 2^{2n}}\right)^{k_3/3} . \tag{11.58}$$

□ ■

We need to prove the Lemmas 11.4.18 and 11.4.19. We first prove Lemma 11.4.18.

Proof. [Lemma 11.4.18] We upper bound $\mathsf{Pr}\left[\mathsf{off\text{-}zz}\text{-}k\right]$ via a compression argument.

The encoding procedure encodes the random oracle h as follows.

1. It first runs $\mathcal{A}_1^{\mathrm{h}}$, answering all its queries using h.

2. If $\mathsf{off\text{-}zz}\text{-}k$ does not happen, it outputs $\varnothing$

3. It finds a set of k salt pairs $\{\{a_{i,1}, a_{i,2}\}_{i\in[k]}\}$ such that all the $a_{i,j}$'s are distinct and $\mathcal{A}_1$ found a zig-zag for each of these pairs.

4. For each of the salt pairs $a_{i,1}, a_{i,2}$, it isolates the four queries $\mathrm{h}(a_{i,1}, M) = y$, $\mathrm{h}(a_{i,1}, M') = y'$, $\mathrm{h}(a_{i,2}, M'') = y$, $\mathrm{h}(a_{i,2}, M''') = y'$. It labels the query that was the last among these four as q_4. It labels the query which has the same answer as q_4 but came earlier as q_3. It labels the earlier of the remaining two queries q_1, and later as q_2.

5. It initializes lists $\mathsf{L}_1, \mathsf{L}_2$ to empty lists, $\mathsf{T}_2, \mathsf{T}_3, \mathsf{T}_4$ to empty sets

6. It starts running $\mathcal{A}_1$ again

7. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:

   (a) If the query is marked as a q_2 query for some zigzag salt pair, it adds the index of the query to the set T_2, inserts the index of the corresponding q_1 in front of the list L_1

   (b) If the query is marked as a q_4 query for some zigzag salt pair, it adds the index of the query to the set T_4, adds the index of the corresponding q_3 in the set T_3

   (c) Otherwise it adds $\mathrm{h}(a, M)$ to L_2.

8. It appends the evaluation of h on the points not queried by $\mathcal{A}_1$ to L_2 in the lexicographical order of the inputs.

9. It outputs $\mathsf{L}_1, \mathsf{L}_2, \mathsf{T}$.

The decoding procedure works as follows.

1. If the encoding is $\varnothing$ it aborts.

2. It runs $\mathcal{A}_1^{\mathrm{h}}$.

3. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:

   (a) If the index of the query is in T_2, it removes the element i on the front of list L_1, and locates the query q which has index i among. It answers with the answer of q.

   (b) Otherwise if the index of the query is in T_4, it locates the query q that has its index in T_2 or L_1 that had input salt a. If such q is not found or more than one such q is found it aborts.

      - If q is in T_2, it finds the query q' in L_1 which was located to answer q – it then finds a query q'' in the set T_3 which has the same input salt as q'. If none or more than one q' is found, it aborts. Otherwise it answers with the answer of q'
      - If q is in L_1, it finds the query q' in T_2 which located q when being answered earlier in the decoding – it then finds a query q'' in the set T_3 which has the same input salt as q'. If none or more than one q' is found, it aborts. Otherwise it answers with the answer of q'

   (c) Otherwise it removes the element in front of L_2 and answers with that.

4. It populates h on the points not queried by $\mathcal{A}_1$ in the lexicographical order by the remaining entries of L_1

Correctness of decoding: First off, observe that for adversary $\mathcal{A}_1$ that causes the event $\mathsf{off\text{-}zz}\text{-}k$ to happen, the encoding algorithm will never return $\varnothing$. Further, we argue that whenever the decoding algorithm receives an output of the encoding algorithm that is not $\varnothing$, the decoding algorithm will decode correctly. This is because the encoding algorithm removes the answer of two of the queries from the pair of zig-zag salts which the decoding algorithm recovers using $\mathsf{L}_1, \mathsf{T}_2, \mathsf{T}_3, \mathsf{T}_4$. It is easy to see decoding answers the queries whose answers were in L_1 and the ones whose answers were not in L_1 but indices were put in T_2 correct. What is left is verifying that the queries whose answers were not in L_1 and whose indices were put in T_4 were answered correctly.

This can be verified by inspecting line 3b. For an output of the encoding algorithm that is not $\varnothing$, the decoding algorithm will never abort here – this is because for the q_4 queries removed, there is exactly one other query among all the queries whose indices are in $\mathsf{L}_1, \mathsf{T}_2$ whose salt is same as q_4, by the definition of the event $\mathsf{off\text{-}zz}\text{-}k$. And this step successfully recovers that value and returns the correct answer. Therefore, whenever $\mathcal{A}_1$ causes $\mathsf{off\text{-}zz}\text{-}k$, the encoding produces an outputs that decodes correctly.

Therefore, we have that

$$\mathsf{Pr}\left[\text{ Decoding is correct }\right] \geqslant \mathsf{Pr}\left[\mathsf{off\text{-}zz}\text{-}k\right] .$$

Observe that the encoding algorithm removes $2 \cdot k$ answers of the random oracle from the encoding, and all the removed answers are distinct because those were queries for different salts. It instead adds an unordered sets $\mathsf{T}_2, \mathsf{T}_2, \mathsf{T}_4$ of k values in $[T_1]$, and an ordered list of k values in $[T_1]$. Using the compression lemma, we have that

$$\mathsf{Pr}\left[\mathsf{off\text{-}zz}\text{-}k\right] \leqslant \mathsf{Pr}\left[\text{ Decoding is correct }\right] \leqslant \frac{\left(\binom{T_1}{k}\right)^3 T_1^k}{2^{n2k}} \leqslant \left(\frac{e^3 T_1^4}{k^3 2^{2n}}\right)^k .$$

□ ■

We next prove Lemma 11.4.19.

Proof. [Lemma 11.4.18] We need to upper bound $\mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}tri}\text{-}k \wedge \neg \mathsf{off\text{-}zz}\text{-}k'\right]$. Note that if $\mathcal{A}_1$ does not cause $\mathsf{off\text{-}zz}\text{-}k'$ and finds triangles of category three for at least k salts, it means that there is a set of at least $k - 2k'$ salts for which $\mathcal{A}_1$ finds triangles of category three such that no two triangles in the set share any query. We use this to intuition to build a compression argument that upper bounds $\mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}tri}\text{-}k \wedge \neg \mathsf{off\text{-}zz}\text{-}k'\right]$.

The encoding procedure encodes the random oracle h as follows.

1. It first runs $\mathcal{A}_1^{\mathrm{h}}$, answering all its queries using h.

2. If $\mathsf{off\text{-}zz}\text{-}k'$ happens or $\mathsf{off\text{-}3\text{-}tri}\text{-}k$ does not happen, it outputs $\varnothing$
3. It finds a set of salts S of size $(k-2k')/2$ such that for each $a \in S$
   - $\mathcal{A}_1$ found a category 3 triangle for a
   - The salt a has no more than $2T_1/(k-2k')$ queries on it
4. If no such S is found, return $\varnothing$
5. It marks the q_3, q_2 queries corresponding to the triangle for each salt in S
6. It initializes lists $\mathsf{L}_1, \mathsf{L}_2$ to empty lists, T to empty set
7. It starts running $\mathcal{A}_1$ again
8. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:
   (a) If the query is marked as a q_2 query for a salt $a \in S$, then it adds the index of this query in T, and it adds the index of q_3 in the list L_2
   (b) Otherwise it adds $\mathrm{h}(a, M)$ to L_1.
9. It appends the evaluation of h on the points not queried by $\mathcal{A}_1$ to L_1 in the lexicographical order of the inputs.
10. It outputs $\mathsf{L}_1, \mathsf{L}_2, \mathsf{T}$.

The decoding procedure works as follows.

1. If the encoding is $\varnothing$ it aborts.
2. It runs $\mathcal{A}_1^{\mathrm{h}}$.
3. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:
   (a) If the index of the query is in T, it removes the element i on the front of list L_2, and locates the query q which has index i. It answers with the answer of q
   (b) Otherwise it removes the element in front of L_1 and answers with that.

4. It populates h on the points not queried by $\mathcal{A}_1$ in the lexicographical order by the remaining entries of L_1

Correctness of decoding: First off, we argue that for adversary $\mathcal{A}_1$ that causes the event $\mathsf{off\text{-}3\text{-}tri}\text{-}k \wedge \neg\mathsf{off\text{-}zz}\text{-}k'$ to happen, the encoding algorithm will never return $\varnothing$. To argue this it suffices to show that the encoding algorithm never returns $\varnothing$ from line 4. Note that the encoding algorithm reaches this line only if the event $\mathsf{off\text{-}2\text{-}tri}\text{-}k \wedge \neg\mathsf{off\text{-}zz}\text{-}k'$ was caused by running $\mathcal{A}_1$, which means there is a set of k salts S' such that $\mathcal{A}_1$ found category three triangles for them. Moreover there does not exist a set of k' salt pairs for which $\mathcal{A}_1$ found a zig-zag. Meaning one can find a set of at least $k - 2k'$ salts such that $\mathcal{A}_1$ found a category three triangle for all of them but no two of these triangles share a q_2 query. Now, since $\mathcal{A}_1$ makes at most T_1 queries, at least half of these salts are such that the adversary makes at most $2Tp/(k-2k')$ queries on them. Therefore, such a set S of size $(k-2k')/2$ always exists.

If the encoding algorithm does not return $\varnothing$, it is easy to see the decoding algorithm decodes correctly, because the query answers that the encoding algorithm does not add in the list L_1 can be recovered by the decoding algorithm correctly using S, L_2. Therefore, we have that

$$\mathsf{Pr}\left[\text{ Decoding is correct }\right] \geqslant \mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}tri}\text{-}k \wedge \neg\mathsf{off\text{-}zz}\text{-}k'\right] .$$

Observe that the encoding algorithm removes $(k-2k')/2$ answers of the random oracle from the encoding, and all the removed answers are distinct because those were q_1 queries for different salts. It instead adds an unordered set T of $(k-2k')/2$ values in $[T_1]$, and an ordered list of $(k-2k')/2$ values in $[T_1]$. Using the compression lemma, we have that

$$\mathsf{Pr}\left[\text{ Decoding is correct }\right] \leqslant \frac{(2T_1/(k-2k'))^{(k-2k')/2}\binom{T_1}{(k-2k')/2}}{2^{n(k-2k')/2}} \leqslant \left(\frac{4eT_1^2}{(k-2k')^2 2^n}\right)^{(k-2k')/2} .$$

Therefore,

$$\mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}tri}\text{-}k \wedge \neg\mathsf{off\text{-}zz}\text{-}k'\right] \leqslant \left(\frac{4eT_1^2}{(k-2k')^2 2^n}\right)^{(k-2k')/2} . \tag{11.59}$$

□ ■

11.4.10 Proof of Lemma 11.4.14

Proof. [Lemma 11.4.14] We say that $\mathcal{A}_1$ has found a triangle for a salt a if it has made queries $\mathrm{h}(a, M) = y$, $\mathrm{h}(y, M') = z$, $\mathrm{h}(a, M'') = z$ for some M, M', M'', y, z. We define the event $\mathsf{off\text{-}dia}\text{-}k$ as $\mathcal{A}_1$ finding diamonds for at least k salts. We have that for any k,

$$\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{off}}\right] \leqslant \mathsf{Pr}\left[\mathsf{dia}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}dia}\text{-}k\right] + \mathsf{Pr}\left[\mathsf{off\text{-}dia}\text{-}k\right] . \tag{11.60}$$

(a) Category 1 diamonds.

(b) Category 2 diamonds.

Figure 11.13: The different categories of diamonds. The edge colored red is the last of the four queries made.

We claim that $\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{off}} \,\middle|\, \neg\mathsf{off\text{-}dia}\text{-}k\right] \leqslant k/2^n$ – this is because if $\mathcal{A}_1$ finds diamonds for at most k different salts, since a is sampled uniformly at random, $\mathsf{dia}_{\mathsf{off}}$ happens with probability at most $k/2^n$.

Towards upper bounding $\mathsf{off\text{-}dia}\text{-}k$, we define a categorization of the diamonds. For a diamond on salt a, let the four queries forming the triangle be $q_1 = \mathrm{h}(a, M) = y$, $q_2 = \mathrm{h}(y, M') = z$, $q_3 = \mathrm{h}(a, M'') = y'$, $q_4 = \mathrm{h}(y', M''') = z$.

- We say that a diamond is of category 1, if the query q_1 or q_3 was the last among the three

- We say that a triangle is of category 2, if the query q_2 or q_4 was the last among the three

We illustrate the categories of diamonds in Fig. 11.13. We define the event $\mathsf{off\text{-}1\text{-}dia}\text{-}k_1$ as $\mathcal{A}_1$ find diamonds of category 1 for k_1 different salts. We define the event $\mathsf{off\text{-}2\text{-}dia}\text{-}k_2$ as $\mathcal{A}_1$ finds diamonds of category 2 for k_2 different salts.

We would have that for any k_1, k_2,

$$\mathsf{Pr}\left[\mathsf{off\text{-}dia}\text{-}2 \cdot \max(k_1, k_2)\right] \leqslant \mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}dia}\text{-}k_1\right] + \mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}dia}\text{-}k_2\right] .$$

We prove the three following claims.

Claim 11.4.20. *For any $k_1 \in \mathbb{N}_{>0}$, such that k_1 is a multiple of 2*

$$\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}dia}\text{-}k_1\right] \leqslant \left(\frac{64e^3T_1^4}{k_1^2 2^{3n}}\right)^{k_1/2} + \left(\frac{4en^2T_1^2}{k_1^2 2^n}\right)^{k_1/2} + \frac{1}{2^n} .$$

Claim 11.4.21. *For any $k_2 \in \mathbb{N}_{>0}$, such that k_2 is a multiple of 30*

$$\begin{aligned}\mathsf{Pr}\left[\text{off-2-dia-}k_2\right] \leqslant \left(\frac{1600eT_1^4}{(k_2)^2 2^{3n}}\right)^{k_2/10} + \left(\frac{100en^2T_1^2}{(k_2)^2 2^n}\right)^{k_2/10} + 3\left(\frac{32e(60e)^3T_1^6}{k_2^3 2^{4n}}\right)^{k_2/30} \\ + 3\left(\frac{256\cdot 5^3\cdot enT_1^3}{k_2^3 2^n}\right)^{k_2/30} + \frac{4}{2^n}\,.\end{aligned}$$

We let $k_1 = \max\left(\left(\frac{128e^3T_1^4}{2^{3n}}\right)^{1/2}, \left(\frac{8en^2T_1^2}{2^n}\right)^{1/2}, 2n\right)$. Then we have that

$$\mathsf{Pr}\left[\text{off-1-dia-}k_1\right] \leqslant \frac{3}{2^n}\,.$$

We let

$$k_2 = \max\left(\left(\frac{3200eT_1^4}{2^{3n}}\right)^{1/2}, \left(\frac{200en^2T_1^2}{2^n}\right)^{1/2}, \left(\frac{64e(60e)^3T_1^6}{2^{4n}}\right)^{1/3}, \left(\frac{512\cdot 5^3\cdot enT_1^3}{2^n}\right)^{1/3}, 30n\right)\,.$$

Then we have that

$$\mathsf{Pr}\left[\text{off-1-dia-}k_2\right] \leqslant \frac{12}{2^n}\,.$$

Also

$$\begin{aligned}\max(k_1,k_2) &\leqslant \left(\frac{128e^3T_1^4}{2^{3n}}\right)^{1/2} + \left(\frac{8en^2T_1^2}{2^n}\right)^{1/2} + \left(\frac{3200eT_1^4}{2^{3n}}\right)^{1/2} + \left(\frac{200en^2T_1^2}{2^n}\right)^{1/2} \\ &\quad + \left(\frac{64e(60e)^3T_1^6}{2^{4n}}\right)^{1/3} + \left(\frac{512\cdot 5^3\cdot enT_1^3}{2^n}\right)^{1/3} + 30n \\ &= \left(\frac{8(2)^{1/2}e^{3/2}T_1^2}{2^{3n/2}}\right) + \left(\frac{(8e)^{1/2}nT_1}{2^{n/2}}\right) + \left(\frac{40(2e)^{1/2}T_1^2}{2^{3n/2}}\right) + \left(\frac{10(2e)^{1/2}nT_1}{2^{n/2}}\right) \\ &\quad + \left(\frac{240e^{2/3}T_1^2}{2^{4n/3}}\right) + \left(\frac{40(en)^{1/3}T_1}{2^{n/3}}\right) + 30n\,.\end{aligned}$$

Let $k = 2\max(k_1,k_2)$. We have that

$$\mathsf{Pr}\left[\text{off-dia-}k\right] \leqslant \frac{15}{2^n}\,.$$

Plugging this into (11.60) we have that

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{dia}_{\mathsf{off}}\right] &\leqslant \left(\frac{8(2)^{1/2}e^{3/2}T_1^2}{2^{5n/2}}\right) + \left(\frac{(8e)^{1/2}nT_1}{2^{3n/2}}\right) + \left(\frac{40(2e)^{1/2}T_1^2}{2^{5n/2}}\right) + \left(\frac{10(2e)^{1/2}nT_1}{2^{3n/2}}\right) \\ &\quad + \left(\frac{240e^{2/3}T_1^2}{2^{7n/3}}\right) + \left(\frac{40(en)^{1/3}T_1}{2^{4n/3}}\right) + \frac{30n}{2^n} + \frac{15}{2^n}\,. \\ &= \frac{30n+15}{2^n} + \frac{40(en)^{1/3}T_1}{2^{4n/3}} + \frac{(8e)^{1/2}nT_1 + 10(2e)^{1/2}nT_1}{2^{3n/2}} \\ &\quad + \frac{240e^{2/3}T_1^2}{2^{7n/3}} + \frac{8(2)^{1/2}e^{3/2}T_1^2 + 40(2e)^{1/2}T_1^2}{2^{5n/2}}\,.\end{aligned}$$

We now prove these claims one by one.

UPPER BOUNDING $\mathsf{Pr}\left[\text{off-1-dia-}k_1\right]$. We have that for any m,

$$\mathsf{Pr}\left[\text{off-1-dia-}k_1\right] \leqslant \mathsf{Pr}\left[\text{off-1-dia-}k_1 \wedge \neg(m+1)\text{-col}\right] + \mathsf{Pr}\left[m+1\text{-col}\right] \ . \tag{11.61}$$

We upper bound $\mathsf{Pr}\left[\text{off-1-dia-}k_1 \wedge \neg(m+1)\text{-col}\right]$ using a compression argument. Before giving the compression argument, we recall the notion of successor-hitting queries we introduced earlier in the proof: we say that a query $q = \mathrm{h}(y, M) = z$ made by $\mathcal{A}_1$ is a successor-hitting query for a salt a if

- $z \neq a, y \neq a, y \neq z$
- there are queries $q' = \mathrm{h}(a, M') = b$, $q'' = \mathrm{h}(b, M'') = z$ with $b \neq a, z \neq b$ made by $\mathcal{A}_1$, for some M', M''

Informally, a query q is a successor-hitting query to salt a if there query $q' = \mathrm{h}(y', M') = z$ such that q' is a successor query to a and $q' \neq q$.

The encoding procedure encodes the random oracle h as follows.

1. It first runs $\mathcal{A}_1^{\mathrm{h}}$, answering all its queries using h.
2. If $m+1$-col happens or off-1-dia-k_1 does not happen, it outputs $\varnothing$
3. It finds a set of salts S of size $k_1/2$ such that for each $a \in S$
   - $\mathcal{A}_1$ found a category 1 triangle for a
   - The salt a has no more than $2m^2T_1/k_1$ successor-hitting queries
4. If no such S is found, return $\varnothing$
5. It marks the q_2, q_3 queries corresponding to the triangle for each salt in S
6. It initializes lists $\mathsf{L}_1, \mathsf{L}_2$ to empty lists, T to empty set
7. It starts running $\mathcal{A}_1$ again
8. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:
   (a) If the query is marked as a q_3 query for a salt $a \in S$, then it adds the index of this query in T, and it adds the lexicographical index of q_2 among all *successor-hitting* queries for the salt a in the list L_2

(b) Otherwise it adds $\mathrm{h}(a, M)$ to L_1.

9. It appends the evaluation of h on the points not queried by $\mathcal{A}_1$ to L_1 in the lexicographical order of the inputs.

10. It outputs $\mathsf{L}_1, \mathsf{L}_2, \mathsf{T}$.

The decoding procedure works as follows.

1. If the encoding is $\varnothing$ it aborts.

2. It runs $\mathcal{A}_1^{\mathrm{h}}$.

3. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:

   (a) If the index of the query is in T, it removes the element i on the front of list L_2, and locates the query q which has lexicographic order i among all the successor-hitting queries of the salt a. It answers with the answer of q

   (b) Otherwise it removes the element in front of L_1 and answers with that.

4. It populates h on the points not queried by $\mathcal{A}_1$ in the lexicographical order by the remaining entries of L_1

Correctness of decoding: First off, we argue that for adversary $\mathcal{A}_1$ that causes the event $\mathsf{off\text{-}1\text{-}dia}\text{-}k_1 \wedge \neg(m+1)\text{-}\mathsf{col}$ to happen, the encoding algorithm will never return $\varnothing$. To argue this it suffices to show that the encoding algorithm never returns $\varnothing$ from line 4. Note that the encoding algorithm reaches this line only if the event $\mathsf{off\text{-}1\text{-}dia}\text{-}k_1 \wedge \neg(m+1)\text{-}\mathsf{col}$ was caused by running $\mathcal{A}_1$, which means there is a set of k_1 salts S' such that $\mathcal{A}_1$ found category one diamonds for them. Now, since $\mathcal{A}_1$ makes at most T_1 queries, and $\neg(m+1)\text{-}\mathsf{col}$ happens, there are at most $m^2 T_1$ salt-successor-hitting query pairs. This is because if there were more than $m^2 T_1$ salt-successor hitting query pairs, there would be at least one query which is a successor hitting query to $m^2 + 1$ salts. Given that there are no $m+1$-multi-collisions, this would be contradiction.

Therefore, for at least $k_1/2$ salts (note that since k_1 is a multiple of 2, $k_1/2$ is an integer) in S' such that it has at most $2T_1 m^2/k_1$ successor queries (because otherwise there would be more than $m^2 T_1$ salt, successor-hitting query pairs). Therefore, such a set $S \subseteq S'$ of size $k_1/2$ always exists.

If the encoding algorithm does not return $\varnothing$, it is easy to see the decoding algorithm decodes correctly, because the query answers that the encoding algorithm does not add in the list L_1 can be recovered by the decoding algorithm correctly using S, L_2. Therefore, we have that

$$\mathsf{Pr}\left[\text{ Decoding is correct }\right] \geqslant \mathsf{Pr}\left[\text{off-1-dia-}k_1 \wedge \neg(m+1)\text{-col}\right] .$$

Observe that the encoding algorithm removes $k_1/2$ answers of the random oracle from the encoding, and all the removed answers are distinct because those were q_3 queries for different salts. It instead adds an unordered set T of $k_1/2$ values in $[T_1]$, and an ordered list of $k_1/2$ elements where each value is at most $2m^2T_1/k_1$. Using the compression lemma, we have that

$$\mathsf{Pr}\left[\text{ Decoding is correct }\right] \leqslant \frac{(2m^2T_1/k_1)^{k_1/2}\binom{T_1}{k_1/2}}{2^{n\,k_1/2}} \leqslant \left(\frac{4em^2T_1^2}{k_1^2 2^n}\right)^{k_1/2} .$$

Therefore,

$$\mathsf{Pr}\left[\text{off-1-dia-}k_1 \wedge \neg(m+1)\text{-col}\right] \leqslant \left(\frac{4em^2T_1^2}{k_1^2 2^n}\right)^{k_1/2} .$$

We know from previous analyses that $\mathsf{Pr}\left[(m+1)\text{-col}\right] \leqslant 1/2^n$ for $m = \max\left(n, \frac{4eT_1}{2^n}\right)$. Plugging this value of m we have

$$\mathsf{Pr}\left[\text{off-1-dia-}k_1 \wedge \neg(m+1)\text{-col}\right] \leqslant \left(\frac{64e^3T_1^4}{k_1^2 2^{3n}}\right)^{k_1/2} + \left(\frac{4en^2T_1^2}{k_1^2 2^n}\right)^{k_1/2} .$$

Plugging this into (11.61), we get that

$$\mathsf{Pr}\left[\text{off-1-dia-}k_1\right] \leqslant \left(\frac{64e^3T_1^4}{k_1^2 2^{3n}}\right)^{k_1/2} + \left(\frac{4en^2T_1^2}{k_1^2 2^n}\right)^{k_1/2} + \frac{1}{2^n} . \tag{11.62}$$

We next upper bound $\mathsf{Pr}\left[\text{off-2-dia-}k_2\right]$. One could hope for an analysis similar to what we did for $\mathsf{Pr}\left[\text{off-dia-}k_1\right]$– with the difference that in the compression argument, the encoding algorithm omits the answer of the last query and instead stores information like the set S, L_2 to recover its answers. However, this approach does not work! The main reason for this is if the adversary finds diamonds of category two for k_2 different salts, it is not necessary that the last queries for the diamond are unique for each of the salts. Therefore, we need to separately handle the case where a particular query is the last query for diamonds of category two of multiple salts.

Now, consider how two salts with category two diamonds can share the last query. There are two possibilities here: either the diamonds share both the queries q_2, q_4 queries or they share one of the queries q_2, q_4. If they share both the queries q_2, q_4, we have that the pair of salts form a zig-zag, and we can re-use our analysis from Lemma 11.4.11. The other possibility is both the category two diamonds share one query that was the last query for both of them. For this to happen for salts a, a', before the shared query is made, there must have been queries

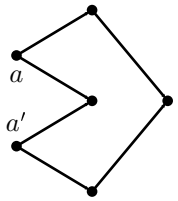


Figure 11.14: The hexagon structure for a salt pair (a', a') in the query graph.

- $q_1 := \mathrm{h}(a, M_1) = w$ for some w, M_1 such that $w \neq a, a'$

- $q_2 := \mathrm{h}(a, M_2) = y$ for some y, M_2 such that $y \neq w, a, a'$

- $q_3 := \mathrm{h}(y, M_3) = z$ for some z, M_3 such that $z \neq a, a', w, y$

- $q_4 := \mathrm{h}(a', M_4) = w$ for some M_4

- $q_5 := \mathrm{h}(a', M_5) = y'$ for some y', M_5 such that $y' \neq w, a, a', y, z$

- $q_6 := \mathrm{h}(y', M_6) = z$ for some M_6

We refer to such a structure as a *hexagon* because this structure consists of a total of six queries and two of these six collide. If for some salt pair $\{a, a'\}$, $\mathcal{A}_1$ has made the above queries, we say that it has found a hexagon for the salt pair. We illustrate the structure of a hexagon in a query graph in Fig. 11.14.

We define the event $\mathsf{off\text{-}hex}\text{-}k$ as follows: there is a set of k salt pairs $\{\{a_{i,1}, a_{i,2}\}_{i\in[k]}\}$ such that

1. $|\cup_{i\in[k]} \{a_{i,1}, a_{i,2}\}| = 2k$

2. $\mathcal{A}_1$ has found a hexagon for each pair $\{a_{i,1}, a_{i,2}\}$ for all $i \in [k]$.

We have that for all k_3, k_4, m

$$\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}dia}\text{-}k_2\right] \leqslant \mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}dia}\text{-}k_2 \wedge \neg\mathsf{off\text{-}zz}\text{-}k_3 \wedge \neg\mathsf{off\text{-}hex}\text{-}k_4 \wedge \neg(m+1)\text{-}\mathsf{col}\right] \tag{11.63}$$

$$+ \mathsf{Pr}\left[\mathsf{off\text{-}zz}\text{-}k_3\right] + \mathsf{Pr}\left[\mathsf{off\text{-}hex}\text{-}k_4\right] + \mathsf{Pr}\left[m+1\text{-}\mathsf{col}\right] \ . \tag{11.64}$$

We prove the following lemmas.

Lemma 11.4.22. *For $k \in \mathbb{N}_{>0}$ such that k is a multiple of 6*

$$\mathsf{Pr}\left[\mathsf{off\text{-}hex}\text{-}k\right] \leqslant 3\left(\frac{32e(12e)^3T_1^6}{k^3 2^{4n}}\right)^{k/6} + 3\left(\frac{256enT_1^3}{k^3 2^n}\right)^{k/6} + \frac{3}{2^n} \ . \tag{11.65}$$

Lemma 11.4.23. *For any $k, k', k'', m \in \mathbb{N}_{>0}$ such that k is a multiple of 2, $k > 2k' + 2k''$,*

$$\begin{aligned}&\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}dia}\text{-}k \wedge \neg\mathsf{off\text{-}zz}\text{-}k' \wedge \neg\mathsf{off\text{-}hex}\text{-}k'' \wedge \neg(m+1)\text{-}\mathsf{col}\right]\\ &\leqslant \left(\frac{4em^2T_1^2}{(k-2k'-2k'')^2 2^n}\right)^{(k-2k'-2k'')/2} .\end{aligned}$$

From Lemma 11.4.18 we had that for $k \in \mathbb{N}_{>0}$,

$$\mathsf{Pr}\left[\mathsf{off\text{-}zz}\text{-}k\right] \leqslant \left(\frac{e^3T_1^4}{k^3 2^{2n}}\right)^k .$$

Plugging this into (11.63), we have that for any k_2, k_3, k_4, m such that k_4 is a multiple of 6, k_2 is a multiple of 2, $k > 2k_3 + k_4$

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}dia}\text{-}k_2\right] \leqslant &\left(\frac{4em^2T_1^2}{(k_2-2k_3-2k_4)^2 2^n}\right)^{(k_2-2k_3-2k_4)/2}\\ &+ 3\left(\frac{32e(12e)^3T_1^6}{k_4^3 2^{4n}}\right)^{k_4/6} + 3\left(\frac{256enT_1^3}{k_4^3 2^n}\right)^{k_4/6}\\ &+ \frac{3}{2^n} + \mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] .\end{aligned}$$

Setting $k_4 = k_3 = k_2/5$, we have

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}dia}\text{-}k_2\right] \leqslant &\left(\frac{100em^2T_1^2}{(k_2)^2 2^n}\right)^{k_2/10} + 3\left(\frac{32e(60e)^3T_1^6}{k_2^3 2^{4n}}\right)^{k_2/30}\\ &+ 3\left(\frac{256\cdot 5^3\cdot enT_1^3}{k_2^3 2^n}\right)^{k_2/30} + \frac{3}{2^n} + \mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] .\end{aligned}$$

Further in the proof of Lemma 11.4.10 we had that for $m = \max(n, 4eT_1/2^n)$, $\mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \leqslant 1/2^n$. Setting m to this value we have that for any k_2 such that k_2 is a multiple of 30,

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}dia}\text{-}k_2\right] \leqslant &\left(\frac{1600eT_1^4}{(k_2)^2 2^{3n}}\right)^{k_2/10} + \left(\frac{100en^2T_1^2}{(k_2)^2 2^n}\right)^{k_2/10}\\ &+ 3\left(\frac{32e(60e)^3T_1^6}{k_2^3 2^{4n}}\right)^{k_2/30} + 3\left(\frac{256\cdot 5^3\cdot enT_1^3}{k_2^3 2^n}\right)^{k_2/30} + \frac{4}{2^n} .\end{aligned}$$

■

We prove Lemma 11.4.22.

Proof. [Lemma 11.4.22] For the adversary to query a hexagon for salts a, a', there must have been queries

- $q_1 := \mathrm{h}(a, M_1) = w$ for some w, M_1 such that $w \neq a, a'$
- $q_2 := \mathrm{h}(a, M_2) = y$ for some y, M_2 such that $y \neq w, a, a'$

- $q_3 := \mathrm{h}(y, M_3) = z$ for some z, M_3 such that $z \neq a, a', w, y$

- $q_4 := \mathrm{h}(a', M_4) = w$ for some M_4

- $q_5 := \mathrm{h}(a', M_5) = y'$ for some y', M_5 such that $y' \neq w, a, a', y, z$

- $q_6 := \mathrm{h}(y', M_6) = z$ for some M_6

We categorize hexagons into three categories based on the query that was the last one among the six

1. category 1 hexagons: q_3 or q_6 is the last query

2. category 2 hexagons: q_2 or q_5 is the last query

3. category 3 hexagons: q_1 or q_4 is the last query

We illustrate the different categories of hexagons in Figure 11.15.

We define these three events.

1. $\mathsf{off\text{-}1\text{-}hex}$-$k_1$: $\mathcal{A}_1$ has found hexagons of category 1 for a set of k_1 salt pairs such that none of the two salt pairs in the set share a salt

2. $\mathsf{off\text{-}2\text{-}hex}$-$k_2$: $\mathcal{A}_1$ has found hexagons of category 2 for a set of k_1 salt pairs such that none of the two salt pairs in the set share a salt

3. $\mathsf{off\text{-}3\text{-}hex}$-$k_3$: $\mathcal{A}_1$ has found hexagons of category 3 for a set of k_1 salt pairs such that none of the two salt pairs in the set share a salt

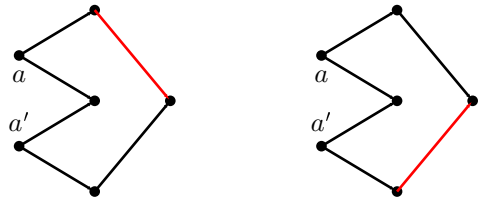
Since if $\mathcal{A}_1$ finds $3k$ hexagons, it must have found k hexagons of one of the categories, we have that for any k

$$\mathsf{Pr}\left[\mathsf{off\text{-}hex}\text{-}3k\right] \leqslant \mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}hex}\text{-}k\right] + \mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}hex}\text{-}k\right] + \mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}hex}\text{-}k\right] \ .$$

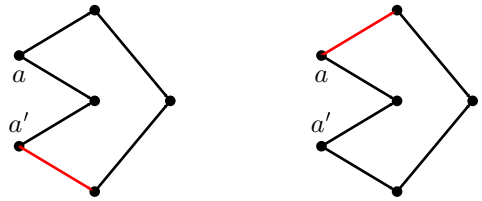
We prove the three following claims.

Claim 11.4.24. *For any $k \in \mathbb{N}_{>0}$ such that k is a multiple of 2*

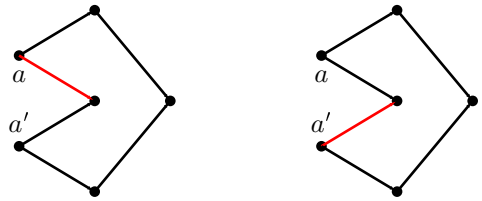
$$\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}hex}\text{-}k\right] \leqslant \left(\frac{32e(4e)^3T_1^6}{k^3 2^{4n}}\right)^{k/2} + \left(\frac{32enT_1^3}{k^3 2^n}\right)^{k/2} + \frac{1}{2^n} \ .$$



(a) Category 1 hexagons.



(b) Category 2 hexagons.



(c) Category 3 hexagons.

Figure 11.15: The different categories of hexagons. The edge colored red is the last of the six queries made.

Claim 11.4.25. *For any $k \in \mathbb{N}_{>0}$ such that k is a multiple of 2*

$$\mathsf{Pr}\left[\text{off-2-hex-}k\right] \leqslant \left(\frac{32e(4e)^3T_1^6}{k^3 2^{4n}}\right)^{k/2} + \left(\frac{32enT_1^3}{k^3 2^n}\right)^{k/2} + \frac{1}{2^n}\,.$$

Claim 11.4.26. *For any $k \in \mathbb{N}_{>0}$ such that k is a multiple of 2*

$$\mathsf{Pr}\left[\text{off-3-hex-}k\right] \leqslant \left(\frac{32e(4e)^3T_1^6}{k^3 2^{4n}}\right)^{k/2} + \left(\frac{32enT_1^3}{k^3 2^n}\right)^{k/2} + \frac{1}{2^n}\,.$$

Putting this all together we have for any k that is a multiple of 2

$$\mathsf{Pr}\left[\text{off-hex-}3k\right] \leqslant 3\left(\frac{32e(4e)^3T_1^6}{k^3 2^{4n}}\right)^{k/2} + 3\left(\frac{32enT_1^3}{k^3 2^n}\right)^{k/2} + \frac{3}{2^n}\,.$$

Equivalently, we have that for any k that is a multiple of 6

$$\mathsf{Pr}\left[\text{off-hex-}k\right] \leqslant 3\left(\frac{32e(12e)^3T_1^6}{k^3 2^{4n}}\right)^{k/6} + 3\left(\frac{256enT_1^3}{k^3 2^n}\right)^{k/6} + \frac{3}{2^n}\,.$$

UPPER BOUNDING $\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}hex\text{-}}k\right]$. We have that for any m,

$$\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}hex\text{-}}k\right] \leqslant \mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}hex\text{-}}k \wedge \neg(m+1)\text{-}\mathsf{col}\right] + \mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \ . \tag{11.66}$$

We upper bound $\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}hex\text{-}}k \wedge \neg(m+1)\text{-}\mathsf{col}\right]$ using a compression argument. The encoding procedure encodes the random oracle h as follows.

1. It runs $\mathcal{A}_1^{\mathrm{h}}$, answering all its queries using h.

2. If $m+1$-col happens or $\mathsf{off\text{-}1\text{-}hex\text{-}}k$ does not happen, it outputs $\varnothing$

3. it finds a set of salt pairs S of size $k/8$ such that for each $\{a, a'\} \in S$

   (a) $\mathcal{A}_1$ found a hexagon for $\{a', a''\}$

   (b) $\mathcal{A}_1$ made no more than $4T_1/k$ queries with salt a'

   (c) $\mathcal{A}_1$ made no more than $4T_1/k$ queries with salt a''

   (d) The salt a has no more than $4mT_1/k$ successor queries

   (e) The salt a has no more than $4mT_1/k$ successor queries

4. If no such S is found, it returns $\varnothing$

5. For every $\{a', a''\}$ pair in S it picks one type 1 hexagon formed by the queries of $\mathcal{A}_1$ (if there is more than one, it picks one arbitrarily). It lets the queries forming the hexagon be $q_1 = \mathrm{h}(a', M_1) = w, q_2 = \mathrm{h}(a, M_2) = y, q_3 = \mathrm{h}(y, M_3) = z, q_4 = \mathrm{h}(a'', M_4) = w, q_5 = \mathrm{h}(a', M_5) = y', q_6 = \mathrm{h}(y', M_6) = z$ for some $M_1, M_2, M_3, M_4, M_5, M_6, w, y, y', z$ such that $|\{a', a'', w, y, y', z\}| = 6$ such that q_6 was the last of the six queries made by $\mathcal{A}_1$.

6. It initializes lists $\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3, \mathsf{L}_4, \mathsf{L}_5$ to empty lists, T to empty set

7. It starts running $\mathcal{A}_1$ again

8. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:

   (a) If the query is marked as a q_6 query for a salt pair $\{a, a'\} \in S$

      i. it adds the index of this query in T

      ii. it adds the lexicographical index of the corresponding q_5 query among all queries with answer a to the list L_1

iii. Let the input salt of the corresponding q_5 query be a''. It adds the lexicographical index of the corresponding q_4 query among all queries with input salt a'' to the list L_2

iv. Let the answer of the corresponding q_4 query be w. It adds the lexicographical index of the corresponding q_1 query among all queries with answer w to the list L_3

v. Let the input salt of the corresponding q_1 query be a'. It adds the lexicographical index of the corresponding q_3 query among all queries which are successor queries of the salt a' to the list L_4.

(b) Otherwise it adds $\mathrm{h}(a, M)$ to L_5.

9. It appends the evaluation of h on the points not queried by $\mathcal{A}_1$ to L_5 in the lexicographical order of the inputs.

10. It outputs $\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3, \mathsf{L}_4, \mathsf{L}_5, \mathsf{T}$.

The decoding procedure works as follows.

1. If the encoding is $\varnothing$ it aborts.

2. It runs $\mathcal{A}_1^{\mathrm{h}}$.

3. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:

(a) If the index of the query is in T,

i. it removes the element i on the front of list L_1, and locates the query q_5 which has lexicographic order i among all queries with answer a.

ii. Let the input of q_5 be a''. It removes the element i on the front of list L_2, and locates the query q_4 which has lexicographic order i among all queries with input salt a''.

iii. Let the answer of q_4 be w. It removes the element i on the front of list L_3, and locates the query q_1 which has lexicographic order i among all queries with answer w.

iv. Let the input salt of q_1 be a'. It removes the element i on the front of list L_4, and locates the query q_3 which has lexicographic order i among all successor queries of the salt a'

It answers with the answer of query q_3.

(b) Otherwise it removes the element in front of L_5 and answers with that.

4. It populates h on the points not queried by $\mathcal{A}_1$ in the lexicographical order by the remaining entries of L_5

Correctness of decoding: First off, we argue that for an adversary $\mathcal{A}_1$ that causes the event $\mathsf{off\text{-}1\text{-}hex}\text{-}k \wedge \neg(m+1)\text{-}\mathsf{col}$ to happen, the encoding algorithm will never return $\varnothing$. To argue this it suffices to show that the encoding algorithm never returns $\varnothing$ from line 4. Since $\mathsf{off\text{-}1\text{-}hex}\text{-}k$ happens, there is a set S' of k salt pair such that no two pairs share a salt and $\mathcal{A}_1$ has found a hexagon for each pair. We first claim that at most $k/4$ of the pairs in S' are such that $\mathcal{A}_1$ has made more than $4T_1/k$ queries on one or both the salts in the pair- this is because otherwise there would need to be more than T_1 queries which is a contradiction. Further, since $\neg(m+1)\text{-}\mathsf{col}$ happens, we have that there are at most mT_1 salt-successor query pairs – this was argued in the proof of 11.4.11. We next claim that at most $k/4$ of the pairs in S' are such that for one or both salts in the pair- there are more than $4mT_1/k$ successor queries – this is because otherwise there would be more than mT_1 salt-successor query pairs.

Therefore, we have that

- The subset of S' such that at least one of the salts is such that there are more than $4T_1/k$ queries on the salt is of size at most $k/4$

- The subset of S' such that at least one of the salts is such that there are more than $4mT_1/k$ successor queries for the salt is of size at most $k/4$

Therefore, this implies the subset of S' such that for at least one of the salts **a)** there are more $4T_1/k$ queries on the salt or, **b)** there are more than $4mT_1/k$ queries that are successor query to the salt is of size at most $k/2$. This in turn means the subset of S' such that for both of the salts **a)** there are at most $4T_1/k$ queries on the salt and, **b)** there are at most $4mT_1/k$ queries that are successor query to the salt is of size at least $k/2$. Therefore, the encoding algorithm will always find such a set S, and never produce $\varnothing$ in line 4.

Further, we argue that whenever the decoding algorithm receives an output of the encoding algorithm that is not $\varnothing$, the decoding algorithm will decode correctly. It is easy to see decoding answers the queries whose answers were in L_5. What is left is verifying that the queries whose answers were not in L_5 and whose indices were put in T were answered correctly.

This can be verified by inspecting line 8a in the encoding algorithm and line 3a in the decoding algorithm. It can be checked that the decoding algorithm always returns the same answer as the encoding algorithm. Therefore, whenever $\mathcal{A}_1$ causes $\mathsf{off\text{-}1\text{-}hex}\text{-}k \wedge \neg(m+1)\text{-}\mathsf{col}$, the encoding produces an outputs that decodes correctly.

Therefore, we have that

$$\mathsf{Pr}\left[\text{ Decoding is correct }\right] \geqslant \mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}hex}\text{-}k \wedge \neg(m+1)\text{-}\mathsf{col}\right] .$$

Observe that the encoding algorithm removes $k/2$ answers of the random oracle from the encoding, and all the removed answers are distinct because those were queries for different salts. It instead adds

1. T: a set of $k/2$ distinct values in $[T_1]$
2. L_1: a sequence of $k/2$ values that are each at most m
3. L_2: a sequence of $k/2$ values that are each at most $4T_1/k$
4. L_3: a sequence of $k/2$ values that are each at most m
5. L_4: a sequence of $k/2$ values that are each at most $4mT_1/k$

Using the compression lemma, we have that

$$\begin{aligned}
\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}hex}\text{-}k \wedge \neg(m+1)\text{-}\mathsf{col}\right] \leqslant & \mathsf{Pr}\left[\text{ Decoding is correct }\right] \\
& \leqslant \frac{\binom{T_1}{k/2} m^{k/2} \left(\frac{4T_1}{k}\right)^{k/2} m^{k/2} \left(\frac{4mT_1}{k}\right)^{k/2}}{2^{nk/2}} \\
& \leqslant \left(\frac{(2eT_1)(4mT_1)(4T_1)m^2}{k^3 2^n}\right)^{k/2} \\
& = \left(\frac{32eT_1^3 m^3}{k^3 2^n}\right)^{k/2} .
\end{aligned}$$

We know from proof of Lemma 11.4.10, that for $m = \max(4eT_1/2^n, n)$, $\mathsf{Pr}\left[(m+1)\text{-}\mathsf{col}\right] \leqslant 1/2^n$. We set m to this value and plug this into (11.66).

$$\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}hex}\text{-}k\right] \leqslant \left(\frac{32e(4e)^3 T_1^6}{k^3 2^{4n}}\right)^{k/2} + \left(\frac{32enT_1^3}{k^3 2^n}\right)^{k/2} + \frac{1}{2^n} .$$

Upper bounding $\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}hex}\text{-}k\right]$. The analysis for this is identical to that for $\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}hex}\text{-}k\right]$ with a slightly modified compression strategy (but one that gives identical amount of compression). For the sake of completeness, we just give the modified encoding and decoding algorithms here.

The encoding procedure encodes the random oracle h as follows.

1. It runs $\mathcal{A}_1^{\mathrm{h}}$, answering all its queries using h.
2. If $m+1$-col happens or $\mathsf{off\text{-}hex}$-k does not happen, it outputs $\varnothing$

3. it finds a set of salt pairs S of size $k/8$ such that for each $\{a, a'\} \in S$

   - $\mathcal{A}_1$ found a hexagon for $\{a', a''\}$
   - $\mathcal{A}_1$ made no more than $4T_1/k$ queries with salt a'
   - $\mathcal{A}_1$ made no more than $4T_1/k$ queries with salt a''
   - The salt a has no more than $4mT_1/k$ successor queries
   - The salt a has no more than $4mT_1/k$ successor queries

4. If no such S is found, it returns $\varnothing$

5. For every $\{a', a''\}$ pair in S it picks one type 1 hexagon formed by the queries of $\mathcal{A}_1$ (if there is more than one, it picks one arbitrarily). It lets the queries forming the hexagon be $q_1 = \mathrm{h}(a', M_1) = w, q_2 = \mathrm{h}(a, M_2) = y, q_3 = \mathrm{h}(y, M_3) = z, q_4 = \mathrm{h}(a'', M_4) = w, q_5 = \mathrm{h}(a', M_5) = y', q_6 = \mathrm{h}(y', M_6) = z$ for some $M_1, M_2, M_3, M_4, M_5, M_6, w, y, y', z$ such that $|\{a', a'', w, y, y', z\}| = 6$ such that q_5 was the last of the six queries made by $\mathcal{A}_1$.

6. It initializes lists $\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3, \mathsf{L}_4, \mathsf{L}_5$ to empty lists, T to empty set

7. It starts running $\mathcal{A}_1$ again

8. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:

   (a) If the query is marked as a q_5 query for a salt pair $\{a, a'\} \in S$

      i. it adds the index of this query in T
      ii. it adds the lexicographical index of the corresponding q_5 query among all queries with input a to the list L_1
      iii. Let the answer of the corresponding q_4 query be w. It adds the lexicographical index of the corresponding q_1 query among all queries with answer w to the list L_2
      iv. Let the input salt of the corresponding q_1 query be a'. It adds the lexicographical index of the corresponding q_3 query among all successor queriesof a' to the list L_3
      v. Let the answer of the corresponding q_3 query be z. It adds the lexicographical index of the corresponding q_6 query among all queries with answer z to the list L_4.

   (b) Otherwise it adds $\mathrm{h}(a, M)$ to L_5.

9. It appends the evaluation of h on the points not queried by $\mathcal{A}_1$ to L_5 in the lexicographical order of the inputs.

10. It outputs $\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3, \mathsf{L}_4, \mathsf{L}_5, \mathsf{T}$.

The decoding procedure works as follows.

1. If the encoding is $\varnothing$ it aborts.

2. It runs $\mathcal{A}_1^{\mathrm{h}}$.

3. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:

   (a) If the index of the query is in T,

      i. it removes the element i on the front of list L_1, and locates the query q_4 which has lexicographic order i among all queries with input salt a.

      ii. Let the answer of q_4 be w. It removes the element i on the front of list L_2, and locates the query q_1 which has lexicographic order i among all queries with answer w.

      iii. Let the input salt of q_1 be a'. It removes the element i on the front of list L_3, and locates the query q_3 which has lexicographic order i among all successor queries of a'.

      iv. Let the answer of q_3 be z. It removes the element i on the front of list L_4, and locates the query q_6 which has lexicographic order i among all queries with answer z.

      It answers with the input of query q_6.

   (b) Otherwise it removes the element in front of L_5 and answers with that.

4. It populates h on the points not queried by $\mathcal{A}_1$ in the lexicographical order by the remaining entries of L_5

Upper bounding $\mathsf{Pr}\left[\mathsf{off\text{-}3\text{-}hex}\text{-}k\right]$. The analysis for this is identical to that for $\mathsf{Pr}\left[\mathsf{off\text{-}1\text{-}hex}\text{-}k\right]$ with a slightly modified compression strategy (but one that gives identical amount of compression). For the sake of completeness, we just give the modified encoding and decoding algorithms here.

The encoding procedure encodes the random oracle h as follows.

1. It runs $\mathcal{A}_1^{\mathrm{h}}$, answering all its queries using h.

2. If $m+1$-col happens or $\mathsf{off\text{-}hex}$-k does not happen, it outputs $\varnothing$

3. it finds a set of salt pairs S of size $k/8$ such that for each $\{a, a'\} \in S$

   - $\mathcal{A}_1$ found a hexagon for $\{a', a''\}$
   - $\mathcal{A}_1$ made no more than $4T_1/k$ queries with salt a'
   - $\mathcal{A}_1$ made no more than $4T_1/k$ queries with salt a''
   - The salt a has no more than $4mT_1/k$ successor queries
   - The salt a has no more than $4mT_1/k$ successor queries

4. If no such S is found, it returns $\varnothing$

5. For every $\{a', a''\}$ pair in S it picks one type 1 hexagon formed by the queries of $\mathcal{A}_1$ (if there is more than one, it picks one arbitrarily). It lets the queries forming the hexagon be $q_1 = \mathrm{h}(a', M_1) = w, q_2 = \mathrm{h}(a, M_2) = y, q_3 = \mathrm{h}(y, M_3) = z, q_4 = \mathrm{h}(a'', M_4) = w, q_5 = \mathrm{h}(a', M_5) = y', q_6 = \mathrm{h}(y', M_6) = z$ for some $M_1, M_2, M_3, M_4, M_5, M_6, w, y, y', z$ such that $|\{a', a'', w, y, y', z\}| = 6$ such that q_4 was the last of the six queries made by $\mathcal{A}_1$.

6. It initializes lists $\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3, \mathsf{L}_4, \mathsf{L}_5$ to empty lists, T to empty set

7. It starts running $\mathcal{A}_1$ again

8. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:

   (a) If the query is marked as a q_4 query for a salt pair $\{a, a'\} \in S$

      i. it adds the index of this query in T
      ii. it adds the lexicographical index of the corresponding q_6 query among all succesor queries of a to the list L_1
      iii. Let the answer of the corresponding q_6 query be z. It adds the lexicographical index of the corresponding q_3 query among all queries with answer z the list L_2
      iv. Let the input salt of the corresponding q_3 query be y'. It adds the lexicographical index of the corresponding q_2 query among all queries with answer y' to the list L_3
      v. Let the input salt of the corresponding q_2 query be a'. It adds the lexicographical index of the corresponding q_1 query among all queries with input salt a' to the list L_4.

   (b) Otherwise it adds $\mathrm{h}(a, M)$ to L_5.

9. It appends the evaluation of h on the points not queried by $\mathcal{A}_1$ to L_5 in the lexicographical order of the inputs.

10. It outputs $\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3, \mathsf{L}_4, \mathsf{L}_5, \mathsf{T}$.

The decoding procedure works as follows.

1. If the encoding is $\varnothing$ it aborts.

2. It runs $\mathcal{A}_1^{\mathrm{h}}$.

3. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:

   (a) If the index of the query is in T,

      i. it removes the element i on the front of list L_1, and locates the query q_6 which has lexicographic order i among all successor queries to a.

      ii. Let the answer q_6 be z. It removes the element i on the front of list L_2, and locates the query q_3 which has lexicographic order i among all queries with answer z.

      iii. Let the input salt of q_3 be y. It removes the element i on the front of list L_3, and locates the query q_2 which has lexicographic order i among all queries with answer y.

      iv. Let the input salt of q_2 be a'. It removes the element i on the front of list L_4, and locates the query q_1 which has lexicographic order i among all queries with input salt a'

   It answers with the answer of query q_1.

   (b) Otherwise it removes the element in front of L_5 and answers with that.

4. It populates h on the points not queried by $\mathcal{A}_1$ in the lexicographical order by the remaining entries of L_5

□ ■

We finally prove Lemma 11.4.23.

Proof. We want to upper bound

$$\mathsf{Pr}\left[\mathsf{off\text{-}2\text{-}dia}\text{-}k \wedge \neg\mathsf{off\text{-}zz}\text{-}k' \wedge \neg\mathsf{off\text{-}hex}\text{-}k'' \wedge \neg(m+1)\text{-}\mathsf{col}\right] .$$

Note that if $\mathcal{A}_1$ does not cause $\mathsf{off\text{-}zz}\text{-}k', \mathsf{off\text{-}hex}\text{-}k''$ and finds diamonds of category two for at least k salts, it means that there is a set of at least $k - 2k' - 2k''$ salts for which $\mathcal{A}_1$ finds diamonds of category two such

that no two diamonds in the set share the last query completing the diamond. We will use this to intuition to build a compression argument that upper bounds this probability.

The encoding procedure encodes the random oracle h as follows.

1. It first runs $\mathcal{A}_1^{\mathrm{h}}$, answering all its queries using h.

2. If $\mathsf{off\text{-}zz}\text{-}k'$ happens or $\mathsf{off\text{-}hex}\text{-}k''$ happens or $(m+1)\text{-}\mathsf{col}$ happens or $\mathsf{off\text{-}2\text{-}dia}\text{-}k$ does not happen, it outputs $\varnothing$

3. It finds a set of salts S of size $(k - 2k' - 2k'')/2$ such that for each $a \in S$

   - $\mathcal{A}_1$ found a category 2 diamond for a
   - The salt a has no more than $2T_1 m/(k - 2k' - 2k'')$ successor queries
   - The category 2 diamond for no two salts in the set share the last query of the diamond

4. If no such S is found, return $\varnothing$

5. For every a' in S it picks the category 2 diamond that does not share the last query with the diamond of any other salt in S found by $\mathcal{A}_1$ (if there is more than one, it picks one arbitrarily). It lets the queries forming the diamond be $q_1 = \mathrm{h}(a', M_1) = y, q_2 = \mathrm{h}(y, M_2) = z, q_3 = \mathrm{h}(a', M_3) = y', q_4 = \mathrm{h}(y', M_4) = z$ for some $M_1, M_2, M_3, M_4, y, y', z$ such that $|\{a', y, y', z\}| = 4$ such that q_4 was the last of the four queries made by $\mathcal{A}_1$.

6. It initializes lists $\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3$ to empty lists, T to empty set

7. It starts running $\mathcal{A}_1$ again

8. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:

   (a) If the query is marked as a q_4 query for a salt $a \in S$, then

      i. it adds the index of this query in T

      ii. it adds the lexicographical order of q_3 among all of the queries made by $\mathcal{A}_1$ with answer a in the list L_1

      iii. Let the input salt of q_3 be a'. It adds the lexicographical order of q_2 among all of the successor queries of the salt a' in the list L_2

(b) Otherwise it adds $\mathrm{h}(a, M)$ to L_3.

9. It appends the evaluation of h on the points not queried by $\mathcal{A}_1$ to L_3 in the lexicographical order of the inputs.

10. It outputs $\mathsf{L}_1, \mathsf{L}_2, \mathsf{L}_3, \mathsf{T}$.

The decoding procedure works as follows.

1. If the encoding is $\varnothing$ it aborts.

2. It runs $\mathcal{A}_1^{\mathrm{h}}$.

3. For every query $\mathrm{h}(a, M)$ made by $\mathcal{A}_1$ it does the following:

   (a) If the index of the query is in T,

      i. it removes the element i on the front of list L_1, and locates the query q_3 which has lexicographical order i among the queries with answer a

      ii. Let a' be the input salt of q_3. it removes the element i on the front of list L_2, and locates the query q_2 which has lexicographical order i among all the successor query of salt a'. It answers with the answer of q_2

   (b) Otherwise it removes the element in front of L_3 and answers with that.

4. It populates h on the points not queried by $\mathcal{A}_1$ in the lexicographical order by the remaining entries of L_3

Correctness of decoding: First off, we argue that for adversary $\mathcal{A}_1$ that causes the event $\mathsf{off\text{-}2\text{-}dia}\text{-}k \wedge \neg\mathsf{off\text{-}zz}\text{-}k' \wedge \neg\mathsf{off\text{-}hex}\text{-}k'' \wedge \neg(m+1)\text{-}\mathsf{col}$ to happen, the encoding algorithm will never return $\varnothing$. To argue this it suffices to show that the encoding algorithm never returns $\varnothing$ from line 4. Note that the encoding algorithm reaches this line only if the event $\mathsf{off\text{-}2\text{-}dia}\text{-}k \wedge \neg\mathsf{off\text{-}zz}\text{-}k' \wedge \neg\mathsf{off\text{-}hex}\text{-}k'' \wedge \neg(m+1)\text{-}\mathsf{col}$ was caused by running $\mathcal{A}_1$, which means there is a set of k salts S' such that $\mathcal{A}_1$ found category two diamonds for them. Moreover there does not exist a set of k' salt pairs for which $\mathcal{A}_1$ found a zig-zag. Also there does not exist a set of k' salt pairs for which $\mathcal{A}_1$ found a hexagon. Now let P be the maximally large set of salt pairs such that

1. each salt in the salt pairs of P is in S'

2. no two salt pairs in P share a salt

3. $\mathcal{A}_1$ found a hexagon for each salt pair in P

Since $\neg$off-hex-k'' happens we have that $|\mathsf{P}| < k''$. Let S'' be all the salts in the salt pairs in P. We have that $|\mathsf{S}'\backslash\mathsf{S}''| = |\mathsf{S}'| - 2|\mathsf{P}| > k - 2k''$. Now let P' be the maximally large set of salt pairs such that

1. each salt in the salt pairs of P' is in $\mathsf{S}'\backslash\mathsf{S}''$

2. no two salt pairs in P' share a salt

3. $\mathcal{A}_1$ found a zigzag for each salt pair in P'

Since $\neg$off-zz-k' happens we have that $|\mathsf{P}'| < k'$. Let S''' be all the salts in the salt pairs in P. We have that $|\mathsf{S}'\backslash\mathsf{S}''\backslash\mathsf{S}'''| = |\mathsf{S}'\backslash\mathsf{S}'| - 2|\mathsf{P}'|| > k - 2k'' - 2k'$. Let Notice that for every salt in a in $\mathsf{S}'\backslash\mathsf{S}''\backslash\mathsf{S}'''$, $\mathcal{A}_1$ found a category 2 diamond for a and category 2 diamond for no two salts in the set share the last query of the diamond by the maximality of $\mathsf{S}'', \mathsf{S}'''$. Further, since $\neg(m+1)$-col happens, the total number of salt-successor query pairs are at most mT_1. Therefore, at least $(k - 2k'' - 2k')/2$ of the salts (note that $(k - 2k' - 2k'')/2$ is an integer because k is a multiple of 2) in $\mathsf{S}'\backslash\mathsf{S}''\backslash\mathsf{S}'''$ are such that there are at most $2mT_1/(k - 2k' - 2k'')$ successor queries for the salt. Therefore, such a set S always exists.

If the encoding algorithm does not return $\varnothing$, it is easy to see the decoding algorithm decodes correctly, because the query answers that the encoding algorithm does not add in the list L_3 can be recovered by the decoding algorithm correctly using $\mathsf{S}, \mathsf{L}_1, \mathsf{L}_2$. This can be verified by inspection. Therefore, we have that

$$\begin{aligned}
&\mathsf{Pr}\left[\text{ Decoding is correct }\right] \\
&\geqslant \mathsf{Pr}\left[\text{off-2-dia-}k \wedge \neg\text{off-zz-}k' \wedge \neg\text{off-hex-}k'' \wedge \neg(m+1)\text{-col}\right] .
\end{aligned}$$

Observe that the encoding algorithm removes $(k - 2k' - 2k'')/2$ answers of the random oracle from the encoding, and all the removed answers are distinct because those were q_1 queries for different salts. It instead adds an set T of $(k - 2k' - 2k'')/2$ distinct values in $[T_1]$, and sequence L_1 of $(k - 2k' - 2k)/2$ values that are at most m, and sequence L_2 of $(k - 2k' - 2k)/2$ values that are at most $2mT_1/(k - 2k' - 2k'')$. Using the compression lemma, we have that

$$\begin{aligned}
&\mathsf{Pr}\left[\text{ Decoding is correct }\right] \\
&\leqslant \frac{\binom{T_1}{(k-2k'-2k'')/2} m^{(k-2k'-2k'')/2} (2mT_1/(k-2k'-2k''))^{(k-2k'-2k'')/2}}{2^{n(k-2k'-2k'')/2}} \\
&\leqslant \left(\frac{4em^2T_1^2}{(k-2k'-2k'')^2 2^n}\right)^{(k-2k'-2k'')/2} .
\end{aligned}$$

Therefore,

$$\Pr\left[\mathsf{off\text{-}2\text{-}dia}\text{-}k \wedge \neg\mathsf{off\text{-}zz}\text{-}k' \wedge \neg\mathsf{off\text{-}hex}\text{-}k'' \wedge \neg(m+1)\text{-}\mathsf{col}\right] \leqslant \left(\frac{4em^2T_1^2}{(k-2k'-2k'')^2 2^n}\right)^{(k-2k'-2k'')/2} .$$

□ ■

11.5 *Upper Bounding Offline-Online advantage via the Multi-Instance Approach*

In [CK18], the authors remark that in personal communication with them, Dan Bernstein noted that a lower bound against multiple-discrete-log algorithms also yields lower bounds on the preprocessing time for discrete-log algorithms with preprocessing. In this section, we explore an example of using this approach to prove guarantees against offline-online adversaries against collision-resistance of a salted random oracle.

This approach upper bounds the advantage of a (T_1, T_2)-adversary $\mathcal{A}$ against the collision-resistance of a salted random oracle in terms of the advantage of an adversary $\mathcal{B}$ against a multi-instance version of the game that makes $T_1 + kT_2$ queries. The oracle game $s\text{-}\mathsf{mi}\text{-}k\text{-}\mathsf{CR}^{\mathrm{h}}$ in Fig 11.16 is the multi-instance version of the salted collision-resistance game: the adversary gets k distinct salts as input, and wins if it finds a collision for each of them. (Note that $k \leqslant 2^s$ since there are at most 2^s salts).

We prove the following theorem.

Theorem 11.5.1. *Let $s, \ell, n \in \mathbb{N}_{>0}$. Let $H_{s,\ell,n}$ be the uniform distribution over $\mathsf{Fcs}(\{0,1\}^s \times \{0,1\}^\ell, \{0,1\}^n)$. Let $\mathcal{A}$ be a (T_1, T_2) adversary such that*

$$\mathsf{Adv}^{s\text{-}\mathsf{pre}\text{-}\mathsf{CR}}_{H_{s,\ell,n}}(\mathcal{A}) = \varepsilon .$$

Then for any $\delta \leqslant \varepsilon$, and any $k \leqslant \delta \cdot 2^s$, there exists an adversary $\mathcal{B}$ such that

$$\mathsf{Adv}^{s\text{-}\mathsf{mi}\text{-}k\text{-}\mathsf{CR}}_{H_{s,\ell,n}}(\mathcal{B}) \geqslant (\varepsilon - \delta)\frac{\binom{\delta \cdot 2^s}{k}}{\binom{2^s}{k}} .$$

Moreover $\mathcal{B}$ makes $T_1 + kT_2$ queries to h.

Proof. We construct the following $T_1 + k \cdot T_2$-adversary $\mathcal{B}$ against $s\text{-}\mathsf{mi}\text{-}k\text{-}\mathsf{CR}^{\mathrm{h}}$ where:

- $\mathcal{B}$ first runs $\mathcal{A}_1^{\mathrm{h}}$.
- It then runs $\mathcal{A}_2^{\mathrm{h}}(a_i)$ for $i \in [k]$, and outputs all the values $\mathcal{A}_2$ outputs.

We next compute a lower bound on $\mathsf{Adv}^{s\text{-}\mathsf{mi}\text{-}k\text{-}\mathsf{CR}}_{H_{s,\ell,n}}(\mathcal{B})$. Let $0 \leqslant \delta \leqslant \varepsilon$. Say h is "good" for $\mathcal{A}$ if $\mathcal{A}$ succeeds on finding collisions for at least $\delta \cdot 2^s$ salts for h. We claim that $\varepsilon - \delta$ fraction of the h are good. This is true

Game s-mi-k-$\mathsf{CR}^{\mathrm{h}}(\mathcal{A})$

$a_1, \ldots, a_k \xleftarrow{\$} \binom{\{0,1\}^s}{k}$

For $i \in k$:

$\{(M_i, M_i')_{i \in k}\} \leftarrow \mathcal{A}^{\mathrm{h}}(a_1, \ldots, a_k)$

If for all $i \in k$, $M_i \neq M_i'$ and $\mathrm{h}(a_i, M_i) = \mathrm{h}(a_i, M_i')$

  Return true

Return false

Figure 11.16: Oracle Game s-mi-k-CR^{h} capturing multi-instance salted collision-resistance of oracle h

because if α fraction of h are good, we have that

$$\mathsf{Adv}^{\mathsf{CR}}_{H_{s,\ell,n}}(\mathcal{A}) \leqslant \mathsf{Pr}\left[\text{h is good}\right] + \delta \ .$$

Therefore, $\alpha \geqslant \varepsilon - \delta$, i.e., $\varepsilon - \delta$ fraction of h are good. Observe that $\mathcal{B}$ always wins if h is good for $\mathcal{A}$, and $a_1, \ldots, a_k$ are sampled from the $\delta \cdot 2^s$ salts on which $\mathcal{A}$ wins. Therefore, for $k \leqslant \delta \cdot 2^s$, we have that

$$\begin{aligned}\mathsf{Adv}^{s\text{-mi-}k\text{-}\mathsf{CR}}_{H_{s,\ell,n}}(\mathcal{B}) &\geqslant \mathsf{Pr}\left[\text{h is good for } \mathcal{A}\right] \mathsf{Pr}\left[\mathcal{A} \text{ win on } a_1, \ldots, a_k \,\middle|\, \text{h is good for } \mathcal{A}\right] \\ &\geqslant (\varepsilon - \delta)\frac{\binom{\delta \cdot 2^s}{k}}{\binom{2^s}{k}}\end{aligned}$$

□ ■

The hope is to then use the upper-bound on the advantage of any $(T_1 + kT_2)$-adversary against s-mi-k-CR to give an upper bound on the advantage of $\mathcal{A}$ against CR. We show an example where the left hand side and the right hand side of the advantage inequality in Theorem 11.5.1 are very far away, meaning we cannot hope for tight-bounds for collision-resistance of salted random oracles using this approach.

Consider a $(2^{n/2}, 0)$-adversary $\mathcal{A}$ that makes $2^{n/2}$ distinct queries for one salt in the offline phase. Let ε be the advantage of $\mathcal{A}$. It is east to see that ε is at most $1/2^s$.

Theorem 11.5.1 requires $k \leqslant \delta \times 2^s \leqslant \varepsilon \cdot 2^s$. Since $\varepsilon \leqslant 1/2^s$, we have that $k \leqslant 1$. Now note that there exists an adversary $2^{n/2}$ against s-mi-1-CR that succeeds with constant probability. On the other hand, for any choice of δ, the right hand side of the inequality is much smaller than 1. So, we cannot hope for tight bounds from this approach.

However, if we were only interested in adversaries that succeed with probability 1, we would have that the advantage of $\mathcal{B}$ would be 1 (since $\mathcal{A}$ wins on all inputs/random oracles). Then we could upper bound the advantage of $\mathcal{A}$ with the advantage of any $(T_1 + kT_2)$ adversary $\mathcal{B}$ against s-mi-k-CR. We can simply use

our analysis in Theorem 11.3.3 of the event off-oneblk-k to upper bound this advantage. Namely off-oneblk-k was the probability that the adversary finds one-block collisions for k different salts using T_1 queries, and we showed that $\mathsf{Pr}\left[\text{off-oneblk-}k\right] \leqslant \frac{\binom{T_1}{2k}}{2^{kn}}$. Clearly the probability that any adversary finds collisions for salts $a_1, \ldots, a_k$ using $T_1 + kT_2$ queries is at least the probability that the adversary finds collisions for k different salts. Therefore the advantage of such $\mathcal{B}$ is at most $\frac{\binom{T_1+kT_2}{2k}}{2^{nk}}$ which is at most $\left(\frac{e(T_1+kT_2)}{2^{n/2}k}\right)^{2k}$. Since we know $\mathcal{B}$ has advantage 1,we would have that

$$1 \leqslant \left(\frac{e(T_1 + kT_2)}{2^{n/2}k}\right)^{2k} .$$

Setting $k = 2^s$ and simplifying we get,

$$\frac{T_1}{2^{s-1}} + T_2 \geqslant \frac{2^{n/2}}{e} .$$

Therefore, for advantage 1 adversaries, we get the right guarantee using this approach.

BIBLIOGRAPHY

[ABD+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 5–17. ACM Press, October 2015. 203

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *CT-RSA 2001*, volume 2020 of *LNCS*, pages 143–158. Springer, Heidelberg, April 2001. 18, 106, 107, 117

[ACDW20] Akshima, David Cash, Andrew Drucker, and Hoeteck Wee. Time-space tradeoffs and short collisions in merkle-damgård hash functions. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 157–186. Springer, Heidelberg, August 2020. 5, 6, 180, 181, 182, 183, 185, 190, 194, 195, 199, 200, 203, 205, 206, 208, 209, 210, 213, 215, 217, 220, 258, 263, 264, 267, 286, 310, 311, 318, 323

[ACFK17] Benedikt Auerbach, David Cash, Manuel Fersch, and Eike Kiltz. Memory-tight reductions. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 101–132. Springer, Heidelberg, August 2017. 4, 9, 10, 13, 14, 16, 17, 18, 19, 20, 22, 24, 46, 48, 49, 50, 52, 61, 70, 104, 151, 156, 157

[ACFK21] Benedikt Auerbach, David Cash, Manuel Fersch, and Eike Kiltz. Personal communication, 2021. 70, 74

[Adl78] Leonard Adleman. Two theorems on random polynomial time. In *Symposium on Foundations of Computer Science, SFCS*, pages 75–83, 1978. 217, 286

[AGL22] Akshima, Siyao Guo, and Qipeng Liu. Time-space lower bounds for finding collisions in merkle-Damgård hash functions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 192–221. Springer, Heidelberg, August 2022. iv, 194, 195, 199, 203, 205, 310, 311, 312, 314, 318, 321, 322, 323, 324, 325

[AHMP08] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C-W Phan. SHA-3 proposal BLAKE. *Submission to NIST*, 92, 2008. 194, 197

[AMPH14] Jean-Philippe Aumasson, Willi Meier, Raphael C.-W. Phan, and Luca Henzen. *The Hash Function BLAKE*. Information Security and Cryptography. 2014. 194, 197

[ASS+16] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. DROWN: Breaking TLS using SSLv2. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 689–706. USENIX Association, August 2016. 200

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, Heidelberg, April 2008. 186, 187, 188, 194

[BDPVA07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, volume 2007. Citeseer, 2007. 186, 187, 194

[BDPVA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the security of the keyed sponge construction. In *Symmetric Key Encryption Workshop*, volume 2011, 2011. 186

[Ber11] Daniel J Bernstein. Extending the salsa20 nonce. In *Workshop record of Symmetric Key Encryption Workshop*, volume 2011. Citeseer, 2011. 14

[Bha20] Rishiraj Bhattacharyya. Memory-tight reductions for practical key encapsulation mechanisms. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 249–278. Springer, Heidelberg, May 2020. 48

[BHK15] Mihir Bellare, Dennis Hofheinz, and Eike Kiltz. Subtleties in the definition of IND-CCA: When and how should challenge decryption be disallowed? *Journal of Cryptology*, 28(1):29–48, January 2015. 74

[BKN04] Mihir Bellare, Tadayoshi Kohno, and Chanathip Namprempre. Breaking and provably repairing the ssh authenticated encryption scheme: A case study of the encode-then-encrypt-and-mac paradigm. *ACM Transactions on Information and System Security (TISSEC)*, 7(2):206–241, 2004. 31

[BL13] Daniel J. Bernstein and Tanja Lange. Non-uniform cracks in the concrete: The power of free precomputation. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2013. 203

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, October 2008. 16

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. 15, 61, 62, 179

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006. 22, 35, 45, 70, 76, 83, 106, 162, 166

[BT16] Mihir Bellare and Björn Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 247–276. Springer, Heidelberg, August 2016. 4, 11, 24, 29

[CDG18] Sandro Coretti, Yevgeniy Dodis, and Siyao Guo. Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 693–721. Springer, Heidelberg, August 2018. viii, 4, 6, 181, 188, 189, 190, 191, 192, 262, 265, 266, 284

[CDGS18] Sandro Coretti, Yevgeniy Dodis, Siyao Guo, and John P. Steinberger. Random oracles and non-uniformity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 227–258. Springer, Heidelberg, April / May 2018. viii, 4, 5, 6, 20, 156, 158, 159, 180, 182, 183, 191, 192, 194, 195, 199, 200, 203, 204, 267, 310, 311, 318

[CGLQ20] Kai-Min Chung, Siyao Guo, Qipeng Liu, and Luowen Qian. Tight quantum time-space tradeoffs for function inversion. In *61st FOCS*, pages 673–684. IEEE Computer Society Press, November 2020. 180, 181, 199, 209, 310, 311, 314, 323

[CK18] Henry Corrigan-Gibbs and Dmitry Kogan. The discrete-logarithm problem with preprocessing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 415–447. Springer, Heidelberg, April / May 2018. 4, 140, 181, 202, 207, 209, 342, 420

[CK19] Henry Corrigan-Gibbs and Dmitry Kogan. The function-inversion problem: Barriers and opportunities. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 393–421. Springer, Heidelberg, December 2019. 3, 179

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 229–235. Springer, Heidelberg, August 2000. 61, 62

[Cor02] Jean-Sébastien Coron. Optimal security proofs for PSS and other signature schemes. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 272–287. Springer, Heidelberg, April / May 2002. 15, 50, 60, 61

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003. 18

[Dam87] Ivan Damgård. Collision free hash functions and public key signature schemes. In *Advances in Cryptology - EUROCRYPT*, pages 203–216, 1987. 181, 186

[Dam89] Ivan Damgård. A design principle for hash functions. In *Advances in Cryptology - CRYPTO*, pages 416–427, 1989. 192, 193

[Dam90] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 416–427. Springer, Heidelberg, August 1990. 205

[DGJL21] Denis Diemert, Kai Gellert, Tibor Jager, and Lin Lyu. Digital signatures with memory-tight security in the multi-challenge setting. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 403–433. Springer, Heidelberg, December 2021. 15, 48, 49, 57, 63

[DGK17] Yevgeniy Dodis, Siyao Guo, and Jonathan Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 473–495. Springer, Heidelberg, April / May 2017. 4, 180, 182, 183, 193, 197, 209, 262, 310, 320

[Din20] Itai Dinur. On the streaming indistinguishability of a random permutation and a random function. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 433–460. Springer, Heidelberg, May 2020. 3, 10, 12, 15, 20, 23, 24, 26, 27, 30, 47, 49, 157

[DTT10] Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 649–665. Springer, Heidelberg, August 2010. 4, 109, 170, 175, 179, 209, 217, 267, 339

[DTZ20] Wei Dai, Stefano Tessaro, and Xihu Zhang. Super-linear time-memory trade-offs for symmetric encryption. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 335–365. Springer, Heidelberg, November 2020. 15

[EGM96] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, March 1996. 56

[EPRS17] Adam Everspaugh, Kenneth G. Paterson, Thomas Ristenpart, and Samuel Scott. Key rotation for authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 98–129. Springer, Heidelberg, August 2017. 89

[FGJ20] Marc Fischlin, Felix Günther, and Christian Janson. Robust channels: Handling unreliable networks in the record layers of quic and dtls 1.3. Cryptology ePrint Archive, Report 2020/718, 2020. https://eprint.iacr.org/2020/718. 31

[FGK22] Cody Freitag, Ashrujit Ghoshal, and Ilan Komargodski. Time-space tradeoffs for sponge hashing: Attacks and limitations for short collisions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 131–160. Springer, Heidelberg, August 2022. xiii, 5, 7, 194, 258

[FGK23] Cody Freitag, Ashrujit Ghoshal, and Ilan Komargodski. Optimal security for keyed hash functions: Avoiding time-space tradeoffs for finding collisions. In *Advances in Cryptology - EUROCRYPT*, pages 440–469, 2023. xiii, 5, 6, 7, 309, 315

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018. 105

[FN99] Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 1999. 4, 179, 182, 188, 191, 261, 276, 283, 284

[GB08] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography, 2008. 309, 317, 318

[GGJT22] Ashrujit Ghoshal, Riddhi Ghosal, Joseph Jaeger, and Stefano Tessaro. Hiding in plain sight: Memory-tight proofs via randomness programming. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 706–735. Springer, Heidelberg, May / June 2022. xiii, 3, 4, 45

[GGKL21] Nick Gravin, Siyao Guo, Tsz Chiu Kwok, and Pinyan Lu. Concentration bounds for almost k-wise independence with applications to non-uniform security. In Dániel Marx, editor, *32nd SODA*, pages 2404–2423. ACM-SIAM, January 2021. 180

[GJT20] Ashrujit Ghoshal, Joseph Jaeger, and Stefano Tessaro. The memory-tightness of authenticated encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 127–156. Springer, Heidelberg, August 2020. xiii, 3, 4, 15, 16, 17, 21, 66, 156

[GK96] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, June 1996. 3

[GK22] Ashrujit Ghoshal and Ilan Komargodski. On time-space tradeoffs for bounded-length collisions in merkle-Damgård hashing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 161–191. Springer, Heidelberg, August 2022. xiii, 5, 7, 194, 195, 199, 203, 205, 208, 310, 311, 318, 321, 322, 323

[GLLZ21] Siyao Guo, Qian Li, Qipeng Liu, and Jiapeng Zhang. Unifying presampling via concentration bounds. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 177–208. Springer, Heidelberg, November 2021. 180

[GPR14] Peter Gaži, Krzysztof Pietrzak, and Michal Rybár. The exact PRF-security of NMAC and HMAC. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 113–130. Springer, Heidelberg, August 2014. 203

[GT00] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *41st FOCS*, pages 305–313. IEEE Computer Society Press, November 2000. 209, 262

[GT16] Peter Gazi and Stefano Tessaro. Provably robust sponge-based PRNGs and KDFs. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 87–116. Springer, Heidelberg, May 2016. 186

[GT20] Ashrujit Ghoshal and Stefano Tessaro. On the memory-tightness of hashed ElGamal. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 33–62. Springer, Heidelberg, May 2020. xiii, 3, 4, 17, 19, 20, 103, 105, 156, 209

[GT23] Ashrujit Ghoshal and Stefano Tessaro. The query-complexity of preprocessing attacks. In *Advances in Cryptology - CRYPTO*, 2023. xiii, 5, 7, 338

[Hel80] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980. 2, 4, 179, 182, 188, 191, 194, 260, 267, 276, 278

[HKR15] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 15–44. Springer, Heidelberg, April 2015. 88

[HR03] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 482–499. Springer, Heidelberg, August 2003. 88

[IK10] Russell Impagliazzo and Valentine Kabanets. Constructive proofs of concentration bounds. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 617–631. Springer, 2010. 180, 209, 217, 311, 323

[IOM12] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and repairing GCM security proofs. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 31–49. Springer, Heidelberg, August 2012. 24

[JT19] Joseph Jaeger and Stefano Tessaro. Tight time-memory trade-offs for symmetric encryption. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 467–497. Springer, Heidelberg, May 2019. 3, 10, 15, 20, 23, 24, 26, 27, 29, 30, 38, 47, 49, 157

[JT20] Joseph Jaeger and Stefano Tessaro. Expected-time cryptography: Generic techniques and applications to concrete soundness. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 414–443. Springer, Heidelberg, November 2020. 204, 205, 344, 345, 346

[KM12] Neal Koblitz and Alfred Menezes. Another look at HMAC. Cryptology ePrint Archive, Report 2012/074, 2012. https://eprint.iacr.org/2012/074. 203

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988. 88

[Mau05] Ueli M. Maurer. Abstract models of computation in cryptography (invited paper). In Nigel P. Smart, editor, *10th IMA International Conference on Cryptography and Coding*, volume 3796 of *LNCS*, pages 1–12. Springer, Heidelberg, December 2005. 104

[Mer82] Ralph C. Merkle. *Secrecy, Authentication and Public Key Systems*. PhD thesis, UMI Research Press, Ann Arbor, Michigan, 1982. 181, 186

[Mer88] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, Heidelberg, August 1988. 181, 186, 194

[Mer90a] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 218–238. Springer, Heidelberg, August 1990. 181, 186, 192, 193

[Mer90b] Ralph C. Merkle. A fast software one-way hash function. *Journal of Cryptology*, 3(1):43–58, January 1990. 205

[MT10] Robin A. Moser and Gábor Tardos. A constructive proof of the general lovász local lemma. *J. ACM*, 57(2):11:1–11:15, 2010. 209

[MV04] David A. McGrew and John Viega. The security and performance of the Galois/counter mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 343–355. Springer, Heidelberg, December 2004. 11, 24, 28

[NRS14] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 257–274. Springer, Heidelberg, May 2014. 16, 67

[Oec03] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 617–630. Springer, Heidelberg, August 2003. 4, 182, 188, 194, 203

[Pol75] J. M. Pollard. A monte carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, Sep 1975. 23

[QD90] Jean-Jacques Quisquater and Jean-Paul Delescaille. How easy is collision search. New results and applications to DES. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 408–413. Springer, Heidelberg, August 1990. 23

[Rog04] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 348–359. Springer, Heidelberg, February 2004. 16, 24, 67

[Rog06] Phillip Rogaway. Formalizing human ignorance. In Phong Q. Nguyen, editor, *Progress in Cryptology - VIETCRYPT 06*, volume 4341 of *LNCS*, pages 211–228. Springer, Heidelberg, September 2006. 203

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, Heidelberg, May / June 2006. 10, 28

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 1–20. Springer, Heidelberg, February 2004. 108

[SHA] Secure hash standard. https://www.nist.gov/publications/secure-hash-standard. 5

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997. 104, 106, 340, 342

[Sho01] Victor Shoup. A proposal for an ISO standard for public key encryption. Cryptology ePrint Archive, Report 2001/112, 2001. https://eprint.iacr.org/2001/112. 18

[SOS20] Ido Shahaf, Or Ordentlich, and Gil Segev. An information-theoretic proof of the streaming switching lemma for symmetric encryption. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 858–863, 2020. 15, 47, 49

[ST79] Robert H. Morris Sr. and Ken Thompson. Password security - A case history. *Commun. ACM*, 22(11):594–597, 1979. 186

[TT18] Stefano Tessaro and Aishwarya Thiruvengadam. Provable time-memory trade-offs: Symmetric cryptography against memory-bounded adversaries. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 3–32. Springer, Heidelberg, November 2018. 3, 15

[Unr07] Dominique Unruh. Random oracles and auxiliary input. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 205–223. Springer, Heidelberg, August 2007. 4, 180, 182, 188, 191, 208, 215, 262, 266, 309, 311, 315, 318, 320

[Vau02] Serge Vaudenay. Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS... In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 534–546. Springer, Heidelberg, April / May 2002. 10

[Wee05] Hoeteck Wee. On obfuscating point functions. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 523–532. ACM Press, May 2005. 262

[WMHT18] Yuyu Wang, Takahiro Matsuda, Goichiro Hanaoka, and Keisuke Tanaka. Memory lower bounds of reductions revisited. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 61–90. Springer, Heidelberg, April / May 2018. 17, 19, 20, 48, 49, 151, 156

[Yao90] Andrew Chi-Chih Yao. Coherent functions and program checkers (extended abstract). In *22nd ACM STOC*, pages 84–94. ACM Press, May 1990. 179, 182, 188