Communication and Round Balanced Oblivious FSM Evaluation

Caleb Horst

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

Master of Science

University of Washington

2016

Reading Committee:

Anderson Nascimento, Chair

Martine De Cock

Program Authorized to Offer Degree:
Computer Science and Systems

University of Washington

Abstract

Communication and Round Balanced Oblivious FSM Evaluation

Caleb Horst

Chair of the Supervisory Committee:
Assistant Professor Anderson Nascimento
Institute of Technology

Privacy is a major issue in the age of the Internet. Many advances are being made in Cryptography regarding performing computations over private data, both in homomorphic encryption, multi-party computation, and applications that put these to use. Herein we present a multiparty protocol for the private evaluation of a finite state machine. We motivate this by noting that many features can be extracted from text using the finite state transducer, an easy extension of the general FSM. For example, this protocol could be used as the feature extraction phase of an end-to-end private machine learning algorithm over text inputs.

Our protocol(s) build on those previously developed by offering a different balance between communication, computation and rounds. Notably, we offer a 2-round protocol with fairly low communication. The previous constant round protocol had higher communication, and the previous low communication protocol had rounds proportional to the input size. A very computation efficient version is provided if a third party is available who is not trusted beyond non-collusion. And a more computationally intensive version removes the need for this helper.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
INTRODUCTION

Advances in data availability and data mining techniques make questions of privacy and security more salient than ever. To address, or potentially avoid, the ethical and moral dichotomy of leveraging Big Data for the betterment of society against the infringement on personal privacy, subfields of cryptography are being explored that allow for computations on data in ways that restrict how much information about that data is leaked.

Conceptually, this is a very powerful and intriguing possibility. Imagine a world where you could have all the benefits of modern Internet based tools, such as the vocal recognition and question answering offered by Apple's Siri, Microsoft's Cortana, Amazon's Alexa, and similar services, but with the promise that these service providers were not actually able to view your data. Or imagine that you are able to complete transactions in such a way that your information is *never decrypted*, guaranteeing the privacy of your financial data.

Such programs are currently too complex and our solutions to this problem too inefficient to be feasibly applied to this extent, but smaller applications may pave the way for more practical applications. The auction in [6], where a large number of independent Danish sugar beet farmers and the purchasers were able to calculate the optimal market price and amounts to be produced without any intermediate information leaking that might have negatively influenced negotiations for any parties, demonstrates that these cryptographic protocols can be applied in real world scenarios.

Finite state machines are a basic, but useful computational model. The ability to evaluate a finite state machine in a private setting, allows for some interesting use cases.

Finite state transducers, which are easily extended from automata, can be used to count occurrences of multiple strings or string types in an input text. This allows for uses such as preprocessing text for machine learning algorithms by extracting linguistic features (e.g. LIWC *cite), or comparing an input text against some certain text of interest. Many of the previous works on oblivious FSM evaluation have been targeted at the application of DNA sequence matching.

1.1 Contribution

Herein we present a pair of protocols that allow two parties, one holding a private finite state machine and the other a private input string, to determine the outputs of that finite state machine on the input string while revealing minimal information, namely the sizes, about the private inputs.

Our protocols are an improvement on previous protocols that achieve a better balance of round complexity and communication overhead for the input string holder as the input alphabet grows in size.

Specifically, two of the best previous protocols offer significant complexity tradeoffs. One [9] features a low constant, 1.5 or 2, round complexity and easy computations at the cost of $nO(|\Sigma||Q|)$ communication complexity. The other [5] has very low communication, potentially as low as $nO(\log_2^2(|Q||\Sigma|)$ at the cost of n rounds and very heavy computation. Our proposition is a 2-round protocol that has communication $nO(|\Sigma| + |Q|)$. Computation is heavy with only two players, but with the assistance of a third party, a non-colluding but otherwise untrusted cryptographic helper, computation is very light.

The asymptotic communication complexity reduction means that the protocol is more efficient than the previous protocols that inspired it as the input alphabet grows in length. This means for applications where the alphabet is significant, such as text input processing where the alphabet could easily be 62 characters or larger (26 letters in lower and upper case along with 10 numeric digits), the communication is significantly improved.

1.2 Organization

This thesis is organized as follows. We begin by presenting some necessary concepts and notation, then proceed to present the previous protocols and our improved protocols. We next include full proofs of security and lastly experimental timing results and suggestions for efficient implementation.

Chapter 2
PRELIMINARIES

Herein we will detail some cryptographic concepts that we directly use or draw heavily on for inspiration, and specify the notation we will use throughout. Each of these cryptographic tools has been the subject of much study and here we will only briefly describe them.

2.1 Cryptographic Concepts

Cryptographic Assumptions Much of cryptography is built off of reductions. Protocols or schemes are reduced to problems that are either provably impossible (information theoretically secure) or have no known, and no readily promising, computationally feasible solutions (computationally secure). One such assumption we will use is the *Random Oracle Assumption* , or Random Oracle Model[4], where we assume for the purposes of reduction that a Hash function behaves like a truly random function. Meaning that for every input the output is uncorrelated and completely random.

Oblivious Transfer and PIR Basic Oblivious Transfer (OT)[19] has a few formulations. These formulations have been shown to be equivalent[7], and the more readily applicable definition is that OT is a protocol where one party holds two pieces of data, x_0 and x_1 and a second party wants to retrieve x_b, but party one should not learn b, and party two should not learn x_{1-b}. This is the case of 1-out-of-2 OT, but 1-out-of-n OT can be constructed by invoking multiple instances of 1-out-of-2 OT.

Oblivious Transfer has been studied extensively, due to its incredible power. It has been shown that OT can be used for secure two party computation of any function, commitments and zero-knowledge proofs [13]. Furthermore, OT can be made very efficient by using OT extension [3], a technique where k OTs are used to perform a practically arbitrarily large number of OTs using symmetric functions instead of additional public key operations.

A closely related field is that of Private Information Retrieval (PIR). In PIR, the server should not know what is being queried, but the client may receive additional information. Generally, a requirement of PIR is also that it requires less communication than simply sending the entire database. PIR also extends to the case where the data may be split among multiple servers to allow for cheaper retrieval. Computational PIR (cPIR), such as [1, 8, 24], relies on the hardness of cryptographic assumptions to achieve PIR for computationally bounded servers. Symmetric PIR is a PIR scheme where the user is also guaranteed to not learn extra information about the server's input and is effectively equivalent to OT.

Homomorphic Encryption Homomorphic encryption (HE) is a field of much study right now. HE allows one to compute functions on encrypted data that will be reflected in the decryption, e.g. $E(x) * E(y) = E(x + y)$. Based on basic Boolean algebra, the ability to calculate both the addition and multiplication operation in this way allows any function to be computed on encrypted data. A seminal paper [11] showed that this is possible, and numerous improvements have been made since. However, these fully homomorphic schemes are still prohibitively expensive for sizeable calculations. Herein we will rely only on a somewhat homomorphic encryption (SHE) scheme which allows for the addition of enciphered values.

The somewhat homomorphic scheme we will consider is Paillier's scheme[18], which is additively homomorphic, based on the hardness of the composite residuosity problem, and allows for multiplication by plaintext constants. There are also other additively homomorphic schemes, based on other computational hardness assumptions, but many of these have drawbacks such as costly decryption or limited homomorphic operations that are not present in Paillier.

We require that the SHE scheme satisfies CPA security. Like other encryption schemes, HE and SHE schemes are considered CPA secure if for adversaries restricted to polynomial time the encryptions of messages reveal absolutely no information about those messages. This is sometimes posed as a game. Given two known messages m_0 and m_1 and an encryption of one of them, $Enc(m_b)$, it must be shown that even with access to a polynomial number of chosen message/ciphertext pairs, there is still only negligibly better than a 1/2 probability of guessing b correctly.

Formally, for any polynomial time algorithm $\mathcal{A}$ and any messages m_0, m_1 with a random $b \in \{0,1\}$

$$P[\mathcal{A}^{Enc(\cdot)}(m_0, m_1, Enc(m_b)) = b] \leq \frac{1}{2} + \epsilon$$

We note that Paillier encryption is CPA secure so long as hardness of the underlying computational problems holds.

Secret Sharing In secret sharing, a secret value is split into parts $x \rightarrow (x_1, ..., x_n)$ called shares given to some number of parties P_1 through P_n (for $n \geq 2$). These shares are chosen such that it is impossible to recover x, or any partial information about x, from any subset of t shares, for $1 < t < n$. One basic secret sharing scheme is the additive secret sharing, where $x = \sum_i x_i$, which has $t = n - 1$. In cases where only a subset of shareholders should also be allowed to recover the secret, Shamir's secret sharing scheme [22], which uses polynomials of degree t, allows $t + 1$ out of n players to recover the secret.

Computations can be done upon these shares in such a way that allows for both addition and multiplication of the shared values (if the parties interact), hence allowing the parties to compute arbitrary functions together in such a way that nobody can recover the shared values until a threshold of parties working together.

Garbled Circuits A major breakthrough in multi-party computation came when Yao proposed the garbled circuit [28]. In essence, the desired function to compute is expressed as a Boolean circuit, then "garbled". Every wire is given a k_0 and k_1 for the values of 0 and 1 on that wire. Then the output of each gate is encrypted using these keys, e.g. for an AND gate, having k_1 for both inputs would allow you to decrypt the k_1 output key, but having k_0 for either input would cause you to decrypt k_0 for that wire. OT is used to retrieve the

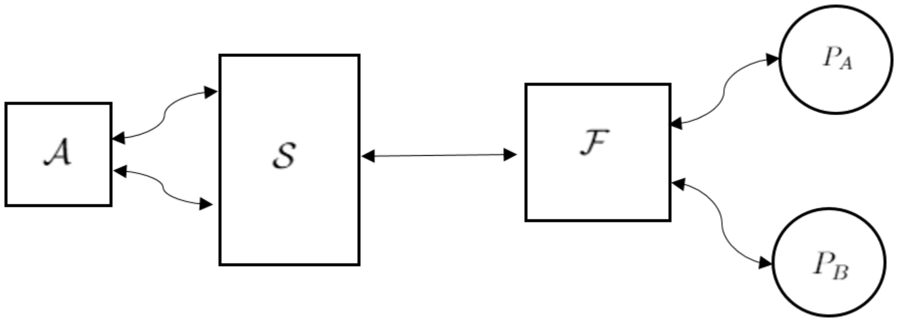

Figure 2.1: The simulator $\mathcal{S}$ interfacing with the adversary $\mathcal{A}$.

correct keys for the input of the person receiving the circuit. Much work has been done to formalize the security of, and dramatically improve the performance of the garbled circuit technique.

Security in garbled circuits intuitively is that the recipient of the circuit is unable to attain any of the keys, or knowledge of which keys he has, except those in the degarbling path based on his inputs.

Simulation Security Proofs To prove the security of the protocols, we present an ideal functionality that achieves the same outputs as our protocol but with indisputable security, modeled generally as the usage of a perfectly trusted third party. We then show that any information seen by a corrupt party in the protocol, provided only one is corrupt, can be generated in this ideal case. Specifically, that for an adversary $\mathcal{A}$ and simulator program $\mathcal{S}$, $\mathcal{S}$ with the ideal function $\mathcal{F}$ can generate a view for $\mathcal{A}$ that is computationally indistinguishable from the output of the protocol π. More formally, for all $\mathcal{A}$ there exists an interface $\mathcal{S}$ such that

$$\text{VIEW}_{\mathcal{A}}(\pi) \overset{\text{COMP}}{\equiv} \text{VIEW}_{\mathcal{A}}(\mathcal{S}(\mathcal{F}))$$

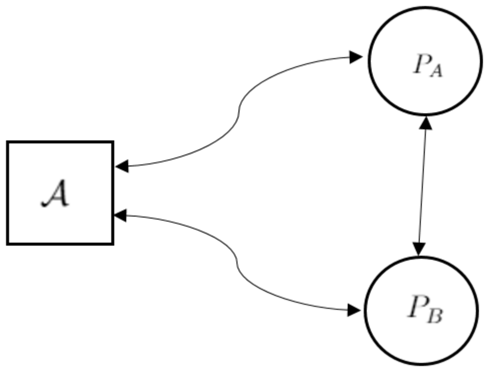

Figure 2.2: The adversary $\mathcal{A}$ running the real protocol with other players.

where the VIEW$_{\mathcal{A}}$ is all the inputs and outputs seen by $\mathcal{A}$ and $X_0 \overset{\text{COMP}}{\equiv} X_1$ iff for all polynomial time algorithms $\mathcal{D}$:

$$P[\mathcal{D}(X_b) = b] \leq \frac{1}{2} + \epsilon$$

for a value of ϵ that can be made arbitrarily small.

Intuitively, security can be envisioned using Figures 2.1 and 2.2 where we see two scenarios, one with $\mathcal{A}$ directly interacting with the other players, and one where $\mathcal{A}$ interacts with $\mathcal{S}$ and $\mathcal{S}$ with $\mathcal{F}$. If these two scenarios appear the exact same to $\mathcal{A}$, since $\mathcal{S}$ only has access to the input of $\mathcal{A}$ and the output $\mathcal{A}$ should receive from $\mathcal{F}$, $\mathcal{A}$ must not be able to learn anything during the protocol he is not supposed to learn.

2.2 Notation

Generally deterministic finite automata (DFAs) are represented as tuples:

$$\mathbb{M} = (Q, q_0, \delta, \Sigma, Q_{accept})$$

Where Q is the set of states, q_0 is the initial state, δ is a transition rule of the form $\delta : Q \times \Sigma \mapsto Q$, Σ is the input alphabet, and $Q_{accept} \subset Q$ is the set of accept states. The output of the FSM is 1 if the execution ends in an accept state and 0 if the execution ends in any other state.

To mathematically represent δ, a matrix Δ is often used. Δ is a $|Q| \times |\Sigma|$ matrix where each entry is $\log_2 |Q|$ bits representing the state transitioned to for each input/state pair.

In the case of transducers, the Q_{accept} parameter is replaced by Γ, an output alphabet (usually $\mathbb{Z}/m\mathbb{Z}$ for some m) and λ, an output rule.

The two common FST models are Moore machines and Mealy machines. In a Moore machine, we have $\lambda : Q \mapsto \Gamma^\rho$, and in a Mealy machine $\lambda : Q \times \Sigma \mapsto \Gamma^\rho$, where ρ represents the number of output variables. We can represent this readily as a matrix Λ of either dimension $1 \times |Q|$ or $|\Sigma| \times |Q|$, respectively, where each entry is $\rho \log_2 |\Gamma|$ bits in size. This can also be represented by appending the outputs to the entries of Δ, however doing so will result in redundancy in a Moore machine.

Our protocols use either three or two parties. We will refer to them based on their input to the protocol. The *Client* provides an input string to be run through the FSM provided by the *Server*. To achieve a higher efficiency, we allow for a third *Helper* party that is expected to not collude with either of the other two parties, but is not trusted to see either's input.

Chapter 3
OBLIVIOUS FSM

In this chapter we provide the previous work on oblivious evaluating FSMs, then our proposed protocol. We include a description of the two variants, how one would extend it to transducers from automata, along with security and complexity analyses.

3.1 Related Work

Several protocols have been previously developed to address the secure evaluation of finite state machines. Other protocols also exist for secure subsequence matching, a weaker task, but one of the potential uses for finite state machines.

Tronosco-Pastoriza et al. in [23] are believed to be the first to publish a paper developing the idea of oblivious FSMs, they specifically targeted the application of DNA sequence matching. It uses basic secret sharing and somewhat HE and/or OT to evaluate the FSM one input symbol at a time.

They considered evaluation being split between two parties, a server and a client. The client holding $\bar{x} \in \Sigma^n$ and the server holding an FSM. At each point in the evaluation, each party holds an additive share of $q_i \in Q$, $i \in \{1...n\}$, the ith state.

To retrieve shares of the next state, the server masks all the entries of his matrices with a round-specific random value r, then rotates the entire transition matrix and output matrix by his share of the previous state. The client uses an Additively Homomorphic Encryption scheme (e.g. Paillier) to encrypt a vector $< v_1, ...v_{|Q|} >$ as a unary encoding, that is $v_i = Enc(0)$ except for the index of his state share, which is $Enc(1)$. The server multiplies his matrix (using the multiplication by constant and ciphertext addition homomorphic properties) by this vector and attains a vector representing the transition function for the current state. The server and client then execute a 1-out-of-$|\Sigma|$ OT protocol to retrieve the x_i'th element. The client can decrypt this to learn his share of the next state, while the server can keep r as his share. This is repeated until the entire input $\bar{x}$ is processed. They note that the order can be switched, by working on the transpose of the matrix, encrypting the input as a vector, and then doing a 1-out-of-$|Q|$ OT, depending on which is more efficient. Furthermore, they provide the basic extension to transducers. For Moore type machines, by repeating the rotation and vector multiplication on the vector of state outputs, the encrypted outputs can be retrieved. For Mealy type machines, the rotation, vector multiplication and OT against the matrix of outputs must all be performed.

Blanton and Aliasgard in [5] slightly build on this framework to use only OT/cPIR for better communication complexity, as well as demonstrating how evaluation can be outsourced to one proxy each, or an arbitrary number of servers, to offload the computational costs from the input and FSM holders.

Instead of using SHE to capture the correct vector then doing OT on the vector, they suggest doing OT on the entire matrix by representing it as a vector then fetching the proper index (as in 2D array indexing). This incurs a 1-out-of-$|\Sigma||Q|$ OT per round instead of $|\Sigma|$ Paillier and 1-out-of$|Q|$ OT (or 1-out-of-$|\Sigma|$ OT and $|Q|$ Paillier).

By also using a more bandwidth optimized cPIR than trivial 1HE, such as the one in [24] this approach can achieve the lowest communication complexity per round of any of the options presented here, at the cost of being the most computationally expensive.

Due to the security of the SHE, OT, or cPIR, these schemes protect the user's input string from the server, and the random rotation of the transition matrix and shared state prevent either party from knowing what state transitions are occurring, protecting the FSM from the client. The corresponding papers include more in depth analyses of the correctness and complexities.

The downside of both of these protocols is that they all suffer from a heavy computational load do to the large amount of SHE or OT called for, commonly requiring modular exponentiations over a large modulus, and are restricted to running in no fewer than n

rounds.

A second approach was presented by Frikken in [9]. He generalizes the idea behind garbled circuits to require only two rounds of communication, at the unfortunate cost of a substantial communication burden, as the entire FSM is sent in garbled form n times. Computationally, however, this technique requires only a few OTs, and is otherwise composed entirely of fast symmetric key operations. Mohassel's protocol in [17] is very similar, but reformats the FSM as using binary inputs, and claims better efficiency than [9]. Their scheme appears to be almost identical however, except the binary restriction, and if it is extended to a nonbinary alphabet, should have equivalent complexity. They do, however, provide a proof of security which is briefly argued but not fully proven by Frikken, and suggest the use of OT extension to reduce the asymmetric operations to a constant instead of being tied to the input length.

The approach posed to solve the problem of the high round counts and heavy computation in the sequential protocols, this protocol builds on basic garbled circuit techniques.

The transition matrix is garbled n times, once for each input character. First, a permutation (rotation) is applied to the set of next states to hide order. Then, for each state and input, the next state is encoded in such a manner that it can be uncovered only using the key for that current state (gained as the FSM is evaluated) and the key for that input (retrieved using OT at the start of the evaluation). You could think of this garbling function generally as

$$\delta^i(\sigma, q) = Enc_{k^i_\sigma, k^i_q}(\Delta_{\sigma,q} || k^{i+1}_{\Delta_{\sigma,q}})$$

To complete evaluation, the data holder uses 1-out-of-$|\Sigma|$ OT to retrieve the keys for each symbol in his input and the FSM holder sends the garbled FSM to the data holder. From the initial state (which can be made public due to the garbling) and the input keys, the data holder evaluates the FSM by ungarbling the correct entry for the current state and input to determine the next state and next state key until the final state is reached.

The advantages of this scheme over the sequential schemes is that the number of rounds and the number of public key operations are very low. The disadvantage is that n times the entire transition matrix is transferred, which creates a high communication overhead.

Frikken noted a few potential optimizations. Even suggesting the use of PIR to avoid sending the whole table, however it appeared that he did not consider selecting entire columns, as he claimed using PIR would increase the round complexity to $O(n)$. We will demonstrate in Chapter 3 that a more judicious application of PIR yields better results.

We note that any FSM with alphabet Σ can be expressed as a binary FSM (for the protocol of [17]), where each input is encoded as a sequence of $\log_2 |\Sigma|$ bits, at the cost of increasing the number of states by a factor of $|\Sigma| - 2$ and the input length by a factor of $\log_2 |\Sigma|$. This new binary FSM may be reducible using known algorithms for state machine optimization, however doing so shouldn't reduce the size of $|Q'|$ below $|Q|$ and the input length will remain $n \log_2 |\Sigma|$. Which generally results in larger communication complexities than if an FSM with the original $|\Sigma|$ can be used.

Laud and Willemsen in [14] utilizes an arithmetic black box, an abstraction of distributed computation, such as secret sharing, and a polynomial representation of the transition function to achieve quite low 'online' communication complexity, however the round complexity is still tied to the length of the input, and the preprocessing still requires a large amount of communication. They also address the evaluation of NFAs, however the benefits of NFAs are unlikely to outweigh the additional costs in the proposed setting of this paper.

Other works further use additional permutations and matrix multiplication to complete DFA and NFA evaluations [25, 27, 21], sometimes even working over encrypted inputs. In general these protocols are less efficient due to heavier security guarantees or harder tasks.

There are other protocols for sequence matching, even counting the occurrences of subsequences, but they are not expected to be more efficient and private in the setting where a large number of sequences are sought, or a number of sequences might need to map to the same output, i.e. multiple words contributing to a count of "happy" words.

3.2 Improved Protocol

By utilizing a garbled FSM scheme and PIR, we can make the $|\Sigma|$ and $|Q|$ communication factors additive instead of multiplicative while keeping the round count constant. The protocol is described with a basic cPIR/PIR scheme for simplicity, and to keep computation minimal.

Garbling Similarly to [9, 17], we will garble the state transition matrix, Δ, n times. This garbling consists of permutation and symmetric encryption. The suggested permutation is adding a random mask value modulo $|Q|$. We will use r_i for these rotation values, and forgo including the modular reduction notation.

The server chooses $n \cdot |Q|$ random keys denoted k, one for each state in all n matrices, and encrypts the state transitions as such:

Let $q' = q - r_i$, the true state

$\forall i \in \{1, \ldots, n\}, q \in \{1, \ldots, |Q|\}, \sigma \in \{1, \ldots, |\Sigma|\}$

$$G_{fsm}(\delta(q,\sigma), i) := Enc_{k_q^i}(\Delta_{q',\sigma} + r_{i+1}, k^{i+1}_{\Delta_{q',\sigma}+r_{i+1}})$$

This garbled form is an encryption of the next permuted state for the given permuted state and input, along with the corresponding key so that it can be decrypted to continue the evaluation. Note that this garbling does not require keys based on the input string, as in previous variants, we will shortly show how coupling this with PIR/OT does not break the security.

This encryption function is a generalization, however. While we could use any CPA-secure symmetric encryption, here we will leverage the Random Oracle Assumption and use a hash function, $H : \{0,1\}^{n+|\Sigma|} \mapsto \{0,1\}^{\kappa+|Q|}$, and encrypt as

$$G_{fsm}(\delta(q,\sigma), i) := H(k_{q_i}||\sigma) \oplus (\Delta_{q',\sigma} + r_{i+1}||k^{i+1}_{\Delta_{q',\sigma}+r_{i+1}})$$

Given the current state's key, it is clear that you can decrypt the permuted state to transition to and the requisite key.

We will address more thoroughly how the output can be included in the garbled FSM, for now, we simply replace the final transition values with the value $1 + r$ if the final state is accepted and r otherwise.

That is

$$G_{fsm}(\delta(q,\sigma), n-1) := \begin{cases} Enc_{k_q^{n-1}}(r) & \delta(q',\sigma) \notin Q_{accept} \\ Enc_{k_q^{n-1}}(r+1) & \delta(q',\sigma) \in Q_{accept} \end{cases}$$

recalling that q' represents $q - r_i$ modulo $|Q|$, the 'true' state.

Next the client executes a PIR/OT subprotocol, fetching only the necessary columns, those corresponding to the client's input. We use PIR/OT interchangeably from here on, because while our subprotocol is not asymptotically better than the trivial solution (treating $|\Sigma|$ as a constant), it can leverage a second server, something common in PIR, but not in OT. We first present a PIR/OT with Helper protocol, and in the next section address how this could be altered in the case where no semi-trusted helper is available.

PIR with Helper The server shares the garbled matrices with the helper. This will not compromise security as the garbled entries will be pseudorandom, as we will prove in section 3.6. This sharing can happen during the protocol execution or at any time in an offline phase between only the server and helper. We further assume that the server and helper share a common random string, likely instantiated as a shared seed for a PRNG, and we will henceforth refer to it as another pseudorandom function G.

The client then generates a bitwise secret sharing of one-hot-encodings for each of his inputs, that is a unary encoding of each symbol in his input string, x. Each symbol x_i is encoded as the shares $\bar{x}_i = \bar{x}_i^0 \oplus \bar{x}_i^1$ for $\bar{x}_i^b \in \{0,1\}^{|\Sigma|}$ and $i \in \{0,\ldots,n-1\}$ such that $\bar{x}_{i,j}^0 \oplus \bar{x}_{i,j}^1 = 1$ iff $j = x_i$ and is zero otherwise. randomly chosen. He then sends one of these shares to the helper and the other to the server. They then take the bitwise inner product of these vectors with the bits of every row of the transition matrix mask them with the next bits of the shared random string, or the output of the PRNG. Mathematically,

$$\bigoplus_{\sigma\in\Sigma} \left(\{\bar{x}_{i,\sigma}^b\}^{\kappa+\log_2|Q|} \wedge [H(k_q^i||\sigma) \oplus (\Delta_{q',\sigma} + r_{i+1}||k^{i+1}_{\Delta_{q',\sigma}+r_{i+1}})] \oplus G(q,\sigma,i) \right)$$

for $b \in \{0,1\}$, $q \in Q$ and $i \in \{0,\ldots,n-1\}$.

This generates a bitwise sharing of the columns corresponding to the clients input string, garbled versions of $\bar{\delta}(\cdot, x_i)$.

The server and helper send these vectors back to the client. Upon recombination, the masks, $G(q, \sigma, i)$, will cancel out and the garbled column will be revealed.

To enable the client to evaluate the FSM, the server supplies the client with the key for the initial state, $k^0_{q_0}$.

Algorithm 1 Garbled OFSM With 3-Party PIR

Inputs: Server has a FSM $\mathcal{M}$, Client has a string $x \in \Sigma^n$, Helper has no input. Operations happen over a public finite field $\mathcal{F}$, $|Q|$ is also public

1. Server garbles $\mathcal{M}$ and sends a copy of the garbled $\mathcal{M}$ to the Helper along with n random seed values.
2. Client generates n length $|\Sigma|$ vectors of bitwise secret sharings for each character of x, such that the x_i element of vector i is 1, and all other elements are 0.
3. Client sends one set of vectors to the Server and one to the Helper.
4. Server and Helper both take the bitwise inner product of each row of the garbled matrices with the corresponding vectors
5. Helper and Server mask the n resultant vectors with the output of a pseudorandom number generator on the n seeds and return these masked vectors to the Client.
6. Client recombines the garbled transition vectors and ungarbles the first state transition using the given key.
7. Client evaluates FSM by ungarbling the entry of the next vector given in the previous vector using the key from the previous transition key.

3.3 Two Party Variant

If we assume that there is no available semi-trusted third party to act as a helper, we have to rework the PIR phase. Note that doing so will the use of additional computational assumptions.

Any symmetric cPIR or OT method can be used to fetch the encrypted columns, some at lower communication or computation complexity than what is suggested here. We use this instantiation of OT mostly to match more closely the previous implementation of [23]. We make this choice considering that since the $n|Q|$ factor of the data being fetched will generally dominate over the query cost, reducing it at the cost of additional computation seems generally like a poor tradeoff. We do acknowledge that depending on the application, a different cPIR may be more practical. The protocol is summarized in Algorithm 2 and described below.

The client initiates basic cPIR by sending one hot encodings of his input $x = x_1, \cdots, x_n$ using an additively homomorphic scheme such as Paillier. That is, for x_i he sends $< Enc(b_1), \ldots, Enc(b_{|\Sigma|}) >$ where $b_{x_i} = 1$ and all other $b_k = 0$.

The server computes the matrix product with each the garbled matrix to find the encrypted garbled transition function $Enc\{\bar{\delta}_i(x_i, \cdot)\}$ in terms of the previous state. These encrypted columns are then sent back to the client, as well as the initial key $k^0_{q_0}$.

This selection is similar to what happens in the original sequential protocol, [23], however all column selections occur at once and are sent at once, as opposed to completing each state selection before proceeding to the next round. The client is then able to decrypt the garbled entries and use these to evaluate the FSM and retrieve the output.

3.4 Output and Transducing

To generate output, we can consider two ways to incorporate the output function into the garbling scheme. The first being the one stated above, which applies only to the basic finite state automaton. On the last state transition, we replace the next state value with the

Algorithm 2 Garbled OFSM with cPIR

Inputs: Server has a FSM $\mathcal{M}$, Client has a string $x \in \Sigma^n$, $|Q|$ is public

1. Server garbles $\mathcal{M}$.

2. Client generates n length $|\Sigma|$ vectors of encryptions for each character of x, such that the x_i element of vector i is $Enc(1)$, and all other elements are $Enc(0)$.

3. Client sends the vectors to the Server.

4. Server takes the inner product of each row of the garbled matrices with the corresponding vectors and returns the resulting column vectors, along with the initial state key, to the Client.

5. Client decrypts the first garbled transition vector and ungarbles the first state transition using the given key.

6. Client evaluates FSM by decrypting and ungarbling the entry of the next vector given in the previous vector using the ungarbled key.

acceptance value 0/1 (masked with a random r).

$$G_{fsm}(\delta(q,\sigma), n-1) := \begin{cases} Enc_{k_q^{n-1}}(r) & \delta(q',\sigma) \notin Q_{accept} \\ Enc_{k_q^{n-1}}(r+1) & \delta(q',\sigma) \in Q_{accept} \end{cases}$$

Note that this can be readily extended to non-zero outputs,

$$G_{fsm}(\delta(q,\sigma), n-1) := Enc_{k_q^{n-1}}(r + \lambda(\delta(q',\sigma)))$$

Furthermore, we can extend the protocol to transducers by adding this into the transitions rules, now using separate random masks for each $i \in \{0, \ldots, n-1\}$ given as z_i.

$$G_{fsm}(\delta(q,\sigma), i) := Enc_{k_q^i}(z_i + \lambda(\delta(q',\sigma)))$$

even if if the output function λ outputs a number of separate outputs, e.g. $\lambda : (Q, \Sigma) \mapsto \Gamma^\rho$ for some number of outputs ρ. In our given encryption function we would have to require that the output of H has a sufficient bitlength to mask the next key, next permuted state, and all the outputs.

These work well for the Mealy type FSM, where the output depends on both the state and the input symbol. In the Moore type FSM where the outputs depend only on the states and not on the inputs, we can cut the communication slightly, as the masked outputs all have to be included in each garbled entry, which would increase communication by $n\rho|Q| \log_2 |\Gamma|$. We can encrypt the outputs for each state using the state keys, then let the client use PIR to retrieve the ones corresponding to his input with either the 3 or 2 party variant or send them all along with the garbled transition vectors. In that case, we would have

$$G_{output}(\lambda(q), i) = H'(k_q) \oplus (\lambda(q') + z_i)$$

for all $i \in \{0, \ldots, n-1\}$ and $q \in Q$.The helper in the three party case would be unable to learn anything from these, because the k_q values will never be reused in the lists and are never used by themselves in the garbled transition functions, and the pseudorandom (hash) function H' need not even be the same as H. If the client is sent all the entries, he is also only able to decrypt the one he has a key for.

Sending the client all the entries only gives a small computation reduction for the server (and helper) since they don't need to perform the selection operation on the extra bits, but using PIR reduces the communication at the cost of an extra two rounds. For the more optimal 3-party version, it would specifically take $2nQ$ bits sent by the client and $2n\rho \log_2 |\Gamma|$ bits returned as opposed to the aforementioned $n\rho|Q| \log_2 |\Gamma|$ cost.

One advantage to these transducer approaches, beyond the practical uses offered by the increased output versatility, is the ability to use an arbitrary number of garbled matrices. While the basic output requires n to be known initially so that the final matrix can incorporate the outputs, here the outputs are incorporated in every matrix (or in closely associated lists) which allows for a long "chain" of matrices to be generated in the offline phase, and shared with the helper if necessary, well before protocol execution begins. When a client wants to evaluate a string, the server and client use the next n matrices, and the server may have to provide r_0 along with the key, since all the matrices will be randomly rotated except the 'first' in the chain.

3.5 A Security Note

Before giving constructions of oblivious finite state machines we would likely to briefly mention that just because the order of states and the transitions are hidden throughout the protocol execution, this does not guarantee the security of the FSM. It is possible to learn an equivalent DFA given only enough strings and their acceptance values.

See e.g. [20] for protocols to derive the internals of a FSM given only the ability to query the machine on specific strings (akin to a chosen plaintext attack).

The DNA sequence matching applications given in most previous works would be vulnerable to this, due to the result being directly opened to the user. Whether or not it is a reasonable concern is dependent on the size of the machine and number of queries permitted. However, any concern can potentially be avoided several ways.

One is by using Nondeterministic Finite Automata (NFAs). It can be shown that learning NFAs can be as hard as breaking public key cryptosystems [2, 12] While an equivalent DFA can be learned, these DFAs *can* be exponentially larger than their NFA counterparts. This is not guaranteed, however, as DFAs are a subset of the NFAs.

A second option is the use of transducers. As opposed to DFAs, which only have an accept/reject output based on the final state, transducers have outputs at every state or transition. In certain cases this could potentially be easier to learn than DFAs, but when coupled with the next fix, the extra complexity should add a layer of security over DFAs.

The safest fix, which works poorly in the DNA matching setting of many previous works, is to leave the output (or outputs) shared in preparation for another protocol. The security added by this naturally is determined the following functions.

3.6 Security Proofs

To prove the security of the protocol and it's two-party variant, we will demonstrate how a simulator in an idealized protocol could generate output that is statistically indistinguishable from what would be seen by a corrupted player in our protocol. We will assume a single, statically corrupted party. That is, one party will be the adversary $\mathcal{A}$, and which party is corrupted is constant throughout the protocol. We allow for $\mathcal{A}$ to only be honest-but-curious (also known as passive or semi-honest), meaning that he must follow the protocol specifications, and we require that all communications happen synchronously over secure channels. Recall that we also rely on the Random Oracle Assumption for our proofs.

We first describe the ideal functionality $\mathcal{F}_{FSM}$, then how the simulator, $\mathcal{S}$, run and give a thorough proof sketch of why the view generated for $\mathcal{A}$ by $\mathcal{S}$ is indistinguishable from what $\mathcal{A}$ would see during a real protocol execution.

Main Protocol

Ideal Functionality We consider $\mathcal{F}_{FSM}$ to be a black-box which takes as input an FSM, $\mathcal{M}$, and a value, r from the Server, an input string, $x \in \Sigma^n$, from the Client and an input,

($help$), from the Helper. Once all three inputs are received, it outputs $r+1$ to the Client if x is accepted on $\mathcal{M}$, and r otherwise.

Helper Corrupted The first input the Helper, $\mathcal{A}$, receives is the garbled FSM and random seed data. $\mathcal{S}$ thus must generate these for the Helper. $\mathcal{S}$ picks the random keys $k_{\sigma,q} \forall \sigma \in \Sigma, q \in Q$ and the n PRNG seeds. $\mathcal{S}$ garbles the dummy FSM where every state returns to itself on every input using the generated keys and $\mathcal{S}$ sends this along with the seeds to the Helper. $\mathcal{S}$ then generates n random bitwise secret sharings of length $|\Sigma|$ and sends these to the Helper as the shares of the Client's inputs. $\mathcal{S}$ receives back from the Helper the masked matrix product and inputs ($help$)to $\mathcal{F}_{FSM}$.

To show that this fake garbled matrix is indistinguishable from any other matrix we use a hybrid argument. We consider a set of distributions $D_0 \dots D_t$ where D_0 is the distribution we get for garbling the fake FSM and D_t is the distribution of a truly random set of n matrices. We start with D_0 and replace one column at a time with a truly random data. There are $n \cdot |\Sigma|$ columns, and we note that each one is assigned a unique random key with overwhelming probability (given the 2^κ keyspace). The value $H(k|x)$ will also be unique with overwhelming probability, and also is indistinguishable from randomly uniform under the Random Oracle Assumption. We therefore conclude that the encrypted values in each column also appear uniformly random, by information theory, and any distinguisher breaks the Random Oracle Assumption for the given H. Since each column appears uniformly random, as we replace them we get indistinguishable distributions and the entire garbled FSM is indistinguishable from uniform randomness. Furthermore, since this was independent of the specific garbled matrix we chose, we can extend this to say that every garbled FSM is indistinguishable from any other and $\mathcal{S}$ has provided the Helper something it cannot discern from what it receives in the real protocol.

The seeds are simply random data in both cases, and the simulated secret shares are indistinguishable from real secret shares unless the Helper colludes with the Server or Client.

$\mathcal{F}_{FSM}$ also should properly complete execution with the correct result, since $\mathcal{S}$ provides (*help*)as required, so if the Helper is corrupted, we have perfectly simulated the real world protocol execution for $\mathcal{A}$ and we conclude that the protocol is as secure as the ideal functionality.

Server Corrupted The Server, $\mathcal{A}$, initially outputs the garbled FSM and random seeds, which $\mathcal{S}$ will hold. $\mathcal{S}$ produces the n random length $|\Sigma|$ binary vectors to represent the client's inputs and passes them to $\mathcal{A}$. At this point $\mathcal{A}$ will do the matrix products and return the masked vectors along with the initial state key. This allows $\mathcal{S}$ to decrypt the entire FSM, extract the value of r from the last garbled matrix, and input these to $\mathcal{F}_{FSM}$.

It is well-known that for this secret sharing scheme shares are indistinguishable from random unless the players collude, and $\mathcal{F}_{FSM}$ receives and returns the proper inputs and outputs so we conclude that the real world protocol can be perfectly simulated by $\mathcal{S}$ and the protocol is as strong as $\mathcal{F}_{FSM}$.

Client Corrupted The Client, $\mathcal{A}$, will provide the bitwise secret sharing vectors representing his input which $\mathcal{S}$ will take and recombine to retrieve the input string x. $\mathcal{S}$ inputs x to $\mathcal{F}_{FSM}$ and receives back r' either r or $r+1$ depending on x's acceptance on $\mathcal{M}$.

$\mathcal{S}$ picks the random keys $k_{\sigma,q} \forall \sigma \in \Sigma, q \in Q$ and the n PRNG seeds, then garbles the dummy FSM where every state returns to itself on every input using the generated keys. As the output of the FSM on x (and every string), $\mathcal{S}$ places r'.

$\mathcal{S}$ multiplies the garbled dummy FSM by the two shares of x, $x^0 = x_0^0 x_1^0 \dots x_{n-1}^0$ and $x^1 = x_0^1 x_1^1 \dots x_{n-1}^1$ and masks them with the outputs of the PRNG,then returns the results to $\mathcal{A}$ along with the initial state key. Formally, $\mathcal{A}$ receives

$$\bigoplus_{\sigma \in \Sigma} \left(\{x_i^b\}^{\kappa + \log_2 |Q|} \wedge [H(k_q^i || \sigma) \oplus (\Delta_{q',\sigma} + r_{i+1} || k^{i+1}_{\Delta_{q',\sigma} + r_{i+1}})] \oplus G(q, \sigma, i) \right)$$

For $b \in \{0,1\}$, $i \in \{0, \dots, n-1\}$, and all $q \in Q$. Recall that G is the seeded pseudorandom number generator that only the Server and Helper share.

To prove the security, we have to show that $\mathcal{A}$ cannot distinguish these dummy returned vectors from the vectors returned in the real protocol. First, note that without recombining the shares properly every entry is masked with a different pseudorandom number, $G(q, \sigma, i)$. This holds for both the real and simulated vectors. The only way to combine them that will not be the combination of two meaningless random masks is the proper pairwise recombinations, which will yield

$$\bigoplus_{\sigma \in \Sigma} \left(\{x_i^0 \oplus x_i^1\}^{\kappa + \log_2 |Q|} \wedge [H(k_q^i || \sigma) \oplus (\Delta_{q',\sigma} + r_{i+1} || k^{i+1}_{\Delta_{q',\sigma} + r_{i+1}})] \right)$$
$$= H(k_q^i || x_i) \oplus (\Delta_{q',x_i} + r_{i+1} || k^{i+1}_{\Delta_{q',x_i} + r_{i+1}})$$

For all $q \in Q$. This shows that all columns except those pertaining to the x_i's could be replaced by $\mathcal{S}$ with truly random data with a negligible chance of $\mathcal{A}$ distinguishing. In the columns recoverable by $\mathcal{A}$, every k_q^i is unique with overwhelming probability, given the 2^κ domain size, and unless that key is known, $H(k_q^i || x_i)$ is pseudorandom.

Once $\mathcal{A}$ decrypts the proper transitions, he uncovers the next key and permuted state. Since the states are permuted randomly and the keys are randomly generated in both the real protocol and by $\mathcal{S}$, we conclude that the two views during the evaluation are indistinguishable, including the final output of r or $r + 1$ based on the acceptance of x on $\mathcal{M}$. Furthermore, since every phase of the real protocol is simulated by $\mathcal{S}$ with $\mathcal{F}_{FSM}$ in a manner that $\mathcal{A}$ cannot distinguish from the protocol, we have shown that the protocol is secure.

Two-Party Protocol

Ideal Functionality The two party ideal functionality, $\mathcal{F}_{FSM}$, differs only in that we no longer a third party to input (*help*). $\mathcal{F}_{FSM}$ still receives $\mathcal{M}$ and r from the Server and x from the Client, then outputs r to the Client if x is rejected on $\mathcal{M}$ and $r+1$ if it is accepted.

Client Corrupted $\mathcal{S}$ sends x the input of $\mathcal{A}$ to $\mathcal{F}_{FSM}$ and receives back the result r'. $\mathcal{S}$ then produces a garbled FSM that evaluates to r' on x and 0 on any other input, and accepts the encrypted input vectors from $\mathcal{A}$. $\mathcal{S}$ performs the matrix multiplications, selection operations, as in the protocol, on his dummy FSM and returns the result to $\mathcal{A}$. $\mathcal{A}$ can decrypt and evaluate the FSM to achieve the proper result of r'.

Server Corrupted $\mathcal{S}$ sends $\mathcal{M}$and r to $\mathcal{F}_{FSM}$ and receives no output in return. $\mathcal{S}$ generates n vectors of $|\Sigma|$ encryption of 0 using the Client's public key and sends these vectors to $\mathcal{A}$ as the protocol dictates. After $\mathcal{A}$ completes the multiplications and returns the encrypted columns to $\mathcal{S}$ the simulation is complete. For any semantically secure SHE scheme, $\mathcal{A}$ cannot distinguish these dummy input encryptions from the proper encryptions of x and the view of $\mathcal{A}$ is indistinguishable with $\mathcal{S}$ and $\mathcal{F}_{FSM}$ from the view in the real protocol.

3.7 Complexity Analysis and Comparison

For a visual comparison of the complexities, refer to Table 3.1 Our analyses are based on the protocols and suggested instantiations. Usage of a different PIR/OT in the two-party case for example would alter the amounts of computation and communication.

Computation:

Three Party This requires the same computation to garble the OFSM as previous works, $nO(|Q||\Sigma|)$, then both the Server and Helper have to perform $nO(|Q||\Sigma|(\kappa+\log_2|Q|))$ binary multiplications and additions to complete the selection and masking. Similarly the Client only has to generate the vectors, recombine the necessary elements, and ungarble suggesting $nO(|\Sigma|)$ total work, all of which is easy operations. Furthermore, the protocol requires no asymmetric operations such as the OT or somewhat homomorphic encryption called for in most previous solutions.

Single Server This protocol has $nO(|Q||\Sigma|)$ asymmetric and symmetric operations for the server. However, instead of each exponentiation being to a power of $\log_2|Q|$ as in the original work [23] each exponentiation is to a power of at least $\kappa + \log_2|Q|$. If the exponentiation takes a time dependent on the size of the exponent, this will be substantially slower. The Client also has to encrypt $n|\Sigma|$ and decrypt at least n entries. This is therefore substantially more computationally costly than the previous results that require no or minimal asymmetric computation ([9, 17, 14]).

Communication:

Three Party The client sends $2n|\Sigma|$ bits combined to the client and server. The query responses are of expected to have a combined size of $2n|Q|(\kappa+\log_2|Q|)$ along with an extra κ bits in the first state key. This is asymptotically reduced from the previous garbling schemes which would require an expected $n|Q||\Sigma|(\kappa+\log_2|Q|)$ bits to be received, not counting the data necessary for the OT protocols. It is not as low as [5], especially paired with the cPIR of [24], but boasts only 2 rounds instead of n and massively reduced computational costs. The server has to send the entire garbled matrices, but as mentioned, this does not affect the client and can happen prior to the client entering the protocol.

Single Server With this variant, the client has to send $n|\Sigma|2\kappa_+$ if we use assume the use of Paillier (where ciphertexts have length $2\kappa_+$). At most $n|Q|2\kappa_+$ is sent to the client and Paillier should allow us to pack multiple entries into each ciphertext by appending them before performing the homomorphic multiplications. Doing this would allow the communication to be reduced to roughly $n|Q|2(\kappa_s+\log_2|Q|)$. Again, this protocol has a higher communication complexity over the most communication efficient previous results, but has slightly lower computation and runs in 2 rounds instead of n. Further note that the incoming communication cost is expected to be slightly higher if $|\Sigma| < 4$ over just garbling, so this protocol should not be used for very small alphabets.

	Client Mod Exp	Server Mod Exp	Client Sends	Server Sends	Helper Sends	Rounds
Tronosco[23]	$nO(\|\Sigma\| + \|Q\|)$	$\boldsymbol{nO(\|\Sigma\|\|Q\|)}$	$nO(\|\Sigma\| + \|Q\|)$	$nO(\|Q\|)$?	N/A	$\boldsymbol{n}$
Blanton[5] [1]	$nO(\log \|\Sigma\|\|Q\|)$	$\boldsymbol{nO(\|\Sigma\|\|Q\|)}$	$nO(\log^2 \|\Sigma\|\|Q\|)$	$nO(\log^2 \|\Sigma\|\|Q\|)$	N/A	$\boldsymbol{n}$
Frikken[9]	$nO(\|\Sigma\|)$	$nO(\|\Sigma\|)$	$nO(\|\Sigma\|)$	$\boldsymbol{nO(\|\Sigma\|\|Q\|)}$	N/A	$O(1)$
Laud [14]	0	0	$nO(\log \|\Sigma\|\|Q\|)$	$nO(\log \|\Sigma\|\|Q\|)$	$nO(\log \|\Sigma\|\|Q\|)$	$\boldsymbol{n}$
2 Party	$nO(\|\Sigma\|)$	$\boldsymbol{nO(\|\Sigma\|\|Q\|)}$	$nO(\|\Sigma\|)$	$nO(\|Q\|)$	N/A	$O(1)$
w/Helper	0	0	$nO(\|\Sigma\|)$	$nO(\|Q\| + \|\Sigma\|\|Q\|)$	$nO(\|Q\|)$	$O(1)$

Table 3.1: Complexities of basic FSM evaluation

Chapter 4
OFSM EXPERIMENTS

In this chapter we detail our proof-of-concept implementation, give basic runtime results and analysis, and make some important notes regarding the application of the protocol.

4.1 Implementation

For proof of concept, we implemented our three- and two-party protocols in Java. We used SHA-256 as our hash function, and a basic implementation of Paillier using Java's BigIntegers using 2048 bit keys. [1]

The three parties were run on a single computer using 3 separate Threads, and Java ObjectStreams were used to pass data between them via Sockets, to be easily extended to actual over-the-Internet implementation.

In the two party protocol, after the server performs the column select operation and returns the encrypted column, we only decrypt the single entry needed for evaluation to avoid unnecessary decryption overhead. Furthermore we exploit the fact that the Paillier selection operation calls for multiple exponentiations of the same base (the ciphertexts) to different powers (the garbled entries) and interweave the modular exponentiations by squaring to avoid repeatedly calculating the powers. Furthermore the calculations are highly parallelizable, but we do not exploit this, to give a better baseline of how the protocols compare.

In the three party protocol, we use the basic and non-cryptographically secure build in Random object for the $G(q, \sigma, i)$ pseudorandom number generator, seeding it with the same value for Helper and Server and then using the same outputs.

[1] adapted from http://www.csee.umbc.edu/ kunliu1/research/Paillier.html

We do not include the timings of the output phase, only the state transitions, for better comparison with previous work. The communication time is not included either, however we know that only two sets of sequential messages are required for the primary protocol between the Client and Server/Helper. The setup time, between Server and Helper, is also excluded, but was trivial without any network delay.

4.2 Performance Results

Experiments were run on a laptop computer with an Intel core i7 6700HQ @ 2.6 GHz nominal and 16 GB DDR4 ram. Trials were run 3 to 10 times and results averaged. The basic performance results as $|Q|$ and $|\Sigma|$ are varied are presented in Figures 4.1 and 4.2.

Execution time for the three party protocol was very short. In fact, it was sufficiently short that the two-round Internet delay could be the performance bottleneck in practice, and the garbling was by far the most costly operation in the experiment. Comparatively, the performance of the two party protocol was very slow. Over 3 orders of magnitude slower. The primary bottleneck is shifted from the garbling and anticipated network delay to the selection and input encryption phases. Notice that as $|\Sigma|$ is increased we see a larger computational burden on the Client in the form of the input encryption.

Unsurprisingly all computations appear to scale effectively linearly. We would expect a slightly nonlinear curve if we used $\kappa + \log_2 |Q|$ bits instead of the full SHA-256 output of 2κ bits.

4.3 Application Notes

While we have provided the evaluation protocol, we have not made any note of how to generate good FSMs for text processing. There does not seem to be a large quantity of research on this topic, likely because it is fairly simple. First create the small FSMs that recognize each word of interest, then merge these FSMs, via a union and state reduction, until you attain the full FSM desired. This basic algorithm does not fully exploit the capabilities of the FSM, but works as a baseline to work from.

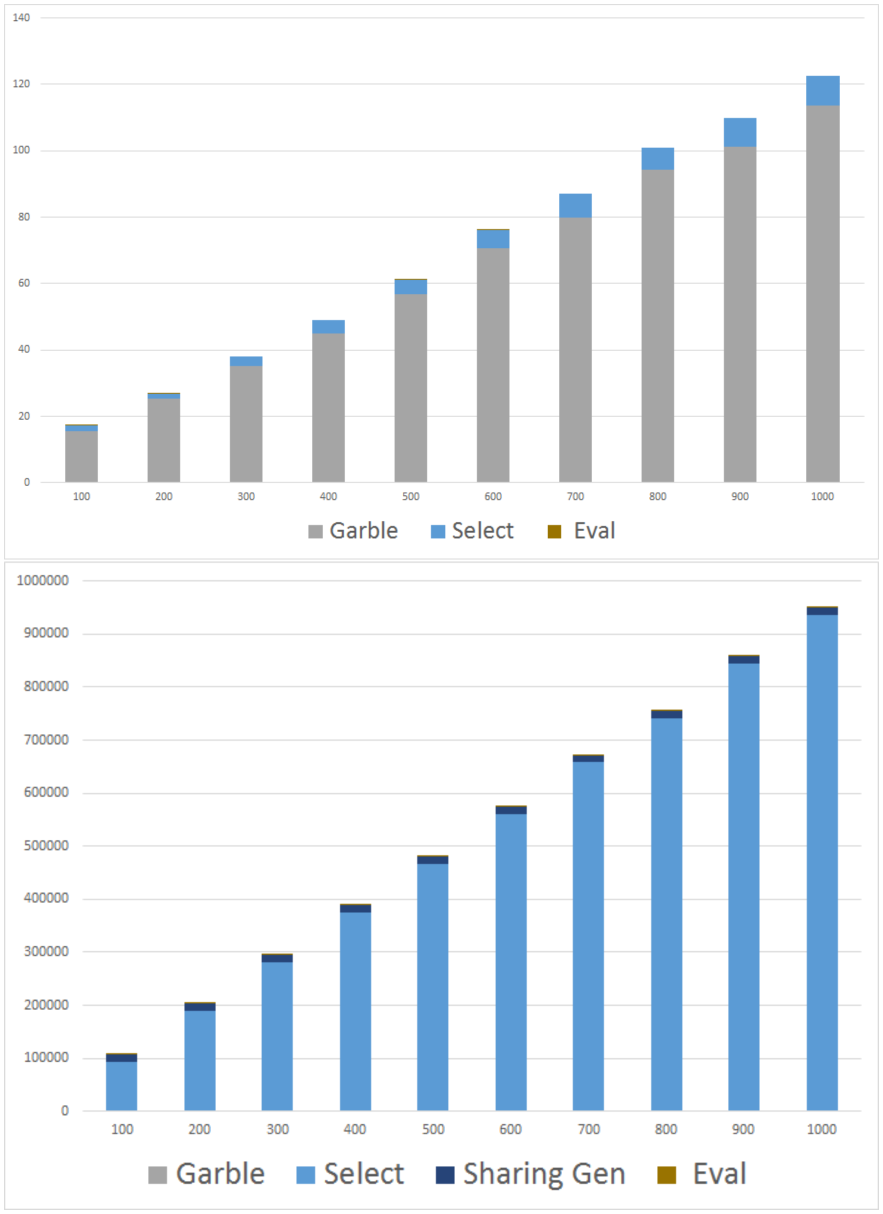

Figure 4.1: Computation times for 3 party (Top) and single server (Bottom) variants with $n = 5$, $|\Sigma| = 32$, and $|Q|$ varying from 100 to 1000. Vertical axis is runtime in milliseconds, horizontal axis is $|Q|$.

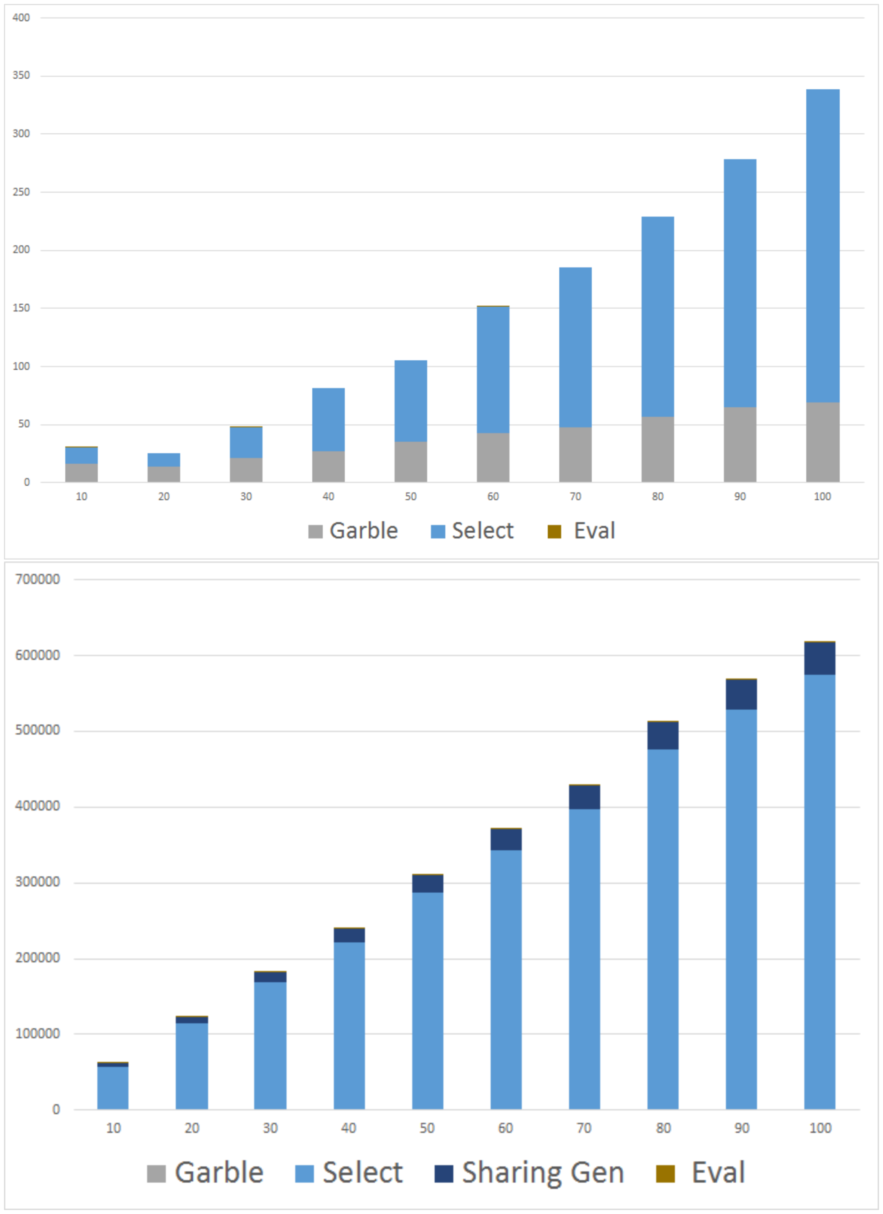

Figure 4.2: Computation times for 3 party (Top) and single server (Bottom) variants with $n = 5$, $|\Sigma|$ varying from 10 to 100, and $|Q| = 200$. Vertical axis is runtime in milliseconds, horizontal axis is $|\Sigma|$.

As a candidate application for this protocol, let us consider the ”spambase” dataset available from UC Irvine [15]. It contains a variety of features extracted from a sample of e-mail texts. From the documentation:

> “48 continuous real [0,100] attributes of type word_freq_WORD = percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail. A ”word” in this case is any string of alphanumeric characters bounded by non-alphanumeric characters or end-of-string.
>
> 6 continuous real [0,100] attributes of type char_freq_CHAR] = percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurences) / total characters in e-mail
>
> 1 continuous real [1,...] attribute of type capital_run_length_average = average length of uninterrupted sequences of capital letters
>
> 1 continuous integer [1,...] attribute of type capital_run_length_longest = length of longest uninterrupted sequence of capital letters
>
> 1 continuous integer [1,...] attribute of type capital_run_length_total = sum of length of uninterrupted sequences of capital letters = total number of capital letters in the e-mail ...”

As shown above, it is not difficult to create the FSM that recognizes, and subsequently is used to count, occurrences of words and characters (we can then convert the results to a fixed point expression obliviously during a Scaling Phase). We can also, use an FSM to find the final variable, “capital_run_length_total”, simply outputting a 1 for this counter every time a capital character occurs. This is a fairly easy extension. The remaining two features, average and longest run length, are not generally within the set of regular languages recognizable by FSAs. We could emulate it given a maximum length, but doing so requires something akin to embedding memory into the FSM, which would be quite costly in terms of states.

Chapter 5
CONCLUSIONS AND FUTURE WORK

Here we have presented an improved protocol for obliviously evaluating finite state machines. As a motivation, we suggested using this as a preprocessing phase for a machine learning protocol, that is, using it to extract relevant features from a sample of text and using those features in a decision tree or other model. It would be a logical next step to develop such an application that actually utilizes this protocol in that manner.

Also, given the dramatic improvements that have been made in the size of binary garbled circuit since their proposal, it seems possible that the size of the garbled FSM might be reducible somewhat. Current garbled circuits have been reduced to roughly half their initial size, and although it is not immediately apparent if any of these techniques can be applied to the garbled FSM, it could merit further investigation, if this protocol were to be used in an application where communication efficiency was essential.

Furthermore, note that while these security proofs assumed the relatively weak passive adversarial model, the proofs are easily extended to the case of the malicious Client using a very similar proof to [17], so long as we posit that $|Q|$ be a power of two.

BIBLIOGRAPHY

[1] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. Xpir: Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies*, 2016(2):155–174, 2015.

[2] Dana Angluin and Michael Kharitonov. When won't membership queries help? In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 444–454. ACM, 1991.

[3] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 479–488. ACM, 1996.

[4] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[5] Marina Blanton and Mehrdad Aliasgari. *Secure Outsourcing of DNA Searching via Finite Automata*, pages 49–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[6] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. Secure multiparty computation goes live. In *International Conference on Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.

[7] Claude Crépeau. *Equivalence Between Two Flavours of Oblivious Transfers*, pages 350–354. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

[8] Yarkın Doröz, Berk Sunar, and Ghaith Hammouri. Bandwidth efficient pir from ntru. In *International Conference on Financial Cryptography and Data Security*, pages 195–207. Springer, 2014.

[9] Keith B Frikken. Practical private dna string searching and matching through efficient oblivious automata evaluation. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 81–94. Springer, 2009.

[10] Rosario Gennaro, Carmit Hazay, and Jeffrey S Sorensen. Text search protocols with simulation based security. In *International Workshop on Public Key Cryptography*, pages 332–350. Springer, 2010.

[11] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[12] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

[13] Joe Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 20–31, New York, NY, USA, 1988. ACM.

[14] Peeter Laud and Jan Willemson. Universally composable privacy preserving finite automata execution with low online and offline complexity. *IACR Cryptology ePrint Archive*, 2013:678, 2013.

[15] M. Lichman. UCI machine learning repository, 2013.

[16] Yehuda Lindell and Benny Pinkas. A proof of security of yaos protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[17] Payman Mohassel, Salman Niksefat, Saeed Sadeghian, and Babak Sadeghiyan. *An Efficient Protocol for Oblivious DFA Evaluation and Applications*, pages 398–415. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[18] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[19] M. O Rabin. How to exchange secrets with oblivious transfer. Technical report, Aiken Computation Lab, Harvard University, 1981.

[20] Dana Ron. *Automata learning and its applications*. PhD thesis, Hebrew University, 1995.

[21] Hirohito Sasakawa, Hiroki Harada, David duVerle, Hiroki Arimura, Koji Tsuda, and Jun Sakuma. Oblivious evaluation of non-deterministic finite automata with application to privacy-preserving virus genome detection. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, pages 21–30, New York, NY, USA, 2014. ACM.

[22] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[23] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Celik. Privacy preserving error resilient dna searching through oblivious automata. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 519–528. ACM, 2007.

[24] Ecem Ünal and Erkay Savaş. Bandwidth-optimized parallel private information retrieval. In *Proceedings of the 7th International Conference on Security of Information and Networks*, page 197. ACM, 2014.

[25] Lei Wei and Michael K. Reiter. *Third-Party Private DFA Evaluation on Encrypted Files in the Cloud*, pages 523–540. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[26] Lei Wei and Michael K Reiter. Third-party private dfa evaluation on encrypted files in the cloud. In *European Symposium on Research in Computer Security*, pages 523–540. Springer, 2012.

[27] Lei Wei and Michael K. Reiter. Toward practical encrypted email that supports private, regular-expression searches. *International Journal of Information Security*, 14(5):397–416, 2015.

[28] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.