Practical Private DNA String Searching and Matching through Efficient Oblivious Automata Evaluation

Keith B. Frikken

Department of Computer Science and Systems Analysis
Miami University, Oxford, OH 45056
frikkekb@muohio.edu

Abstract. In [18] it was shown that the ability to perform oblivious automata evaluation was useful for performing DNA searching and matching. By oblivious automata evaluation we mean that one participant has a finite state machine and the other participant has a sequence, and at the end of the protocol the sequence owner learns whether the machine accepts the sequence. A protocol was given in [18], but it required $O(n)$ rounds (where n is the number of characters in the sequence) and $O(mn)$ modular exponentiations (where m is the number of states in the automata). Both of these factors limit the applicability of this approach. In this paper we propose a new protocol that requires only $O(1)$ rounds and reduces the number of modular exponentiations to $O(n)$ without revealing any additional information. We have implemented both schemes and have shown experimentally that our scheme is two to three orders of magnitude faster than the previous scheme.

Keywords: Privacy-Preserving Protocols, DNA matching, and Cryptography

1 Introduction

In this paper we consider the problem of evaluating an automata in an oblivious manner. That is, one participant, Alice, has an automata that she regards as private, and the other participant, Bob, has a sequence which he regards as private. We desire a protocol where Bob learns whether Alice's automata accepts his sequence, but Bob learns no other information about the automata and Alice learns no information about Bob's sequence. This problem was considered in [18] with the motivation of applying this technology to oblivious DNA searching and matching. For example, it is possible to build a finite state machine that accepts only DNA sequences that contain a specific sub-sequence that is close to a marker sequence. Assuming Alice has found such a marker (perhaps Alice is a large pharmaceutical company) and would like to sell a service to Bob where he learns whether he has a predisposition to the disease, then this technology would make it possible for Bob to learn whether he has a predisposition to the

specific disease without revealing his DNA sequence to Alice or revealing Alice's proprietary information to Bob.

In the near future, it is envisioned that it will be possible to sequence one's own DNA for a modest price. For example, the US National Institute of Health has set a goal that by 2014 it should be possible to sequence a human genome for under 1000 dollars [1]. When such technology exists, genomic information in electronic form will become ubiquitous. This raises serious privacy concerns, as it is easy to envision possible abuses of such information. And while recent legislation in the US will make it illegal to discriminate based on DNA [2], it is desirable to have technology that would allow this data to be useful without having to share it with other entities.

In [18], a protocol was proposed for evaluating an automata in an oblivious manner. However, while their scheme was an excellent first step towards practical DNA searching and matching, their scheme has two performance drawbacks: i) it requires rounds linear in the size of Bob's sequence, and ii) it requires modular exponentiations proportional to the number of states in the automata times the number of characters in Bob's sequence. Since modular exponentiations require significantly more computation than many other operations, they are an accurate estimator of performance for protocols. In this paper we introduce a new scheme that reduces the number of rounds to a small constant and reduces the number of modular exponentiations to a value that is linear in the length of Bob's sequence. Furthermore, we have implemented both our scheme and the scheme in [18] and we show experimentally that our scheme is 2 to 3 orders of magnitude faster than the previous scheme.

1.1 Problem Definition/Notation

The server has a automata denoted by (Σ, S, q, M, F) where Σ is the alphabet (we denote the alphabet as $\Sigma_1, \ldots, \Sigma_{|\Sigma|}$), S is a set of states denoted by $\{s_0, \ldots, s_{m-1}\}$, $q \in S$ is the initial state, M is a $\Sigma \times m$ matrix representing the transition function, i.e., $M(i, j)$ represents the state that the automata enters when processing input Σ_i in state s_j, and $F \subseteq S$ is the set of final states. The server regards this automata as private. The client has a sequence of letters $\ell_1 \cdots \ell_n \in \{\Sigma\}^n$, that it regards as private. For convenience we denote the state that the automata is in after processing the string $\ell_1 \cdots \ell_m$ by $s_{\ell_1 \cdots \ell_m}$. More formally, $s_{\ell_1} = M(\ell_1, q)$ and $s_{\ell_1 \cdots \ell_i} = M(\ell_i, s_{\ell_1 \cdots \ell_{i-1}})$.

The goal of the protocol is for the client to learn whether the server's automata accepts its sequence (i.e., if $s_{\ell_1 \cdots \ell_n} \in F$), without revealing information about the sequence to the server (including the automata's result) and the client learns nothing other than what can be deduced from this result.

In this paper we assume that the automata owner is honest-but-curious in that it will follow the protocol exactly but will try to infer additional information. We consider this model because: i) if a protocol cannot be made practical in this restricted model then there is little hope that it could be made secure in a stronger adversary model and ii) by auditing the server and levying fines for misbehavior would give disincentives to keep the server from misbehaving.

1.2 Our Contributions

The contributions of this work are as follows:

1. We introduce a new protocol for oblivious automata evaluation that requires only a constant number of rounds (a reduction from $O(n)$ in the previous scheme), and reduces the number of modular exponentiations from $O(mn + n|\Sigma|)$ to $O(n)$
2. We have implemented our scheme along with the scheme in [18] and we show experimentally that our proposed scheme is 2-3 orders of magnitude faster than the previous approach.
3. We extend our scheme to support transducers (i.e., finite state machines that produce more than Boolean output).

1.3 Organization of Manuscript

The remainder of this document is organized as follows. In section 2 we describe related work. In section 3 we describe tools needed for our protocol. In section 4 we describe the scheme in [18] in more detail so that it is possible to compare the schemes. In section 5 we introduce our new scheme. In section 7 we introduce experimental results, and we summarize our results in section 8.

2 Prior Work

2.1 Generic SMC

Secure Multiparty Computation (SMC) is the problem of creating a privacy-preserving protocol for any function f[1]; that is, creating a protocol that computes f over distributed inputs while revealing only the result and inferences that can be made from this result. General results state that any function can be computed in such a secure manner. The first constructions for secure two-party SMC were given in [19, 20]; these assumed that the adversary of the protocol was honest-but-curious (HBC) in that the adversary will follow the protocol exactly but will attempt to make additional inferences. Later a construction was given for multiple parties [10] in the malicious adversary model (where the adversary deviates arbitrarily from the protocol) assuming that a majority of the participants are honest. There have also been many other papers attempting to improve the efficiency of these protocols to make the general results practical. Another approach that has been deployed is to introduce a domain-specific protocol that is more efficient than the general results. We describe specific secure protocols for genomic problems in the next section.

One crucial difference between the problem considered in this paper and the above-mentioned general results is that in this paper, the server's input is a

[1] We are assuming that f can be computed in polynomial time when given all of the inputs.

function (i.e., an automata) and the function being evaluated is private. Some prior approaches include: i) *Selective private function evaluation (SPFE)* and ii) Cryptocomputuing. The goal of SPFE [8] is for one party to compute a private function over a subset of another party's database without revealing the function. However, this model is different than the one used in our approach. Cryptocomputing [17] allows two parties to compute a private function on another party's input. This work was extended in [4, 5] to support more general functions. In this paper we introduce a more efficient approach for the specific problem of oblivious automata evaluation.

2.2 Privacy-Preserving Protocols for Genomic Computation

The work most closely related to this paper is [18], which introduced a protocol for oblivious automata evaluation, with the goal of being able to perform robust DNA searching. We describe the protocol proposed in this paper in more detail in section 4. While there are many applications for doing oblivious automata evaluation, a primary motivation is to be able to perform DNA searching and matching. There has been other work that has considered the problem of privacy-preserving DNA matching. For example, the work in [3] introduced a protocol that computes the edit distance between two sequences and a new protocol was introduced in [11] that improved the efficiency of this scheme. Thus the work in this paper is semi-orthogonal to the work in [3, 11] in that: i) for computing the edit distance between two sequences the approach described in this manuscript is less general than the other papers, but this allows the scheme to be more efficient, and ii) however, the scheme proposed in this paper allows for many other computations not supported in [3, 11] in that it can evaluate an arbitrary automata.

3 Building Blocks

3.1 Oblivious Transfer

In this paper (and in [18]) we require the usage of a chosen 1-out-k Oblivious Transfer (OT). In this protocol the sender has k values $v_1, \ldots, v_k$ and the chooser has a specific index $i \in [1, k]$. At the end of the protocol the chooser obtains v_i but does not learn any information about the other values and the sender does not learn which value was chosen. We specifically, use the protocol for OT described in [15], which requires $O(1)$ rounds of communication, $O(1)$ modular exponentiations by both participants[2], $O(k)$ modular multiplications by the server and $O(k)$ communication.

[2] Technically, the sender must perform $O(k)$ exponentiations in an initialization phase but these can be amortized over multiple runs of the protocol

3.2 Yao's Scrambled Circuit Evaluation

In Yao's Scrambled Circuit Evaluation [20], one participant will generate an oblivious circuit that computes the desired outcome and that the other participant will evaluate this circuit. One crucial idea behind this construction is that the generator will choose two random encodings[3] for each wire of the circuit–one corresponding to 0 (false) and the other to 1(true). The evaluator learns the encoding corresponding to the actual value of the wires, but does not know what this encoding corresponds to. While the generator would know the meaning of the encoding, the generator does not know the encoding known to the evaluator. Thus neither participant knows the true value of the wire, but together they do. For a more detailed description of this protocol and a proof of security see [13].

3.3 Additively-Homomorphic Encryption

While the scheme we propose in this paper does not utilize a homomorphic encryption scheme, the scheme in [18] did, and thus to make a detailed comparison between the schemes we provide a brief overview of such encryption schemes. In an additive homomorphic encryption scheme, the product of two ciphertexts is a ciphertext of the sum of the two plaintexts. This allows basic operations to be performed on encrypted values. One specific homomorphic encryption scheme is described in [16]. More specifically, when we refer to homomorphic encryption, we are using an encryption scheme with the following properties: public-key, semantically-secure, additive homomorphism–Given $E(x)$ and $E(y)$, we require that $D(E(x) * E(y)) = x + y$ and $D(E(x)^c) = xc$, and re-encryption–Given $E(x)$ and the public parameters of the homomorphic encryption scheme it is possible to re-encrypt the value to obtain another ciphertext for x (usually this is done by multiplying by $E(0)$). Note that encryption, decryption, and re-encryption all require $O(1)$ modular exponentiations.

4 Previous Scheme and Analysis

In this section we give a high level overview of the scheme introduced in [18], and we also provide a detailed analysis of the cost of this approach. A principle idea of this scheme, is that the sequence owner (hereafter referred to as the client) will learn the current state of the automata obfuscated by an additive permutation factor chosen by the automata owner (hereafter referred to as the server). That is, the current state will be additively split between the two parties so that neither will know the actual value. Initially, the protocol is bootstrapped by having the client and server engage in a 1-out-of-$|\Sigma|$ OT protocol to learn the state of the automata after the first character of the sequence.

Protocol: To set up this protocol, it is assumed that server has received the client's public key for a semantically-secure homomorphic encryption system, and that client has received the setup information to be a chooser in an OT protocol (as in [15]).

[3] At a high level these can be thought of as cryptographic keys.

1. The server chooses n integer values, $h_1, \ldots, h_n$ where each value is chosen uniformly from $[0, m-1]$. The server constructs $|\Sigma|$ values, $v_1, \ldots, v_{|\Sigma|}$ where $v_i = M(i, q) + h_1 \bmod m$ (Recall that q is the initial state of the automata). The server and the client engage in a 1-out-of-$|\Sigma|$ OT protocol where the server inputs $v_1, \ldots, v_{|\Sigma|}$ and the client chooses ℓ_1. The client stores this value as s'_{ℓ_1} (The prime is used to denote that this is the current state additively permuted).
2. In sequence, for each value i from 2 to n they do:
   (a) The client creates a list of encrypted values $E(a'_0), \ldots, E(a'_{n-1})$ where $a'_j = 1$ if $j = s'_{\ell_1 \cdots \ell_{i-1}}$ and is 0 otherwise. The client sends these values to the server.
   (b) The server shifts these values by h_{i-1} positions to obtain $E(a_0), \ldots, E(a_{n-1})$ where $E(a_j) = E(a'_{j-h_{i-1} \bmod m})$. Note that a_j is 1 if and only if $j = s_{\ell_1 \cdots \ell_{i-1}}$. The server creates an $n \times 1$ matrix A with these values. The server then creates a $|\Sigma| \times n$ matrix M' where $M'(i, j) = M(i, j) + h_i \bmod m$. Using standard techniques for computing with homomorphically-encrypted values the server calculates $M'A$ to obtain the values $r_1, \ldots, r_{|\Sigma|}$.
   (c) The server and the client engage in a 1-out-of-$|\Sigma|$ OT protocol where the server inputs $r_1, \ldots, r_{|\Sigma|}$ and the client chooses ℓ_i. The client stores this value as $s'_{\ell_1 \cdots \ell_i}$.
3. The server creates m messages $z_1, \ldots, z_m$ where z_i is 1 if $s_{i-h_n \bmod m}$ is a final state and is 0 otherwise. The server and the client engage in a 1-out-of-m OT protocol where the server inputs $z_1, \ldots, z_m$ and the client chooses $s_{\ell_1 \cdots \ell_m}{}'$. If the client receives 1 then its sequence was accepted by the automata and if the client learns 0 then its sequence was not accepted by the automata.

Analysis: The above protocol requires $O(n)$ rounds as step 2 must be run sequentially. The server's computation for each letter in the sequence is $O(|\Sigma|m)$ as this is the number of modular multiplications to perform the matrix multiply and requires $O(|\Sigma|)$ modular exponentiations, thus in total the server performs $O(|\Sigma|mn)$ operations and $O(|\Sigma|n)$ modular exponentiations. The client must perform $O(m)$ operations and modular exponentiations per character in the sequence, and thus it has to perform $O(nm)$ such operations in total. Finally, the protocol requires $O(\rho_1 mn + \rho_2 n|\Sigma|)$ bits of communication, where ρ_1 is the size of an semantically-secure homomorphic encryption (e.g., 2048 bits for security comparable to 1024-bit RSA keys for the Paillier scheme [16]) and ρ_2 is the size of a hash function (e.g., 160 bits for SHA-1). Table 1 summarizes the performance of this scheme. Two performance bottlenecks with this scheme are: i) the number of rounds grows linearly with the length of the sequence being checked, and ii) the number of modular exponentiations is prohibitively large.

5 Proposed Scheme, Analysis, and Comparison

In this section we introduce our new scheme for oblivious automata evaluation; this scheme borrows ideas from Yao's Scrambled Circuit Evaluation [20]. That

is, a single step of automata processing is similar to processing a single gate in a circuit. Like the previous scheme the client will learn the current state of the automata obfuscated by an additive hiding factor. Also, for each character of the sequence the server will create an encoding[4] for each possible state and the client will learn the specific encoding corresponding to the current state. Also, for each sequence character, the server will choose an encoding for each letter of the alphabet and the client learns the encoding corresponding to his sequence's current letter. Using this permuted state and the state and alphabet encodings, we describe an oblivious transition function that allows the client to compute the next state encoding and permuted state from the previous values. More details are described below:

1. The server chooses $(n-1)|\Sigma|$ random encodings $k_{i,j}$ for $i \in [2,n]$ and $j \in [1,|\Sigma|]$. In the following, these will be referred to in the following as the alphabet encodings. In this protocol the client will learn, k_{i,ℓ_i} for each $i \in [2,n]$ and will not learn any of the other alphabet encodings. These encodings will be revealed via $n-1$ oblivious transfer protocols, which can all be done in parallel.
2. The server chooses mn random encodings $e_{i,j}$ for $i \in [1,n]$ and $j \in [1,m]$. In the following, these will be referred as the state encodings. The client will learn $e_{1,s_{\ell_1}}, e_{2,s_{\ell_1\ell_2}}, \ldots, e_{n,s_{\ell_1\cdots\ell_n}}$ and no other state encodings. That is, the client will learn one state encoding for each letter of his sequence, and this state encoding will be the one corresponding to the actual state of the automata after processing that portion of the client's sequence.
3. The server will choose n state permutation values $h_1, \ldots, h_n$ and the client will learn $s_{\ell_1} + h_1 \bmod m$, $s_{\ell_1\ell_2} + h_2 \bmod m, \cdots, s_{\ell_1\cdots\ell_n} + h_n \bmod m$. That is, the client will learn the obfuscated state of the automata after processing each letter.

We will now describe an oblivious transition function that maps $(k_{g,j}, e_{g-1,s}, s + h_{g-1} \bmod M)$ to $(e_{g,M(j,s)}, M(j,s)+h_g \bmod M)$. That is if after $g-1$ characters the client is in state s, and its gth letter was Σ_j, then this oblivious function allows the client to learn the encoding for $M(j,s)$ (i.e., the next state of the automata) along with the value of this state permuted by the hiding factor. We will denote this gth transition function as a $|\Sigma| \times m$ matrix as G_g, where

$$G_g[j,i] = (e_{g,M(j,s)} || M(j,s) + h_g \bmod m) \oplus f(e_{g-1,s}, k_{g,j})$$

where $s = i - h_g \bmod m$ and f is a pseudorandom function[5]. Note that in order to obtain the value $(e_{g,M(j,s)} || M(j,s) + h_g \bmod m)$ the client must have both $e_{g-1,s}$ and $k_{g,j}$, and since the client has at most one state encoding for $g-1$ and

[4] Hereafter when referring to the term encoding, we are referring to a value chosen from $\{0,1\}^\rho$ for some security parameter ρ

[5] Note this idea is very similar to the ideas used in [20, 14] in that these scheme built an oblivious gate evaluation using similar encodings/ permuted values. Furthermore, such schemes were proven secure in [13]

one alphabet encoding for g it will be able to only obtain one state encoding for g.

Now suppose the client has G_g and that $s = s_{\ell_1 \cdots \ell_{g-1}}$. Further suppose that the client has $i = s + h_{g-1} \bmod m$, $e_{g-1,s}$, k_{g,ℓ_g}, and ℓ_g the client can compute $G_g[\ell_g, i] \oplus f(e_{g-1,s}, k_{g,\ell_g})$ to obtain $e_{g,M(\ell_g,s)}$ and $M(\ell_g, s) + h_g \bmod m$

To help clarify the above process, we now give an example which demonstrates the above protocol. Suppose the server has an automata with two states that processes an alphabet of size two where the automata transition function is as follows: $M[0,0] = 0, M[0,1] = 1, M[1,0] = 0, M[1,1] = 1$. Further suppose that h_{g-1} is 1 and h_g is 0, then the values of G_g are as follows:

- $G_g[0,0] = e_{g,0}||0 \oplus f(e_{g-1,1}, k_{g,0})$
- $G_g[0,1] = e_{g,1}||1 \oplus f(e_{g-1,0}, k_{g,0})$
- $G_g[1,0] = e_{g,0}||0 \oplus f(e_{g-1,1}, k_{g,1})$
- $G_g[1,1] = e_{g,1}||1 \oplus f(e_{g-1,0}, k_{g,1})$

Further suppose that the client is in state 0 after $g-1$ characters and that his next character is 1. Now the client has the following values: $e_{g-1,0}$ (the encoding corresponding to its state), $i = 1$ which is its current state permutes with h_{g-1}, and $k_{g,1}$ (its alphabet encoding for ℓ_g). The client computes $G_g[\ell_g, i] \oplus f(e_{g-1,0}, k_{g,1}) = G_g[1,1] \oplus f(e_{g-1,0}, k_{g,1}) = e_{g,1}||1$ and thus it learns its new permuted state. Note that the permuted state is 1, and since the new hiding factor is 0, this corresponds to the automata being in state 1.

In what follows we describe the protocol in detail:

1. The client and the server engage in $(n-1)$ 1-out-of-$|\Sigma|$ OTs (in parallel) where in the ith protocol the server acts as the sender and uses $k_{i,\Sigma_1}, \ldots, k_{i,\Sigma_{|\Sigma|}}$ and the client inputs ℓ_i for $i \in [2,n]$. After this step the client will have k_{i,ℓ_i} for all $i \in [2,n]$.
2. The client and server engage in 1-out-of-$|\Sigma|$ OT where the server inputs $a_1, \ldots, a_{|\Sigma|}$ where $a_i = e_{1,M(\Sigma_i,q)}||M(\Sigma_i, q)+h_1 \bmod m$ and the client inputs ℓ_1. After this step the client will have the permuted state and the encoding for the automata after processing the client's first character.
3. The server calculates $G_2, \ldots, G_n$ and sends them to the client. The client then evaluates the functions in sequence as described above and after evaluating G_n we will denote the permuted state by $f = s_{\ell_1 \cdots \ell_n} + h_n \bmod m$.
4. The server creates m messages $z_1, \ldots, z_m$ where z_i is $Enc(1, e_{n,i-h_n \bmod m})$ if $s_{i-h_n} \bmod m \in F$ (i.e., it is a final state) and is $Enc(0, e_{n,i-h_n \bmod m})$ otherwise [6]. Note that by encrypting the result with the final encoding this prevents the client from choosing an arbitrary state to learn information about the automata. The server and the client engage in a 1-out-of-m OT protocol where the server inputs $z_1, \ldots, z_m$ and the client chooses f. The client decrypts the value with the encoding of the final state and if the client receives 1 then its input was accepted by the automata and if the client learns 0 then it was not accepted by the automata.

[6] We are denoting the encryption of a message m with a key k was $Enc(M, k)$.

Analysis: Note that the above protocol can be done in $O(1)$ rounds. Also, the server needs to perform $O(mn|\Sigma|)$ work to calculate all of the oblivious transition functions, but only needs to perform $O(n)$ modular exponentiations (to perform OTs). Similarly the same amount of computation and modular exponentiations need to be performed by the client. Finally, the size of the oblivious transition function information is $O(\rho_2 mn|\Sigma|)$ (where ρ_2 is the size of a secure pseudorandom function) and to perform the n OTs the communication requirements are $O(\rho_1 n\Sigma)$ (where ρ_1 is the size of a modulus where the discrete logarithm problem is hard).

The careful reader may be wondering why the above scheme does not simply just use Yao's garbled circuit evaluation to compute the state of the automata. That is, what is the efficiency of building a circuit that evaluates the automata? The efficiency of this approach is a function depends on two factors: i) the number of inputs into the circuit, and ii) the number of gates in the circuit. Clearly, the number of inputs would be $O(n \log |\Sigma|)$ as each of the n letters has $O(\log \Sigma)$ bits. The evaluation of each state is essentially a table lookup in a table with $O(m|\Sigma|)$ entries. Thus in the straightforward circuit, each letter will require $O(m|\Sigma|)$ equality checks each involving $O(\log m + \log |\Sigma|)$ bits. And so, each state would require $O(m|\Sigma|(\log m + \log |\Sigma|))$ computation, and thus the total computation would be $O(mn|\Sigma|(\log m + \log |\Sigma|))$.

Table 1 summarizes the performance of all three approaches. We now compare the scheme in this paper and the scheme in [18]: Clearly, the number of rounds required by our scheme is substantially improved in our scheme. Furthermore, we have essentially replaced the modular exponentiations of the previous protocol with the evaluation of a pseudorandom function. And since pseudorandom functions are two to three orders of magnitude faster than homomorphic encryptions, we would expect to see a large performance difference between these schemes. Finally, it might appear that the communication required by our protocol is higher than that of the previous scheme, for the specific problem of DNA matching (i.e., when $|\Sigma| = 4$), then $\rho_2|\Sigma| \leq \rho_1$, so the communication between these schemes is comparable. Also, the performance of the proposed scheme is asymptotically superior to the generic construction based on Yao's protocol.

Metric	**Original [18]**	**Proposed**	**Yao-based**
Rounds	$O(n)$	$O(1)$	$O(1)$
Server Computation	$O(mn\|\Sigma\|)$	$O(mn\|\Sigma\|)$	$O(mn\|\Sigma\|(\log m + \log \|\Sigma\|)))$
Server Mod Exps	$O(n\|\Sigma\|)$	$O(n)$	$O(n \log \|\Sigma\|)$
Client Computation	$O(mn)$	$O(mn)$	$O(mn\|\Sigma\|(\log m + \log \|\Sigma\|)))$
Client Mod Exps	$O(mn)$	$O(n)$	$O(n \log \|\Sigma\|)$
Communication	$O(\rho_1 mn + \rho_2 n\|\Sigma\|)$	$O(\rho_1 n\|\Sigma\| + \rho_2 mn\|\Sigma\|)$	$O(\rho_2 mn\|\Sigma\|(\log m + \log \|\Sigma\|))$ $+\rho_1 n \log \|\Sigma\|)$

Table 1. Performance Comparison Summary

Security Analysis In the full version of the paper we will provide a detailed security proof, but we give only a brief overview here. The basic argument rests

on the composition theorem from [7], that is when we replace the OT protocols and oblivious gate evaluation function with ideal versions (that utilize a Trusted Third Party) of the protocol it is straightforward to show that the resulting protocol is secure. Thus when we replace these functions with secure protocols, the resulting protocol is also secure. More specifically, we will first utilize ideal implementations of Oblivious Transfer (OT) and of Oblivious State Evaluation (OSE), specifically, OT will allow the client to input a value to a TTP $\ell \in [1, |\Sigma|]$ and the server to input Σ values $k_1, \ldots, k_\Sigma$ to the TTP. After receiving all of these values the OT TTP sends to the client the values k_ℓ and the server receives $\perp$. In the ideal OSE protocols, the client sends $(k_{g,j}, e_{g-1,s}, s+h_{g-1} \bmod M)$ and the server sends $(k_{g,1}, \ldots, k_{g,|\Sigma|}, e_{g-1,1}, \ldots, e_{g-1,n}, e_{g,1}, \ldots, e_{g,n}, M, h_g, h_{g+1})$ and after receiving all of the inputs the TTP sends the client $(e_{g,M(j,s)}, M(j,s)+h_g \bmod m)$ and sends the server $\perp$. Note that the ideal OT functionality can be achieved with the protocol in [15] and the ideal OSE functionality follows from [13].

6 Extensions

6.1 Efficiency Improvements

In this section we propose three techniques to further improve the efficiency of our approach.

Communication Reduction By utilizing Private Information Retrieval (PIR), it is possible to reduce the communication to something sub-linear in mn, however this performance increase comes at a cost of increasing the rounds to $O(n)$. Recall that in PIR a sender has a database of values and a chooser learns at least one of the items of his choosing from the database without the sender learning which value was chosen. Single server solutions for PIR exist that require sub-linear communication [9, 12, 6]. To integrate this with our scheme, notice that when processing G_g in Step 3, the client only needs the one specific value from this function (the one corresponding to the permuted state and current letter). Thus the client and server could engage in a PIR protocol where the client learns the value that it needs to move to the next state of the automata.

Modular Exponentiation Reduction Using the techniques described in [15] it is possible to reduce the number of modular exponentiations. The principle idea behind this is that it is possible to replace the n 1-out-of-$|\Sigma|$ OTs with n/k 1-out-of-$|\Sigma|^k$ OTs (i.e., we merge k OTs together). This reduces the number of modular exponentiations to n/k, but it increases the computation/communication due to OTs to $\frac{n|\Sigma|^k}{k}$. However for small $|\Sigma|$ and k this approach may improve the performance of the scheme, because the additional expense is outweighed by performance gain from removing the modular exponentiations.

Precomputation Another advantage of the scheme proposed in this paper over the scheme in [18] is that in this scheme the server can precompute a large portion of its work. More specifically, in our scheme the server can precompute all of the oblivious transition functions as they do not depend on the client's input. However, in the scheme outlined in [18], the server must wait until it receives the client's homomorphic encryption key (which is different for every client) before it can do any of the operations for that client.

6.2 Extending protocol to support transducers

In this section we describe modifications to our protocol that allow us to evaluate a transducer instead of an automata, note that the protocols in [18] also had such a generalization. In what follows we describe schemes for both Moore machines and Mealy machines.

Moore machines At a high level a Moore machine is a automata that produces an output vector with one character of output for each sequence character and furthermore the output is determined by the current state of the machine (i.e., each state produces a specific output character). More formally, a Mealy machine is a 6-tuple $(\Sigma, \Pi, S, q, M, \lambda)$ where Σ, S, q, M have the same meaning as in a automata, Π is the output alphabet, and $\lambda : S \rightarrow \Pi$ maps each state to an ouptut symbol. In such a protocol, if the machine is in states $t_1, \ldots, t_n$ when processing the sequence, the client will learn $\lambda(t_1) \cdots \lambda(t_n)$ [7]

To modify our protocol to support such machines, the server will include the output character in the oblivious transition function. More specifically, it will change $G_g[j, i] =$

$$(e_{g,M(j,s)}||M(j,s) + h_g||\lambda(M(j,s))) \oplus f(e_{g-1,s}, k_{g,j})$$

where $s = i - h_g \bmod m$. Of course the pseudorandom function would have to be expanded to include enough bits to encrypt the output symbol. Another change to the protocol is that the final step which decodes the final state can be omitted.

Mealy machines The principle difference between a Mealy machine and a Moore machine is that a Mealy machines output depends on the state and the current character of the sequence. More formally, a Moore machine is a 6-tuple $(\Sigma, \Pi, S, s, M, \lambda)$ where Σ, S, s, M have the same meaning as in a automata, Π is the output alphabet, and $\lambda : S \times \Sigma \rightarrow \Pi$ maps each state and character combination to an output symbol. The only change that needs to be done is to change $G_g[j, i] =$

$$(e_{g,M(j,s)}||M(j,s) + h_g||\lambda(M(j,s), j)) \oplus f(e_{g-1,s}, k_{g,j})$$

(i.e., we change $\lambda(M(j,s))$ to $\lambda(M(j,s), j)$.

[7] It is straightforward to modify this protocol so that this output sequence is additively split between the client and the server.

7 Experimental Evaluation

We implemented both the protocol in [18] and the one described in this paper. The implementation was in Java, and the experiments were run on two Dell machines with 2.0 GHz Intel Core Duo processors and 512MB of RAM machines connected in a LAN setting. For all experiments we set the size of the alphabet to be 4. In the first experiment we ran we set the size of the client's sequence to 10 and we varied the states in the server's automata from 10 to 200. We ran each test 10 times. In figure 1 we show the performance difference between the previous schemes and our scheme. By this we mean this is the value of (previous scheme's running time)/(our scheme's running time). Notice that as the number of states increases the performance difference also appears to increase. Also notice that for 200 states our scheme is about 400 times faster than the previous approach.

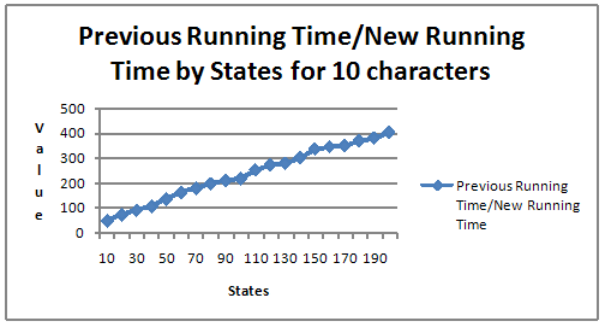

Fig. 1. Performance Difference by States

In the second experiment we set the number of states to be 10 and varied the number of letters from 10 to 200. We also ran these tests 10 times. Like the previous test we considered the performance difference between the two schemes. Figure 2 shows the results. Notice that as the number of letters gets large our scheme is about 100 times better, but that this also appears to level off at this value. This would be expected because the number of modular exponentiations in our scheme is $O(n)$, and thus n has a significant impact on performance for our scheme.

In our final experiment (see Figure 3) we consider the running time in milliseconds for our scheme on 10 characters but we vary the number of states from 500 to 10000. Notice that to process 10 characters in a 10000 state automata our scheme seems to require about 8 seconds. When we extrapolate our data for the previous scheme we estimate that the previous scheme would take about 4 hours to handle this size of automata (however due to the length of such an experiment we did not run this test to verify if this was true). To put this in perspective, according to [18] an automata with 2000 states can be built that

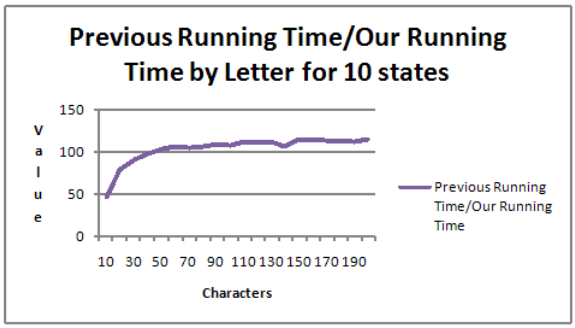

Fig. 2. Performance Difference by Letters

accepts all sequences which contain a subsequence with edit distance 2 (or less) from another sequence of length 50.

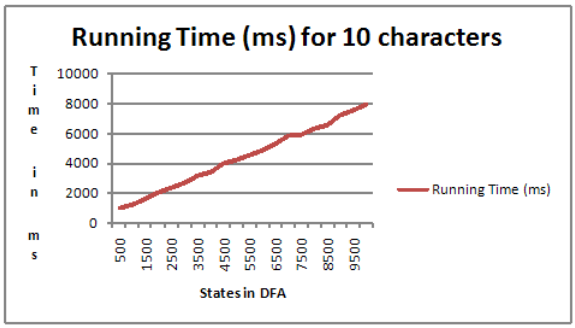

Fig. 3. Performance of Our Scheme for 10 characters by States

8 Summary

In this paper we introduced a scheme for oblivious automata evaluation. Previous work in this area has highlighted the importance of this problem when doing DNA searching and matching. The advantages of our approach is that we reduce the rounds to $O(1)$ and we significantly reduce the number of modular exponentiations. This turns out to be significant in practice as we show experimentally that our scheme is 2 to 3 orders of magnitude faster than the scheme in [18].

References

1. The 100 dollars Genome,Technology Review, published by MIT, April 17, 2008. `http://www.technologyreview.com/Biotech20640/page1/`.

2. Genetic Information Nondiscrimination Act, http://en.wikipedia.org/wiki/Genetic_Information_Nondiscrimination_Act.
3. M. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *WPES '03: Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, pages 39–44, New York, NY, USA, 2003. ACM.
4. D. Beaver. Minimal-latency secure function evaluation. In *EUROCRYPT 2000, Lecture Notes in Computer Science*, volume 1807, pages 335–350, 2000.
5. C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-round secure computation and secure autonomous mobile agents. In *Lecture Notes in Computer Science (ICALP)*, volume Volume 1853, 2000.
6. Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. *Lecture Notes in Computer Science*, 1592:402–414, 1999.
7. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
8. R. Canetti, Y. Ishai, R. Kumar, M. Reiter, R. Rubinfeld, and R. Wright. Selective private function evaluation with applications to private statistics. In *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 293–304. ACM Press, 2001.
9. Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.
10. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 218–229. ACM Press, 1987.
11. S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 216–230, May 2008.
12. E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, pages 364–373. IEEE Computer Society, 1997.
13. Y. Lindell and B. Pinkas. A proof of yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004. http://eprint.iacr.org/.
14. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *Proceedings of Usenix Security*, 2004.
15. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.
16. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology: EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999.
17. Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC1. In *40th Annual Symposium on Foundations of Computer Science*, pages 554–566, 1999.
18. J. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient dna searching through oblivious automata. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 519–528, New York, NY, USA, 2007. ACM.

19. A. Yao. Protocols for secure computations. In *Proceedings of the 23th IEEE Symposium on Foundations of Computer Science*, pages 160–164. IEEE Computer Society Press, 1982.
20. A. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Society Press, 1986.