Secure Outsourcing of DNA Searching via Finite Automata

Marina Blanton and Mehrdad Aliasgari

Department of Computer Science and Engineering, University of Notre Dame,
{mblanton,maliasga}@cse.nd.edu

Abstract. This work treats the problem of error-resilient DNA searching via oblivious evaluation of finite automata, where a client has a DNA sequence, and a service provider has a pattern that corresponds to a genetic test. Error-resilient searching is achieved by representing the pattern as a finite automaton and evaluating it on the DNA sequence (which is treated as the input), where privacy of both the pattern and the DNA sequence must be preserved. Interactive solutions to this problem already exist, but can be a burden on the participating parties. Thus, in this work we propose techniques for secure outsourcing of oblivious evaluation of finite automata to computational servers, such that the servers do not learn any information. Our techniques are applicable to any type of finite automata, but the optimizations are tailored to the setting of DNA searching.

1 Introduction

The need to protect private or sensitive information about an individual is widely recognized. Recent advances in bioinformatics and biomedical science promise great potential in our ability to understand and compute over genome data, but the DNA of an individual is highly sensitive data. In recent years, several publications appeared that allow for computing over DNA data in a private manner with the purpose of identifying ancestry relationships or genetic predisposition. In particular, results are known for sequence comparisons that compute the edit-distance [3, 18], error-resilient pattern matching based on finite automata (FA) evaluation [28, 14], and specific DNA processing for the purposes of ancestry testing [8].

DNAs or DNA fragments used in such computations are large in size. For that reason, recent work [18, 29] concentrated on improving the efficiency of such protocols, but they still remain computation- and communication-intensive. Thus, if a customer would like to engage in a private computation that uses her DNA, she might not necessarily have computational resources and/or bandwidth to carry out the protocol. When this is the case, it is natural to consider outsourcing the computation to powerful servers or a large distributed network such as a computational grid. Obviously, in such a setting the privacy of all sensitive inputs (the customer's DNA, the service provider's tests, etc.) must be preserved from the servers participating in the computation.

Results for privacy-preserving outsourcing of the edit distance computation of two strings are known [4, 5], but outsourcing of more general type of computation over DNA via finite automata has remained unexplored. Thus, the focus of this work is on secure outsourcing of oblivious evaluation of a finite automaton on a private input. We use the

work of Troncoso-Pastoriza et al. that pioneered techniques for oblivious finite automata evaluation (OFAE) [28] as a starting point for out solution and develop techniques for outsourcing such computations.

Using FA for DNA searching and matching is motivated by the fact that queries to DNA data need to take into account various errors such as clinically irrelevant mutations, sequencing errors, incomplete specifications, etc. Such errors can be tolerated if the pattern is expressed using regular expressions, implemented as FA. We refer the reader to [28] for a more detailed description of the types of searching and alignment algorithms that can be implemented using this technique. Then a service provider (such as, e.g., 23andMe [1]) can build a FA that implements a genomic test, and a customer who possess a private DNA sequence will use it as an input to the automaton. A DNA sequence is specified as a string of characters over the alphabet $\Sigma = \{A, C, T, G\}$ of length N, and a deterministic finite automaton (also called a finite state machine (FSM)) corresponding to a DNA test is specified as a tuple $M = (Q, \Sigma, \Delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\Delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. Without loss of generality, the transition matrix is assumed to be complete, i.e., it specifies a transition from each state on each input, and can be represented as a table of size $|Q| \times |\Sigma|$, where each value stores a state. The states are represented as integers in $\mathbb{Z}_{|Q|}$ and input characters are represented as integers in $\mathbb{Z}_{|\Sigma|}$. A FA M accepts a string $x = x_0 x_1 \ldots x_{N-1} \in \Sigma^N$ if on input x it transitions from q_0 to a state $q_N \in F$.

Contributions. Our contributions can be summarized as follows:

- We first show how the solution of [28] can be simplified to improve both the computation and communication in practice for typical values of the parameters (i.e., when $|\Sigma|$ is small). We also provide a detailed analysis of both the original and modified solutions (more specific than just asymptotic analysis) and show that the communication cost can be rather high and not suitable for all clients. Since most of the communication overhead of the solutions comes from the oblivious transfer (OT) protocol, we analyze the performance of the protocols using different OT realizations that allow us to achieve a computation-computation tradeoff.
- We then give a protocol for outsourcing the computation of both the client and the FA owner (service provider) to two computational servers without increasing either the communication or computational complexity of the protocol. The communication complexity of the client and service provider becomes linear in the size of their data and involves virtually no computation.
- Next, we give a protocol that works for outsourcing the computation to any number of servers (i.e., the multi-party case). To minimize the overhead, we use a different structure from that used in the two-party outsourcing solution. To lower the communication complexity (and in part the computation overhead), we represent the transition matrix Δ as a square, so that the communication is decreased from $O(|\Sigma| + |Q|)$ to $O(\sqrt{|\Sigma||Q|})$.
- We also develop a threshold version of the multi-party outsourcing protocol which makes the solution suitable to work in unstable or dynamic environments such as grids. Due to space limitation, it could not be included in this article and can be found in the full version [7].

2 Related Work

There is a considerable number of publications on secure DNA comparison and matching (see, e.g., [3, 5, 27, 28, 8, 19]). The majority of them (such as [3–5, 18, 29]) use dynamic programming (DP) to securely compute the edit distance between a pair of genomic sequences: There are two parties, each holding its respective sequence, and the algorithms compute the edit distance between the sequences without revealing any information besides the output. Since the DP techniques involve computation quadratic in the size of the inputs, such solutions are computation and/or communication heavy. For that reason, consecutive work [4, 5] considered outsourcing the edit distance computation to more powerful helper servers, and another line of research [18, 29] concentrated on making such solutions more efficient. Related to them, [13] gives secure computation and outsourcing of the longest common subsequence (LCS) using an optimized DP algorithm and a communication-efficient Private Information Retrieval protocol [15]. This work is continued in [17] and gives secure structures for the LCS computation and privacy-protecting equivalence and sampling algorithms for finite regular languages.

While these techniques are likely to improve the communication and/or computation complexity of the original DP solution, one might consider the edit distance computation to be a specific type of DNA comparison that might not be suitable when, e.g., error-resilient searching is necessary (handling sampling errors, incomplete specifications, etc.). For that reason, another line of research [28, 14] uses FSMs to implement error-resilient searching over DNA data, and can support any searches that can be formulated as regular languages. These publications provide secure two-party protocols for OFAE, which can be used in any context and is not limited to DNA searching. We use the first publication in this domain [28] as a starting point for our outsourcing construction. A follow-up work [14] uses techniques similar to generic Boolean circuit evaluation to significantly lower the round complexity of the protocol (from $O(N)$ to $O(1)$) and lower the computation complexity as well. The circuit-based approach, however, does not generalize to the outsourcing scenario, since it assumes that the function to be evaluated (i.e., a FA in our case) is known to the participants. Similarly, other general secure function evaluation approaches are not suitable for the same reason.

Other work on privacy-preserving computing over DNA data includes [27], where the authors introduce a strategy for enhancing data privacy in a distributed network deploying the Smith-Waterman algorithm for sequence comparison. In [8], the authors build secure multi-party protocols for specific genetic tests such as parental tests; the approach can also handle a small number of errors, but the complexity of the protocol rapidly increases with the number of errors it can tolerate. Lastly, [19] presents a cryptographic framework for executing queries on databases of genomic data, where data privacy is achieved by relying on two non-colluding third parties.

3 Preliminaries

Homomorphic encryption. Prior and our work relies on a semantically secure homomorphic public-key encryption scheme. Let a public-key encryption scheme be defined as $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, where Gen is a key generation algorithm that, on input a security parameter 1^κ, produces a public-private key pair (pk, sk); Enc is an encryption

algorithm that, on input pk and message m, produces a ciphertext c; and Dec is a decryption algorithm that, on input (pk, sk) and ciphertext c, produces decryption of c m. For brevity of exposition, we use notation $\mathsf{Enc}_{pk}(m)$ and $\mathsf{Dec}_{sk}(c)$. Let n denote the public modulus associated a public key pk; the message space is then $\mathbb{Z}_n^*$. In our description we will assume that $|n| = \kappa$.

With homomorphic encryption, operations on ciphertexts translate into certain operations on the underlying plaintexts. For additively homomorphic schemes, $\mathsf{Enc}_{pk}(m_1) \cdot \mathsf{Enc}_{pk}(m_2) = \mathsf{Enc}_{pk}(m_1 + m_2)$, which implies $\mathsf{Enc}_{pk}(m)^a = \mathsf{Enc}_{pk}(m \cdot a)$ for known a. A ciphertext $\mathsf{Enc}_{pk}(m)$ can be re-randomized by multiplying it to $\mathsf{Enc}_{pk}(0)$; this makes it infeasible to link the new ciphertext to the original one.

Oblivious transfer. A 1-out-of-t oblivious transfer, OT_1^t, allows the receiver to retrieve one item from the t items at the sender in such a way that the receiver does not learn anything other than the item it received and the sender learns nothing. This is a heavily studied cryptographic tool, with many available realizations. The use of different OT protocols from the literature allows one to achieve tradeoffs between sender computation, receiver computation, and their computation. In particular, OT_1^t protocol from [25] has very efficient amortized cost (one modulo exponentiation per OT for both the sender and the receiver) and linear communication cost $O(t)$. Other protocols (e.g., [21, 15]) can achieve sub-linear communication, but generally have larger computation requirements. Depending on the parameters used in OFAE (such as the number of states, input length, etc.) and resources available to the participants, one scheme might be preferred over another. We use different OT schemes in our detailed analysis in Section 5.

Oblivious evaluation of finite automata. Here we review the solution of [28], which is used as a starting point in this work. The service provider $\mathcal{S}$ holds Δ and the client $\mathcal{C}$ holds input x. The evaluation processes one input character at a time, and the current state is shared between $\mathcal{C}$ and $\mathcal{S}$ modulo $|Q|$. Throughout this paper, we will assume that the rows of the matrix are numbered 0 through $|Q| - 1$, and the columns of the matrix are numbered from 0 to $|\Sigma| - 1$. The solution consists of three sub-protocols: (i) a protocol for performing the first state transition, (ii) a protocol for executing a general kth state transition (for $k = 1, \ldots, N - 1$), and (iii) a protocol for announcing the result to the client. Our description of the (main) kth state transition protocol here is slightly different from its original presentation in [28]: it is described for a transposed matrix to improve efficiency of the protocol (as was suggested in [28]). We will use q_i to denote the current state in the execution after processing i input characters. Notation $a \stackrel{R}{\leftarrow} A$ means that a is chosen uniformly at random from the set A. The protocol uses a homomorphic encryption scheme $\mathcal{E}$ for which $\mathcal{C}$ knows (pk, sk) and $\mathcal{S}$ knows pk.

Protocol for 1st state transition. This protocol allows $\mathcal{C}$ and $\mathcal{S}$ to evaluate the automaton on the first input symbol, i.e., compute $q_1 = \Delta(q_0, x_0)$, and share it in an additively split form, i.e., $\mathcal{S}$ learns $q_1^{(1)}$ and $\mathcal{C}$ learns $q_1^{(2)}$ such that $q_1^{(1)} + q_2^{(2)} \bmod |Q| = q_1$.

1. $\mathcal{S}$ chooses $r \stackrel{R}{\leftarrow} \mathbb{Z}_{|Q|}$ and blinds each value in row q_0 by adding r to it modulo $|Q|$.
2. The parties engage in an $\mathrm{OT}_1^{|\Sigma|}$, where the sender $\mathcal{S}$ uses the blinded row q_0 as its database and receiver $\mathcal{C}$ retrieves the element at position x_0.

At the end, $\mathcal{S}$ has $q_1^{(1)} = -r \bmod |Q|$ and $\mathcal{C}$ has $q_1^{(2)} = q_1 + r \bmod |Q|$.

Protocol for kth state transition. In the beginning of the protocol, $\mathcal{C}$ and $\mathcal{S}$ additively share the kth state (i.e., $\mathcal{S}$ has $q_k^{(1)}$ and $\mathcal{C}$ has $q_k^{(2)}$ such that $q_k = q_k^{(1)} + q_k^{(2)} \bmod |Q|$); $\mathcal{C}$ also holds the next input character x_k and $\mathcal{S}$ holds the transition matrix Δ. The output consists of $\mathcal{C}$ and $\mathcal{S}$ additively sharing the $(k+1)$st state q_{k+1}.

1. $\mathcal{S}$ chooses $r \xleftarrow{R} \mathbb{Z}_{|Q|}$ and blinds each element of Δ by adding r to the element modulo $|Q|$. $\mathcal{S}$ rotates the matrix $q_k^{(1)}$ rows up to obtain modified matrix by Δ_k.
2. $\mathcal{C}$ generates a binary vector of length $|\Sigma|$ consisting of a 1 at position x_k and 0's in other positions. $\mathcal{C}$ encrypts the vector with pk and sends encrypted bits $e = (e_0, \ldots, e_{|\Sigma|-1})$ to $\mathcal{S}$, where each $e_i = \mathsf{Enc}_{pk}(b_i)$ and $b_i \in \{0, 1\}$.
3. $\mathcal{S}$ performs matrix multiplication of e and Δ_k using the homomorphic properties of the encryption. As a result, $\mathcal{S}$ obtains a new vector $v = (v_0, \ldots, v_{|Q|})$, that corresponds to an element-wise encryption of the column at position x_k.
4. Both parties engage in an $\mathrm{OT}_1^{|Q|}$, where $\mathcal{S}$ plays the role of the sender using vector v and $\mathcal{C}$ plays the role of the receiver and retrieves the element at position $q_k^{(2)}$.
5. $\mathcal{C}$ decrypts the value and obtains its share $q_{k+1}^{(2)}$, while $\mathcal{S}$ sets its share to $q_{k+1}^{(1)} = -r$.

Protocol for announcement of result. In the beginning of the protocol, $\mathcal{C}$ and $\mathcal{S}$ additively share state q_N modulo $|Q|$. As a result of this protocol, $\mathcal{C}$ learns whether the evaluation resulted in an accept state or not, i.e., it learns a bit.

1. $\mathcal{S}$ generates a random binary vector f of length $|Q|$ by setting its element at position $j + q_N^{(1)}$ to 1 if the state $j \in F$, and to 0 otherwise.
2. Both parties engage in an $\mathrm{OT}_1^{|Q|}$, where $\mathcal{S}$ plays the role of the sender using vector f and $\mathcal{C}$ plays the role of the receiver and retrieves the element at position $q_N^{(2)}$.

4 Security Model

We identify the requirements that a scheme for secure outsourcing of OFAE must meet:

Correctness**:** The protocol computation should provide the client with correct evaluation of the service provider's finite state machine M on the client's input x.

Efficiency**:** Communication and computation complexity of $\mathcal{C}$ ($\mathcal{S}$) should be linear in the size of its input x (in the size of the automaton M (i.e., the size of Δ), respectively). Communication and computation complexity (including round complexity) of the servers should be minimized if possible.

Security**:** The servers should not learn any information throughout the protocol execution. We assume that the servers are trusted to perform their computation correctly, i.e., they are semi-honest or honest-but-curious in that that they will follow the protocol as prescribed, but might attempt to learn additional information from the intermediate values.

We now can formally define security using the standard definition in secure multi-party computation for semi-honest adversaries. Since the computational servers do not contribute any data to the computation, this should be interpreted as no private input to the function they are evaluating. Then for the purposes of the security definition, all data the servers receive before or during the computation (i.e., the transition matrix and client's input) will be considered to be a part of the function evaluation and therefore must not leak any information. We denote "no data" by a special character $\perp$.

Definition 1. *Let parties $P_0, \ldots, P_{m-1}$ engage in a protocol π that computes function $f(\perp, \ldots, \perp) = (o_0, \ldots, o_{m-1})$, where o_i denotes output of party P_i. Let $\mathrm{VIEW}_\pi(P_i)$ denote the view of participant P_i during the execution of protocol π. It is formed by P_i's input and any internal random coin tosses r_i, as well as messages $m_1, \ldots, m_t$ passed between the parties during protocol execution $\mathrm{VIEW}_\pi(P_i) = (\perp, r_i, m_1, \ldots, m_t)$. We say that protocol π is secure against semi-honest adversaries if for each party P_i there exists a probabilistic polynomial time simulator S_i such that $\{S_i(f(\perp, \ldots, \perp))\} \equiv \{\mathrm{VIEW}_\pi(P_i), \perp\}$, where $\equiv$ denotes computational indistinguishability.*

Note that this standard model allows the computational servers to collude with each other (i.e., share the information) in the multi-party case. The security guarantees must hold as long as the coalition size does not exceed a specific threshold. The computational servers do not receive any output, but rather communicate the result to $\mathcal{C}$.

5 Secure FSM Evaluation

Before proceeding with outsourcing solutions, we give a simplification of the original approach that simultaneously improves communication and computation requirements for DNA computation. Our simplification involves representing the matrix Δ as a one-dimensional list (as opposed to a two-dimensional table), and does not affect either the functionality or security of the solution while allowing us to skip encryption and manipulation of encrypted data. When we represent the matrix as a one-dimensional list, we reference element (i, j) of the matrix as the element at index $|\Sigma|i + j$ in the list.

Protocol for 1st state transition. The same as before.

Protocol for kth state transition. Prior the protocol, $\mathcal{C}$ and $\mathcal{S}$ additively share the kth state modulo $|Q|$, and the output of the protocol consists of $\mathcal{C}$ and $\mathcal{S}$ additively sharing the $(k+1)$st state.

1. $\mathcal{S}$ chooses $r \xleftarrow{R} \mathbb{Z}_{|Q|}$ and blinds each element of Δ by adding r to it modulo $|Q|$.
2. $\mathcal{S}$ rotates the matrix Δ $q_k^{(1)}$ rows up. Let Δ_k denote the modified matrix. $\mathcal{S}$ then represents Δ_k as a list of $|Q| \cdot |\Sigma|$ elements.
3. $\mathcal{C}$ and $\mathcal{S}$ engage in $\mathrm{OT}_1^{|Q| \cdot |\Sigma|}$, at the end of which $\mathcal{C}$ obtains the element at position $|\Sigma| \cdot q_k^{(2)} + x_k$ from the list corresponding to Δ_k.

Protocol for announcement of result. The same as before.

We now can compare performance of the protocol above with the original solution from [28]. As suggested in [28], we assume that the efficient OT_1^t protocol with amortized single exponentiation per transfer [25] is used. Also, since in this application $|\Sigma| \ll |Q|$, we assume that the transition matrix is transposed (as presented in Section 3) to result in maximal savings from the OT protocol.

In the analysis, we include all modular exponentiations and also count modular multiplications if their number is large; the overall complexity is expressed in the number of modular exponentiations (1 mod exp $= \kappa$ mod mult). The results for k executions of the main state transition protocol are presented in Table 1 (the rest of the work is significantly lower). In the original scheme, in each protocol round, $\mathcal{C}$ performs $|\Sigma|$

	Original [28]	Modified
$\mathcal{C}$'s exps	$(\|\Sigma\|+2)N$	N
$\mathcal{S}$'s exps	$\|Q\|+N(1+(\log(\|Q\|)\|\Sigma\|+\|\Sigma\|-1)\|Q\|/\kappa)$	$\|Q\|\|\Sigma\|+N(1+\|\Sigma\|\|Q\|/\kappa)$
Comm	$2\kappa N(\|\Sigma\|+\|Q\|)$	$\log(\|Q\|)N\|\Sigma\|\|Q\|$

Table 1. Analysis of original and modified oblivious automata evaluation solutions.

encryptions, 1 decryption, and 1 modular exponentiation (for the OT protocol). $\mathcal{S}$'s work to execute one OT protocol involves $|Q|+1$ modular exponentiations and $|Q|$ modular multiplications. To process the client's response in each round, it performs $|Q||\Sigma|$ modular exponentiations with small exponents (or length $\log|Q|$), which results in $N(\log(|Q|)|\Sigma|)/\kappa$ regular modular exponentiations overall. Since the client sends $|\Sigma|$ encrypted values and the OT protocol involves the transfer of $|Q|$ encrypted messages in each round, the overall communication is $2\kappa N(|\Sigma|+|Q|)$.

In the modified scheme, only OT is used, and thus $\mathcal{C}$'s work drops by a factor of $|\Sigma|+2$. $\mathcal{S}$'s work is also lowered, as the dominating term in the original solution is $|Q||\Sigma|N\log(|Q|)/\kappa$, while in the modified scheme it is $|Q||\Sigma|N/\kappa$. This means that the server's work drops by a factor of $\log|Q|$ (which is an improvement by at least an order of magnitude). Even though the communication complexity is now proportional to $N|\Sigma||Q|$ instead of $N(|\Sigma|+|Q|)$ in the original protocol, it can be two orders of magnitude lower due to the overhead caused by the security parameter κ in the original solution (that is, for any feasible finite automaton size, $\log|Q| \ll \kappa$; a typical setup can consist of $\log(|Q|) \leq 20$, $|\Sigma|=4$, and κ = 1024).

One of our original motivations for conducting this analysis was large communication overheads of the scheme. For instance, genome sequences can be billions of characters long, but even with the current ability to sample them, the sequences are in the thousands. The FSM that represents a search pattern can have significantly more states than the length of the pattern itself due to the need to handle errors. Thus, for a sample setup of $N=10,000$, $|Q|=50,000$, and $\kappa=1024$, the communication cost of the original solution is 10^{12} bits ≈ 0.125 TB (this is lowered to $\approx 3\cdot 10^{10}$ bits in the modified solution). This amount of communication is prohibitively large for many clients (e.g., it can take several days or even months on a reasonably fast DSL link). Thus, we investigate the use of other OT protocols, which can lower the communication requirements of the protocol. Then depending on the computation resources and the bandwidth one has, the most suitable choice can be used.

Besides existing OT protocols, the OT functionality can be achieved by utilizing an efficient Private Information Retrieval (PIR) protocol, in which the receiver may learn additional information about the database besides the item or block it retrieves, and the sender learns nothing. Transferring a PIR protocol to a Symmetric PIR (SPIR) protocol (in which privacy of the database is also preserved, and the receiver learns information only about a single item) can be done at low computation and communication cost using the techniques from [24] or [11], which will give us an OT protocol. We chose to compare the performance of OFAE using three recent and efficient PIR protocols, which make use of very different techniques. In particular, several PIR protocols (such as [9, 20, 26, 21, 15]) were studied in [23], and we select most communication

	Lipmaa OT	GR OT	AG OT
$\mathcal{C}$'s op.	$K_1 N \log(\|Q\|)(\log(\|Q\|)/2-1)$	$(4NK_e\sqrt{\|Q\|})$	$N(K_3^{\log 10}+2K_3^{\log 5}+\|Q\|K_e)$
$\mathcal{S}$'s op.	$(2\|Q\|-\log(\|Q\|))K_1 N$	$2\|Q\|K_e N$	NK_3^2
Comm	$N((K_1/2)\log^2(\|Q\|)+$ $+3K_e\log(\|Q\|))$	$N(\log(\|Q\|)+K_e+4$ $+\log(\log(\|Q\|)))$	$N\|Q\|K_eK_3^2$

Table 2. Performance of the original OFAE protocol (except matrix multiplication) using different OT protocols.

	Lipmaa OT	GR OT	AG OT
$\mathcal{C}$'s op.	$K_1 N \log(\|Q\|\|\Sigma\|)\times$ $\times(\log(\|Q\|\|\Sigma\|)/2-1)$	$(4N\log(\|Q\|)\sqrt{\|Q\|\|\Sigma\|})$	$N(K_3^{\log 10}+2K_3^{\log 5}$ $+\|Q\|\|\Sigma\|\log(\|Q\|))$
$\mathcal{S}$'s op.	$(2\|Q\|\|\Sigma\|-\log(\|Q\|\|\Sigma\|))K_1 N$	$2\|Q\|\|\Sigma\|\log(\|Q\|)N$	NK_3^2
Comm	$N((K_1/2)\log^2(\|Q\|\|\Sigma\|)+$ $+3\log(\|Q\|)\log(\|Q\|\|\Sigma\|))$	$N(\log(\|Q\|\|\Sigma\|)+\log(\|Q\|)+$ $+4+\log(\log(\|Q\|\|\Sigma\|)))$	$N\|Q\|\|\Sigma\|\log(\|Q\|)K_3^2$

Table 3. Performance of the simplified OFAE protocol using different OT protocols.

efficient solutions of Lipmaa [21] and Gentry-Ramzan (GR) [15], as well as a more recent lattice-based protocol of Aguilar Melchor-Gaborit (AG) [2] which has very light computation overhead. We replace the original OT protocol ([25]) by an OT protocol based on one of those three PIR protocols in both OFAE protocols of Sections 3 and 5.

Before presenting our analysis, we need to point out the differences between these protocols because they are based on different setups, which will require the use of different security parameters and underlying operations. More precisely, the Lipmaa's protocol is based on the use of a length-flexible additively homomorphic encryption scheme (such as [12]), the GR protocol uses groups with special properties (in which Φ-hiding assumption holds), and the AG protocol is a lattice-based PIR scheme. Thus, to achieve as precise analysis as possible, we measure the computation overhead in the number of group operations, and describe what a group operation involves in each solution.

The complexity analysis of the original OFAE approach (except the matrix multiplication over an encrypted vector and Δ in step 3 of the kth state transition protocol in Section 3) is given in Table 2, where work is measured in group operations. The matrix multiplication overhead (the same regardless of the OT protocol used) is given below:

	Matrix Multiplication
$\mathcal{C}$'s group op.	$(\|\Sigma\|+2)K_eN$
$\mathcal{S}$'s group op.	$N\|Q\|K_e(\log(\|Q\|)\|\Sigma\|+\|\Sigma\|-1)$
Comm	$NK_e(\|Q\|+\|\Sigma\|)$

Similarly, Table 3 presents analysis of our modified scheme. In the tables, K_1, K_2, and K_3 are security parameters for each scheme and K_e is the security parameter for the homomorphic encryption scheme (i.e., $K_e = \kappa$). In Lipmaa's solution, K_1 is the same as K_e, and thus $K_1 = \kappa$, which can be set to 1024. (The table reports performance of the original Lipmaa's protocol; however, according to [6], the sender's computation cost can be reduced by almost 38% through optimization.) In GR approach, K_2 is a security parameter of a similar length, but it also depends on the configuration of the OT

protocol for which it is used. In particular, $K_2 = \max(\kappa, \ell, f(\log(t)))$ for OT_1^t, where ℓ is the size of an element in the OT protocol and $f(\cdot)$ is a polynomial function. K_2 is not used in the tables, but it determines the cost of the group operation (multiplication modulo K_2-bit numbers). Note that in the original solution, the OT protocol is called on blocks of size 2κ, and to reduce the computation overhead of operating over very large numbers, each block can be partitioned into several blocks of smaller size (so that the OT protocol will need to be executed more than once).

In the AG solution, the value of the security parameter K_3 is suggested to be set to 50, but the group operations are performed using elements in $\mathbb{Z}_p$ for prime p of size $3(\lceil \log(tK_3) \rceil + 1)$ on the database of size t. Note that the value of t in OT_1^t is different in the original and modified solutions ($|Q|$ and $|Q||\Sigma|$, respectively), which will affect the overhead associated with group operations when they depend on t.

From these options, the AG solution has the highest communication cost (which can be further increased to lower the computation), but it is very computation efficient unlike other protocols (also see [22] for further discussion). Thus, it is ideally suited for parties with very fast communication links. The GR approach, on the other hand, has the lowest communication cost, although the amount of computation carried on the server side as well as the client side are more pronounced. Thus, the first two methods based on Lipmaa's and GR PIR schemes should be used when the bandwidth is an issue of consideration, while the third approach gives the fastest performance with respect to the execution time assuming a fast data link between the participants.

6 Secure Outsourcing of FSM Computation

6.1 Secure two-party outsourcing

The idea behind this solution is that the client $\mathcal{C}$ additively splits (modulo $|\Sigma|$) each character of its x between computational servers P_0 and P_1. Likewise, $\mathcal{S}$ splits (modulo $|Q|$) each element of its matrix Δ between P_0 and P_1. We refer to the P_i's share (for $i = 0, 1$) of the string x as $x^{(i)}$ and its share of the kth character of x as $x_k^{(i)}$. Similarly, we refer to the P_i's share of Δ as $\Delta^{(i)}$ and its share of the element of Δ at position (j_1, j_2) as $\Delta^{(i)}(j_1, j_2)$. The computational servers are also given q_0, i.e., they know what row in the matrix is the starting state (which gives no information about the automaton itself). Finally, P_0 and P_1 also receive information about final states F in a split form. We represent F as a bit vector of length $|Q|$ that has jth bit set to 1 iff state $j \in F$. This vector is additively split modulo 2 (i.e., XOR-split) between P_0 and P_1.

During the kth state transition, P_0 acts as $\mathcal{S}$ in the previous solution and P_1 as $\mathcal{C}$, except that the share of the matrix P_0 possesses is rotated by both P_0's share of the next input character $x_k^{(0)}$ and its share of the current state $q_k^{(0)}$. At the end of this execution, P_0 and P_1 additively share some value q'. The same steps are also performed with the roles of P_0 and P_1 reversed (using P_1's share of the transition matrix), which results in P_0 and P_1 additively sharing another value q''. Finally, P_0 and P_1 each locally add their shares of q' and q'', which results in state q_{k+1} being split (modulo $|Q|$) between them.

Protocol for 1st state transition.

1. For $i = 0, 1$, P_i chooses value $r_i \stackrel{R}{\leftarrow} \mathbb{Z}_{|Q|}$, blinds each element of row q_0 by adding r_i to it modulo $|Q|$ and rotates the row $x_0^{(i)}$ elements left.
2. For $i = 0, 1$, P_i engages in $\text{OT}_1^{|\Sigma|}$ with P_{1-i}, where the sender P_i holds the modified row q_0, and receiver P_{1-i} obtains the element at position $x_0^{(1-i)}$, denoted s_i.
3. For $i = 0, 1$, P_i sets its share of state q_1 to $q_1^{(i)} = s_{1-i} - r_i \bmod |Q|$.

Protocol for kth state transition. Prior to the protocol, P_0 and P_1 additively share the kth state q_k (modulo $|Q|$), the kth input character x_k (modulo $|\Sigma|$), and each element $\Delta(i, j)$ of the transition matrix for $0 \leq i < |Q|$ and $0 \leq j < |\Sigma|$ (modulo $|Q|$). The output consists of P_0 and P_1 additively sharing the $(k + 1)$st state q_{k+1} modulo $|Q|$.

1. For $i = 0, 1$, P_i chooses $r_i \stackrel{R}{\leftarrow} \mathbb{Z}_{|Q|}$ and adds its to each $\Delta^{(i)}(j_1, j_2)$ modulo $|Q|$.
2. For $i = 0, 1$, P_i rotates the resulting matrix $\Delta^{(i)}$ $q_k^{(i)}$ rows up and $x_k^{(i)}$ elements left, and represents it as a list of $|Q| \cdot |\Sigma|$ elements, which we denote by $\Delta_k^{(i)}$.
3. For $i = 0, 1$, P_i engages with P_{1-i} in $\text{OT}_1^{|Q|\cdot|\Sigma|}$ (where P_i acts as the sender), at the end of which P_{1-i} obtains the element at position $|\Sigma| \cdot q_k^{(1-i)} + x_k^{(1-i)}$ from the database $\Delta_k^{(i)}$ prepared by P_i. Denote the element that P_{1-i} retrieves by s_i.
4. For $i = 0, 1$, P_i sets its share of state q_{k+1} to $q_{k+1}^{(i)} = s_{1-i} - r_i \bmod |Q|$.

In the above $q' = s_0 - r_0 \bmod |Q|$ and $q'' = s_1 - r_1 \bmod |Q|$, and also $q_{k+1}^{(0)} = s_1 - r_0 \bmod |Q|$ and $q_{k+1}^{(1)} = s_0 - r_1 \bmod |Q|$.

Protocol for announcement of result. In the beginning, P_0 and P_1 share XOR-split bit vector F, and at the end $\mathcal{C}$ learns the bit of F at position q_N.

1. For $i = 0, 1$, P_i generates a random bit b_i and blinds its vector $F^{(i)}$ by XORing it with b_i. P_i then rotates its $q_N^{(i)}$ bits left.
2. For $i = 0, 1$, P_i engages in $OT_1^{|Q|}$ with P_{1-i}, where P_i uses it modified vector $F^{(i)}$ as the sender and P_{i-1} retrieves the bit c_i at position $q_N^{(1-i)}$.
3. For $i = 0, 1$, P_i sets it share of the result to $f^{(i)} = b_i \oplus c_{1-i}$.
4. P_0 and P_1 send their bits $f^{(0)}$ and $f^{(1)}$ to $\mathcal{C}$, who XORs them and learns the result.

6.2 Secure multi-party outsourcing

To generalize the above solution to multiple parties $P_0, \ldots, P_{m-1}$, we first need to have $\mathcal{C}$ and $\mathcal{S}$ split their data among all of them. For a split item a, we use $a^{(i)}$ to denote the share party P_i has. Since now both the input characters and the current state will be split among m participants, any solution that involves data rotation by a share of the state or input character becomes more expensive. In particular, at least $m - 1$ parties need to rotate the data in a predetermined order using their own shares. This means that the data to be rotated must be obfuscated from others (i.e., encrypted) when it leaves the owner and it also means that each party needs to re-randomize the data to hide the amount of rotation. With this (or any other secure) approach, the work performed by one party in a single execution of the state transition protocol is inevitably $O(|Q||\Sigma|)$ (and is also a function of a security parameter), and we wish to minimize the amount of work other parties need to perform, as well as their communication complexity. Therefore, we reduce the overhead of most parties to $O(\sqrt{|Q||\Sigma|})$ by representing the transition

matrix Δ as a two-dimensional array of size $\sqrt{|Q||\Sigma|} \times \sqrt{|Q||\Sigma|}$. The interaction is then similar at the high-level to the interaction in the original protocol and proceeds as follows: one party generates a vector of encrypted bits of size $\sqrt{|Q||\Sigma|}$, $m-2$ parties sequentially rotate and randomize it, and the last party performs matrix multiplication to create a new vector of the same size. This vector is also passed to $m-2$ parties for rotation and re-randomization, after which the last party obtains the decryption of one element of it. This process is repeated for each share of the transition matrix $\Delta^{(i)}$.

Our solution requires the parties to convert shares $v^{(i)}$ of value v additively split modulo n to additive shares of it modulo $|Q|$. To do this, the parties will need to compute the quotient $u = \lfloor \sum_{i=1}^{m} v^{(i)}/n \rfloor$ and use it to adjust the shares. To prevent the parties from learning u, we additively split it among the participants over integers. Since $0 \leq u < m$, we define $B > m2^{\kappa'}$, where κ' is a security parameter. Then if we hide u using shares r and $u - r$, where $r \xleftarrow{R} [-B, B]$, the value of u will be statistically hidden.

Finally, the parties now use a threshold homomorphic encryption scheme, in which the public key pk is known to everyone, but the decryption key sk is split among the parties. In this solution, we require all m parties to participate in decryption (i.e., use (m, m)-threshold encryption), and the threshold multi-party solution given in [7] will have the threshold set to t (i.e., (t, m)-threshold encryption).

Before presenting the main protocols, we describe a sub-protocol, RotateAndShare, that will be utilized in all of them, but will be called on different types of data. This sub-protocol assumes that one party, P_i, has a vector, which will be encrypted, and then rotated by a certain amount, re-randomized, and blinded by every party. P_i will be the data owner and plays a special role in the protocol. The amount of rotation is determined by some value additively split among all parties (e.g., the current state q_k). Blinding involves adding a random value r_i to the encrypted contents by each party. Then when the last party chooses an element of the vector, other parties jointly decrypt that value for it. At this point, all parties jointly hold additive shares of the result modulo n. As the last (and optional) step, they engage in the computation to convert the additive shares modulo n to additive shares modulo a different modulus n'.

RotateAndShare: The input consists of value $i, 0 \leq i \leq m-1$, encryption $\mathcal{E}$ with public key pk, modulus n, and distributed secret key sk, final modulus n' (if no conversion is necessary, n' is set to $\perp$), party P_i inputs vector $v = (v_0, \ldots, v_{\ell-1})$ and its length ℓ, and each party $P_j, 0 \leq j \leq m-1$ inputs amount of rotation $rt^{(j)}$. The output consists of the parties additively sharing value o modulo n' (or modulo n if $n' = \perp$), which corresponds to one of the values from vector v.

1. P_i chooses $r_i \xleftarrow{R} \mathbb{Z}_n$, adds it modulo n to each v_j, and encrypts each result with pk to obtain $e = (e_0, \ldots, e_{\ell-1})$, where $e_j = \mathsf{Enc}_{pk}(v_j + r_i)$ for $j = 0, \ldots, \ell - 1$. P_i circularly rotates the elements of e $rt^{(i)}$ positions left and sends the result to P_{i+1}.
2. P_{i+1} circularly rotates the vector it received $rt^{(i+1)}$ positions left. It also chooses $r_{i+1} \xleftarrow{R} \mathbb{Z}_n$ and multiplies each element of its resulting vector by different encryptions $\mathsf{Enc}_{pk}(r_{i+1})$ (or by the same encryption, but then re-randomizes each element). This adds r_{i+1} to the encrypted values. P_{i+1} sends the result to P_{i+2}.
3. Each of $P_{i+2}, \ldots, P_{m-1}, P_0, \ldots, P_{i-2}$ sequentially perform the same steps at P_{i+1} using their respective values of randomness r and rotation amount rt.

4. Parties P_{i-2} and P_{i-1} engage in OT_1^ℓ, where P_{i-2} plays the role of the sender using the final encrypted vector and P_{i-1} plays the role of the receiver using index $rt^{(i-1)}$. This results in P_{i-1} obtaining an encryption of the value at position $\left(\sum_{j=0}^{m-1} rt^{(j)}\right) \bmod \ell$ in v blinded with $\left(\sum_{j\in[0,m-1],j\neq i-1} r_j\right) \bmod n$. P_{i-1} re-randomizes the item it received (by multiplying it with $\mathsf{Enc}_{pk}(0)$), asks the rest of participants to decrypt it, and sets r_{i-1} to the decrypted value.
5. Now, if the value of n' was not $\perp$, the parties re-share the result modulo n'. To do so, they need to compute the number of times the sum of the shares "wraps around" the modulus n and use it in their computation. The parties engage in secure multi-party computation, e.g., using a standard general multi-party Boolean circuit [16]. Here each party inputs its share, they jointly compute $u = \lfloor(\sum_{j=0}^{m-1} r_j)/n\rfloor$ (e.g., by repeated subtraction of n from the sum) and the output is additively shared over the integers. That is, party P_j for $j = 0, \ldots, m-2$ receives a random $s_j \in [-B, B]$ and party P_{m-1} receives $s_{m-1} = u - \sum_{j=0}^{m-2} s_j$.
6. Party P_j, for $j = 0, \ldots, m-1$, sets its output $o^{(j)}$ to $(s_j \cdot n - r_j) \bmod n'$.

We are now ready to present the main protocols of the multi-party outsourcing solution.

Protocol for 1st state transition.

1. For $i = 0, \ldots, m-1$, execute in parallel: party P_i sets v to be the q_0th row of its matrix $\Delta^{(i)}$ and all parties execute $\mathsf{RotateAndShare}(i, \mathcal{E}, pk, sk, |Q|, v, |\Sigma|, x_0^{(0)}, \ldots, x_0^{(m-1)})$. Let $o_i^{(j)}$ denote the output P_j receives after such execution on P_i's data.
2. For $i = 0, \ldots, m-1$, party P_i sets its share of q_1 to $q_1^{(i)} = \sum_{j=0}^{m-1} o_j^{(i)} \bmod |Q|$.

Protocol for kth state transition. Prior to the protocol execution, the parties additively share the kth state q_k (modulo $|Q|$), the kth input character x_k (modulo $|\Sigma|$), and each element $\Delta(i, j)$ of the transition matrix for $0 \leq i < |Q|$ and $0 \leq j < |\Sigma|$ (modulo $|Q|$). At the end, they additively share state q_{k+1} (modulo $|Q|$).

For $i = 0, \ldots, m-1$, perform the following steps in parallel using the share $\Delta^{(i)}$ of the transition matrix.

1. P_i rotates the matrix $\Delta^{(i)}$ $q_k^{(i)}$ rows up and $x_k^{(i)}$ elements left. We denote the resulting matrix by $\Delta_k^{(i)}$. P_i represents $\Delta_k^{(i)}$ as a two-dimensional array of roughly square size as follows[1]: P_i computes the size of the first dimension of the matrix as $d_1 = \lceil\sqrt{|Q||\Sigma|}\rceil$ and the size of the second dimension as $d_2 = \lceil|Q|/d_1\rceil|\Sigma|$. P_i then creates columns 0 through $|\Sigma|-1$ of the modified matrix using rows 0 through $d_1 - 1$ of $\Delta_k^{(i)}$, columns $|\Sigma|$ through $2|\Sigma| - 1$ using rows d_1 through $2d_1 - 1$ of $\Delta_k^{(i)}$, etc. In other words, the modified square matrix, denoted $\tilde{\Delta}_k^{(i)}$, is filled in stripes of width $|\Sigma|$ until all of $|Q|$ rows are used (note that part of the square might be incomplete due to rounding in the computation). Empty cells are then filled with dummy entries to make it a full matrix of size $d_1 \times d_2$.
2. Party P_{i+1} creates a vector of encrypted values $e = (e_0, \ldots, e_{d_1-1})$ using homomorphic encryption, where the value at position $q_k^{(i+1)} \bmod d_1$ corresponds to encryption of 1, and all other e_j's correspond to encryption of 0.

[1] In the current discussion we assume that $|\Sigma| < |Q|$, but the technique can be used when either $|\Sigma| < |Q|$ or $|Q| < |\Sigma|$ (and it is not necessary for the matrix to be close to a square size).

3. Party P_{i+1} sends the vector to P_{i+2}, who performs a circular rotation of it $q_k^{(i+2)}$ values left and re-randomizes the encrypted values. The encrypted vector is sequentially processed by parties $P_{i+2}, \ldots, P_{m-1}$, $P_0, \ldots, P_{i-1}$ who perform the same operations as P_{i+2} using their respective shares of q_k.
4. P_{i-1} sends the final vector $\tilde{e} = (\tilde{e}_0, \ldots, \tilde{e}_{d_1-1})$ to P_i. P_i performs matrix multiplication using $\tilde{e}$ and $\tilde{\Delta}^{(i)}$ as follows: to compute the jth element of the resulting vector v, perform $v_j = \prod_{\ell=0}^{d_1-1} \tilde{e}_\ell^{\tilde{\Delta}_k^{(i)}(\ell,j)}$. Now the vector v corresponds to the element-wise encryption of the row of the matrix $\tilde{\Delta}_k^{(i)}$ at index $q_k \bmod d_1$.
5. All parties execute a modified algorithm RotateAndShare$(i, \mathcal{E}, pk, sk, |Q|, v, d_2, (x_k^{(0)}, \lfloor q_k^{(0)}/d_1 \rfloor |\Sigma|), \ldots, (x_k^{(m-1)}, \lfloor q_k^{(m-1)}/d_1 \rfloor |\Sigma|))$ with the following changes:
   (a) The vector v is already in an encrypted form, so no encryption is performed in step 1 of RotateAndShare.
   (b) Instead of each P_j rotating the vector by amount $rt^{(j)}$, rotation is performed as follows: now $rt^{(j)}$ consists of two parts, $rt_1^{(j)}$ and $rt_2^{(j)}$. Starting from $j = i$, party P_j divides the vector v into blocks of size $|\Sigma|$ and circularly rotates each block $rt_1^{(j)}$ positions left, and then rotates the overall resulting vector $rt_2^{(j)}$ positions left.
   (c) Using two different values for the amount of rotation also affects the oblivious transfer in step 4 of the protocol. Now party P_{i-1} selects the element at position $rt_1^{(i-1)} + rt_2^{(i-1)}|\Sigma|$.

   Let $o_j^{(i)}$ denote the output party P_j receives as a result of such execution.

After executing these steps on all shares of the database $\Delta^{(i)}$, party P_j sets its share of q_{k+1}, $q_{k+1}^{(j)}$, to the sum of the values it received in step 5 of the protocol executions, i.e., $q_{k+1}^{(j)} = \sum_{i=0}^{m-1} o_i^{(j)} \bmod |Q|$.

Protocol for announcement of result. Prior to the protocol, parties $P_0, \ldots, P_{m-1}$ additively share the state q_N and also share vector F XOR-split between all of them.

1. For $i = 0, \ldots, m-1$, execute in parallel: the parties call RotateAndShare$(i, \mathcal{E}, pk, sk, \perp, F^{(i)}, |Q|, q_N^{(0)}, \ldots, q_N^{(m-1)})$. Let $o_j^{(i)}$ denote the output party P_j receives.
2. For $i = 0, \ldots, m-1$, P_i computes $f^{(i)} = \sum_{j=0}^{m-1} o_j^{(i)} \bmod n$ and sends $f^{(i)}$ to $\mathcal{C}$.

$\mathcal{C}$ recovers the result by computing bit $b = \sum_{i=0}^{m-1} f^{(i)} \bmod n$.

The above protocol calls RotateAndShare without modulus conversion. The reason is that the client can easily recover the result by adding the shares it received modulo n. It, however, would involve less work for the client if, prior to sending the shares to the client, they were converted to additive shares modulo 2 (i.e., XOR-shares). Then the client's work would consists of only $m-1$ bit XORs. Thus, if the client is extremely weak, the above protocol can be modified to include modulus conversion at the cost of the increased work for the computational servers.

Also note that the protocol for announcement of the result can have a similar structure to the kth state transition protocol if the vector F is represented as a matrix of size $\sqrt{|Q|} \times \sqrt{|Q|}$. Then the computation and communication complexity of all parties will be reduced by a significant amount. But since this protocol is executed only once (as opposed to the kth state transition protocol), we leave it in the simple form above.

Remark. The above technique allow us to have communication associated with processing a square two-dimensional grid to be linear in the size of its one dimension. One might ask if it might be possible to further reduce the communication by represented the matrix as a high-dimensional hypercube and still have communication to be proportional to its single dimension. Such technique was employed in private information retrieval systems to dramatically decrease communication cost to $O(\ell^{\epsilon})$ for any $\epsilon > 0$ [20] or $O(\log^2(\ell))$ [21] with stronger privacy guarantees for a database of size ℓ. Here we note that such a solution would not work in our setting because decreasing the dimension of the matrix (represented as a hypercube of any dimension) by one requires interaction of all of the participants, and thus would involve communication close to linear in the matrix size in our case (this technique worked for PIR systems when the entire database is stored at a single location).

7 Analysis

We now evaluate correctness and security requirements and give complexity analysis.

Correctness. Correctness of the protocols follows by examination. That is, during each round of the protocol, the parties additively share the value of the next state that can be found in matrix $\Delta^{(i)}$ for each participant P_i and add them all together to correctly share the next state. The same applies to the protocol for announcement of the result.

Security. The argument for achieving security in presence of semi-honest parties that we use in this solution is very standard, and is based on the following components:

- The composition theorem due to Canetti [10] states that composition of secure protocols remains secure. This means that the security of the overall solution reduces to ensuring that sub-protocols or other tools used as a part of it are secure against semi-honest adversaries.
- Semantic security of homomorphic encryption ensures that no information about the underlying plaintext can be learned by observing its encryption. Threshold encryption ensures that participation of a predefined number of parties (including all parties) is necessary for decryption.
- Additive secret sharing ensures unconditional security as long as there is at least one honest party that does not collude with the rest of the participants.

Given the above, it is straightforward to build a simulator that will simulate the view of the computational parties without access to $\mathcal{C}$'s or $\mathcal{S}$'s data. That is, every time encryption is used, it can produce encryptions of random values that will be indistinguishable from real data due to the security property of encryption, and every time shares are used, it will also produce random shares that will be indistinguishable from the shares used in the real execution. Since only secure and composable components are used in the protocols, the overall solution is secure as well.

Complexity. We analyze computation and communication complexity of two-party and multi-party outsourcing protocols separately. The analysis corresponds to the N executions of the kth state transition protocol (as the rest of the overhead will be orders of magnitude lower).

Two-party outsourcing: The client $\mathcal{C}$ only splits its input between two servers, therefore the computation is near N (no cryptography is used) and communication is $2N \log(|\Sigma|)$.

The service provider $\mathcal{S}$ splits the representation of its automaton M among two servers, with the computation being near $|M|$ and communication approximately twice the size of representing M (i.e., near $|Q||\Sigma|\log(|Q|)$). Each computational server incurs computation and communication overhead of both $\mathcal{C}$ and $\mathcal{S}$ in the solution with no outsourcing (as given in Table 1). That is, each server performs about $|Q||\Sigma|+N(2+|\Sigma||Q|/\kappa)$ modulo exponentiations and communicates about $2\log(|Q|)N|\Sigma||Q|$ bits.

Multi-party outsourcing: The work and communication of $\mathcal{C}$ and $\mathcal{S}$ remain similar to the two-party case, except that splitting of their data and communication needs to be done for m servers instead of two. This means that work becomes proportional to m (with no cryptographic operations, as before), which for $\mathcal{C}$ means mN and for $\mathcal{S}$ is $m|M|$, and their communication is $mN\log(|\Sigma|)$ and near $m|Q||\Sigma|\log(|Q|)$, respectively. The computation and communication requirements for the computational servers also now increase by a factor of m and are as follows. The main computation overhead comes from (i) $2\sqrt{|Q||\Sigma|}(m-1)$ modular exponentiations in each round due to re-randomization; (ii) $|Q||\Sigma|\log(|Q|)$ modular multiplications in each round for matrix multiplication; (iii) κ OT_1^2 executions for the Boolean circuit and one $\text{OT}_1^{\sqrt{|Q||\Sigma|}}$ in each round. We assume that the OT protocol with low amortized cost (one mod exp per transfer) is used. The communication complexity is dominated by the transmission of encrypted vectors and the OT protocol and is near $4\kappa(m-1)N\sqrt{|Q||\Sigma|}$.

8 Conclusions

This work studies the problem of outsourcing oblivious evaluation of a finite state machine to computational servers. We present solutions for outsourcing the computation to two (two-party) or more (multi-party) computational servers that rely on different techniques. The two-party solution has the same complexity as secure computation by the owners of the data themselves without outsourcing. An interesting research direction that remains is to explore the applicability of alternative techniques with the goal of reducing the overhead of the protocol. In particular, it would be desirable to eliminate executing multiple instances of the same protocol on different shares of the automaton and use a single copy of M instead.

Acknowledgments. Portions of this work were sponsored by grant AFOSR-FA9550-09-1-0223. The first author would like to thank Scott Emrich for useful discussions regarding DNA processing technology.

References

1. Genetic Testing for Health, Disease & Ancestry; DNA Test – 23andMe. `http://www.23andme.com`.
2. C. Aguilar-Melchor and P. Gaborit. A lattice-based computationally-efficient private information retrieval protocol. In *WEWORC*, 2007.
3. M. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *WPES*, pages 39–44, 2003.
4. M. Atallah and J. Li. Secure outsourcing of sequence comparisons. In *PET*, pages 63–78, 2004.

5. M. Atallah and J. Li. Secure outsourcing of sequence comparisons. *International Journal of Information Security*, 4(4):277–287, 2005.
6. H. Bae. Design and analysis for log-squared and log private information retrieval, 2008.
7. M. Blanton and M. Aliasgari. Secure outsourcing of dna searching via finite automata. Technical Report 2010–03, Department of CSE, University of Notre Dame, 2010.
8. F. Bruekers, S. Katzenbeisser, K. Kursawe, and P. Tuyls. Privacy-preserving matching of DNA profiles. ePrint Cryptology Archive Report 2008/203, 2008.
9. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with poly-logarithmic communication. In *EUROCRYPT*, pages 402–414, 1999.
10. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
11. G. Crescenzo, T. Malkin, and R. Ostrovsky. Single database private information retrieval implies oblivious transfer. In *Advances in Cryptology – EUROCRYPT*, pages 122–138, 2000.
12. I. Damgard and M. Jurik. A length-flexible threshold cryptosystem with applications. In *Australasian Conference on Information Security and Privacy*, 2007.
13. M. Franklin, M. Gondree, and P.Mohassel. Communication-efficient private protocols for longest common subsequence. In *RSA*, pages 265–278, 2009.
14. K. Frikken. Practical private dna string searching and matching through efficient oblivious automata evaluation. In *DBSec*, pages 81–94, 2009.
15. C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. In *ICALP*, pages 803–815, 2005.
16. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, pages 218–229, 1987.
17. M. Gondree and P. Mohassel. Longest common subsequence as private search. In *WPES*, pages 81–90, 2009.
18. S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. In *IEEE Symposium on Security and Privacy*, pages 216–230, 2008.
19. M. Kantarcioglu, W. Jiang, Y. Liu, and B. Malin. A cryptographic approach to securely share and query genomic sequences. *IEEE Transactions on Information Technology in Biomedicine*, 12(5):606–617, 2008.
20. E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *IEEE FOCS*, pages 364–373, 1997.
21. H. Lipmaa. An oblivious transfer protocol with log-squared communication. In *Information Security*, volume 3650 of *LNCS*, pages 314–328, 2005.
22. C. Melchor, B. Crespin, P. Gaborit, and V. Jolivet. High-speed private information retrieval computation on GPU. In *IEEE SECURWARE*, 2008.
23. C. Aguilar Melchor and Y. Deswarte. Single-database private information retrieval schemes: Overview, performance study, and usage with statistical databases. In *Privacy in Statistical Databases*, pages 257–265, 2006.
24. M. Naor and B. Pinkas. Oblivious transfer and polynomial evaluation. In *STOC*, 1999.
25. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA*, pages 448–457, 2001.
26. J. Stern. A new efficient all-or-nothing disclosure of secrets protocol. In *ASIACRYPT*, pages 357–371, 1998.
27. D. Szajda, M. Pohl, J. Owen, and B. Lawson. Toward a practical data privacy scheme for a distributed implementation of the Smith-Waterman genome sequence comparison algorithm. In *NDSS*, 2006.
28. J. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient DNA searching through oblivious automata. In *ACM CCS*, pages 519–528, 2007.
29. R. Wang, X. Wang, Z. Li, H. Tang, M. Reiter, and Z. Dong. Privacy-preserving genomic computation through program specialization. In *ACM CCS*, pages 338–347, 2009.