Privacy-Preserving Deterministic Automata Evaluation with Encrypted Data Blocks

Giovanni Di Crescenzo(✉), Brian Coan, and Jonathan Kirsch

Vencore Labs, Basking Ridge, NJ 07920, USA
{gdicrescenzo,bcoan,jkirsch}@vencorelabs.com

Abstract. Secure computation (i.e., performing computation while keeping inputs private) is a fundamental problem in cryptography. In this paper, we present an efficient and secure 2-party computation protocol for deterministic automata evaluation, a problem of large practical relevance. Our result is secure under standard assumptions and bypasses roadblocks in previous general solutions, like Yao's garbled circuits and Gentry's lattice-based fully homomorphic encryption, by performing secure computations over data blocks (instead of bits) and using typical-size (instead of impractically large) cryptographic keys. An important efficiency property achieved is that the number of both asymmetric and symmetric cryptographic operations in the protocol is *sublinear* in the size of the circuit representing the computed function (specifically, improving linear-complexity protocols by a multiplicative factor equal to a block size). All previous protocols for deterministic automata evaluation required a linear number of asymmetric cryptographic operations. Moreover, we use quantitative comparison techniques to show that in typical parameter settings, our protocols' latency is at least 1 to 2 orders of magnitude smaller than the protocol obtained by a direct application of both state-of-the-art general-purpose secure 2-party computation protocols. Even though not as general as in these two general-purpose techniques, our result is applicable to the class of all constant-space computations.

1 Introduction

Managing data privacy for real-life systems is a complex endeavor with many different areas in need of investigation. Cryptography research has traditionally produced cornerstone technical solutions to a large variety of data privacy problems. In some domains, like communication security, a wide variety of cryptography solutions with various dimensions of desirable properties have been produced, and a typical system designer has several valid options to choose from at development stage. Unfortunately this is not the case for other areas, many of which related to data privacy. This paper focuses on one of these areas, 2-party secure computation [22], where several solutions have been proposed, but still different types of gaps remain towards regular deployment of this technology in real-life systems. Existing solution paradigms, like garbled circuits [22] and

J. Garcia-Alfaro et al. (Eds.): DPM/CBT 2017, LNCS 10436, pp. 275–294, 2017.
DOI: 10.1007/978-3-319-67816-0_16

fully homomorphic encryption [10], address a large spectrum of assumptions and satisfy many desirable properties, but do not exhaustively cover needs that may arise from real-life systems.

In this paper, we propose new cryptography solutions for 2-party secure computation based on a recent paradigm of privacy-preserving computations over encrypted data blocks [5]. We show that solutions can be exhibited for the important problem of deterministic automata evaluation, going beyond a previous result of [5] that only applied to monotone formulae over equality statements.

Automata evaluation. Deterministic automata evaluation is a well-known problem in computer science, also equivalent to regular expression matching, with several applications (most notably, pattern matching). We consider the design of secure 2-party protocols for deterministic automata evaluation, where Alice holds the (pattern) automata, Bob holds the (text) string, and one of the two parties obtains the match result, while the two parties learn no other information on the other party's input. A practical and secure 2-party protocol for deterministic automata evaluation is expected to have several interesting applications, including DNA identity testing, firewall policy checking on web traffic, keyword search on emails, etc.

Secure computation: state of the art. Secure two-party computation is a fundamental cryptographic primitive with significant application potential. In the formulation of interest for this paper, there are two parties, Alice and Bob, who would like to interactively compute a function f on their inputs x and y, respectively, such that at the end of the protocol: Bob obtains $f(x, y)$; an efficient adversary corrupting Alice learns nothing new about Bob's input y; and an efficient adversary corrupting Bob learns nothing new about Alice's input x, in addition to what is efficiently computable from $f(x, y)$. The first general solution to this problem for any arbitrary function f was presented by Yao in [22], assuming that the adversary is semi-honest (i.e., he follows the protocol as the corrupted party but may at the end try any polynomial-time algorithm to learn about the other party's input). The generality of this solution is so attractive that, even decades after their introduction, researchers are considering improvements and optimizations (see, e.g., [14,17]), thus bringing them closer to being usable in practice, at least in some specific scenarios (i.e., with the help of additional servers [1]). An important roadblock in this process is represented by the fact that Yao's protocol, using a boolean circuit representation of the function f, requires cryptographic operations for all input bits and binary gates in the circuit.

Recently, another general and powerful cryptographic primitive, fully homomorphic encryption, has been realized [10]. This primitive allows arbitrary polynomial-time computations over encrypted data and thus can be applied to construct secure 2-party computation protocols for any arbitrary polynomial-size arithmetic circuit (and therefore any polynomial-size boolean circuit). Even in this case, researchers are recently considering improvements and optimizations, trying to bring it closer to being usable in practice (see, e.g., [3]). The roadblock for garbled circuits does not apply here, when using arithmetic circuits, since in that case fully homomorphic encryption solutions typically do operate over data blocks (instead of bits). However, another roadblock on the way to efficiency appears here: the security of all known constructions of fully homomorphic encryption is based on problems whose required key lengths are significantly high and the overall scheme is only theoretically efficient, but not in practice.

Computations over encrypted data blocks, as introduced in [5], attempt to combine the best features from both cited general-purpose approaches: computing over encrypted data blocks (as in fully homomorphic encryption over arithmetic circuits), limited requirements on key lengths (as in garbled circuits), and achieving solutions for a large class of problems (as in both). The solution proposed in [5], shows secure protocols over encrypted data blocks for the class of monotone formulae over string equality statements.

Our contribution. Our main result in this paper is an efficient and secure 2-party protocol, based on computations over encrypted data blocks, for deterministic automata evaluation, thus being applicable to all constant-space computations. The security of our protocol holds under standard cryptographic assumptions and is proved based on the existence of secure 2-party protocols for simpler tasks: (a) pseudo-random function evaluation, which, in turn, were previously proved secure based on standard number-theoretic assumptions with conventional key lengths (see, e.g., [8,16]); and (b) conditional transfer [6] for string equality and AND of string equality statements, which, in turn, can be based on symmetric encryption alone, given the information shared between the two parties during the protocol for pseudo-random function evaluation. We give two instantiations of the secure 2-party protocols for these two simpler tasks, resulting in two instantiations of our main protocol with different desirable efficiency properties.

The main efficiency property is the protocol's time complexity, as we show, in our main protocol's first instantiation, that it only requires a number of cryptographic operations *sub-linear* in the size of the circuit computing the function. Specifically, it improves over the natural application of the garbled circuit technique from [22] by a factor equal to the length of alphabet symbols. In practice, depending on the alphabet required by the specific application, this can be anywhere between a small and a very large improvement. In our main protocol's second instantiation, we also show a variant that improves multiplicative constants for small alphabets, by using an alternative implementation of the secure 2-party protocol for pseudo-random function evaluation. We show a performance analysis of both variants, and comparisons with previously known protocols in the literature [9,15,18], all requiring at least a linear number of asymmetric cryptographic operations. Moreover, we use quantitative comparison techniques to compare the latency of both our protocols with the protocol obtained by a direct application of both state-of-the-art general-purpose secure 2-party computation protocols. We obtain that our protocols' latency is at least 1 to 2 orders of magnitude smaller in typical parameter settings.

Organization of the paper. In Sect. 2 we detail definitions and models of interest, including a formal definition for secure function evaluation protocols, and for tools used in our constructions, such as symmetric encryption schemes, pseudo-random functions, oblivious PRF evaluation protocols, and conditional OT protocols.

In Sect. 3 we present our main result: a practical and secure protocol for 2-party evaluation of a deterministic automata, based on building blocks such as a PRF, an oblivious PRF evaluation protocol, and a conditional OT protocol for string equality and AND of string equality conditions.

In Sect. 4 we describe a first instantiation of our main result that is particularly efficient for large automata alphabets, based on an adaptation of an oblivious PRF evaluation protocol from [16], and a simple variant of conditional OT protocols in [4,6].

In Sect. 5 we describe a second instantiation of our main result that is particularly efficient for small automata alphabets. This differs from the previous instantiation in that the oblivious PRF evaluation protocol is now replaced by a suitable combination of results from [19,20].

In Sect. 6 we discuss the practical performance of the two instantiations of our protocols, showing improved efficiency with respect to previous work, including a protocol that can be constructed by an application of the original Yao's general-purpose protocol [22].

2 Definitions and Background

In this section we give definitions and background useful in the rest of the document. Definitions in Sect. 2.1 are specific to the main problem of interest in the paper, and include deterministic automata, secure 2-party function evaluation protocols, and efficiency requirements. Definitions in Sect. 2.2 are specific to our solutions to the main problem considered, and include pseudo-random functions, and secure 2-party protocols for pseudo-random function evaluation and conditional transfer.

2.1 Secure 2-Party Evaluation of Deterministic Automata

Deterministic automata. A *deterministic automata* is formally defined as a tuple $\mathrm{DA} = (S, s_0, F, A, \tau)$, where S is the set of automata states, s_0 is the initial state, F is a subset of S representing the set of final states, A is an alphabet, and $\tau : S \times A \to S$ is a transition function that maps any state and any alphabet element to the next state (when defined). We also denote as $|S| = s$ the number of states, as $|F| = f$ the number of final states, and as $|A| = a$ the number of alphabet symbols. An input string $x = (x_1, \ldots, x_m)$ is a sequence of alphabet symbols $x_i \in A$, for $i = 1, \ldots, m$.

The *deterministic automata evaluation* (briefly, DAE) problem consists of computing $s_i = \tau(s_{i-1}, x_i)$, for $i = 1, \ldots, m$, and then returning as output $out_{ae} = 1$ if s_m is in F (denoting that a final state is reached) or $out_{ae} = 0$ otherwise.

In the *2-party DAE* problem, the two parties, called Alice and Bob, are given as input the automata objects S, s_0, A and the parameters s, a, m; Alice is given as input F, τ; Bob holds the input string x; and at the end of the 2-party protocol, Bob obtains the output out_{ae}, defined as for the DAE problem.

Secure 2-party function evaluation protocols. The expression $z \leftarrow D$ denotes the probabilistic process of randomly and independently choosing x according to distribution D. By $\text{Prob}[z \leftarrow D : E]$ we denote the probability of event E after the execution of the probabilistic process $z \leftarrow D$. Let σ denote a security parameter. A function over the set of natural numbers is *negligible* if for all sufficiently large natural numbers $\sigma \in \mathcal{N}$, it is smaller than $1/p(\sigma)$, for all polynomials p. Two distribution ensembles $\{D_\sigma^0 : \sigma \in \mathcal{N}\}$ and $\{D_\sigma^1 : \sigma \in \mathcal{N}\}$ are *computationally indistinguishable* if for any efficient algorithm A, the quantity $|\text{Prob}[x \leftarrow D_\sigma^0 : A(x) = 1] - \text{Prob}[x \leftarrow D_\sigma^1 : A(x) = 1]|$ is negligible in σ (i.e., no efficient algorithm can distinguish if a random sample came from one distribution or the other). In a 2-party protocol execution, a party's *view* is the sequence containing the party's input, the party's random string, and all messages received during the execution.

We use the simulation-based definition from [11] for security of 2-party function evaluation protocols in the presence of semi-honest adversaries (i.e., adversaries that corrupt one party, follow the protocol as that party and then attempt to obtain some information about the other party's input). According to this definition, a protocol π to evaluate a (possibly probabilistic) function f satisfies *simulation-based security* in the presence of a semi-honest adversary, if there exists two efficient algorithms Sim_A, Sim_B (called the *simulators*), such that:

1. let $out_{S,A}$ be Sim_A's output on input Alice's input and Alice's output (if any); then, it holds that the pair ($out_{S,A}$, Bob's output) is computationally indistinguishable from the pair (Alice's view, Bob's output); and
2. let $out_{S,B}$ be Sim_B's output on input Bob's input and Bob's output (if any); then, it holds that the pair (Alice's output, $out_{S,B}$) is computationally indistinguishable from the pair (Alice's output, Bob's view).

In the above, the first (resp., second) condition says that a semi-honest adversary's view when corrupting Alice (resp., Bob), can be generated by an efficient algorithm not knowing Bob's (resp., Alice's) input, and thus the adversary does not learn anything about the uncorrupted party's input. This definition also implies correctness of the protocol's output: that is, the intended recipient of the 2-party problem formulation's output does receive this output at the end of the protocol.

Efficiency requirements. We will consider the following efficiency metrics, relative to a single execution of a given secure 2-party protocol:

1. *time complexity*: time between the protocol execution's beginning and end;
2. *communication complexity*: length of all messages exchanged; and
3. *round complexity*: number of messages exchanged.

All efficiency metrics are expressed as a function of the security parameter σ, and parameters s, a, m associated with the deterministic automata and input string that are input to the protocol. In evaluating protocol latency, we will pay special attention to the number of asymmetric cryptography operations (e.g., modular exponentiations in a large group) and of symmetric cryptographic operations (e.g., block cipher executions), since the former are typically orders of magnitude more expensive than the latter (although the latter might be applied a larger number of times). As a comparison result, we will target the general solution from [22] for the 2-party secure evaluation of function $f(x, y)$, where x is Alice's input and y is Bob's input, which requires $O(|y|)$ asymmetric cryptography operations and $O(|C_f|)$ symmetric cryptography operations, if C_f denotes the size of the boolean circuit computing f. Even if we will mainly focus our efficiency analysis on time complexity, our design targets minimization of all the mentioned efficiency metrics.

2.2 Cryptographic Primitives and Protocols Used in Our Solutions

Pseudo-random function families. A family of functions $\{r_n : n \in \mathcal{N}\}$ is a *random function family* if, for each value of the security parameter n, the function r_n associated with that value is chosen with distribution uniform across all possible functions of the pre-defined input and output domains. A family of keyed functions $\{F_n(k, \cdot) : n \in \mathcal{N}\}$ is a *pseudo-random function family* (briefly, a PRF family, first defined in [12]) if, after key k is randomly chosen, no efficient algorithm allowed to query an oracle function O_n can distinguish whether O_n is $F_n(k, \cdot)$ or O_n is a random function $R_n(\cdot)$ over the same input and output domain, with probability greater than 1/2 plus a negligible (in n) quantity. We consider *symmetric-type PRFs*, which are implemented in practice using symmetric-key cryptography primitives (e.g., block ciphers like AES), and *asymmetric-type PRFs*, which are based on a public and a secret key, usually implemented using number-theoretic functions, the most expensive often being modular exponentiations.

Secure evaluation protocols for specific functions. In our solutions, we use or build constructions of 2-party secure evaluation protocols for the following functionalities: pseudo-random function, scalar product, and real-or-random conditional transfer.

A *secure pseudo-random function evaluation protocol* (briefly, sPRFeval protocol) is a protocol between two parties: Alice, having as input a key k for a PRF F, and Bob, having as input a string x, where the description of F is known to both parties. The protocol is defined as a secure function evaluation of the value $F(k, x)$, returned to Bob (thus, without revealing any information about x to Alice, or any information about k to Bob in addition to $F(k, x)$). Efficient constructions of sPRFeval protocols, based on the hardness of number-theoretic problems, were given in [8,16].

A *secure conditional transfer protocol* for the condition predicate p (briefly, p-sCTeval protocol, or sCTeval protocol when p is clear from the context) is a protocol between two parties: Alice, having as input a message m and a string x, and Bob, having as input a string y. The protocol is defined as a secure function evaluation of the value m', returned to Bob, where $m' = m$ if $p(x, y) = 1$ or m' is computationally indistinguishable from a string random and independent from m, and of the same length as m, if $p(x, y) = 0$. Thus, an execution of the protocol does not reveal any information about y to Alice, or any information about x to Bob in addition to m', and m' only reveals m when $p(x, y) = 1$ or the (possibly padded) length of m when $p(x, y) = 0$. Also, note that if m is a pseudo-random string, then at the end of a p-sCTeval protocol, Bob does not obtain any information about the value of predicate p. The notion of a p-sCTeval protocol is a generalization of the symmetrically-private conditional transfer notion in [4], which, in turn, generalizes the conditional oblivious transfer from [6]. Specifically, it differs in formalizing privacy according to the secure computation notion. Both notions from [4,6] are, in turn, variants of the much studied oblivious transfer (OT) protocol notion from [21].

3 Secure Evaluation of Deterministic Automata

In this section we present our 2-party protocol for secure evaluation of a deterministic automata. The protocol consists of a private evaluation of Alice's deterministic automata on Bob's input string, using cryptographic primitives such as encrypted data blocks (also called pseudonyms), a symmetric-type and an asymmetric-type pseudo-random function, a secure pseudo-random function evaluation protocol, and a secure conditional transfer protocol. Formally, our protocol satisfies the following result.

Theorem 1. Assume the existence of:

1. symmetric-type pseudo-random function family prF_s
2. asymmetric-type pseudo-random function family prF_a,
3. an sPRFeval protocol for the evaluation of prF_a, and
4. an sCTeval protocol for equalities, and AND of equalities condition predicates.

There exists a (black-box) construction of a 2-party sDAeval protocol π, requiring $O(m)$ executions of the sPRFeval protocol, and $O(sam)$ applications of an sCTeval protocol, where s, a denote the number of states and alphabet symbols of the Alice's input automata, and m denotes the number of alphabet symbols in Bob's input string.

We note that the sPRFeval protocol from [8] only requires $O(1)$ asymmetric cryptography operations, and thus an execution of π based on them only requires $O(m)$ asymmetric cryptography operations, which is linear in the number of alphabet symbols input to Bob, and thus sublinear in the length $n = O(m \log a + sa \log s)$ of the input to the 2-party DAE problem. Instead, a direct application of the general solution from [22] would require $O(m \log a)$ asymmetric cryptography operations. We now prove Theorem 1 with a description of protocol π, and then show its efficiency and security properties.

Narrative and formal description of π. The description of protocol π can be divided into 4 phases: Alice's input processing, Bob's input processing, transition processing and output computation. At a high-level, π can be summarized as follows: in the first two phases, Alice and Bob compute encrypted data blocks or pseudonyms for their inputs; in the transition processing phase, Alice and Bob compute the transition steps in the DAE problem over encrypted data blocks; in the output computation phase, Alice and Bob compute the output of the DAE problem over encrypted data blocks, in a way that Bob receives the cleartext output. We now describe all 4 phases of protocol π in greater detail.

Alice's input processing. In this phase, Alice randomly chooses two keys: k_s for the symmetric-type pseudo-random function prF_s, and k_a for the asymmetric-type pseudo-random function prF_a. Then, Alice computes an initial set of encrypted pseudonyms for all s DFA states, as the output of the pseudo-random function prF_s on input the state symbol s_j and a position index $i = 0$, for $j = 1, \ldots, s$. Moreover, Alice computes an encrypted pseudonym for the a DFA alphabet symbols, as the output of the pseudo-random function prF_a on input the alphabet symbol a_h, for $h = 1, \ldots, a$. The detailed steps of this phase go as follows:

```
1. Alice randomly chooses keys $k_s, k_a$
2. For $j = 1, \ldots, s$, Alice computes $p_{S,j,0} = prF_s(k_s, (0|j))$
3. For $h = 1, \ldots, a$, Alice computes $p_{A,h} = prF_a(k_a, h)$
```

Bob's input processing. In this phase, Bob transforms each symbol in Bob's input string x into an encrypted pseudonym, to be computed as output of the pseudo-random function prF_a on input the symbol $x(i)$. This computation is performed by an execution of the sPRFeval protocol for each $i = 1, ..., m$, where Alice uses key k_a as input, and Bob uses x_i as input and receives $prF_a(k_a, x_i)$ as output. By the end of this phase, Bob has obtained the encrypted pseudonyms associated with all his input symbols $x_1, \ldots, x_m$. A formal description of this phase goes as follows:

```
1. For $i = 1, \ldots, m$,
     Alice and Bob run the sPRFeval protocol for function $prF_a$, where
        Alice's input is key $k_a$
        Bob's input is $x_i$
        Bob's output is $p_{x,i}$, intended to be $= prF_a(k_a, x_i)$
```

Circuit processing. In this phase, Alice sends to Bob the encrypted pseudonym associated with the initial state s_0 (set, wlog, =1), also being the current state. The invariant that Bob holds a valid encrypted pseudonym for the current state will be maintained throughout the protocol execution. Alice and Bob perform private evaluation of the deterministic automata, using the sCTeval protocol and the encrypted pseudonyms computed in the input processing phases. In the private evaluation of the deterministic automata, the execution continues in n iterations, where the i-th iteration, for $i = 1, \ldots, n$, goes as follows. First, Alice randomly chooses permutations α_i of $(1, \ldots, s)$ and β_i of $(1, \ldots, a)$, and computes an encrypted pseudonym for i-th variants of the s DFA states, as follows: the pseudonyms are outputs of the pseudo-random function prF_s on input the state symbol s_j and the position index i, for $j = 1, \ldots, s$. Then, for each symbol and state, Alice transfers the next state pseudonym to Bob, using an sCTeval protocol, where the condition is an AND of 2 equalities, defined so that Bob obtains the next state pseudonym sent by Alice in correspondence to the current state pseudonym and the current symbol pseudonym held by Bob. Alice will perform one execution of an sCTeval protocol for each of the possible current states and each of the possible symbols (in random orders according to permutations α_i, β_i), but only for one such pair is Bob holding the valid pseudonyms that meet both equalities; thus, Bob will receive the next state pseudonym only in correspondence of one such pair, in a random position, except with negligible probability. The detailed steps of this phase go as follows:

```
1. Alice computes $p_{S,1,0} = prF_s(k_s, (0|1))$ and sends $p_{S,1,0}$ to Bob
2. Bob sets $q_{S,0} = p_{S,1,0}$
3. For $i = 1, \ldots, m$,
     Alice randomly chooses permutations $\alpha_i$ of $(1, \ldots, s)$ and $\beta_i$ of $(1, \ldots, a)$
     for $j = 1, \ldots, s$,
       Alice computes $p_{S,j,i} = prF_s(k_s, (i|j))$
4. For $i = 1, \ldots, m$,
     for $j = \alpha_i(1), \ldots, \alpha_i(s)$,
       for $h = \beta_i(1), \ldots, \beta_i(a)$,
         Alice and Bob run the sCTeval protocol for the AND-of-equality function, where
           Alice uses as input key $k_s$ and pseudonyms $p_{S,j,i-1}, p_{A,h}$ and pseudonym $p_{S,j,i}$
           Bob uses as input pseudonyms $q_{S,i-1}$ and $p_{x,i}$
           Bob's output is in $\{\perp, z\}$ for some string $z \in \{0,1\}^\ell$, and
             it is intended to be $= p_{S,j,i} \neq \perp$ if $(p_{S,j,i-1} = q_{S,i-1})$ AND $(p_{A,h} = p_{x,i})$
     if Bob's output is $z \neq \perp$ then Bob sets $q_{S,i} = z$
```

Output computation. In the output computation phase, after the last symbol from string x is processed, Alice computes encrypted pseudonyms for the two expressions in set {yes, no} = {final-state, non-final-state} and transfers each of these two pseudonyms using an sCTeval protocol, using an equality condition, defined so that Bob obtains the appropriate pseudonym sent by Alice in correspondence to the current state pseudonym held by Bob. Alice will perform one execution of an sCTeval protocol for each of the possible current states, but only for one of these states, Bob is holding the valid pseudonym; thus, Bob will receive the final-state or non-final-state pseudonym only in correspondence of one such state, except with negligible probability. The detailed steps of this phase go as follows:

```
1. For $j = 1, \ldots, s$,
     Alice and Bob run the sCTeval protocol for the equality function, where
       Alice sets $s_j$ ='yes' if $j \in F$ or $s_j$ ='no' if $j \notin F$
       Alice uses as input key $k_s$, pseudonym $p_{S,j,m}$, and string $s_j$
       Bob uses as input pseudonym $q_{S,m}$
       Bob's output is in $\{\perp, z\}$ for some string $z \in \{0,1\}^\ell$, and
         it is intended to be $= s_j \neq \perp$ if $(p_{S,j,m} = q_{S,m})$ and $\perp$ otherwise
     if Bob's output is $z \neq \perp$ then Bob returns $z$ and halts.
```

Pictorial description. A pictorial description of this protocol can be found in Fig. 1. We remark that the circuit processing phase may be actually run in parallel across all $i = 1, \ldots, n$, and that the protocol in Fig. 1 can be easily adapted if we require Alice to be the party receiving the computation output, as follows. In the output computation phase, instead of obliviously transferring to Bob a yes or no string, Alice obliviously transfers a large random pseudonym for such strings, and then Bob sends back the received string to let Alice determine the output.

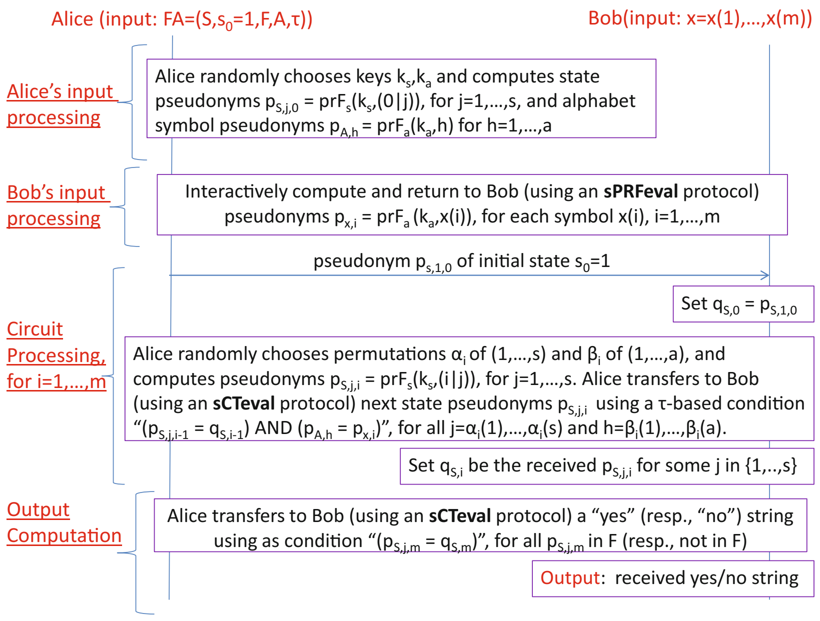
Alice (input: FA=(S,s_0=1,F,A,τ))
Bob(input: x=x(1),…,x(m))
Alice's input processing
Alice randomly chooses keys k_s,k_a and computes state pseudonyms $p_{S,j,0}$ = prF_s(k_s,(0|j)), for j=1,…,s, and alphabet symbol pseudonyms $p_{A,h}$ = prF_a(k_a,h) for h=1,…,a
Bob's input processing
Interactively compute and return to Bob (using an **sPRFeval** protocol) pseudonyms $p_{x,i}$ = prF_a (k_a,x(i)), for each symbol x(i), i=1,…,m
pseudonym $p_{s,1,0}$ of initial state s_0=1
Set $q_{S,0}$ = $p_{S,1,0}$
Circuit Processing, for i=1,…,m
Alice randomly chooses permutations $α_i$ of (1,…,s) and $β_i$ of (1,…,a), and computes pseudonyms $p_{S,j,i}$ = prF_s(k_s,(i|j)), for j=1,…,s. Alice transfers to Bob (using an **sCTeval** protocol) next state pseudonyms $p_{S,j,i}$ using a τ-based condition "($p_{S,j,i-1}$ = $q_{S,i-1}$) AND ($p_{A,h}$ = $p_{x,i}$)", for all j=$α_i$(1),…,$α_i$(s) and h=$β_i$(1),…,$β_i$(a).
Set $q_{S,i}$ be the received $p_{S,j,i}$ for some j in {1,..,s}
Output Computation
Alice transfers to Bob (using an **sCTeval** protocol) a "yes" (resp., "no") string using as condition "($p_{S,j,m}$ = $q_{S,m}$)", for all $p_{S,j,m}$ in F (resp., not in F)
Output: received yes/no string

Fig. 1. Our sDAeval protocol

Properties of π. It is easy to calculate Alice and Bob's runtime by inspection of protocol π. Specifically, Alice's runtime is dominated by her program in m executions of an sPRFeval protocol, $(ma+1)s$ executions of an sCTeval protocol, $(m+1)s$ computations of pseudo-random function prF_s and a computations of pseudo-random function prF_a. Bob's runtime is dominated by his program in m executions of an sPRFeval protocol and in $(ma+1)s$ executions of an sCTeval protocol. We now show the security properties of protocol π, considering two cases, depending on which of the two participants is corrupted by the adversary Adv.

Security, part 1 (Adv corrupts Alice): We note that in protocol π Alice's program consists of running a polynomial number of sPRFeval and sCTeval protocols, and sending an initial state pseudonym to Bob. Since these subprotocols are secure, they admit an efficient simulator whose output is computationally indistinguishable from the adversary's view during protocol execution. By suitably composing these simulators and running some of Alice's instructions, we obtain an efficient simulator for Adv's view when corrupting Alice in π. Specifically, simulator Sim_A runs Alice's program to simulate Alice's view during her input processing phase, runs the simulator for the sPRFeval protocol to simulate Alice's view during Bob's input processing phase, runs Alice's program to simulate the sending of the initial state pseudonym from Alice to Bob, and runs the simulator for the sCTeval protocol to simulate Alice's view during the circuit processing and output computation phases. The simulation's output is computationally indistinguishable from Alice's view since an analogue property holds for the simulators for the sPRFeval protocol and the sCTeval protocol.

Security, part 2 (Adv corrupts Bob): We note that in protocol π Bob's program consists of running a polynomial number of sPRFeval and sCTeval protocols, and receiving multiple pseudo-random state pseudonyms. Since these subprotocols are secure, they admit an efficient simulator whose output is computationally indistinguishable from the adversary's view during protocol execution. For every $i = 1, \ldots, m$, sa sCTeval protocols are executed by Bob and only one results in a pseudo-random state pseudonym (different than the error symbol) as output. The position of this non-erroneous execution can be simulated as a random position in the $s \times a$ matrix, and the received pseudo-random state pseudonyms can be simulated using a random string of the same length. By suitably composing these simulators and the generation of the next-state pseudonyms, we obtain an efficient simulator for Adv's view when corrupting Bob in π. Specifically, simulator Sim_B runs the simulator for the sPRFeval protocol to simulate Bob's view during Bob's input processing phase, randomly chooses pseudonym $p_{s,1,0}$ of initial state $s_0 = 1$ and sets $q_{s,0} = p_{s,1,0}$, randomly chooses next state pseudonym $p_{s,j,i}$ and sets $q_{s,j} = p_{s,j,i}$, runs the simulator for the sPRFeval protocol to simulate Bob's view during the executions of the sPRFeval protocol in the circuit processing phase, and runs the simulator for the sCTeval protocol to simulate the output computation phase. The simulation's output is computationally indistinguishable from Alice's view since an analogue property holds for the simulators for the sPRFeval protocol and the sCTeval protocol, and since a random permutation of both rows and columns of the transition matrix at each iteration $i = 1, \ldots, m$ implies that the distribution of the position of the state pseudonym received by Bob during the protocol is random within the $s \times a$ matrix.

4 Our sDAeval Protocol: A First Instantiation

In this section we describe a first instantiation of our main result that is asymptotically more efficient than previous schemes in the literature, having, in particular, running time sublinear in the length of Bob's input string x. The instantiation is obtained by an adaptation of the family of asymmetric-type PRF and related sPRFeval protocol from [2,7,13,16], and a simple sCTeval protocol for an AND of equalities between pseudo-random cryptographic pseudonyms, based on any symmetric-type PRF. In the rest of this section, we describe these 3 ingredients, and the efficiency properties of the resulting instantiation, denoted as π_1, of our main protocol.

A family of asymmetric-type PRFs. In this instantiation of π, the family of asymmetric-type pseudo-random functions, denoted as prF_a, is realized as an adaptation of the family used in [16], as we detail here. First, on input a security parameter 1^σ, the function's parameters are generated by running the following steps:

1. randomly choose $p_1, p_2, p'_1, p'_2 \in \{0,1\}^\sigma$ such that
   $p_1 = 2p'_1 + 1$, $p_2 = 2p'_2 + 1$, and p_1, p_2, p'_1, p'_2 are primes
2. set $n = p_1 p_2$
3. randomly choose an element g_1 of order n in a group Z_p^* such that
   p is the first prime such that p divides $(n-1)$
4. output: parameters (n, g_1)

Then, on input a randomly chosen key k in Z_n^* and an input string x in $\{0,1\}^q$, the function prF_a returns $g_1^t \mod p$, for $t = 1/(k+x)$ if $\gcd(k+x, n) = 1$ and 1 otherwise. Two remarks are necessary on this definition. First, the event $\gcd(k+x, n) \neq 1$ happens with negligible probability over the random choice of k, assuming the hardness of factoring numbers of the same distribution as n. Second, the length q of the input string was first thought in [16] to be limited by the number-theoretic assumption needed to prove the pseudo-randomness of this function family, but later [13] observed that such restriction is not needed (see below for more details). This function family, defined in [16], is a variant of the one in [7], in turn based on an unpredictable function from [2], the only modification being of using a group whose order is a safe RSA modulus instead of a group of prime order. The function from [2] was proved to be unpredictable under a number-theoretic assumption on the underlying group (i.e., the computational q-DHI assumption). A proof from [7] can be extended to show that this same function is a pseudo-random function under the mentioned number-theoretic assumption. Moreover, as stated in [16], the same arguments from [2] for prime-order groups also imply that (1) the function family considered here in a composite-order group is pseudo-random assuming the decisional q-DHI assumption on such groups and the hardness of factoring, and (2) the same generic-group argument which motivated trust in the q-DHI assumption on the prime-order groups carries to composite-order groups as well. Here, we recall a sketch of the definition of the q-DHI assumption: all efficient algorithms, given n and g, can only distinguish the two tuples

- $(g, g^u, g^{u^2}, \ldots, g^{u^q}, g^{1/u})$,
- $(g, g^u, g^{u^2}, \ldots, g^{u^q}, h)$

for random $h \in Z_p^*$ and $u \in Z_n^*$, with negligible probability. In later work [13], it has been showed that no restriction of value q is needed by observing that the function family we consider is (almost) a permutation.

An sPRFeval protocol for function prF_a. This protocol is a simplified version of the oblivious evaluation protocol from Fig. 1 of [16] for the above pseudo-random function prF_a. Specifically, this protocol evaluates the pseudo-random function f_K in Sect. 2.1 of the same paper and uses the encryption scheme in Sect. 2.2 of the same paper. The simplification is possible since we only require our protocol to be secure against semi-honest adversaries, and is thus obtained by removing the three zero-knowledge proofs of knowledge π_1, π_2, and π_3 from the protocol in Fig. 1 of [16]. The resulting protocol is an sPRFeval protocol for pseudo-random function prF_a, which will be run interactively by Alice and Bob and evaluate function prF_a on input Bob's input symbols. The proof for this fact is obtained as a corollary of the proof in [16]. As an optimization that does not affect the theorem validity, our implementation for π_1 also avoids the initial exponentiation to the k-th power of generator g_2 and just randomly chooses a symmetric key k instead. In this instantiation of π, function prF_a will also be computed non-interactively by Alice, on input relatively short strings denoting the alphabet symbols of the DA states.

An sCTeval protocol for 2 equalities conditions. We describe an sCTeval protocol for an equality condition and then one for an AND-of-equality condition. In both cases, we assume that all inputs to the equality statements are (large-size and pseudo-random) cryptographic pseudonyms. First, assume Alice wants to transfer some pseudonym p to Bob under the condition that Alice's pseudonym p_A is equal to Bob's pseudonym p_B. Based on any symmetric-type PRF prF_s with output length, the sCTeval protocol goes as follows:

1. Alice sets $k_A = p_A$, randomly chooses $r \in \{0,1\}^\sigma$, computes $u = prF_s(k_A, r)$, $v = u \oplus (p|0^\sigma)$, and sends (r, v) to Bob;
2. Bob sets $k_B = p_B$, computes $u' = prF_s(k_B, r)$; if $u' \oplus v = (m|0^\sigma)$ for some m, it returns: p; if not, it returns a special error symbol.

Note that if Alice's pseudonym p_A is equal to Bob's pseudonym p_B then Bob returns the same pseudonym p sent by Alice with probability 1; moreover, if Alice's pseudonym p_A is not equal to Bob's pseudonym p_B then Bob returns p only with negligible probability; finally, by the pseudo-randomness properties of prF_s, Alice learns no information about p_B and Bob learns no information about p_A, which implies the security of the protocol. This protocol is run s times in the output computation phase of π_1. Now, assume Alice wants to transfer some pseudonym p to Bob under the condition that Alice's pseudonym p_A is equal to Bob's pseudonym p_B and Alice's pseudonym q_A is equal to Bob's pseudonym q_B. Based on any symmetric-type PRF prF_s with output length, the sCTeval protocol goes as follows:

1. Alice randomly chooses p_1 and computes $p_2 = p \oplus p_1$
2. Alice transfers p_1 via the above sCTeval protocol for the string equality predicate, using the equality $(p_A = p_B)$ as condition;
3. Alice transfers p_1 via the above sCTeval protocol for the string equality predicate, using the equality $(q_A = q_B)$ as condition;

4. If Bob returns p_1', p_2' (and thus no error symbol) on any of these two executions, he returns $p' = p_1' + p_2'$; else Bob returns an error symbol.

Note that if both equalities are satisfied then Bob can compute $p_1' = p_1$ and $p_2' = p_2$ and return $p' = p$ with probability 1; moreover, if at least one equality is not satisfied then Bob returns $p' = p$ only with negligible probability. The security of this protocol directly follows from the security of the individual single-equality sCTeval protocols used.

Efficiency properties of π_1. Protocol π_1 only requires 3 messages between Alice and Bob, as the sPRFeval protocols require 3 messages (where Alice sends first), the sCTeval protocols require a single messages between Alice and Bob, and this latter message can be combined with the last message in the sPRFeval protocols. Alice's input processing phase requires $O(s)$ symmetric cryptography operations in Alice's executions of prF_s and $O(a)$ asymmetric cryptography operations in Alice's executions of prF_a. Bob's input processing phase requires $O(m)$ applications of an sPRFeval protocol, which only need $O(m)$ asymmetric cryptography operations. The circuit processing and output computation phase require $(sa+1)m$ applications of an sCTeval protocol, which need $O(sam)$ symmetric cryptography operations. Thus, in total, π_1 only requires $O(sam)$ symmetric cryptography and $O(m)$ asymmetric cryptography operations (instead of $O(m \log a)$, as required in a direct application of the general solution from [22]). An analogue improvement is observed in the protocol's communication complexity.

5 Our sDAeval Protocol: A Second Instantiation

In this section we describe a second instantiation of our main protocol that, although asymptotically less efficient than the first instantiation, is actually efficient for small automata alphabets, including some encountered in practice. The instantiation is obtained by replacing the use of asymmetric-type PRFs with any symmetric-type PRFs, and realizing a sPRFeval protocol by a suitable combinations of protocols from [19,20]. This realization uses the fact that the automata alphabet is small (i.e., polynomial in the security parameter), and that the PRF needs to be evaluated over any one of the a alphabet elements. In the rest of this section, we describe this different sPRFeval protocol, and the efficiency properties of the resulting instantiation, denoted as π_2, of our main protocol.

An sPRFeval protocol for any small-domain symmetric-type function prF_s. While in the first instantiation of π, we used a simplified version of the oblivious pseudo-random function evaluation protocol from Fig. 1 of [16], here, assuming that the number a of automata's alphabet symbols is small (i.e., polynomial in the security parameter, as opposed to super-polynomial), we use a protocol that can be based on: (1) any arbitrary symmetric-type pseudo-random function, including the already assumed prF_s, which we implement using a block cipher (e.g., AES), and (2) any secure (1-out-of-a)-OT protocol, such as the one in [19], which is in turn based on any arbitrary symmetric-type pseudo-random function, including the already assumed prF_s, and any arbitrary (1-out-of-2)-OT protocol, such as the one in [20], based on the hardness of the Decisional Diffie-Hellman problem. Specifically, the new oblivious pseudo-random function evaluation protocol will be an oblivious protocol for the evaluation of function on input (i, x_i), where x_i is the i-th element from Bobs input string $x = x_1, \ldots, x_m$, for all $i = 1, \ldots, m$. This protocol goes as follows:

1. Alice computes $z_h = prF_s(h, as_h)$, for $h = 1, \ldots, a$, where as_h denotes the h-th alphabet symbol according to a standard, lexicographic, ordering;
2. Alice uses the (1-out-of-a)-OT protocol from [19] to transfer z_t to Bob, where Alice uses as input strings $z_1, \ldots, z_a$, Bob uses as input $t \in \{1, \ldots, a\}$ such that $as_t = x_j$

The resulting protocol is a sPRFeval protocol for pseudo-random function prF_s, whenever a is polynomial in the security parameter. To summarize, protocol π_1 and π_2 differ in how the design of the sPRFeval protocol, affecting the type of PRF used and the assumption on the size of the automata alphabet. In π_1, the sPRFeval protocol is designed as an adaptation of the scheme from [16], and works for a specific asymmetric-type PRF, and for arbitrary-size automata alphabets. On the other hand, in π_2, the sPRFeval protocol is designed building from a (1-out-of-a)-OT scheme from [19], works for any arbitrary symmetric-type PRF, and for polynomial-size automata alphabets.

Efficiency properties of π_2. Protocol π_2 only requires 2 messages between Alice and Bob, as the sPRFeval protocols require 2 messages (where Bob sends first), the sCTeval protocols require a single messages between Alice and Bob, and this latter message can be combined with the last message in the sPRFeval protocols. As in π_1, Alice's input processing phase requires $O(s)$ symmetric cryptography operations in Alice's executions of prF_s and $O(a)$ asymmetric cryptography operations in Alice's executions of prF_a. Bob's input processing phase requires $O(m)$ applications of an sPRFeval protocol, which need $O(m \log a)$ asymmetric and symmetric cryptography operations. The circuit processing and output computation phase require $(sa + 1)m$ applications of an sCTeval protocol, which need $O(sam)$ symmetric cryptography operations. Thus, in total, π_2 requires $O(sam)$ symmetric cryptography and $O(m \log a)$ asymmetric cryptography operations, which is asymptotically similar to a direct application of the general solution from [22], but, as later shown in the performance evaluation of our implementation, comes with considerable runtime improvements.

6 Performance Analysis

In this section we discuss the practical performance of the two instantiations π_1, π_2 of our main protocol π, showing improved latency with respect to the original Yao's protocol, as well as previous protocols in the literature. We performed two types of analysis of our protocols' on-line computation (or latency): an asymptotic analysis, with comparison with previous sDAeval protocols, and a more practical numerical analysis based on measurements of our implementations running time.

Implementation setup, metric and parameter settings. Testing of our implementations of protocols π_1 and π_2 was done on 2 Dell PowerEdge 1950 processors and one Dell PowerEdge 2950 processor, Intel(R) Xeon(R) CPU E5405 @ 2.00 GHz. We run Alice and Bob's programs on the 2 PowerEdge 1950 processors, and the testing control was run on the 2950 processor. All offline and online communication traffic was run over a dedicated gigabit Ethernet LAN. Testing control and collection of timing measurement traffic was isolated on a separate dedicated gigabit Ethernet LAN.

In our performance experiments, we mainly evaluated on-line computation time (or latency), divided into asymmetric cryptographic operations, with security parameter σ, and symmetric cryptography operations, with security parameter λ. Recall that in practice the former type of operations is expected to require computing resources greater than the latter type by orders of magnitude (slightly more than 3 orders on our machines, when setting $\sigma = 2048$ and $\lambda = 128$). In addition than σ and λ, latency was evaluated over input length parameters s (the number of DA states), a (the number of DA alphabet symbols) and m (the number of symbols in Bob's input string x).

Performance evaluation of π_1, π_2. Off-line computation performance (used for the generation of one-time public keys and parameter value settings) required 200 s in π_1 and less than 4 s in π_2. Memory used by π_1 (resp., π_2) was 337 Mbytes (resp., 2.553 Mbytes).

Practical numeric latency times of our protocols are heavily dependent on the subset of values that we consider as settings for parameters s, a, n. We considered three main cases for parameter a, reflecting related application scenarios; namely:

- $a = 2$ (binary alphabet),
- $a = 27$ (English alphabet plus one special symbol for all other alphabet symbols),
- $a = 128$ (smallest power of 2 that includes all ASCII symbols).

For these three values, we measured running times and extrapolated them into estimates of close-to-maximum values of s and n such that the overall latency of the protocols would remain below 30 s or 10 s.

The found values are captured in Table 1 below.

Comparison of asymptotic performance with previous sDAeval protocols. We compared our protocols π_1, π_2 with previous sDAeval protocols achievable from results in [9,15,18,22]. This comparison was performed by evaluation of asymptotic expressions for their latency, and is summarized in Table 2 below. (While we were able to extend the protocol from [22] to a non-binary automata alphabet size, it was unclear how to do the same with the protocol in [18], which is defined for binary alphabets.)

We remark that the number $O(m)$ of asymmetric cryptography operations in π_1 is sublinear in the total length of the input $n = O(m \log a + sa \log s)$ to Alice and Bob.

Table 1. Max s, a, n parameter values obtained for our protocols, under constraints of latency ≤ 30 s (columns 2, 3, 4) and of latency ≤ 10s s (columns 5, 6, 7).

Protocol	s	a	n	s	a	n
π_1	200	2	84	200	2	28
π_1	70	27	70	70	27	23
π_1	80	128	42	80	128	14
π_2	200	2	6200	200	2	2100
π_2	100	27	1500	100	27	500
π_2	80	128	75	80	128	25

Table 2. Asymptotic latency analysis, relative to alphabet type.

Protocol	Asymmetric crypto	Symmetric crypto	Alphabet
	Operations	Operations	Type
[22]	$O(m \log a)$	$O(sm \log a \log s)$	a-ary, for any $a \geq 2$
[9]	$O(ms \log a)$	None	a-ary, for any $a \geq 2$
[15]	$O(ms \log a)$	None	a-ary, for any $a \geq 2$
[18]	$O(m \log a)$	$O(sm \log a)$	Binary
π_1	$O(m)$	$O(sam)$	a-ary, for any $a \geq 2$
π_2	$O(m \log a)$	$O(sam)$	a-ary, for any $a \geq 2$

Performance comparison with Yao's protocol. To obtain some insights on the 'computing with encrypted data blocks' paradigm underlying our sDAeval protocol, we compared our sDAeval protocols' performance with sDAeval protocols obtained by instantiating the state-of-the-art solutions in the area of general-purpose secure 2-party function evaluation (specifically, the garbled circuit paradigm, starting with [22], and the fully homomorphic encryption paradigm, starting with [10]). We expanded the quantitative performance comparison framework used in [5] to obtain numeric (as opposed to asymptotic) performance estimates and derive a comparison of these 3 approaches. It was quickly apparent that an sDAeval protocol obtained using the fully homomorphic encryption paradigm, would be the least efficient. Accordingly, we focused our analysis on comparing our sDAeval protocols obtained using the computing with encrypted data blocks paradigm with the sDAeval protocol obtained using the garbled circuit paradigm from [22].

Let π_Y denote the 2-party sDAeval protocol obtained using the garbled circuit paradigm from [22]. Then, for any 2-party sDAeval protocol π, we define the *latency ratio* for π, as follows:

$$\text{latency ratio}(\pi) = \text{latency}(\pi_Y)/\text{latency}(\pi).$$

In our analysis, we have found that the latency ratio is essentially independent on parameter m, which makes it much easier to analyze as a function of the

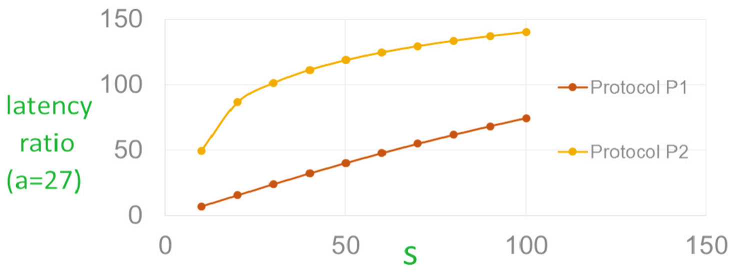

Fig. 2. Latency ratio of π_1 and π_2, with alphabet size 27, as a function of the number s of states

remaining parameters s and a. Using a combination of runtime measurements and estrapolations based on protocol analysis, we characterized the latency ratio for our protocols π_1 and π_2 in Fig. 2 below. We observe that the improvement of π_1 (respectively, π_2) over π_Y varies between almost 1 to about 1.8 (respectively, almost 1.5 to about 2.2) orders of magnitude in the shown parameter value space and further increases with the number s of states.

7 Conclusions and Open Directions

This work can be considered a next step in the direction of [5], where we have introduced a paradigm for the design of more efficient secure function evaluation protocols, performing the most computationally expensive operations (i.e., asymmetric cryptography operations) over input data blocks instead of input data bits, while maintaining efficient key sizes. This addresses performance shortcomings of the 2 main general-purpose secure function evaluation approaches in the literature: Yao's garbled circuits, which has efficient key sizes but operates on single data bits, and Gentry's fully homomorphic encryption, which operates on input data blocks when applied to arithmetic circuits, but with inefficient key sizes. In [5], we had shown efficient secure function evaluation protocols satisfying both requirements, for the class of monotone formulae over equality statements. In this paper, we show such a protocol for deterministic automata evaluation, which, being equivalent to regular expression matching, captures all constant-space computations. A first open research direction is to improve the practical performance of secure protocols for pseudo-random function evaluation protocols, which would improve the performance of the first instantiation of our protocol on practical parameter settings. A second and more general open research direction is that of finding more instances of protocols following the paradigm of computations over encrypted data blocks, possibly including large classes of polynomial-size circuits.

Acknowledgments. We thank Jonathan Katz for pointing us to a useful reference. This work was supported by the Defense Advanced Research Projects Agency (DARPA) via Air Force Research Laboratory (AFRL), contract number FA8750-14-C-0057. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation hereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, AFRL or the U.S. Government.

References

1. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M., Toft, T.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 325–343. Springer, Heidelberg (2009). doi:10.1007/978-3-642-03549-4_20
2. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24676-3_4
3. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, pp. 97–106, 22–25 October 2011
4. Crescenzo, G.: Private selective payment protocols. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 72–89. Springer, Heidelberg (2001). doi:10.1007/3-540-45472-1_6
5. Di Crescenzo, G., Coan, B.A., Kirsch, J.: Efficient computations over encrypted data blocks. In: Proceedings of the 40th International Symposium on Mathematical Foundations of Computer Science, Part II, MFCS 2015, Milan, Italy, pp. 274–286, 24–28 August 2015
6. Di Crescenzo, G., Ostrovsky, R., Rajagopalan, S.: Conditional oblivious transfer and timed-release encryption. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 74–89. Springer, Heidelberg (1999). doi:10.1007/3-540-48910-X_6
7. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (2005). doi:10.1007/978-3-540-30580-4_28
8. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (2005). doi:10.1007/978-3-540-30576-7_17
9. Gennaro, R., Hazay, C., Sorensen, J.S.: Automata evaluation and text search protocols with simulation-based security. J. Crypt. **29**(2), 243–282 (2016)
10. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the of 41st ACM STOC, pp. 169–178 (2009)
11. Goldreich, O.: The Foundations of Cryptography - Basic Applications, vol. 2. Cambridge University Press, New York (2004)
12. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM **33**(4), 792–807 (1986)
13. Hazay, C., Nissim, K.: Efficient set operations in the presence of malicious adversaries. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 312–331. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13013-7_19

14. Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: Proceedings of 20th USENIX Security Symposium (2011)
15. Ishai, Y., Paskin, A.: Evaluating branching programs on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 575–594. Springer, Heidelberg (2007). doi:10.1007/978-3-540-70936-7_31
16. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 577–594. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00457-5_34
17. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: Proceedings of 13th USENIX Security Symposium, pp. 287–302 (2004)
18. Mohassel, P., Niksefat, S., Sadeghian, S., Sadeghiyan, B.: An efficient protocol for oblivious DFA evaluation and applications. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 398–415. Springer, Heidelberg (2012). doi:10.1007/978-3-642-27954-6_25
19. Naor, M., Pinkas, B.: Oblivious transfer with adaptive queries. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 573–590. Springer, Heidelberg (1999). doi:10.1007/3-540-48405-1_36
20. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, Washington, DC, USA, pp. 448–457, 7–9 January 2001
21. Rabin, M.O.: How to exchange secrets with oblivious transfer. IACR Cryptology ePrint Archive, 2005:187 (2005)
22. Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: Proceedings of 27th IEEE FOCS, pp. 162–167 (1986)