Multi-Party Computation of Polynomials and Branching Programs without Simultaneous Interaction

S. Dov Gordon[1⋆], Tal Malkin[2 ⋆⋆], Mike Rosulek[3⋆⋆⋆], and Hoeteck Wee[4†]

[1] Applied Communication Sciences
[2] Columbia University
[3] University of Montana
[4] George Washington University

Abstract. Halevi, Lindell, and Pinkas (CRYPTO 2011) recently proposed a model for secure computation that captures communication patterns that arise in many practical settings, such as secure computation on the web. In their model, each party interacts only once, with a single centralized server. Parties do not interact with each other; in fact, the parties need not even be online simultaneously.

In this work we present a suite of new, simple and efficient protocols for secure computation in this "one-pass" model. We give protocols that obtain optimal privacy for the following general tasks:

- Evaluating any multivariate polynomial $F(x_1, \ldots, x_n)$ (modulo a large RSA modulus N), where the parties each hold an input x_i.
- Evaluating any read once branching program over the parties' inputs.

As a special case, these function classes include all previous functions for which an optimally private, one-pass computation was known, as well as many new functions, including variance and other statistical functions, string matching, second-price auctions, classification algorithms and some classes of finite automata and decision trees.

1 Introduction

Most of the literature on secure multi-party computation assumes that all parties remain on-line throughout the computation. Unfortunately, this assumption

⋆ Parts of this work was completed while the author was a postdoctoral researcher at Columbia University.

⋆⋆ Supported in part by NSF grant CCF-1116702 and by the the Intelligence Advanced Research Project Activity (IARPA) via Department of Interior National Business Center (DoI / NBC) contract Number D11PC20194. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. 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 IARPA, DoI/NBC, or the U.S. Government.

⋆⋆⋆ Supported by NSF grant CCF-1149647.

† Supported by NSF CAREER Award CNS-1237429

is problematic in many emerging environments, where the parties are often disconnected from the network due to geographic or power constraints. Moreover, the protocols typically require each party to broadcast a large number of messages to the other parties, which can be quite impractical in large distributed networks. We would like to minimize interaction to the greatest extent possible due to practical communication and bandwidth considerations — ideally, each party would need to send only one message.

We consider secure computation in a one-pass client-server model put forth in a recent work of Halevi, Lindell and Pinkas [12].[5] In this model, there is a single server and multiple clients, and the goal is for the server to securely compute some function of the inputs held by the respective clients. Each client connects to the server once (hence "one-pass") and interacts with it, without any other client necessarily being connected at the same time. In particular, there is no need for any two clients to interact. This model is applicable in settings where maintaining constant network connectivity can be problematic — for example, when deployed troops are communicating with the central command center. It is also applicable in situations where the participants cannot be coordinated for social reasons. Imagine trying to get thirty program committee members across different time zones online at the same time to cast a vote. Instead, in the one-pass model, each will receive an email instructing them to login to the server at their leisure. When all participants have done so, the server can compute the output and post the data to a website (or email it out). Similarly, if a website would like to gather data from its visitors, it is unreasonable to ask that they remain logged-in to the site for the duration of the computation. Instead, as they login, they can upload the relevant data according to the protocol, assured of their privacy, and the server can compute the agreed-upon function offline.

1.1 Security for the One-Pass Model

We briefly outline the security model for the one-pass client-server setting and previous results of Halevi et al. [12] — hereafter, "HLP." First, observe that secure computation in this setting is easy if the server is always honest, and is trusted with user data: each client simply sends its input to the server, encrypted under the server's public key; the server will then perform all of the computation. However, assuming that the server is completely honest is not realistic. Instead, we aim to protect the privacy of the honest parties' inputs even amidst a malicious server that may collude with some subset of the clients. Together with the requirement that the protocol be one-pass, this imposes inherent limitations on what we can securely compute in this model. To see why this is the case, consider parties $P_1, P_2, \ldots, P_n$ computing some function $f(x_1, \ldots, x_n)$, where party P_i holds x_i and the parties go in order $P_1, P_2, \ldots, P_n$. If the server colludes with the last t parties, then the correctness and one-pass nature of the protocol imply that the coalition can compute the "residual function"

[5] The ideas of "non-interactive" and "one-pass" computations can be further traced back to [18, 14]. See Section 1.3.

$f(x_1, \ldots, x_{n-t}, \cdot, \cdots, \cdot)$, on *any* choice of a t-tuple $(z_{n-t+1}, \ldots, z_n)$, and for arbitrarily many such choices. In other words, *inherent* to this one-pass model is the fact that parties $P_1, \ldots, P_{n-t}$ must disclose enough information about their inputs to allow the remaining parties to correctly evaluate the residual function $f(x_1, \ldots, x_{n-t}, \cdot, \cdots, \cdot)$. Once the last parties have this information, nothing can prevent them from using it repeatedly. This is in stark contrast to the standard interactive model for secure computation, where the adversary only learns the output of the computation on a single set of inputs, and which allows us to securely compute every efficiently computable function [19, 10].

Due to these inherent limitations of the one-pass model, the "best possible" security guarantee that one could hope for is that the protocol reveals *no more information* than what is revealed by oracle access to this residual function $f(x_1, \ldots, x_{n-t})$. Throughout this paper, this will be the notion we mean when we refer to security (following [12], we will also refer to this notion as *optimal privacy*); for completeness, we provide the formal definitions in Section 2.2. HLP [12] presented practical optimally private protocols for sum of inputs, selection, and symmetric functions like majority, and leave as an open problem whether we can obtain practical optimally private protocols for some larger classes of functions. Indeed, there is no clear candidate for such a larger class of functions as the previous protocols are somewhat ad-hoc and seem to rely on different ideas.

Even ignoring the issue of practical efficiency, the aforementioned functions are essentially the only ones for which we have optimally private protocols. The main technical challenge in designing optimally private protocols is as follows: on one hand, the view y_i of the server after interaction with party P_i should encode sufficient information about the first i inputs $x_1, \ldots, x_i$ to be able to compute the function f; on the other hand, in order to establish security, the simulator needs to be able to efficiently reconstruct the view y_i given only oracle access to the residual function $f(x_1, \ldots, x_i, \cdot, \cdots, \cdot)$. HLP formalize this via the notion of *minimum-disclosure decomposition*, which is a combinatorial property of the function itself, providing a necessary condition for the existence of an optimally private protocol. In addition, they demonstrate that every function with this combinatorial property admits *some* optimally private protocol, albeit a highly inefficient one. However, beyond the small classes of functions mentioned above, they do not demonstrate that any function has such a property. Indeed, using pseudorandom functions, they demonstrate that not all functions have a minimum-disclosure decomposition.

1.2 Our results

We present practical, optimally private protocols for two broad classes of functions: (1) sparse polynomials over large domains, which capture many algebraic and arithmetic functions of interest, such as mean and variance, and (2) read-once branching programs, which capture symmetric functions, string matching, classification algorithms and some classes of finite automata and

decision trees (c.f. [15, 14]).[6] Together, these two classes capture all of the functions addressed in the previous work of HLP, and also include many more functions of interest. One such concrete example is a second-price auction (the n-party functionality that returns the *index* of the largest value along with the second largest value). This function is asymmetric, but can be represented as a branching program. A second-price auction with n parties and discrete bids in the range $\{1, \ldots, k\}$ has an associated branching program of width nk^2.

We begin by giving a simplified exposition of the protocols (achieving security against semi-honest adversaries), and outlining the simulation strategies used in the proof of security. In particular, the simulation strategies provide a solution to the *minimum-disclosure decomposition* problem.

Computing sparse polynomials. Consider a sparse[7] polynomial F in n variables $X_1, \ldots, X_n$, where party P_i holds an input x_i for variable X_i. The parties go in the order $P_1, \ldots, P_n$. Consider the following polynomial:

$$F_i(X_{i+1}, \ldots, X_n) := F(x_1, \ldots, x_i, X_{i+1}, \ldots, X_n).$$

Informally, party P_i will post to the server an encryption of the coefficients of polynomial F_i. The next party P_{i+1} will homomorphically evaluate an encryption of (the coefficients of) F_{i+1} given its input x_{i+1} and the previous encryption of F_i (Figure 1). To do so, the encryption scheme must be homomorphic with respect to affine functions over the integers. We are able to realize such an encryption scheme from the DCR assumption, which leads to a one-pass protocol for computing sparse polynomials over $\mathbb{Z}_N$, where N is a RSA modulus. Overall, each party does $O(M)$ group operations and sends $O(M)$ group elements, where M is an upper bound on the number of monomials in F.

To establish security of this protocol, we must show a simulator that can efficiently reconstruct the coefficients of F_i given oracle access to appropriate residual function, which in this case is F_i itself. (For technical reasons, the simulator needs to reconstruct not just the encrypted coefficients but the coefficients themselves.) We show that by querying F_i on sufficiently many random points, the simulator can obtain the coefficients of F_i by solving a suitable system of linear equations.

Computing branching programs. Consider a layered read-once branching program, where party i holds the input x_i in the i'th layer. Our protocol proceeds by evaluating the branching program in a *bottom-up* manner, "percolating" output labels from the end of the branching program towards the start node. Accordingly, we label the output layer of the branching program L_0, and layers $L_1, \ldots, L_n$ proceed up from there. The parties go in order $P_1, \ldots, P_n$, and party P_i will post to the server an encryption of the output labels on all of

[6] For technical reasons outlined below, our protocol for computing polynomials relies on having a large input domain (namely, $\mathbb{Z}_N$). On the other hand, the nature of branching programs makes them well-suited to functions with small input domains. Thus these two classes of functions are somewhat incomparable.

[7] That is, F can be written as the sum of $\text{poly}(n)$ monomials.

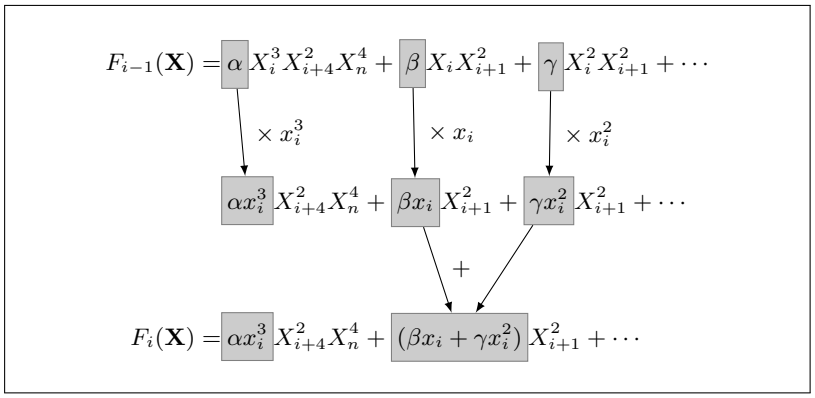

Fig. 1. Obtaining coefficients of F_i using the coefficients of F_{i-1} and the value of x_i. Shaded boxes are encrypted values. Operations on arrows are homomorphic operations possible in an additively homomorphic scheme.

the nodes in the i'th layer. The next party, P_{i+1}, generates an encryption of labels in layer $i+1$, given x_{i+1} and an encryption of labels in the i'th layer (Figures 2 & 3). Due to the simplicity of the percolation operation, it suffices to use an encryption scheme which is homomorphic with respect to the identity map (i.e., *re-randomizable*). Such an encryption scheme may be realized from the DCR, DDH and DLIN assumptions (the latter two instantiations are important for compatibility with Groth-Sahai proofs [11]). Overall, each party does $O(w)$ group operations and sends $O(w)$ group elements, where w is an upper bound on the width of the branching program.

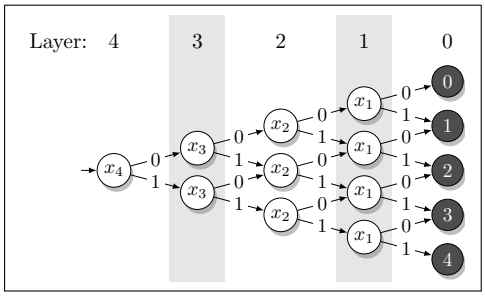

Fig. 2. A layered branching program for computing a tally among 4 parties. Output nodes are darkly shaded.

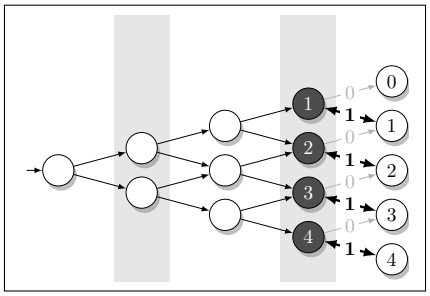

Fig. 3. How party #1 truncates the branching program, corresponding to input $x_1 = 1$.

To establish security of this protocol, we must show a simulator that can efficiently compute the labels that the protocol assigns to the layer corresponding to the last honest party, given oracle access to the appropriate residual function. For each node u in the i'th layer, the simulator runs a depth-first search to find a path to u from the start node in the branching program. The path determines a set of inputs on which to query the residual function; the result of the query will be the label on the node u.

The full-fledged protocol: more details. The outline above is a little over-simplified. The parties will in fact need to use a homomorphic *threshold* encryption scheme, which is also re-randomizable, in order to provide "circuit privacy" (that is, hide the homomorphic operations). Roughly speaking, the i'th party P_i's message will be encrypted under the public keys of parties $P_{i+1}, \ldots, P_n$ and the server, so that the message will be private unless all of these parties and the server are corrupted. The use of homomorphic threshold encryption here is analogous to previous constructions [12].

The protocols outlined above obtain optimal privacy against only semi-honest adversaries. To achieve security against malicious adversaries, we can use a generic GMW-style compiler via non-interactive zero-knowledge proofs in the random oracle model, in line with previous work. For our branching-program protocol, we provide an alternative method, in the standard model, that relies on Groth-Sahai proofs. The same approach does not apply to our polynomial-evaluation protocol, since it requires an additively homomorphic encryption scheme, and none are known that are compatible with Groth-Sahai proofs.

As with previous constructions, our protocols can often be extended to handle arbitrary ordering of the players (which is useful in such an asynchronous interaction setting). Indeed, this is the case for our polynomial evaluation protocols. Our branching-program protocol can also allow for arbitrary ordering if the function computed is such that the branching program can be adjusted "on the fly" based on the order in which the parties show up; this is the case for all symmetric functions, as well as some asymmetric ones such as the second-price auction mentioned above.

Finally, we note that while the previously known constructions of [12] are captured as special cases of our two protocols, our technical novelty over these previous constructions is two-fold. First, for our polynomial-evaluation protocol we provide a novel threshold homomorphic encryption scheme based on the DCR assumption. This is important for extending the expressivity from simple summations to more general polynomials while keeping the protocol practical.[8] Second, proving security for our constructions (in particular, proving that the functions admit minimum-disclosure decompositions) requires much more sophisticated simulation strategies than those required by the previous work. In particular, for the classes of functions considered previously, there is no need to solve systems of linear equations or solve s-t connectivity, as we do in this work.

1.3 Additional related work

Related constructions. Surprisingly, our result statements are similar to the results of Harnik, Ishai and Kushilevitz [13, Section 4] for a very different problem. They showed how to securely compute branching programs and sparse

[8] Recall that if efficiency is not an issue, then we could instead rely on threshold fully homomorphic encryption, or a threshold variant of i-hop garbled circuits [8], as shown in [12].

polynomials[9], where every pair of parties makes a single call to an oblivious transfer channel. In their setting, as in ours, the parties incrementally maintain a succinct representation of the inputs of the first i parties. Beyond that similarity, however, the security goal and the underlying communication model are very different. Specifically, they achieve security in the standard MPC setting where the simulator calls the ideal functionality once (there is no "one pass" restriction); indeed, our simulation strategy is very different from theirs. An interesting open problem is to adapt their result on linear branching programs to our setting; the key technical obstacle appears to be solving the analogue of s-t connectivity on the computation graph for linear branching programs.

Related models. There is a large body of work considering the general theme of secure computation with a restricted communication pattern. Sander, Young and Yung [18] were the first to put forth the notion of 'non-interactive' secure computation, but only in the context of two-party computation. Extensions to the multi-party setting were addressed recently in the work of Ishai et al. [16]. These are essentially 'two-pass' protocols, where it is still possible to securely compute any efficiently computable function. Secure computation in two passes was also recently considered by Asharov et al. [**?**].

The notion of one-pass computation was considered by Ibrahim, Kiayias, Yung and Zhou [14]. The notion of security is however quite different – roughly speaking, they do not allow the server to collude with the clients, which is in some sense the main source of technical difficulty in the model we study here; their main goal is to minimize server's storage. Ibrahim et al. also provided an efficient protocol for computing branching programs in their model. We note that their protocol is very different from ours: (1) the computation is done in a top-down manner, whereas ours is done in a bottom-up manner; and (2) the transitions from one layer to the next is encoded using a degree w polynomial where w is the width of the branching program, and the parties homomorphically evaluate a degree w polynomial on ciphertexts. The authors showed how to realize the latter based on only the DCR assumption, whereas our protocol may be based on either the DDH, DLIN, or DCR assumptions. The idea for evaluating branching programs in a bottom-up manner originates in a paper of Ishai and Paskin [15] in a different context; their main result exploits the DCR assumption to obtain short ciphertexts.

Other related works. We also point out that both classes of functions we consider in this work have been studied in several recent works in a variety of different settings [3, 2, 17, 15, 14].

Organization. We summarize the general one-pass framework [12] (including minimum-disclosure decomposition) Section 2. We provide a generic protocol

[9] They handle sparse polynomials over bits, whereas we consider sparse polynomials over $\mathbb{Z}_N$. In addition, they evaluate the branching programs top-down, whereas we do it bottom-up.

construction in Section 3, and show how to apply it to computing polynomials and branching programs in Sections 4 and 5 respectively. We provide concrete instantiations for underlying primitives in Section 6.

2 General Framework

We design our protocols in the registered public-key infrastructure (PKI) model [1]. We assume that in an initial setup phase every party registers a public and private key pair with a central authority and all the public keys are made known to everyone. We discuss the exact assumptions in the full version.

2.1 Decompositions

As described above, we prove that our protocols leak only the minimum possible information, even if the server colludes with some of the players. We assume that parties $P_1, \ldots, P_n$ interact with the server in order, with P_1 going first and P_n going last.[10] As in [12], we define a decomposition of the function f that the players are computing, by a sequence of functions $f_1, \ldots, f_n$.

Definition 1 (Decomposition). *For a function $f : D^n \to R$, we define a* decomposition *of f by a tuple of n functions, $f_1, \ldots, f_n$, where $f_1 : D \to \{0,1\}^*$, $f_i : \{0,1\}^* \times D \to \{0,1\}^*$ for $1 < i < n$, and $f_n : \{0,1\}^* \times D \to R$, such that for all $(x_1, \ldots, x_n) \in D^n$, it holds that $f_n(f_{n-1}(\cdots f_2(f_1(x_1), x_2) \cdots, x_{n-1}), x_n) = f(x_1, \ldots, x_n)$. We define a partial decomposition inductively as $\tilde{f}_1(x_1) = f_1(x_1)$ and $\tilde{f}_i(x_1, \ldots, x_i) = f_i(\tilde{f}_{i-1}(x_1, \ldots, x_{i-1}), x_i)$.*

Minimum-Disclosure Decompositions: As in the work of Halevi et al. [12], we use the notion of a *minimum-disclosure decomposition* to argue that our protocols reveal as little information as possible. For a function f, a decomposition of f given by $f_1, \ldots, f_n$, some fixed inputs $\mathbf{x} = (x_1, \ldots, x_n)$, and for all $i \in [n]$, we define the residual function $g_i^{\mathbf{x}}(z_{i+1}, \ldots, z_n) = f(x_1, \ldots, x_i, z_{i+1}, \ldots, z_n)$.

Definition 2 ([12]). *A decomposition of function f, given by $f_1, \ldots, f_n$, is a* minimum-disclosure decomposition *if there exists a probabilistic, black-box simulator $\mathcal{S}$ that for any set of inputs $\boldsymbol{x} = (x_1, \ldots, x_n)$ having total length m, and any $i \in [n]$, when $\mathcal{S}$ is given black-box access to an oracle computing $g_i^{\boldsymbol{x}}(\cdot)$, the output of the simulator satisfies $\mathcal{S}^{g_i^{\boldsymbol{x}}(\cdot)}(m, n, i) = \tilde{f}_i(x_1, \ldots, x_i)$, and the running time of $\mathcal{S}^{g_i^{\boldsymbol{x}}(\cdot)}(m, n, i)$ is polynomial in m and n.*

2.2 Defining Security

Security is defined using the real/ideal world paradigm [9, 12]. In the ideal world, there is a trusted party that computes f, which is represented by some fixed

[10] As noted before, the parties can actually interact with the server in arbitrary order for our polynomial evaluation protocol and in many cases for the branching program protocol as well.

decomposition, $f_1, \ldots, f_n$. Each party P_i gives input x_i to the trusted party. If P_i is honest, or semi-honest, he simply uses the value x_i that was found on his input tape; a malicious $P_i(z)$, with auxiliary information z, may use any input of his choice. We denote the corrupted set of parties by $\mathcal{I} \subset \{P_1, \ldots, P_{n+1}\}$. If $P_{n+1} \notin \mathcal{I}$ (i.e. if the server is honest), the trusted party sends output $f(x_1, \ldots, x_n)$ to the server. If $P_{n+1} \in \mathcal{I}$, then we let i^* denote the largest index such that $P_{i^*} \notin \mathcal{I}$ (i.e. P_{i^*} is the last honest party). The trusted party ignores inputs $(x_{i^*+1}, \ldots, x_n)$ and sends $\tilde{f}_{i^*}(x_1, \ldots, x_{i^*})$ to the adversary controlling $\mathcal{I}$. In this case, we stress that the trusted party does *not* send $f(x_1, \ldots, x_n)$, although this can of course be computed by the adversary once he is given $\tilde{f}_{i^*}(x_1, \ldots, x_{i^*})$. This subtlety becomes important while proving security, because the simulator will have no way to extract the input of malicious party P_j for $j > i^*$.

In the real world, f is computed by a sequence of protocols $\pi = (\pi_1, \ldots, \pi_n)$, where π_i is a two-party protocol between the server and P_i. Each party P_i uses input x_i in π_i, and, as above, if they are honest or semi-honest, they use the input found on their input tape. The server uses his output from π_{i-1} as input to π_i. Each player is also given all $n+1$ public keys, denoted by $\widetilde{\mathrm{PK}}$, which are set up as described at the beginning of this Section.

Let $\mathcal{S}(z)$ denote an ideal-world adversary holding auxiliary input z and corrupting some set of parties $\mathcal{I}$. On input set $\mathbf{x} = (x_1, \ldots, x_n)$ and security parameter κ, we denote the output of $\mathcal{S}(z)$ and server P_{n+1} by $\mathsf{Ideal}_{\bar{f},\mathcal{S}(z),\mathcal{I}}(\mathbf{x}, z, 1^\kappa)$. Let $\mathcal{A}(z)$ denote a real-world adversary holding auxiliary input z and corrupting the set of parties $\mathcal{I}$. On input set $\mathbf{x} = (x_1, \ldots, x_n)$ and security parameter κ, we denote the output of $\mathcal{A}(z)$ and server P_{n+1} by $\mathsf{Real}_{\bar{f},\mathcal{A}(z),\mathcal{I}}(\mathbf{x}, z, \widetilde{\mathrm{PK}}, 1^\kappa)$.

Definition 3 ([12]). *We say that a protocol $\pi = (\pi_1, \ldots, \pi_n)$ securely computes a decomposition $\bar{f} = (f_1, \ldots, f_n)$ with optimal privacy, if π is a minimum decomposition for $\bar{f}$, and if for any non-uniform, PPT adversary $\mathcal{A}(z)$ corrupting some subset of parties $\mathcal{I}$ in the real-world, there exists a non-uniform, PPT adversary $\mathcal{S}(z)$ corrupting $\mathcal{I}$ in the ideal-world such that*

$$\left\{\mathsf{Ideal}_{\bar{f},\mathcal{S}(z),\mathcal{I}}(\boldsymbol{x}, z, 1^\kappa)\right\} \stackrel{c}{=} \left\{\mathsf{Real}_{\pi,\mathcal{A}(z),\mathcal{I}}(\boldsymbol{x}, z, \widetilde{\mathrm{PK}}, 1^\kappa)\right\}.$$

2.3 Homomorphic threshold encryption

Our constructions require a (n-out-of-n) threshold encryption scheme which supports the following properties in addition to the standard Enc, Dec, and Gen procedures: (These properties generalize the "layer re-randomizable encryption" in [12, Definition 4.1].)

- To encrypt to a set of users whose corresponding public keys comprise the set S, one simply *aggregates* their public keys via $\widetilde{\mathrm{PK}} \leftarrow \mathsf{Aggregate}(S)$, and then encrypts normally treating $\widetilde{\mathrm{PK}}$ as a normal public key.
- The scheme is *homomorphic* (with respect to a class of functions we specify later when describing our main protocols). More formally, there is a procedure Eval which takes a (possibly aggregated) public key, a ciphertext,

and a function, and outputs another ciphertext. We then require that for all valid keypairs $(\mathrm{SK}, \mathrm{PK})$, all supported functions f, and all ciphertexts C:

$$\mathsf{Dec}(\mathrm{SK}, \mathsf{Eval}(\mathrm{PK}, C, f)) = f(\mathsf{Dec}(\mathrm{SK}, C))$$

– Given an encryption C under public keys $\mathrm{PK}_1, \ldots, \mathrm{PK}_n$, the owner of any corresponding secret key SK_i, $i \in [n]$, can transform C into a (*fresh*) encryption of the same message, under the remaining $n-1$ public keys. More formally, there is a procedure Strip which takes a (aggregated) public key, a secret key, and a ciphertext, and outputs another ciphertext. We require that, for all valid keypairs $(\mathrm{SK}^*, \mathrm{PK}^*)$, all $S \ni \mathrm{PK}^*$, all plaintexts M, and all C in the support of $\mathsf{Enc}(\mathsf{Aggregate}(S), M)$, we have

$$\mathsf{Strip}(\mathsf{Aggregate}(S), \mathrm{SK}^*, C) \approx_s \mathsf{Enc}(\mathsf{Aggregate}(S \setminus \{\mathrm{PK}^*\}), M).$$

Semantic Security. For an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ we define the advantage $\mathsf{AdvThEnc}^{\mathcal{A}}(k)$ to be:

$$\left| \Pr\left[U \setminus U^* \neq \emptyset \wedge b = b' : \begin{array}{l} (\mathrm{PK}_i, \mathrm{SK}_i) \leftarrow \mathsf{Gen}(1^k), i = 1, \ldots, n; \\ (U, U^*, M_0, M_1) \leftarrow \mathcal{A}_1(1^k, \mathrm{PK}_1, \ldots, \mathrm{PK}_n); \\ b \xleftarrow{\$} \{0,1\}; \\ C \leftarrow \mathsf{Enc}(\mathsf{Aggregate}(\{\mathrm{PK}_i \mid i \in U\}), M_b); \\ b' \leftarrow \mathcal{A}_2(C, \{\mathrm{SK}_i \mid i \in U^*\}); \end{array} \right] - \frac{1}{2} \right|$$

A threshold encryption scheme is said to be *indistinguishable against chosen plaintext attacks* (IND-CPA) if for all PPT adversaries $\mathcal{A}$, the advantage $\mathsf{AdvThEnc}^{\mathcal{A}}(k)$ is a negligible function in k.

3 Our General Protocol

Our protocols are designed using the following high-level approach, which is essentially an abstraction of that in [12].

1. We begin with a decomposition for the class of functions we are interested in, namely sparse polynomials and read-once branching programs, as described in Sections 4 and 5 respectively. We show that our decompositions are in fact minimal, proving that our protocols are optimally private for these classes of functions.
2. We construct a semi-honest protocol by combining the decomposition with a threshold homomorphic encryption scheme. (See Section 3.1.) For our constructions, the only homomorphic operations we need to support are the identity function and affine functions. In Section 6, we provide concrete instantiations from DDH, DCR and DLIN.
3. We construct a protocol that is secure against malicious parties by having the participants first encrypt their inputs and then prove consistency using suitable NIZKs. We provide a detailed treatment in the design of NIZKs, where we completely specify the witnesses used by the honest provers. (Some of these details were omitted in [12].) These results appear in the full version.

3.1 Protocol for Semi-Honest Adversaries

We consider n parties $P_1, \ldots, P_n$, with corresponding registered key pairs $\{(\mathrm{PK}_i, \mathrm{SK}_i)\}_{i \in [n]}$. Let $f_1, \ldots, f_n$ be a decomposition for f in which the parties go in order $1, \ldots, n$. Our protocol is as follows: At a high level, party i sends to the server the ciphertext C_i, which is an encryption of the value $y_i := f_i(y_{i-1}, x_i) = \tilde{f}_i(x_1, \ldots, x_i)$ under the aggregated public key $\widetilde{\mathrm{PK}}_i = \mathsf{Aggregate}(\mathrm{PK}_{i+1}, \ldots, \mathrm{PK}_{n+1})$. Ciphertext C_i is generated by applying the encryption scheme's homomorphic properties to ciphertext C_{i-1}. In more detail:

1. Party P_1 computes $C_1 \xleftarrow{\$} \mathsf{Enc}(\widetilde{\mathrm{PK}}_1, f_1(x_1))$ and sends C_1 to the server P_{n+1}.
2. For $i = 2, \ldots, n$: party P_i receives C_{i-1} from the server, and sends C_i to the server, where:
$$C_i \xleftarrow{\$} \mathsf{Strip}(\widetilde{\mathrm{PK}}_i, \mathrm{SK}_i, \mathsf{Eval}(\widetilde{\mathrm{PK}}_i, C_{i-1}, f_i(\cdot, x_i))).$$
3. Upon receiving C_n from P_n, the server P_{n+1} decrypts the ciphertext using its secret key SK_{n+1} and outputs the result.

From the properties of Eval and Strip, it is easy to see that if all players are honest, then $C_i \approx_s \mathsf{Enc}(\widetilde{\mathrm{PK}}_i, y_i)$ for all i. Correctness then follows from the fact that $f_1, \ldots, f_n$ is a correct decomposition.

Lemma 1 (Semi-honest security). *If* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Aggregate}, \mathsf{Eval}, \mathsf{Strip})$ *is a secure threshold encryption scheme (Section 2.3), then the above protocol is an optimally private protocol for decomposition* $(f_1, \ldots, f_n)$*, against semi-honest adversaries.*

4 Computing Sparse Multivariate Polynomials

In this section we instantiate our general framework to obtain a protocol for evaluating a multivariate polynomial on the parties' inputs. We begin with a simple lemma about learning the coefficients of a multivariate polynomial via oracle queries:

Lemma 2. *Let* $F \in \mathbb{Z}_N[X_1, \ldots, X_n]$ *be a known multivariate polynomial with total degree* d*, where* N *is square-free, and* $d \leq p/2$ *for every prime* p *dividing* N*. Let* M *be the number of monomials in* F*. Fix an (unknown) input to the polynomial* $(x_1, \ldots, x_n) \in (\mathbb{Z}_N)^n$ *and define:*
$$F_i(X_{i+1}, \ldots, X_n) := F(x_1, \ldots, x_i, X_{i+1}, \ldots, X_n)$$
Then, for each i*, it is possible to learn the coefficients of the polynomial* F_i *by making a polynomial number (in* M *and* $\log N$*) of queries to an oracle for* F_i*.*

Proof. Our approach for learning the coefficients of F_i is to simply query F_i on a sufficiently large number of random points (the number of points to be determined later). Then the coefficients of F_i can be viewed as unknowns in a

linear system over $\mathbb{Z}_N$, which can be solved via Gaussian elimination. We must show that the linear system uniquely determines F_i with high probability.

Fix i and recall that F is fixed and known. Let us say that a monomial m' in variables $\{X_{i+1}, \ldots, X_n\}$ is *valid* if there exists some monomial $m \in F$ (with nonzero coefficient) such that the degree of X_j is the same in both m' and m, for all $j \in \{i+1, \ldots n\}$. Since F_i is of the form $F(\hat{x}_1, \ldots, \hat{x}_i, X_{i+1}, \ldots, X_n)$, every monomial of F_i must be valid. Then we may restrict our linear system to polynomials whose monomials are all valid, by including unknowns only for the coefficients of valid monomials. Recall that there are at most M valid monomials. Now, it suffices to show that the linear system uniquely determines F_i, among polynomials that contain only valid monomials.

Let p be a prime divisor of N. Fix any polynomial $F' \neq F_i$, where F' contains only valid monomials. Then by the Schwartz-Zippel lemma, we have that F_i and F' agree on q randomly selected points (modulo p) with probability at most $(d/p)^q \leq 1/2^q$. There are at most N^M such multivariate polynomials F', and at most $\log N$ prime divisors of N, so choose $q = Mk \log N \log\log N$. Then by a union bound, we have that F_i agrees with *some* other F' on all q random points modulo *some* prime divisor with probability at most $1/2^k$. By the Chinese Remainder Theorem, the linear system over $\mathbb{Z}_N$ uniquely determines F_i with probability at least $1 - 1/2^k$.

Function decomposition. The preceding lemma suggests that, given a sparse polynomial F, we may compute its minimum-disclosure decomposition as follows:

> $f_i(\cdot, x_i)$ takes as input the list of coefficients for a polynomial $P(X_i, X_{i+1}, \ldots, X_n)$ and outputs the list of coefficients for the polynomial $P'(X_{i+1}, \ldots, X_n)$ where $P'(X_{i+1}, \ldots, X_n) := P(x_i, X_{i+1}, \ldots, X_n)$.

Specifically, f_i proceeds as follows:

1. For each monomial of P that contains a term of the form X_i^t, multiply that coefficient by x_i^t.
2. For each set of monomials whose degrees in $X_{i+1}, ..., X_n$ are identical, add the coefficients together.

This next Lemma follows directly from Lemma 2.

Lemma 3. *The decomposition described above is a minimum-disclosure decomposition.*

Secure, one-pass protocols. It is easy to see that $f_i(\cdot, x_i)$ is an affine function of its inputs. Therefore, using our general framework in the preceding section, it suffices to construct a threshold homomorphic encryption scheme that supports computing affine functions on encrypted values. Indeed, we provide such an instantiation based on DCR in the full version.

Theorem 1. *Under the DCR assumption, there is a one-pass protocol, secure against a semi-honest adversary, for evaluating any $F \in \mathbb{Z}_N[X_1, \ldots, X_n]$ with M monomials, where N is a RSA modulus and M and the total degree of F satisfy the bounds given in Lemma 2. The protocol achieves optimal privacy, its runtime is polynomial in M, n, and $\log N$, and it requires $O(M)$ exponentiations per party.*

In Section 6, with further details in the full version, we demonstrate concrete instantiations of NIZKs appropriate to ensure security against malicious adversaries. This leads to the following Theorem.

Theorem 2. *Under the DCR assumption, there is a one-pass protocol in the random-oracle model, secure against malicious adversaries, for evaluating any $F \in \mathbb{Z}_N[X_1, \ldots, X_n]$ expressed as a sum of monomials, where N is as in Lemma 2. The protocol achieves optimal privacy and it requires $O(nM|\mathcal{D}|)$ exponentiations per party (where $\mathcal{D}$ denotes the input domain for each party).*

5 Computing Branching Programs

In this section we describe our protocol for computing branching programs.

Overview. A (deterministic) branching program P is defined by a directed acyclic graph in which the nodes are labeled by input variables and every nonterminal node has two outgoing edges, labeled by 0 and 1.[11] An input naturally induces a computation path from a distinguished initial node to a terminal node, whose label determines the output. We rely on a technique of Ishai and Paskin [15] for computing branching programs (BPs) in a bottom-up manner. Let $x_1, \ldots, x_n$ be the inputs to the BP. First, without loss of generality we may make the BP *layered* (defined below), incurring at most a quadratic blow-up in its size (this blow-up may be avoided in specific cases, see [15]). In a layered BP, all nodes can be partitioned into layers $L_0, \ldots, L_n$, with the property that all nodes in layer $i \in \{1, \ldots, n\}$ correspond to input variable X_i and have outgoing edges only into layer $i-1$. (Because we work in a bottom-up manner, we label the output layer L_0, and the topmost layer L_n.) Layer 0 contains only output nodes.

Imagine evaluating a layered BP by "percolating" output labels from the end of the BP towards the start node, as follows.[12] Starting at layer L_0, we do the following: For every edge (u, v) between layer L_i and L_{i-1} that is labeled with

[11] We note that our protocols work also for more general "linear branching programs", where the edges are labeled with affine functions.

[12] We note that computing branching programs in a top-down manner may also be considered in the one-pass model. Each party simply posts an encryption of the unique active node in its layer. This leads to a minimum-disclosure decomposition if the BP does not have redundant states, which can be achieved using a variant of the Myhill-Nerode algorithm. However, this top-down approach requires the threshold encryption to support the BP's transition function as a homomorphic operation, whereas our bottom-up approach requires only re-randomizability.

the value x_i (that is, if we are at node u and X_i assumes the value x_i, we proceed to node v), copy the output label from node v to node u (there will not be a conflict by the deterministic property of the branching program). Finally, the start node in layer L_n is labeled with the output of the computation.

This process naturally lends itself to a decomposition of the branching program's functionality. Namely, the ith phase of the decomposition outputs the labels of all nodes in layer i. To show that this decomposition is minimum-disclosure, we must argue that an adversary could also learn this information by corrupting the server and parties $i+1$ through n in the ideal world. To see why, first assume that all nodes in the branching program are reachable from the start node. Then a path from the start node to some node v in layer i naturally corresponds to a set of inputs that the adversary could query to the residual function. The result of the query is the label that this process would have assigned to node v.

Definitions. We proceed with the details of our protocol:

Definition 4 (Branching program). *A* branching program *on variables $X = (X_1, \ldots, X_n)$ with input domain $\mathcal{D}$ and output range $\mathcal{R}$ is defined by a tuple $\{G = \{V, E\}, \mathcal{S}_{\mathsf{out}}, \phi_V, \phi_E\}$. V contains a single start node with in-degree 0, and a set of designated leaf nodes, $\mathcal{S}_{\mathsf{out}}$, along with any internal nodes. The function ϕ_V assigns each node in $\mathcal{S}_{\mathsf{out}}$ with an output value from $\mathcal{R}$, and every other node with a variable from X. ϕ_E is a function that labels each edge $(u, v) \in E$ with values from $\mathcal{D}$.*

Definition 5 (Read-Once, Layered BP). *In a* layered *branching program, V can be partitioned into layers $L_n, \ldots, L_1, L_0 = \mathcal{S}_{\mathsf{out}}$ such that for any node $u \in L_i$ and $v \in L_j$, with $i > j$, the length of every path from u and v is exactly $i - j$. A layered branching program is read-once if every node in layer i is labeled with variable X_i (possibly after re-naming the variables).*

Informally, we can think of every node in layer i as having the same height, and the same variable assignment. Looking ahead, layer i will coincide with the input variable of player P_i. We note that any branching program can be turned into a layered branching program with at most a quadratic blowup in the size of V. For simplicity, we will assume that our branching programs are already read-once, layered branching programs.

Function decomposition. Let $F : \mathcal{D}^n \to \mathcal{R}$ denote the function on $X = (X_1, \ldots, X_n)$ described by a read-once, layered branching program BP. Let $s_i = |\{v \in L_i\}|$ denote the size of layer i in BP. We assume some (arbitrary) ordering on the nodes in each layer: let $(v_1, \ldots, v_{s_i})$ be the ordered nodes of layer i, and $(u_1, \ldots, u_{s_{i-1}})$ the ordered nodes in layer $i-1$. We define $f_i : \mathcal{R}^{s_{i-1}} \times \{x_i\} \to \mathcal{R}^{s_i}$ as follows. Let $\mathsf{in}_j \in \mathcal{R}$ denote the jth input to f_i, and $\mathsf{out}_k \in \mathcal{R}$ denote the kth output. Then $\mathsf{out}_k = \mathsf{in}_j$ if and only if $(v_k, u_j) \in E$, and $\phi_E(v_k, u_j) = x_i$.

Intuitively, this decomposition percolates the output "up" the graph, stripping off layers as it goes. For example, $f_1(\phi_V(\mathcal{S}_{\mathsf{out}}), x_1)$ fixes the variable $X_1 = x_1$ in layer 1, and percolates the resulting output values from layer 0 up to each node in layer 1. The output nodes in $\mathcal{S}_{\mathsf{out}}$ now become irrelevant to the computation. Similarly, $\tilde{f}_i = f_i(\cdots f_2(f_1(\phi_V(\mathcal{S}_{\mathsf{out}}), x_1), x_2) \cdots, x_i)$ strips off layers 0 through $i-1$, labeling all the nodes in layer i with the correct output, and making all layers $j < i$ irrelevant. More specifically, consider two nodes $u_j \in L_i$ and $v_k \in \mathcal{S}_{\mathsf{out}}$. If there exists some path $p = (e_i, \ldots, e_1)$ from u_j to v_k such that $(\phi_E(e_i), \ldots, \phi_E(e_1)) = x_i, \ldots, x_1$, then $\tilde{f}_i(x_1, \ldots, x_i)$ assigns $\phi_V(v_k)$ to node u_j.

Lemma 4. *The decomposition of F described above is a minimum-disclosure decomposition.*

Proof. We must show that for every $i \in [n]$, there exists a simulator $\mathcal{S}^{g_i^{\mathbf{x}}(\cdot)}(m, n, i)$, that outputs $\tilde{f}_i(x_1, \ldots, x_i)$. Recall that the output of $\tilde{f}_i$ contains $s_i = |\{v \in L_i\}|$ values, $\mathsf{out}_1, \ldots \mathsf{out}_{s_i} \in \mathcal{R}$. To compute the value of out_j, the simulator takes the jth node u_j in layer L_i and runs a breadth-first-search on G to find a path from the start node to u_j. Let $x_n, \ldots, x_{i+1}$ denote the input assignments associated with the edges along this path (according to ϕ_E). $\mathcal{S}$ queries his oracle and sets $\mathsf{out}_j = g_i^{\mathbf{x}}(x_{i+1}, \ldots, x_n)$.

Secure, one-pass protocols. To obtain a secure protocol using our framework in Section 2, we need to specify the homomorphic operation required by party P_i. It is easy to verify that we only need to re-randomize ciphertexts. By our conventions for homomorphic encryption (Section 2.3), re-randomization is performed when P_i strips his secret key's contribution from the ciphertext. We do not require any homomorphic operations beyond this. A formal description of the protocol is in Figure 4.

Theorem 3. *Assuming an encryption scheme satisfying the conditions of Section 2.3 w.r.t. the identity function, the protocol in Figure 4 is a one-pass protocol, secure against a semi-honest adversary, for evaluating any read-once, layered branching program. The protocol achieves optimal privacy. For branching programs of width w, the runtime is polynomial in w and n, and it requires $O(w)$ exponentiations per party.*

In Section 6 we provide instantiations of the NIZKs that are necessary to make this protocol secure against malicious adversaries. This gives us the following theorem as well.

Theorem 4. *Assuming an encryption scheme satisfying the conditions of Section 2.3 w.r.t. the identity function, and that the NIZK schemes mentioned above are secure, there is a one-pass protocol, secure against a malicious adversary, for evaluating any read-once, layered branching program. The protocol achieves optimal privacy. For branching programs of width w and output domain $\mathcal{D}$, the runtime is polynomial in w, n and $|\mathcal{D}|$, and it requires $O(nw|\mathcal{D}|)$ exponentiations per party.*

Branching Programs

Inputs: Player P_i holds input $x_i \in \{0,1\}$. Each also has a full description of the branching program, $BP = \{G = \{V, E\}, \mathcal{S}_{\mathsf{out}}, \phi_V, \phi_E\}$ Let $L_i = \{v_1, \ldots, v_{s_i}\}$ denote the nodes in layer i.

Protocol:

Player P_1 begins the protocol. For each $v_j \in L_1$,

- P_1 finds $u \in \mathcal{S}_{\mathsf{out}}$ such that $(u, v_j) \in E$ and $\phi_E(u, v_j) = x_1$.
- He computes $\psi_j = \mathsf{Enc}(\widetilde{\mathrm{PK}}_2, \phi_V(u))$.

P_1 sends $C_1 := (\psi_1, \ldots, \psi_{s_1})$ to the server.

For $i = 2 \ldots n$:

- Party P_i receives ciphertexts $C_{i-1} = (\psi_1, \ldots, \psi_{s_{i-1}})$ from the server.
- For every $v_j \in L_i$,
  - P_i finds $u_k \in L_{i-1}$ such that $(u_k, v_j) \in E$ and $\phi_E(u_k, v_j) = x_i$. We let ψ_k denote the ciphertext corresponding to u_k.
  - P_i sets $\psi'_j = \mathsf{Strip}(\widetilde{\mathrm{PK}}_i, \mathrm{SK}_i, \psi_k)$.
- P_i sends $C_i := (\psi'_1, \ldots, \psi'_{s_i})$ to the server.

Output: Let C_n be the (single) ciphertext sent from P_n to the server. The server computes and outputs $\mathsf{Dec}(\mathrm{SK}_{n+1}, C_n)$.

Fig. 4. A protocol secure for computing branching program BP.

6 Realizing the Required Encryption & NIZK Schemes

In the full version, we present three threshold homomorphic encryption schemes. Two are based on the DDH and DLIN assumptions, respectively, and support homomorphic evaluation of the identity function (i.e., re-randomization). The third is based on the DCR assumption, and supports homomorphic evaluation of affine functions over $\mathbb{Z}_N$. We rely on the first two schemes for branching programs and the last for sparse polynomials. The full details of our malicious-secure protocol are given in the full version. There we also describe concrete and efficient NIZK proofs, consistent with our instantiations of homomorphic threshold encryption, for the statements described in the malicious-secure protocol.

In the random oracle model, it suffices to construct appropriate Σ-protocols and then apply the Fiat-Shamir technique. We additionally use techniques of Cramer et al. [6] to compose simple Σ-protocols using logical conjunction and disjunction. The main challenge then is to show how party P_i can prove that the ciphertexts C_{i-1} and C_i are consistent, in that C_i was derived from C_{i-1} according to the protocol (with the encryption scheme's Strip and Eval operations). We eventually reduce this problem to the task of proving that two ciphertexts encrypt the same value (under different aggregated public keys), for which we provide efficient Σ-protocols.

Our instantiations based on the DDH and DLIN assumptions are compatible with our protocol for evaluating branching programs. For these homomorphic threshold schemes, we describe efficient NIZK proofs in the *standard model*, using the NIZK scheme of Groth and Sahai [11].

Acknowledgements. We thank Yuval Ishai and Yehuda Lindell for helpful discussions.

References

[1] B. Barak, R. Canetti, J. B. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195, 2004.
[2] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, 2011.
[3] D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, pages 149–168, 2011.
[4] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004.
[5] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144, 2003.
[6] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.
[7] I. Damgård and T. Toft. Trading sugar beet quotas - secure multiparty computation in practice. *ERCIM News*, 2008(73), 2008.
[8] C. Gentry, S. Halevi, and V. Vaikuntanathan. i-hop homomorphic encryption and rerandomizable Yao circuits. In *CRYPTO*, pages 155–172, 2010.
[9] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521830842.
[10] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
[11] J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, pages 415–432, 2008.
[12] S. Halevi, Y. Lindell, and B. Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO*, 2011.
[13] D. Harnik, Y. Ishai, and E. Kushilevitz. How many oblivious transfers are needed for secure multiparty computation? In *CRYPTO*, pages 284–302, 2007.
[14] M. H. Ibrahim, A. Kiayias, M. Yung, and H.-S. Zhou. Secure function collection with sublinear storage. In *ICALP (2)*, pages 534–545, 2009.
[15] Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *TCC*, pages 575–594, 2007.
[16] Y. Ishai, E. Kushilevitz, R. Ostrovsky, M. Prabhakaran, and A. Sahai. Efficient non-interactive secure computation. In *EUROCRYPT*, pages 406–425, 2011.
[17] L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure function evaluation with ordered binary decision diagrams. In *ACM Conference on Computer and Communications Security*, pages 410–420, 2006.
[18] T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for NC^1. In *FOCS*, pages 554–567, 1999.
[19] A. C.-C. Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167, 1986.