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.

* An extended abstract of this work appears in the proceedings of *EUROCRYPT 2013*. This is the full version.

** 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

1 Introduction

Most of the literature on secure multi-party computation assumes that all parties remain online throughout the computation. Unfortunately, this assumption 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" $f(x_1, \ldots, x_{n-t}, \cdot, \cdots, \cdot)$, 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].

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

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.

For convenience, we provide a summary of our results in Figure 1.

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

[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 poly(n) monomials.

	Section 4	Section 5
class of functions	sparse polynomials	branching programs
hardness assumption	DCR	DLIN/SXDH/DCR
fixed ordering of parties?	no	depends
semi-honest complexity	$O(M)$	$O(w)$
malicious complexity	$O(nM\|\mathcal{D}\|)$	$O(nw\|\mathcal{D}\|)$
examples	mean, variance	symmetric functions, first price auction
other		DLIN/SXDH instantiations support GS proofs

Fig. 1. Summary of our constructions. n = number of parties; w = width of branching program; M = number of monomials; $|\mathcal{D}|$ = size of input domain.

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 2). 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.

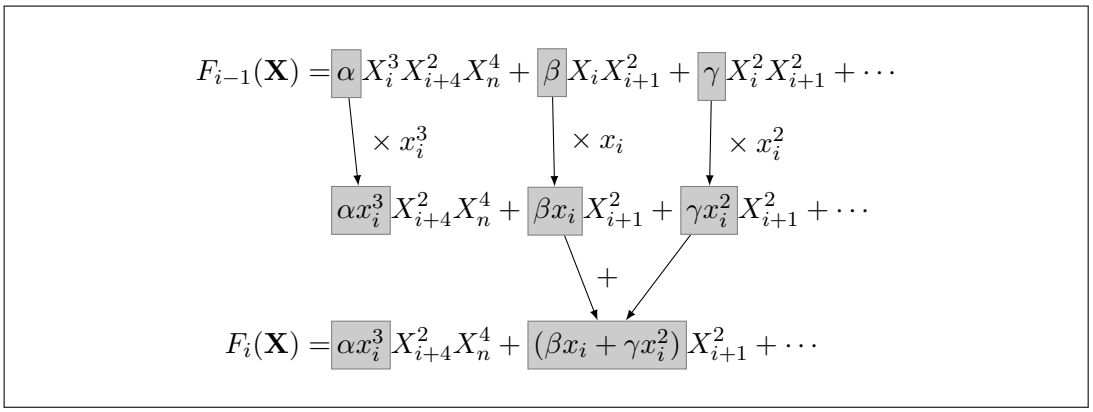

Fig. 2. Obtaining coefficients of F_i using the coefficients of F_{i-1} and the value of x_i. Shaded boxes are encrypted values. Operations on arrows are homomorphic operations possible in an additively homomorphic scheme.

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 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 3 & 4). 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.

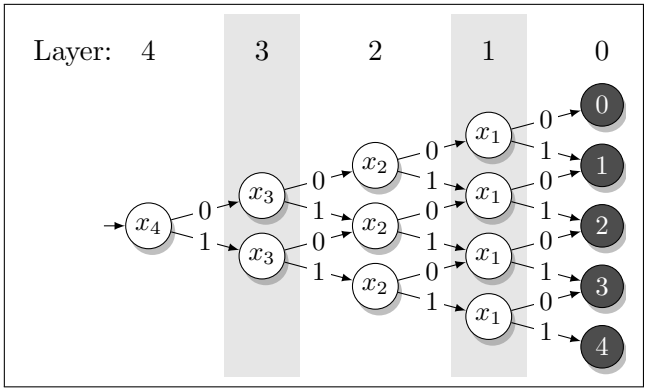

Fig. 3. A layered branching program for computing a tally among 4 parties. Output nodes are darkly shaded.

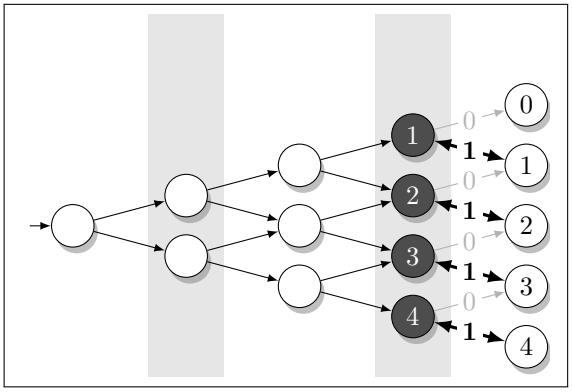

Fig. 4. How party #1 truncates the branching program, corresponding to input $x_1 = 1$.

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 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. [1].

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

[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].

[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.

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 [4, 3, 17, 15, 14].

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

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.*

[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.

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 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 $\widetilde{\text{PK}}$, 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 $\mathsf{Real}_{\bar{f},\mathcal{A}(z),\mathcal{I}}(\mathbf{x}, z, \widetilde{\text{PK}}, 1^\kappa)$.

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*

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

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 $\widetilde{\text{PK}} \leftarrow \mathsf{Aggregate}(S)$, and then encrypts normally treating $\widetilde{\text{PK}}$ 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 (SK, PK), all supported functions f, and all ciphertexts C:
$$\mathsf{Dec}(\text{SK}, \mathsf{Eval}(\text{PK}, C, f)) = f(\mathsf{Dec}(\text{SK}, C))$$
- Given an encryption C under public keys $\text{PK}_1, \ldots, \text{PK}_n$, 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 $(\text{SK}^*, \text{PK}^*)$, all $S \ni \text{PK}^*$, all plaintexts M, and all C in the support of $\mathsf{Enc}(\mathsf{Aggregate}(S), M)$, we have

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

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

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

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 Appendix B.

3.1 Protocol for Semi-Honest Adversaries

We consider n parties $P_1, \ldots, P_n$, with corresponding registered key pairs $\{(\text{PK}_i, \text{SK}_i)\}_{i \in [n]}$. 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 $\widetilde{\text{PK}}_i = \mathsf{Aggregate}(\text{PK}_{i+1}, \ldots, \text{PK}_{n+1})$. 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 $C_1 \xleftarrow{\$} \mathsf{Enc}(\widetilde{\text{PK}}_1, f_1(x_1))$ 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:

$$C_i \xleftarrow{\$} \mathsf{Strip}(\widetilde{\text{PK}}_i, \text{SK}_i, \mathsf{Eval}(\widetilde{\text{PK}}_i, C_{i-1}, f_i(\cdot, x_i))).$$

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 $C_i \approx_s \mathsf{Enc}(\widetilde{\text{PK}}_i, y_i)$ 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 Appendix C.3.

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 and Appendix D, we will demonstrate concrete instantiations of the NIZKs described in Appendix B. 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.

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

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 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

[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.

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 5.

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. 5. A protocol secure for computing branching program BP.

Theorem 3. *Assuming an encryption scheme satisfying the conditions of Section 2.3 w.r.t. the identity function, the protocol in Figure 5 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 and Appendix D, 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.*

6 Realizing the Required Encryption & NIZK Schemes

In Appendix C, 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 Appendix B. In Appendix D we 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. [7] 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] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501. Springer, 2012. ISBN 978-3-642-29010-7.

[2] B. Barak, R. Canetti, J. B. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195, 2004.

[3] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, 2011.

[4] D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, pages 149–168, 2011.

[5] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004.

[6] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, pages 126–144, 2003.

[7] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, pages 174–187, 1994.

[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.

A Additional Definitions

PKI Model. 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.

- For honest parties, these key pairs are always generated using some fixed and publicly known key generation algorithm. (In the proof of security, the simulator generates these key pairs according to the key generation algorithm.)
- Dishonest parties may choose their key pairs based on the public keys of the honest parties, with the restriction that the key pairs must constitute a valid output of the key generation algorithm. (In the proof of security, the simulator knows the private key for each dishonest party, as provided to the central authority.)

In the full version of this paper, we will consider a relaxation to the bare PKI model, where the dishonest party is not required to register a private key. The idea is to have the parties encrypt their inputs using some fixed public key, provided by the trusted set-up, instead of their individual public key.

Non-interactive Zero Knowledge. A non-interactive zero-knowledge (NIZK) proof scheme for an NP-relation R consists of algorithms GenCRS, Prove and Verify. We require the scheme to satisfy the following properties:

1. **Completeness:** For all x, w such that $R(x, w) = 1$, the following probability is overwhelming:
$$\Pr[crs \leftarrow \mathsf{GenCRS}(1^k); \mathsf{Verify}(crs; x, \mathsf{Prove}(s; x, w)) = 1]$$
2. **Adaptive Soundness:** For all polynomial-time adversaries A, the following probability is negligible:
$$\Pr[crs \leftarrow \mathsf{GenCRS}(1^k); (x, \pi) \leftarrow A(crs) : \mathsf{Verify}(crs; x, \pi) = 1 \wedge \nexists w : R(x, w) = 1]$$

3. **Zero-knowledge:** There exist algorithms SimCRS and $\mathsf{SimProof}$ such that for all polynomial-time adversaries A and all (x, w) satisfying $R(x, w) = 1$, the following two distributions are indistinguishable:

$$\begin{aligned} &\{(crs, x, \pi) \mid crs \leftarrow \mathsf{GenCRS}; \pi \leftarrow \mathsf{Prove}(crs; x, w)\} \\ \approx &\{(crs, x, \pi) \mid (crs, \tau) \leftarrow \mathsf{SimCRS}; \pi \leftarrow \mathsf{SimProof}(crs, \tau, x)\} \end{aligned}$$

B Handling Malicious Adversaries

To handle malicious adversaries, we will require each party P_i to carry out the above semi-honest protocol for generating ciphertexts C_i, and, in addition, to also perform the following operations:

1. P_i encrypts his input under his own public key, $\widehat{C}_i \xleftarrow{\$} \mathsf{Enc}(\mathrm{PK}_i, x_i)$.
2. P_i prepares a proof that $(\widehat{C}_i, C_{i-1}, C_i)$ is "well-formed", corresponding to encryptions of values x_i, y_{i-1}, y_i such that $x_i \in \mathcal{D}$ and $y_i = f_i(y_{i-1}, x_i)$. More formally, it generates a NIZK proof π_i of the following statement (using his input, secret key, and randomness as the witness):

$$\begin{aligned} \exists x_1, r, r' : C_i = \mathsf{Enc}(\widetilde{\mathrm{PK}}_1, f_1(x_1); r) \text{ and } x_1 \in \mathcal{D} \text{ and } \widehat{C}_i = \mathsf{Enc}(\mathrm{PK}_i, x_i; r') \qquad &\text{when } i = 1 \\ \exists \mathrm{SK}_i, x_i, r, r' : C_i = \mathsf{Strip}(\widetilde{\mathrm{PK}}_{i-1}, \mathrm{SK}_i, \mathsf{Eval}(\widetilde{\mathrm{PK}}_{i-1}, C_{i-1}, f_i(\cdot, x_i); r)) \text{ and } x_i \in \mathcal{D} & \\ \text{and } \widehat{C}_i = \mathsf{Enc}(\mathrm{PK}_i, x_i; r') \text{ and } \mathrm{SK}_i \text{ is consistent with } \mathrm{PK}_i \qquad &\text{when } i \geq 2 \end{aligned}$$

   where, as before, $\widetilde{\mathrm{PK}}_i = \mathsf{Aggregate}(\mathrm{PK}_{i+1}, \ldots, \mathrm{PK}_n)$.
3. P_i signs the message $(C_i, \widehat{C}_i, \pi_i)$ under his verification key VK_i. Call this signature σ_i.
4. At the start of P_i's turn, the server sends all of the previous parties' messages to party P_i. Upon receiving $\{(\widehat{C}_j, C_j, \pi_j, \sigma_j) \mid j < i\}$ from the server, party P_i verifies the following for each $j < i$:
   - σ_j is a valid signature of $(C_j, \widehat{C}_j, \pi_j)$.
   - π_j is a valid proof of the statement described above (with respect to the value C_{j-1} included in the message from the server).

   If any such condition fails, then P_i aborts; otherwise P_i sends $(\widehat{C}_i, C_i, \pi_i, \sigma_i)$ to the server.

The server also verifies these signatures and proofs from each party. If any fail to verify, then the server does not give output.

Lemma 5 (malicious security). *If* $(\mathsf{Dec}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Strip})$ *is a secure threshold encryption scheme, then the above protocol is an optimally private protocol for the decomposition* $(f_1, \ldots, f_n)$*, against malicious adversaries.*

Proof (Proof of Lemma 5). We first define the simulator and then show that it is sound. Recall that we must simulate all (and only) the messages sent from honest parties to malicious parties. We break the description of our simulator into two cases: if the server is honest, then we need to simulate the messages sent from the server to each malicious party $P_i \in \mathcal{I}$. If the server is malicious, then we need to simulate the messages sent from each honest party $P_i \notin \mathcal{I}$ to the server. We let i^* denote the index of the last honest party.

The simulator begins by playing the part of the certificate authority, collecting and distributing keys. He will create the keys on behalf of the honest players by using the honest key-generation protocol. We stress that the simulator knows all of the secret keys in the system.

Case 1: If the server is honest, we must simulate his message to each party $P_i \in \mathcal{I}$. Recall, the server sends to P_i the set of messages $\{(\widehat{C}_j, C_j, \pi_j, \sigma_j)\}_{j<i}$, where $(\widehat{C}_j, C_j, \pi_j, \sigma_j)$ is the message the server received from P_j in round j.

For each $P_i \in \mathcal{I}$, taken in order, the simulator does the following:

– *message simulation:* The simulator must create the message sent from the honest server to P_i. Specifically, for each $j < i$, the simulator must generate $(\widehat{C}_j, C_j, \pi_j, \sigma_j)$. If $P_j \in \mathcal{I}$, then the simulator simply uses the message that P_j had previously sent to the server in round j. If $P_j \notin \mathcal{I}$, and the simulator has not yet generated $(\widehat{C}_j, C_j, \pi_j, \sigma_j)$ in a previous step, then it generates these values according to the protocol, using input $x_j = 0$. (Here we use the fact that the simulator chose the secret keys of the honest party P_j.)
– *input extraction:* The simulator then receives $(\widehat{C}_i, C_i, \pi_i, \sigma_i)$ from P_i, which is intended for the server. He verifies that π_i is a valid proof, and that σ_i is a valid signature. If not, he aborts the simulation. Otherwise, he uses SK_i to decrypt $\widehat{C}_i$, thereby extracting input x_i. He sends x_i to the trusted party, and stores $(\widehat{C}_i, C_i, \pi_i, \sigma_i)$ for simulating later players.

The simulator has now extracted x_i for each $P_i \in \mathcal{I}$. He submits these values to the trusted party, and outputs the views of each corrupt P_i.

Case 2: If the server is corrupt, then we only need to simulate the view of the server. Specifically, we have to simulate messages sent from honest parties to the server. We do so as follows:

– *message simulation ($i < i^*$):* For every $i \notin \mathcal{I}$ with $i < i^*$, the simulator creates $(\widehat{C}_i, C_i, \pi_i, \sigma_i)$ honestly, using input $x_i = 0$. (Here we use the fact that the simulator chose the secret keys of the honest party P_i.) He sends this value to the server on behalf of P_i.
– *input extraction:* In round i^* the server sends a message intended for P_{i^*}. Recall, this message has the form: $\{(\widehat{C}_j, C_j, \pi_j, \sigma_j)\}_{j<i^*}$. For $j < i^*$, $P_j \in \mathcal{I}$, the simulator verifies that π_j is a valid proof, and that σ_j is a valid signature. If not, he aborts the simulation. Otherwise, he extracts input x_j from $(\widehat{C}_j, C_j, \pi_j, \sigma_j)$ using P_j's secret key. For $j > i^*$, $P_j \in I$, he sets $x_j = 0$ (these inputs are ignored by the trusted party anyway). He sends $\{x_j\}_{j\in\mathcal{I}}$ to the trusted party.
– *message simulation (i^*):* Recall that in the HLP model, the trusted party ignores the inputs from players $j > i^*$ and sends $y_{i^*} = \tilde{f}_{i^*}(x_1, \ldots, x_{i^*})$ to the simulator. When this happens, the simulator generates $C_{i^*} \leftarrow \mathsf{Enc}(\widetilde{\mathrm{PK}}_{i^*}, y_{i^*})$; $\widehat{C}_{i^*} \leftarrow \mathsf{Enc}(\mathrm{PK}_{i^*}, 0)$; π_{i^*} as a simulated NIZK proof; and σ_{i^*} as a signature on these values. The simulator sends $(\widehat{C}_{i^*}, C_{i^*}, \pi_{i^*}, \sigma_{i^*})$ to the server on behalf of P_{i^*}, then halts outputting the view of the corrupt server.

To show the validity of the simulator, we consider a sequence of hybrid interactions $H_1, \ldots, H_{n+1}$. H_{n+1} is the real interaction in which honest parties execute with their true input; H_j is the same as H_{j+1} except that if P_j is honest and $j \leq n$, then P_j runs as in the simulation described above. Then H_1 is identical to our simulation. Clearly $H_j \approx H_{j+1}$ when P_j is corrupt, so it suffices to show that $H_j \approx H_{j+1}$ when P_j is honest.

If an honest party P_j rejects any of the NIZK proofs or signatures received from the server, then $H_j \approx H_{j+1}$ trivially, so we condition on the event that these proofs & signatures are valid. By the adaptive soundness of the NIZK scheme and the correctness properties of the homomorphic encryption scheme, we have that C_{j-1} is in the support of $\mathsf{Enc}(\widetilde{\mathrm{PK}}_{j-1}, \tilde{f}_{j-1}(x_1, \ldots, x_{j-1}))$. Also by the soundness of the NIZK scheme, whenever P_ℓ is corrupt, the ciphertext $\widehat{C}_\ell$ is a valid encryption of x_ℓ under PK_ℓ. In the PKI model, the simulator has access to the secret keys of the corrupt parties. Thus the simulator can compute the value $y_{j-1} = \tilde{f}_{j-1}(x_1, \ldots, x_{j-1})$.

As before, let i^* denote the index of the last honest party. We show that $H_j \approx H_{j+1}$ in the case of $j = i^* < n + 1$ (i.e., the server is corrupt). Consider the following sequence of intermediate hybrids:

1. H'_{i^*+1} is the same as H_{i^*+1} except that π_{i^*} is generated as a simulated NIZK proof. Since the statement being proven is true, we have that $H_{i^*+1} \approx H'_{i^*+1}$.

2. H_{i^*} is the same as H'_{i^*+1} except that C_{i^*} is generated as $C_{i^*} \xleftarrow{\$} \mathsf{Enc}(\widetilde{\mathrm{PK}}_{i^*}, y_{i^*})$ rather than $C_{i^*} \xleftarrow{\$} \mathsf{Strip}(\widetilde{\mathrm{PK}}_{i^*-1}, \mathrm{SK}_{i^*}, \mathsf{Eval}(\widetilde{\mathrm{PK}}_{i^*}, C_{i^*-1}, f_{i^*}(\cdot, x_{i^*}))$. By the soundness property mentioned above, and by the correctness properties of the encryption scheme, we see that the two ways of generating C_{i^*} are statistically indistinguishable, hence $H'_{i^*+1} \approx_s H_{i^*}$.

We now consider the case where $j < i^*$. Again we consider the following sequence of intermediate hybrids:

1. $H^{(1)}_{j+1}$ is the same as H_{j+1} except that π_j is generated as a simulated NIZK proof. Since the statement being proven is true, we have that $H_{j+1} \approx H^{(1)}_{j+1}$.
2. $H^{(2)}_{j+1}$ is the same as $H^{(1)}_{j+1}$ except that C_j is generated as $C_j \xleftarrow{\$} \mathsf{Enc}(\widetilde{\mathrm{PK}}_j, f_j(y_{j-1}, x_j))$ rather than $C_j \xleftarrow{\$} \mathsf{Strip}(\widetilde{\mathrm{PK}}_{j-1}, \mathrm{SK}_j, \mathsf{Eval}(\widetilde{\mathrm{PK}}_j, C_{j-1}, f_j(\cdot, x_j))$. By a similar argument as above, we have $H^{(1)}_{j+1} \approx_s H^{(2)}_{j+1}$.
3. $H^{(3)}_{j+1}$ is the same as $H^{(2)}_{j+1}$ except that C_j is generated as $C_j \xleftarrow{\$} \mathsf{Enc}(\widetilde{\mathrm{PK}}_j, f_j(y_{j-1}, 0))$ and $\widehat{C}_j$ is generated as $\widehat{C}_j \xleftarrow{\$} \mathsf{Enc}(\mathrm{PK}_j, 0)$. Note that $\widetilde{\mathrm{PK}}_j$ includes PK_{i^*}, the public key of an honest party. Since neither SK_j nor SK_{i^*} are included in the adversary's view, we have $H^{(2)}_{j+1} \approx H^{(3)}_{j+1}$ by the (threshold) semantic security of the encryption scheme.
4. $H^{(4)}_{j+1}$ is the same as $H^{(3)}_{j+1}$ except that C_j is generated as $C_j \xleftarrow{\$} \mathsf{Strip}(\widetilde{\mathrm{PK}}_{j-1}, \mathrm{SK}_j, \mathsf{Eval}(\widetilde{\mathrm{PK}}_j, C_{j-1}, f_j(\cdot, 0))$. Similar to above, we have that $H^{(4)}_{j_1} \approx_s H^{(3)}_{j+1}$.
5. H_j is the same as $H^{(4)}_{j+1}$ except that the proof π_j is generated honestly. Note that the statement being proven is true — party P_j is indeed carrying out the protocol honestly with effective input 0. Thus, similar to above, we have $H^{(4)}_{j+1} \approx H_j$.

C Realizing Threshold Homomorphic Encryption

In this section, we present three threshold homomorphic encryption schemes, similar to those given in [12]. 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.

C.1 Instantiations from DDH

Let $\mathbb{G}$ be a group of prime order q specified using a generator g. The DDH assumption asserts that g^{ab} is pseudorandom given g, g^a, g^b where $g \xleftarrow{\$} \mathbb{G}; a, b \xleftarrow{\$} \mathbb{Z}_q$.

Key Set-Up. $\mathrm{PP} := (\mathbb{G}, q, g)$. Sample $\mathrm{SK} \xleftarrow{\$} \mathbb{Z}_q$ and output $\mathrm{PK} := g^{\mathrm{SK}}$.

Encryption. $\mathsf{Enc}(\mathrm{PK}, M)$ where $M \in \{0, 1\}$. Sample $r \xleftarrow{\$} \mathbb{Z}_q$ and output $(g^r, \mathrm{PK}^r \cdot g^M)$.

Key Aggregation. $\mathsf{Aggregate}(\mathrm{PK}_1, \ldots, \mathrm{PK}_\ell)$ outputs $\prod_{i=1}^{\ell} \mathrm{PK}_i$ (combines multiple public keys into a single public key)

Evaluation. This scheme is only compatible with our protocol for evaluating branching programs. In that protocol, we only require a re-randomization capability, and no further homomorphic operations. Re-randomization is included as a part of the Strip operation:

STRIP. On input a ciphertext (C_0, C_1), first sample $r' \xleftarrow{\$} \mathbb{Z}_q$. Output the new ciphertext:

$$(C_0 g^{r'}, C_1 C_0^{-\mathrm{SK}} \mathrm{PK}^{r'})$$

C.2 Instantiations from DLIN

Let $\mathbb{G}$ be a group of prime order q. The decision linear (DLIN) assumption is that h^{a+b} is pseudorandom given u, v, h, u^a, v^b, where $u, v, h \xleftarrow{\$} \mathbb{G}$ and $a, b \xleftarrow{\$} \mathbb{Z}_q$. We describe a threshold encryption scheme based on the *linear encryption* scheme described by Boneh, Boyen, and Shacham [5].

KEY SET-UP. $\mathrm{PP} := (\mathbb{G}, q, g)$ where g is a generator of $\mathbb{G}$. Sample $u, v, w \xleftarrow{\$} \mathbb{G}$, and $x, y \xleftarrow{\$} \mathbb{Z}_q^*$ satisfying $u^x = v^y = w$. Set $\mathrm{SK} = (x, y)$ and $\mathrm{PK} := (u, v, w)$.

KEY AGGREGATION. $\mathsf{Aggregate}(\mathrm{PK}_1, \ldots, \mathrm{PK}_\ell)$: Parse $\mathrm{PK}_i = (u_i, v_i, w_i)$ and set $W = \prod_{i=1}^{\ell} w_i$. Output $\{(u_i, v_i) \mid i \leq \ell\}$ and W.

ENCRYPTION. $\mathsf{Enc}(\mathrm{PK}, M)$ where $M \in \mathbb{G}$. Parse PK as an aggregated public key, $\mathrm{PK} = (\{(u_i, v_i) \mid i \leq n\}, W)$. Sample $a, b \xleftarrow{\$} \mathbb{Z}_q$ and output $(u_1^a, \ldots, u_n^a, v_1^b, \ldots, v_n^b, W^{a+b} m)$.

EVALUATION. This scheme is only compatible with our protocol for evaluating branching programs. In that protocol, we only require a re-randomization capability, and no further homomorphic operations. re-randomization is included as a part of the STRIP operation:

STRIP. To strip the contribution of $\mathrm{SK}_n = (x, y)$ from ciphertext $(U_1, \ldots, U_n, V_1, \ldots, V_n, C)$, sample $a', b' \xleftarrow{\$} \mathbb{Z}_q$ and output the new ciphertext:

$$(U_1^{a'}, \ldots, U_{n-1}^{a'}, V_1^{b'}, \ldots, V_{n-1}^{b'}, C \cdot (U_n^x V_n^y)^{-1} (W')^{a'+b'})$$

where $W' = \prod_{i=1}^{n-1} w_i$.

The threshold semantic security of this scheme easily reduces to the semantic security of the standard linear encryption scheme. We note that this scheme is more efficient (requiring $2n + 1$ group elements) than the generic transformation presented by HLP (requiring $3n$ elements in the case of linear encryption) [12].

C.3 Instantiations from DCR

We use the ElGamal variant of Paillier's encryption given in [6]. Fix a Blum integer $N = PQ$ for safe primes $P, Q \equiv 3 \pmod 4$ (such that $P = 2p + 1$ and $Q = 2q + 1$ for primes p, q).

KEY SET-UP. $\mathrm{PP} := (N, g)$ where $g' \xleftarrow{\$} \mathbb{Z}_{N^2}^*$ and $g := (g')^N$. Sample $\mathrm{SK} \xleftarrow{\$} [N^2/4]$ and output $\mathrm{PK} := g^{\mathrm{SK}}$.

ENCRYPTION. $\mathsf{Enc}(\mathrm{PK}, M)$ where $M \in [N]$. Sample $r \xleftarrow{\$} [N^2/4]$ and output $(g^r, \mathrm{PK}^r \cdot (1 + N)^M)$.

KEY AGGREGATION. $\mathsf{Aggregate}(\mathrm{PK}_1, \ldots, \mathrm{PK}_\ell)$ outputs $\prod_{i=1}^{\ell} \mathrm{PK}_i$ (combines multiple public keys into a single public key)

EVALUATION. Given a function $f_{a,b} : \mathbb{Z}_N \to \mathbb{Z}_N$ that maps $x \mapsto ax + b$, on input a ciphertext (C_0, C_1), output $(C_0^a, C_1^a (1 + N)^b)$. In addition, given encryptions (C_0, C_1) and (C_0', C_1') of x and x' respectively, $(C_0 \cdot C_0', C_1 \cdot C_1')$ is an encryption of $x + x'$.

STRIP. On input a ciphertext (C_0, C_1), first sample $r' \xleftarrow{\$} [N^2/4]$. Output the new ciphertext:

$$(C_0 g^{r'}, C_1 C_0^{-\mathrm{SK}} \mathrm{PK}^{r'})$$

D Instantiating Non-interactive Zero-Knowledge Proofs

When players are malicious, we use a NIZK proof to enforce correct behavior (cf. Appendix B). In this section, we show how to realize these NIZK proofs. In Section D.1, we work in the random oracle model, so it suffices to obtain Σ-protocols and then apply the Fiat-Shamir paradigm. The idea is to break down the relation we need to enforce as the conjunction and disjunction of DDH relations (or variants there-of) and then apply the techniques of Cramer et al. [7] to combine these individual relations. In Section D.2, we demonstrate how to efficiently instantiate the NIZK proofs in bilinear groups without random oracles via Groth-Sahai proofs [11].

Overview. Recall that our goal is to enforce that $(\widehat{C}_i, C_{i-1}, C_i)$ satisfies the following relation:

$$\begin{aligned}\exists \mathrm{SK}_i, x_i, r, r' : C_i = \mathsf{Strip}(\widetilde{\mathrm{PK}}_{i-1}, \mathrm{SK}_i, \mathsf{Eval}(\widetilde{\mathrm{PK}}_{i-1}, C_{i-1}, f_i(\cdot, x_i); r)) \text{ and } x_i \in \mathcal{D} \\ \text{and } \widehat{C}_i = \mathsf{Enc}(\mathrm{PK}_i, x_i; r') \text{ and } \mathrm{SK}_i \text{ is consistent with } \mathrm{PK}_i \qquad \text{for } i \geq 2\end{aligned}$$

where, as before, $\widetilde{\mathrm{PK}}_i = \mathsf{Aggregate}(\mathrm{PK}_{i+1}, \ldots, \mathrm{PK}_n)$.

Clearly, we can write this statement as the disjunction of $|\mathcal{D}|$ statements, ranging over $x_i \in \mathcal{D}$. That is, we fix $x_i \in \mathcal{D}$, and show:

$$\begin{aligned}\exists \mathrm{SK}_i, r, r' : C_i = \mathsf{Strip}(\widetilde{\mathrm{PK}}_{i-1}, \mathrm{SK}_i, \mathsf{Eval}(\widetilde{\mathrm{PK}}_{i-1}, C_{i-1}, f_i(\cdot, x_i); r)) \\ \text{and } \widehat{C}_i = \mathsf{Enc}(\mathrm{PK}_i, x_i; r') \text{ and } \mathrm{SK}_i \text{ is consistent with } \mathrm{PK}_i \qquad \text{for } i \geq 2\end{aligned}$$

We will provide a Σ-protocol for this simpler statement, and then use the techniques of Cramer et al. [7] to combine these into a single protocol for the disjunction.

In the case of branching programs, $f_i(\cdot, x_i)$ is essentially just the identity function. For polynomials, $f_i(\cdot, x_i)$ is an affine function.

D.1 NIZK in the Random Oracle Model

NIZK for Branching Programs. For simplicity, let us work with boolean domains for now and our ElGamal encryption instantiation. It suffices to show how a player can prove that he is behaving consistently with input 0 (respectively 1). Recall, player P_i sends $|L_i|$ ciphertexts to the server in this protocol, each corresponding to a node in layer L_i of the branching program. We focus for now on a single node $v \in L_i$, and the denote the corresponding ciphertext by C_i. Technically, the malicious player will have to prove the conjunction over $|L_i|$ of the statements we describe below, in order to prove that *all* $|L_i|$ ciphertexts were correctly formed. Let $u \in L_{i-1}$ be the node such that $(u, v) \in E$, and $\phi_E(u, v) = 0$. Let C_{i-1} denote the ciphertext corresponding to this node that player P_{i-1} sent to the server in the previous round. Recall that in our protocol for branching programs, $f_i(\cdot, 0)$ is simply the identity function, so we can simplify the NIZK statement to be the following.

($\widehat{C}_i$ is an encryption of 0 under PK_i) AND
(C_i and C_{i-1} are encryptions of the same value under $\widetilde{\mathrm{PK}}_i$ and $\widetilde{\mathrm{PK}}_{i-1}$ respectively)

The first statement corresponds to just proving that $\widehat{C}_i$ is a valid DDH tuple (using as witness randomness for encryption). For completeness, we provide the appropriate Σ-protocol in Figure 6. Let us now focus on the second part. We may write:

$$C_{i-1} = (u_{i-1}, \psi_{i-1}) = (g^r, \widetilde{\mathrm{PK}}_{i-1}^r \cdot M) \quad \text{and} \quad C_i = (u_i, \psi_i) = (g^{r+r_i}, \widetilde{\mathrm{PK}}_i^{r+r_i} \cdot M)$$

We stress that P_i knows (r_i, SK_i) but not r or M; here, r_i denotes the randomness for Strip. Now, observe that:

$$(u_i/u_{i-1}, \psi_i/\psi_{i-1}) = (g^{r_i}, \mathrm{PK}_i^{-r} \cdot \widetilde{\mathrm{PK}}_i^{r_i})$$

For a fixed generator g, write $\mathsf{dh}(g^a, g^b)$ to denote g^{ab}. Therefore, we can write ψ_i/ψ_{i-1} as

$$\mathsf{dh}(g^r, \mathrm{PK}_i) \cdot \mathsf{dh}(\widetilde{\mathrm{PK}}_i, g^{r_i}) = \mathsf{dh}(u_{i-1}, \mathrm{PK}_i) \cdot \mathsf{dh}(\widetilde{\mathrm{PK}}_i, u_i/u_{i-1})$$

Indeed, P_i can prove that ψ_i/ψ_{i-1} is of this form by using (SK_i, r_i) as his witness. More specifically, he must prove that the following tuple is of the form $(g^{a_1}, g^{a_2}, g^{b_1}, g^{b_2}, g^{a_1 b_1 + a_2 b_2})$:

$$(g^{\widetilde{\mathrm{SK}}_i}, u_{i-1}, u_i/u_{i-1}, g^{-\mathrm{SK}_i}, \psi_i/\psi_{i-1}) = (g^{\widetilde{\mathrm{SK}}_i}, g^r, g^{r_i}, g^{-\mathrm{SK}_i}, g^{\widetilde{\mathrm{SK}}_i r_i - \mathrm{SK}_i r}).$$

Letting $\mathbf{a} = (\widetilde{\mathrm{SK}}_i, r)$, and $\mathbf{b} = (r_i, -\mathrm{SK}_i)$, party P_i uses $\mathbf{b}$ as his witness in the Σ-protocol of Figure 7.

Proof of DDH Membership in DCR groups

Inputs: The public parameters are (N, g). The language is $\{(g, h, u, v) \in (\mathbb{Z}_{N^2}^*)^4 : \exists\, w \in [N/4] \text{ s.t.} u = g^w, v = h^w\}$. Both parties have access to the instance (g, h, u, v). The prover P also has the witness w.

1. P chooses a random $r \xleftarrow{\$} [N/4]$ and sends $(g_0, h_0) := (g^r, h^r)$ to the verifier V.
2. V chooses a random $e \xleftarrow{\$} [N^3/4]$ and sends e to P.
3. P responds with $z = r + we$. V checks that $(g^z, h^z) = (g_0 u^e, h_0 v^e)$.

Fig. 6. A Σ-protocol for proving DDH membership in DCR groups

---

NIZK for Polynomials In our protocol for computing polynomials, party P_i receives encryptions of a set of coefficients, denoted by $C_{i-1}^{(1)}, \ldots C_{i-1}^{(\ell)}$. We let $m_1, \ldots, m_\ell$ denote the corresponding plaintexts. As before, P_i creates a ciphertext C_i and proves that it is correctly formed for some input x_i. He does this by proving a disjunction over $|D|$ statements, one for each possible input value. Also as before, he may actually have to create more than one ciphertext for each possible input (depending on how many coefficients he is expected to add together after substituting his own input). We will simply focus on a single one of these ciphertexts, and simply note that he will actually have to prove that all are correctly formed for some input. For simplicity, then, we assume that the monomials corresponding to $m_1, \ldots, m_\ell$ each have the same degree in each variable other than x_i, and, consequently, all ℓ coefficients are collapsed into the single one encrypted by C_i.

Intuitively, P_i must prove the following statement, given some publicly known values $a_1, \ldots, a_\ell$, where each a_j corresponds to $x_i^{d_j}$ for some (known) d_j.

If $C_{i-1}^{(1)}, \ldots C_{i-1}^{(\ell)}$ are encryptions of $m_1, \ldots, m_\ell$ under $\widetilde{\mathrm{PK}}_{i-1}$, then
C_i is an encryption of $a_1 m_1 + \cdots + a_\ell m_\ell$ under $\widetilde{\mathrm{PK}}_i$

Note that both the verifier and the prover, independently, can compute an encryption of $a_1 m_1 + \cdots + a_\ell m_\ell$ under $\widetilde{\mathrm{PK}}_{i-1}$ from $C_{i-1}^{(1)}, \ldots C_{i-1}^{(\ell)}$. Let's denote this by $\widetilde{C}_{i-1}$. Now the statement above can be reduced to the one we saw for branching programs. Specifically, P_i simply has to prove that C_i and $\widetilde{C}_{i-1}$ are encryptions of the same value, under $\widetilde{\mathrm{PK}}_i$ and $\widetilde{\mathrm{PK}}_{i-1}$ respectively.

Proof of ℓ-DHMULT Membership

Inputs: The public parameters are $(\mathbb{G}, q, g)$. The language is $\{(g^{\mathbf{a}}, g^{\mathbf{b}}, u) \in \mathbb{G}^\ell \times \mathbb{G}^\ell \times \mathbb{G} : u = g^{\mathbf{a}\cdot\mathbf{b}}\}$. Both parties have access to the instance $(g^{\mathbf{a}}, g^{\mathbf{b}}, u)$. The prover P also has the witness $\mathbf{b}$.

1. P chooses a random $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^\ell$, computes $c := g^{\mathbf{a}\cdot\mathbf{r}}$ and sends $(g^{\mathbf{r}}, c)$ to the verifier V.
2. V chooses a random $e \xleftarrow{\$} \mathbb{Z}_q$ and sends e to P.
3. P responds with $\mathbf{z} := \mathbf{r} + e\mathbf{b} \in \mathbb{Z}_q^\ell$. V checks that $g^{\mathbf{z}} = g^{\mathbf{r}} \cdot (g^{\mathbf{b}})^e$ and $g^{\mathbf{a}\cdot\mathbf{z}} = c \cdot u^e$.

(where $\cdot$ in the exponent refers to dot product, and for $\mathbb{G}$-vectors denotes entry-wise product.)

Fig. 7. A Σ-protocol for proving ℓ-DHMULT membership

D.2 Instantiations via Groth-Sahai Proofs

We can remove the use of random oracles by using the Groth-Sahai non-interactive proof system [11]. This proof system is compatible with both our DDH and DLIN instantiations (used in our protocol for evaluating branching programs).

SXDH instantiation. Here, we have a pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Informally, the SXDH assumption is that DDH is hard in both $\mathbb{G}_1$ and in $\mathbb{G}_2$. We can then instantiate our protocol for branching programs using ElGamal encryptions in $\mathbb{G}_1$, where DDH holds.

We first show how to use Groth-Sahai proofs to prove the kinds of clauses needed for the NP statement. Let h be any fixed generator of $\mathbb{G}_2$. Then we have:

$$\begin{aligned}
&\mathrm{SK}_j \text{ is consistent with } \mathrm{PK}_j && \Leftrightarrow && e(\mathrm{PK}_j, h) \cdot e(g^{-1}, h^{\mathrm{SK}_j}) = 1 \\
&(\widehat{C}_0, \widehat{C}_1) = \mathsf{Enc}(\mathrm{PK}_j, m; r') && \Leftrightarrow && \begin{cases} e(\widehat{C}_0, h) \cdot e(g^{-1}, h^{r'}) = 1 \\ e(\widehat{C}_1, h) \cdot e\big((\mathrm{PK}_j)^{-1}, h^{r'}\big) \cdot e(m^{-1}, h) = 1 \end{cases} \\
&(C_0, C_1) = \mathsf{Strip}(\widetilde{\mathrm{PK}}, \mathrm{SK}_j, (X, Y); r) && \Leftrightarrow && \begin{cases} e(C_0, h) \cdot e(X^{-1}, h) \cdot e(g^{-1}, h^r) = 1 \\ e(C_1, h) \cdot e(Y^{-1}, h) \cdot e(X, h^{\mathrm{SK}_j}) \cdot e\big((\widetilde{\mathrm{PK}}_{j+1})^{-1}, h^r\big) = 1 \end{cases}
\end{aligned}$$

In the terminology of Groth-Sahai, the values SK_j, r, r' are *variables* (witness values), and all other values appearing in these pairing-product equations are either public parts of the NP statement, or can be easily computed from them. Because the right-hand side of each equation is the identity element $1 \in \mathbb{G}_T$, the Groth-Sahai scheme can prove them in zero-knowledge (not just witness hiding).

For notational simplicity here, we consider a boolean input domain. Let Φ_b denote the NP statement that the prover behaved consistently with input b. Thus it suffices to show how to prove the statement $\Phi_0 \vee \Phi_1$.

Groth-Sahai proofs natively support proving conjunctions of pairing-product equations, possibly reusing variables. Thus Φ_0 (resp. Φ_1) can be expressed as a conjunction of the pairing-product equations given above:

$$\Phi_0 \iff (\exists \mathbf{r}) : \bigwedge_k \left(\prod_\ell e(X_{k\ell}, h^{r_{k\ell}}) = 1 \right)$$

where each $X_{k\ell}$ is a public constant, and each $r_{k\ell}$ is either a public constant 1, or refers to one of the variables of $\mathbf{r}$. For each variable $r_{k\ell}$, let $\widehat{r}_{k\ell}$ denote a new variable. Then define $\widehat{\Phi}_0(b)$ to be the

following formula:

$$\widehat{\Phi}(b) \iff (\exists \mathbf{r}, \widehat{\mathbf{r}}) : \left(\bigwedge_k \left[\prod_\ell e(X_{k\ell}, h^{\widehat{r}_{k\ell}}) = 1 \right] \right) \wedge \left(\bigwedge_{k,\ell} e(g^b, h^{r_{k\ell}}) \cdot e(g^{-1}, h^{\widehat{r}_{k\ell}}) = 1 \right)$$

This formula replaces each $r_{k\ell}$ with $\widehat{r}_{k\ell}$ in Φ_0, and further enforces that $\widehat{r}_{k\ell} = b \cdot r_{k\ell}$. Thus, when $b = 0$, the statement $\widehat{\Phi}_0(b)$ is trivially true (and satisfied by $\widehat{\mathbf{r}} = \mathbf{0}$ and any assignment to $\mathbf{r}$). When $b = 1$, the statement $\widehat{\Phi}_0(b)$ is logically equivalent to Φ_0. Thus, our approach is to prove the following conjunction, which is logically equivalent to the disjunction $\Phi_0 \vee \Phi_1$:

$$(\exists b_0, b_1) : (b_0 \cdot b_1 = 0) \wedge (b_0 + b_1 = 1) \wedge \widehat{\Phi}_0(b_0) \wedge \widehat{\Phi}_1(b_1) \tag{D.1}$$

Intuitively, if the prover satisfies Φ_1, he can set $b_0 = 0$, $b_1 = 1$, choose any witness for $\widehat{\Phi}_0(b_0)$ and use his witness $\widehat{\Phi}_1(b_1)$. We can prove the new clauses introduced in equation (D.1) via the following pairing-product equations:

$$\begin{aligned} b_0 \cdot b_1 = 0 \quad &\Leftrightarrow \quad e(g^{b_0}, h^{b_1}) = 1 \\ b_0 + b_1 = 1 \quad &\Leftrightarrow \quad e(g^{-1}, h) \cdot e(g, h^{b_0}) \cdot e(g, h^{b_1}) = 1 \end{aligned}$$

As before, all pairing-product equations have right-hand side $1 \in \mathbb{G}_T$, so can be proven in zero-knowledge.

DLIN instantiation. Consider an instantiation of our protocol for branching programs in which we use our DLIN threshold encryption in $\mathbb{G}_1$. We use the same basic approach as above, and it suffices to show the following pairing-product equations. Below, $\text{SK}_j = (x, y)$; $\text{PK}_j = (u, v, w)$; $\widetilde{\text{PK}} = (\{(u_i, v_i)\}_{i \le j}, W)$; and $W' = \prod_{i<j} w_i$:

$$\text{SK}_j \text{ is consistent with } \text{PK}_j \quad \Leftrightarrow \quad \begin{cases} e(w, h) \cdot e(u^{-1}, h^x) = 1 \\ e(w, h) \cdot e(v^{-1}, h^y) = 1 \end{cases}$$

$$(U, V, C) = \mathsf{Enc}(\text{PK}_j, m; (a, b)) \quad \Leftrightarrow \quad \begin{cases} e(U, h) \cdot e(u^{-1}, h^a) = 1 \\ e(V, h) \cdot e(v^{-1}, h^b) = 1 \\ e(C, h) \cdot e(w^{-1}, h^a) \cdot e(w^{-1}, h^b) \cdot e(m^{-1}, h) = 1 \end{cases}$$

$$\begin{aligned} &(\overrightarrow{U'}, \overrightarrow{V'}, C') \\ &= \mathsf{Strip}(\widetilde{\text{PK}}, \text{SK}_j, (\overrightarrow{U}, \overrightarrow{V}, C); (a, b)) \end{aligned} \quad \Leftrightarrow \quad \begin{cases} e(U'_i, h) \cdot e(U_i^{-1}, h) \cdot e(u_i^{-1}, h^a) = 1 \quad (\forall i < j) \\ e(V'_i, h) \cdot e(V_i^{-1}, h) \cdot e(v_i^{-1}, h^b) = 1 \quad (\forall i < j) \\ e(C', h) \cdot e(C^{-1}, h) \cdot e(U_j, h^x) \cdot e(V_j, h^y) \\ \qquad \cdot e((W')^{-1}, h^a) \cdot e((W')^{-1}, h^b) = 1 \end{cases}$$