Private Function Evaluation with Linear Complexity

Jonathan Katz*   Lior Malka†

Abstract

We consider the problem of *private function evaluation* (PFE) in the two-party setting. Here, informally, one party holds an input x while the other holds (a circuit describing) a function f; the goal is for one (or both) of the parties to learn $f(x)$ while revealing nothing more to either party. In contrast to the usual setting of secure computation — where the function being computed is known to both parties — PFE is useful in settings where the function (i.e., algorithm) itself must remain secret, e.g., because it is proprietary or classified.

It is known that PFE can be reduced to standard secure computation by having the parties evaluate a *universal circuit*, and indeed this is the approach taken in most prior work. Using a universal circuit, however, introduces additional overhead and results in a more complex implementation. We show here a completely new technique for PFE that avoids universal circuits, and results in constant-round protocols with communication/computational complexity *linear* in the size of the circuit computing f. To the best of our knowledge, these are the first protocols for PFE with linear complexity (without using fully homomorphic encryption), regardless of the number of rounds, and even restricted to semi-honest adversaries.

*Dept. of Computer Science, University of Maryland. Email: `jkatz@cs.umd.edu`. This work was supported in part by DARPA.

†Intel. Work done while at the University of Maryland. Email: `lior34@gmail.com`

1 Introduction

In the setting of two-party *private function evaluation* (PFE), a party P_1 holds an input x while another party P_2 holds (a circuit C_f describing) a function f; the goal is for one (or both) of the parties to learn the result $f(x)$ while not revealing to either party any information beyond this. (The parties do agree in advance on the *size* of the circuit being computed, as well as the input/output length. See Section 2.1 for further discussion.) PFE is useful when the function being computed must remain private, say because the function is classified, because revealing the function would lead to security vulnerabilities, or because the *implementation* of the function (e.g., the circuit C_f itself) is proprietary even if the function f is known [29, 6, 8, 9, 10, 11, 12, 15, 5, 28, 26, 3].

PFE stands in contrast to the standard setting of secure two-party computation [32, 13], where the parties hold inputs x and y, respectively, and wish to compute the result $f(x, y)$ for some *mutually known* function f using an agreed-upon circuit C_f for computing f. On the other hand, it is well known that the problem of PFE can be reduced to the problem of secure computation through the use of *universal circuits*. In more detail, let U_n be some (fixed) universal circuit such that $U_n(x, C) = C(x)$ for every circuit C having at most n gates. (We implicitly assume here some fixed representation for circuits.) Then if $\mathcal{C}$ is the class of circuits having at most n gates, PFE for this class is solved by having the parties run a (standard) secure computation of U_n.

There are, however, some drawbacks to using universal circuits to implement PFE. First is the resulting complexity: although PFE using universal circuits has been implemented [30], it is fair to say that it is more challenging, tedious, and error-prone to write code involving universal circuits than it is to implement secure computation "directly" using Yao's garbled circuit approach (as done, e.g., in [23, 22, 27, 16]). Using universal circuits also impacts efficiency. Valiant [31] showed a construction of a universal circuit achieving (optimal) $|U_n| = O(n \log n)$; the construction is complex, however, and the constant terms (as well as the low-order terms) are significant. Recently, Kolesnikov and Schneider [19, 30] gave a simpler construction of universal circuits: they obtain the worse asymptotic bound $|U_n| = O(n \log^2 n)$, but their techniques yield smaller universal circuits than Valiant's construction for "reasonable" values of n. (The exact improvement depends also on the number of inputs and outputs. We refer the reader to their work for a detailed comparison.)

The overhead from universal circuits is not just of theoretical concern. As secure two-party computation moves closer to practice [23, 22, 27, 16], even factors that appear "small" in theory can be significant. To take an example involving "moderately sized" circuits, consider PFE for circuits having $n \approx 2^{15}$ gates. (This is roughly the size of a circuit implementing AES, as reported in [27].) A universal circuit for this value of n contains more than $285 \cdot n$ gates [31]. (The universal circuit obtained using the techniques of [19] is even larger.) As reported by Pinkas et al. last year [27], semi-honest secure computation of a circuit of size $n \approx 2^{15}$ takes 7–14 seconds (depending on the optimizations used and assumptions one is willing to make). Making the rough approximation that the time scales linearly with the number of gates, running semi-honest PFE using universal circuits for this value of n — i.e., running a secure computation of a (universal) circuit of size $285 \cdot n$ — would take somewhere between 30 minutes to 1 hour. Indeed, the implementation of PFE by Kolesnikov and Schneider [19, 30] can handle circuits of only a few thousand gates [26].

1.1 Contributions of our Work

We measure complexity of a PFE protocol in terms of the number of gates n of the circuit being computed, and do not take into account factors polynomial in the security parameter. In this work we show what are, to the best of our knowledge, the first PFE protocols for general circuits with *linear* complexity without relying on fully homomorphic public-key encryption.[1] We begin by showing such a protocol in the semi-honest setting; this illustrates our core techniques and represents what we consider to be our main contribution. We also discuss extensions of our protocol that handle a malicious P_1, still with linear complexity. (While it is possible to extend our protocol to also achieve security against a malicious P_2, we only know how to do so with *quadratic* complexity. In this case, however, one could run a secure computation of a universal circuit, coupled with "cut-and-choose" techniques for achieving malicious security [20], which would give a protocol of complexity $O(n \log n)$ using [31] or $O(n \log^2 n)$ using [19].) Our protocols rely on (singly) homomorphic public-key encryption and symmetric-key encryption secure against (a certain type of) related-key attacks.

In addition to the theoretical improvement, we believe our approach will yield better performance in practice for PFE on circuits of moderate size. Specifically, although our protocol uses $O(n)$ public-key operations — in contrast to universal-circuit-based approaches that use $O(n \log n)$ or $O(n \log^2 n)$ symmetric-key operations[2] — the *communication complexity* of our protocol is also linear, and network communication has been identified as having a significant impact on the overall run-time in prior experimental work on secure computation [23, 27]. Moreover, there are several ways our protocol can be improved (e.g., using elliptic-curve cryptography along with fast algorithms for performing multiple fixed-base exponentiations) to reduce its computational cost.

1.2 Overview of our Techniques

Our main technical contribution, as noted above, is our idea for achieving PFE with linear complexity in the semi-honest setting; we describe this here. Our description is fairly detailed and we will refer to this description in the formal description of our protocol later; nevertheless, it should also be possible to skim this section so as to obtain the main ideas. At a very high level, our technique is to have P_1 and P_2 use (singly) homomorphic encryption so as to allow P_2 to connect gates together in a manner that is oblivious to P_1, while still enabling P_1 to prepare a garbled circuit corresponding to the function f held by P_2. This idea of having one party connect gates of the circuit together is vaguely reminiscent of the "soldering" approach taken in [25]; our setting, however, is different than theirs (in [25] it was required that both parties know the circuit being computed), as is our implementation of the "soldering" step.

Say $x \in \{0,1\}^\ell$, and assume that f outputs a single bit and that C_f is known to contain exactly n NAND gates. (Neither of these assumptions is necessary, but we avoid complications for now.) It will be useful to distinguish between *outgoing wires* and *ingoing wires* of a circuit. Outgoing wires include the ℓ input wires of the circuit, along with the n output wires from each gate of the circuit; thus, in a circuit with ℓ inputs and n gates there are exactly $\ell + n$ outgoing wires. The ingoing wires are exactly the input wires to each gate of the circuit; thus, in a circuit with n two-input gates there are exactly $2n$ ingoing wires. A circuit is defined by giving a correspondence between outgoing wires and ingoing wires; e.g., specifying that outgoing wire i (which may be an input wire or an output wire of some internal gate) connects to ingoing wires j, k, and ℓ. We stress that even though we speak of each internal gate as having only a single outgoing wire, we handle arbitrary fan-out since a single outgoing wire can be connected to several ingoing wires.

[1] It is easy to construct linear-complexity PFE from fully homomorphic encryption. But it is of theoretical interest to reduce the assumptions used, and of practical importance to avoid the overhead of fully homomorphic encryption.

[2] Note that this does not account for any oblivious transfers performed in the universal-circuit-based approaches. However the number of oblivious transfers scales linearly in the input length, not the circuit size.

The protocol begins by having P_1 generate and send a public key pk for a (singly) homomorphic encryption scheme Enc. Similar to Yao's garbled-circuit technique, P_1 then chooses $\ell + n$ pairs of random *keys* that will be assigned to each of the outgoing wires. Let s_i^b denote the key corresponding to bit b on wire i. Then P_1 sends

$$\left[\mathsf{Enc}_{pk}(s_1^0), \mathsf{Enc}_{pk}(s_1^1)\right], \ldots, \left[\mathsf{Enc}_{pk}(s_{\ell+n}^0), \mathsf{Enc}_{pk}(s_{\ell+n}^1)\right]$$

to P_2. (It will become clear from what follows that P_1 need not send the final encrypted pair $\left[\mathsf{Enc}_{pk}(s_{\ell+n}^0), \mathsf{Enc}_{pk}(s_{\ell+n}^1)\right]$. We include it above for clarity.)

P_2 will, in turn, *obliviously* define keys for each of the $2n$ ingoing wires. P_2 sorts the gates of C_f topologically, so that if the outgoing wire from some gate i connects to an ingoing wire of some other gate j then we have $i < j$. This defines a natural enumeration of all the outgoing wires in the circuit: the outgoing wires numbered from 1 to ℓ correspond to the input wires of the circuit, and outgoing wire $\ell + i$ (for $i \in \{1, \ldots, n\}$) corresponds to the output wire from gate i. Note that the output wire of the circuit corresponds to outgoing wire $\ell + n$.

For each ingoing wire of the circuit, P_2 does as follows. Say the ingoing wire is connected to outgoing wire j. Then P_2 chooses random a, b and defines the (encrypted) keys for this ingoing wire to be

$$\left[\mathsf{Enc}_{pk}(a \cdot s_j^0 + b),\ \mathsf{Enc}_{pk}(a \cdot s_j^1 + b)\right],$$

where the above is computed using the homomorphic properties of the encryption scheme. (In the above, the ciphertexts are re-randomized in the usual way.) Two observations are in order: first, the (unencrypted) keys $(r^0, r^1) \stackrel{\text{def}}{=} \left(a \cdot s_j^0 + b,\ a \cdot s_j^1 + b\right)$ are random and independent of j. Second, given s_j^b it is possible for P_2 to compute r^b (using a, b); without s_j^{1-b}, however, P_2 learns no information about r^{1-b}. (Recall we are in the semi-honest setting, so a, b are chosen at random.)

Expanding upon the above, say gate i of the circuit has its left ingoing wire connected to outgoing wire j and right ingoing wire connected to outgoing wire k. (As always, the outgoing wire from this gate is numbered $\ell + i$.) Then P_2 defines the encrypted "garbled gate"

$$\mathsf{encGG}_i = \begin{pmatrix} \left[\mathsf{Enc}_{pk}(a_i \cdot s_j^0 + b_i),\ \mathsf{Enc}_{pk}(a_i \cdot s_j^1 + b_i)\right] \\ \left[\mathsf{Enc}_{pk}(a_i' \cdot s_k^0 + b_i'),\ \mathsf{Enc}_{pk}(a_i' \cdot s_k^1 + b_i')\right] \\ \left[\mathsf{Enc}_{pk}(s_{\ell+i}^0),\ \mathsf{Enc}_{pk}(s_{\ell+i}^1)\right] \end{pmatrix},$$

where a_i, b_i, a_i', b_i' are chosen uniformly at random. Finally, P_2 sends

$$\mathsf{encGG}_1, \ldots, \mathsf{encGG}_n$$

to P_1. (In fact P_2 need not transmit the final pair $\left[\mathsf{Enc}_{pk}(s_{\ell+i}^0),\ \mathsf{Enc}_{pk}(s_{\ell+i}^1)\right]$ of each encrypted garbled gate, since P_1 already knows it. We include it above for clarity.)

Upon receiving this message, P_1 decrypts each encGG to obtain, for each gate i, the three pairs of keys $\left([L_i^0, L_i^1], [R_i^0, R_i^1], [s_{\ell+i}^0, s_{\ell+i}^1]\right)$. It then prepares a garbled version GG_i of this gate in the usual way: namely, it computes the four ciphertexts

$$C'_{b,c} \leftarrow \mathsf{sEnc}_{L_i^b}\left(\mathsf{sEnc}_{R_i^c}\left(s_{\ell+i}^{\text{NAND}(b,c)}\right)\right)$$

(where sEnc denotes a symmetric-key encryption scheme), and sets GG_i to be the four ciphertexts $\left(C'_{0,0}, \ldots, C'_{1,1}\right)$ in random permuted order. P_1 sends $\mathsf{GG}_1, \ldots, \mathsf{GG}_n$ to P_2. In addition, P_1 sends the appropriate input-wire keys $s_1^{x_1}, \ldots, s_\ell^{x_\ell}$, as well as both output-wire keys $\left(s_{\ell+n}^0, s_{\ell+n}^1\right)$.

P_2 now has enough information to compute the result, using a procedure analogous (but not identical) to what is done in a standard application of Yao's garbled-circuit methodology. P_2 begins knowing a key s_i for each outgoing wire $i \in \{1, \ldots, \ell\}$. (Recall these are the input wires of the circuit that correspond to P_1's input.) Inductively, P_2 can compute a key for every outgoing wire as follows: Consider the $(\ell + i)$th outgoing wire exiting from gate i, where the left ingoing wire to this gate is connected to outgoing wire $j < i$ and the right ingoing wire to this gate is connected to outgoing wire $k < i$. Assume P_2 has already determined keys s_j, s_k for outgoing wires j, k, respectively. P_2 computes keys $L_i = a_i s_j + b_i$ and $R_i = a'_i s_k + b'_i$ for the left and right *ingoing* wires to gate i. Then P_2 tries to decrypt each of the four ciphertexts in GG_i. With overwhelming probability, only one of these decryptions will be successful; the result of this successful decryption defines the key $s_{\ell+i}$ for outgoing wire $\ell + i$. Once P_2 has determined key $s_{\ell+n}$, it can check whether this corresponds to an output of '0' or '1' using the ordered pair $\left(s^0_{\ell+n}, s^1_{\ell+n}\right)$ sent by P_1.

Further details, intuition for security of the above, proofs of security, and extensions to handle malicious behavior of P_1 are described in the sections that follow. A more efficient variant of the above protocol is described in Section 3.3.

1.3 Other Related Work

To the best of our knowledge, only two prior approaches have been suggested for efficient PFE for general functions: either using secure computation of universal circuits [19, 30, 28] or using fully homomorphic encryption. Several works, however, have explored weaker variants of this problem. Paus et al. [26] consider *semi-private* function evaluation where the circuit *topology* (i.e., the connections between gates) is assumed to be known to both parties, but the boolean function computed by each gate can be hidden. Here we treat the more difficult case where *everything* about the circuit (except an upper bound on its size and the number of inputs/outputs) remains hidden. Another direction has been to consider PFE for more limited classes of functions: e.g., functions defined by low-depth circuits [29, 4], branching programs [15, 3], or polynomials [9, 24]. In this work we handle functions defined by arbitrary (polynomial-size) circuits.

2 Definitions

We denote the security parameter by k. A *distribution ensemble* $X = \{X(1^k, a)\}_{k\in\mathbb{N},\, a\in\mathcal{D}}$ is an infinite sequence of random variables indexed by $k \in \mathbb{N}$ and $a \in \mathcal{D}$. Ensembles $X = \{X(1^k, a)\}_{k\in\mathbb{N},\, a\in\mathcal{D}}$ and $Y = \{Y(1^k, a)\}_{k\in\mathbb{N},\, a\in\mathcal{D}}$ are *computationally indistinguishable*, denoted $X \overset{c}{\equiv} Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every k and every $a \in \mathcal{D}$

$$\Big|\Pr[D(X(1^k, a)) = 1] - \Pr[D(Y(1^k, a)) = 1]\Big| \le \mu(k).$$

2.1 Private Function Evaluation

Our definitions of security are standard, but we include them here for completeness. For simplicity, we treat the case where P_1 holds some value $x \in \{0,1\}^\ell$ as input while P_2 holds a circuit C_f computing some deterministic function f; the goal of the protocol is for P_2 to learn $f(x)$. The definitions we provide here, as well as our protocols, extend rather easily to handle, e.g., additional input provided by P_2 (this can simply be incorporated into the circuit C_f), randomized functions f, or the case where P_1 is to receive output (in addition to, or instead of, P_2).

The problem of PFE is meaningless (in practice) if f (resp., C_f) is allowed to be arbitrary: in that case P_2 could take $f(x) = x$ and learn P_1's entire input! It is thus reasonable to assume that P_1 knows *something* about C_f in advance. The most general formulation of the problem is to assume that both parties fix some class $\mathcal{C}$ of circuits, and require that $C_f \in \mathcal{C}$; in that case we refer to the problem as $\mathcal{C}$-PFE. This encompasses both the case when P_1 knows some partial information about f (as in [26]), as well as the case where C_f is restricted in some way (e.g., to have low depth). In this work, we assume only that P_1 knows the input length ℓ, and upper bounds on the output length m and the number of gates n.

There are two ways one could incorporate a security parameter into the definition of the problem. The usual way, which we find less natural in our setting, is to allow the sizes of the inputs to grow and to set the security parameter equal to the input size(s). We prefer instead to treat the input domains (namely, $\{0,1\}^\ell$ and some class of circuits $\mathcal{C}$) as fixed, and to treat the security parameter k as an additional input.

A two-party protocol for $\mathcal{C}$-PFE is a protocol running in polynomial time and satisfying the following correctness requirement: if party P_1, holding input 1^k and x, and party P_2, holding input 1^k and $C_f \in \mathcal{C}$, run the protocol honestly, then (except with probability negligible in k) the output of P_2 is $C_f(x)$.

Security in the semi-honest case. In the semi-honest case we assume both parties follow the protocol honestly but may each try to learn some additional information from their (respective) view. Fix $\mathcal{C}$ and let Π be a protocol for $\mathcal{C}$-PFE. The *view* of the ith party during an execution of Π when the parties begin holding inputs x and C_f, respectively, and security parameter 1^k is denoted by $\textsc{view}_i^\Pi(1^k, x, C_f)$. The view of P_i contains P_i's input and random tape, along with the sequence of messages received from the other party P_{3-i}.

As noted in [13], when f is deterministic it suffices to consider the views of the parties in isolation without considering the joint distribution with the output. We thus have:

Definition 1 *Protocol* Π *is a* secure $\mathcal{C}$-PFE protocol for semi-honest adversaries *if there exist probabilistic polynomial-time simulators* $\mathcal{S}_1, \mathcal{S}_2$ *such that*

$$\left\{\mathcal{S}_1\left(1^k, x\right)\right\}_{k\in\mathbb{N},\, x\in\{0,1\}^\ell,\, C_f\in\mathcal{C}} \stackrel{\rm c}{\equiv} \left\{\textsc{view}_1^\Pi\left(1^k, x, C_f\right)\right\}_{k\in\mathbb{N},\, x\in\{0,1\}^\ell,\, C_f\in\mathcal{C}}$$

$$\left\{\mathcal{S}_2\left(1^k, C_f, C_f(x)\right)\right\}_{k\in\mathbb{N},\, x\in\{0,1\}^\ell,\, C_f\in\mathcal{C}} \stackrel{\rm c}{\equiv} \left\{\textsc{view}_2^\Pi\left(1^k, x, C_f\right)\right\}_{k\in\mathbb{N},\, x\in\{0,1\}^\ell,\, C_f\in\mathcal{C}}.$$

Security against malicious behavior. We define security for a malicious adversary via the usual real/ideal framework [13]. Since we only show a protocol for the case when P_1 is malicious, we tailor our definition to that case.

Let Π be a protocol for $\mathcal{C}$-PFE, and let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine corrupting P_1 and holding auxiliary input z. We let $\textsc{view}_{\mathcal{A}(z)}^\Pi(1^k, x, C_f)$ be the random variable denoting the entire view of the adversary following an execution of Π with the indicated inputs, and let $\textsc{out}_{\mathcal{A}(z)}^\Pi(1^k, x, C_f)$ be the random variable denoting the output of the honest party P_2 after this execution. Set

$$\textsc{real}_{\mathcal{A}(z)}^\Pi(1^k, x, C_f) \stackrel{\rm def}{=} \left(\textsc{view}_{\mathcal{A}(z)}^\Pi(1^k, x, C_f),\ \textsc{out}_{\mathcal{A}(z)}^\Pi(1^k, x, C_f)\right).$$

An ideal execution of $\mathcal{C}$-PFE proceeds as follows:

Inputs: P_1 and P_2 hold inputs $x \in \{0,1\}^\ell$ and $C_f \in \mathcal{C}$, respectively; the adversary $\mathcal{A}$ (who corrupts P_1) also has 1^k and z as inputs.

Send inputs to trusted party: P_2 sends C_f to the trusted party. $\mathcal{A}$ sends a value $x' \in \{0,1\}^\ell$ (if $\mathcal{A}$ sends nothing, or some $x' \notin \{0,1\}^\ell$, then some default input in $\{0,1\}^\ell$ is used instead).

Trusted party sends outputs: The trusted party computes $C_f(x)$ and sends the result to P_2. The honest P_2 outputs what it was sent by the trusted party, and $\mathcal{A}$ outputs an arbitrary (probabilistic polynomial-time computable) function of its view.

We let $\textsc{view}^{\text{PFE}}_{\mathcal{A}(z)}(1^k, x, C_f)$ denote the output of $\mathcal{A}$, and $\textsc{out}^{\text{PFE}}_{\mathcal{A}(z)}(1^k, x, C_f)$ denote the output of P_2, following an execution in the ideal model as described above. Set

$$\textsc{ideal}^{\text{PFE}}_{\mathcal{A}(z)}(1^k, x, C_f) \stackrel{\text{def}}{=} \left(\textsc{view}^{\text{PFE}}_{\mathcal{A}(z)}(1^k, x, C_f),\ \textsc{out}^{\text{PFE}}_{\mathcal{A}(z)}(1^k, x, C_f)\right).$$

Definition 2 *Protocol Π is a* secure $\mathcal{C}$-PFE protocol for a malicious P_1 *if it is a secure $\mathcal{C}$-PFE protocol for semi-honest adversaries and if, in addition, for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ corrupting P_1 in the real model there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal model such that*

$$\left\{\textsc{ideal}^{\text{PFE}}_{\mathcal{S}(z)}(1^k, x, C_f)\right\}_{k\in\mathbb{N},\, x\in\{0,1\}^\ell,\, C_f\in\mathcal{C},\, z\in\{0,1\}^*} \stackrel{\text{c}}{\equiv} \left\{\textsc{real}^{\Pi}_{\mathcal{A}(z)}(1^k, x, C_f)\right\}_{k\in\mathbb{N},\, x\in\{0,1\}^\ell,\, C_f\in\mathcal{C},\, z\in\{0,1\}^*}.$$

2.2 Tools

Our protocol uses a (singly) homomorphic public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. In the description of our protocol, we require the ability to add or multiply by a scalar: namely, given a, b and $\mathsf{Enc}_{pk}(m)$ it should be possible to compute a (random) encryption $\mathsf{Enc}_{pk}(am+b)$. We remark, however, that our protocol is more flexible; the property we require is the ability to evaluate a *pairwise-independent function* on the plaintext space. In particular, the plaintext space need not have prime order. Moreover, by adapting the protocol appropriately, the homomorphic encryption scheme we use could be instantiated using (standard) El Gamal encryption: for that encryption scheme it is possible, given a, b, and $\mathsf{Enc}_{pk}(m)$, to compute $\mathsf{Enc}_{pk}(m^a g^b)$, which defines a pairwise-independent mapping.

We also use a symmetric-key encryption scheme $(\mathsf{sEnc}, \mathsf{sDec})$ whose key space $\mathcal{K}(k)$ is, for simplicity, assumed to be equal to the plaintext space of the public-key encryption scheme being used. (This implies, in particular, that the plaintext space of the homomorphic encryption scheme must grow with k.) We impose the same requirements as in [21]: namely, that this scheme have *elusive* and *efficiently verifiable* range. In addition, we require the encryption scheme to satisfy a weak form of *related-key security* where, roughly, encryption remains secure even when performed using linearly related keys (where the linear relations are chosen at random). That is:

Definition 3 *Encryption scheme* $(\mathsf{sEnc}, \mathsf{sDec})$ *is* secure under linear related-key attacks *if the following is negligible in k for all polynomials d and all* PPT *adversaries $\mathcal{A}$:*

$$\left|\Pr\left[\begin{array}{c} s \leftarrow \mathcal{K}(k); c \leftarrow \{0,1\}; \\ a_1, b_1, \ldots, a_d, b_d \leftarrow \mathcal{K}(k) \end{array} : \mathcal{A}^{\mathsf{sEnc}^c_{a_1 s+b_1}(\cdot,\cdot), \ldots, \mathsf{sEnc}^c_{a_d s+b_d}(\cdot,\cdot)}(a_1, b_1, \ldots, a_d, b_d) = c\right] - \frac{1}{2}\right|,$$

where $\mathsf{sEnc}^c_s(m_0, m_1) \stackrel{\text{def}}{=} \mathsf{sEnc}_s(m_c)$.

We remark that a weaker definition (where $\mathcal{A}$ queries each $\mathsf{sEnc}^c_{a_i s+b_i}(\cdot,\cdot)$ only on two nonadaptively-chosen inputs) suffices for our proof. It is easy to construct an encryption scheme satisfying the above definition using a (non-programmable) random oracle, and it would be surprising if standard encryption schemes based on AES could be shown not to satisfy the above definition. Moreover, recent work of Applebaum et al. [1] can be used to construct a scheme satisfying the above based on the decisional Diffie-Hellman assumption.

3 A $\mathcal{C}$-PFE Protocol for Semi-Honest Adversaries

3.1 Description of the Protocol

We now formally define our $\mathcal{C}$-PFE protocol for semi-honest adversaries. In our description here, we assume the reader is familiar with the protocol overview provided in Section 1.2.

We assume that all circuits in $\mathcal{C}$ are composed solely of NAND gates. This is for simplicity only, and it is clear that our protocol can be easily modified to handle circuits over an arbitrary basis of 2-to-1 gates with only a constant-factor impact on the efficiency. Let n be an upper bound on the size of any circuit in $\mathcal{C}$, and let m be an upper bound on the number of outputs. By adjusting n appropriately, we may freely assume that every circuit in $\mathcal{C}$ has exactly m outputs (P_2 can always add "dummy" outputs that are fixed to some constant); that the output wires of the circuit do not connect to any other gates (this can be achieved by adding at most m gates to the circuit); and that every circuit in $\mathcal{C}$ contains exactly n gates (P_2 can add "dummy" gates whose output wires are connected to nothing). We make all these assumptions in what follows.

Recall from Section 1.2 that we distinguish between *outgoing wires* and *ingoing wires* of C_f. (Recall also that although each gate has only a single outgoing wire, we handle circuits with arbitrary fan-out since a single outgoing wire can be connected to several ingoing wires.) As in Section 1.2, party P_2 sorts the gates of C_f topologically and this defines an enumeration of the $N \stackrel{\text{def}}{=} \ell + n$ outgoing wires. The outgoing wires numbered from 1 to ℓ correspond to the ℓ input wires of the circuit, and outgoing wire $\ell+i$ (for $i \in \{1,\ldots,n\}$) corresponds to the output wire from gate i. The output wires of the circuit correspond to the m outgoing wires $N-m+1,\ldots,N$.

It will be useful to define some notation before we describe the protocol. We define an algorithm encYao that prepares garbled gates as in Yao's protocol: encYao takes as input three pairs of keys and outputs four ciphertexts, and is defined as

$$\mathsf{encYao}\left([L^0,L^1],[R^0,R^1],[s^0,s^1]\right) \stackrel{\text{def}}{=} \left\{\mathsf{sEnc}_{L^b}\left(\mathsf{sEnc}_{R^c}\left(s^{\text{NAND}(b,c)}\right)\right)\right\}_{b,c\in\{0,1\}},$$

where the four ciphertexts are in random permuted order. We analogously define an algorithm decYao that takes as input two keys (for each of two ingoing wires) and a garbled gate, and outputs a key for the outgoing wire; this algorithm, given keys L, R and four ciphertexts $\{C'_0, C'_1, C'_2, C'_3\}$, computes $\mathsf{sDec}_L(\mathsf{sDec}_R(C'_i))$ for all i and outputs the unique non-$\perp$ value that is obtained. (If more than one non-$\perp$ value results, this algorithm outputs $\perp$.)

Our protocol is described in Figure 1. It is not difficult to see that correctness holds with all but negligible probability, via an argument similar to the one in [21].

In our description of the protocol we have aimed for clarity rather than efficiency, and it is clear that several improvements are possible. For one, P_2 need not include $\left[\mathsf{Enc}_{pk}(s^0_{\ell+i}),\ \mathsf{Enc}_{pk}(s^1_{\ell+i})\right]$ as part of encGG_i since P_1 already knows these values. Furthermore, P_1 need not send the encrypted

Inputs: The security parameter is k. The input of P_1 is a value $x \in \{0,1\}^\ell$, and the input of P_2 is a circuit C_f with ℓ, n, m as described in the text.

Round 1 P_1 computes $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$ and sends pk to P_2. In addition, P_1 chooses $N = \ell + n$ pairs of random keys s_i^0, s_i^1 for $i \in \{1, \ldots, N\}$. It then sends to P_2 the ciphertexts

$$\left[\mathsf{Enc}_{pk}(s_1^0), \mathsf{Enc}_{pk}(s_1^1)\right], \ldots, \left[\mathsf{Enc}_{pk}(s_N^0), \mathsf{Enc}_{pk}(s_N^1)\right].$$

Round 2 For each gate $i \in \{1, \ldots, n\}$ of C_f, with left ingoing wire connected to outgoing wire j and right ingoing wire connected to outgoing wire k (as always, the outgoing wire from gate i is numbered $\ell + i$), P_2 chooses a_i, b_i, a_i', b_i' uniformly and computes

$$\mathsf{encGG}_i = \begin{pmatrix} \left[\mathsf{Enc}_{pk}(a_i \cdot s_j^0 + b_i),\ \mathsf{Enc}_{pk}(a_i \cdot s_j^1 + b_i)\right] \\ \left[\mathsf{Enc}_{pk}(a_i' \cdot s_k^0 + b_i'),\ \mathsf{Enc}_{pk}(a_i' \cdot s_k^1 + b_i')\right] \\ \left[\mathsf{Enc}_{pk}(s_{\ell+i}^0),\ \mathsf{Enc}_{pk}(s_{\ell+i}^1)\right] \end{pmatrix},$$

using the homomorphic properties of Enc. (In the above, it is required that each ciphertext is re-randomized in the usual way.) Then P_2 sends $\mathsf{encGG}_1, \ldots, \mathsf{encGG}_n$ to P_1.

Round 3 For $i \in \{1, \ldots, n\}$, party P_1 decrypts encGG_i using sk to obtain the three pairs of keys $\mathsf{keys}_i \stackrel{\text{def}}{=} \left([L_i^0, L_i^1], [R_i^0, R_i^1], [s_{\ell+i}^0, s_{\ell+i}^1]\right)$. It then computes $\mathsf{GG}_i \leftarrow \mathsf{encYao}(\mathsf{keys}_i)$, and sends $\mathsf{GG}_1, \ldots, \mathsf{GG}_n$ to P_2. Finally, P_1 sends

$$\mathsf{input\text{-}wires}\colon\ s_1^{x_1}, \ldots, s_\ell^{x_\ell} \quad \text{and} \quad \mathsf{output\text{-}wires}\colon \left(s_{N-m+1}^0, s_{N-m+1}^1\right), \ldots, \left(s_N^0, s_N^1\right).$$

Output computation Say P_1 sent input-wires: $s_1, \ldots, s_\ell$ to P_2 in the previous round. Then for all $i \in \{\ell+1, \ldots, \ell+n\}$, party P_2 does: If the left ingoing wire of gate i is connected to outgoing wire $j < i$ and the right ingoing wire of gate i is connected to outgoing wire $k < i$, then (1) compute $L_i = a_i s_j + b_i$ and $R_i = a_i' s_k + b_i'$, and then (2) set $s_i = \mathsf{decYao}(L_i, R_i, \mathsf{GG}_i)$.

Once P_2 has computed $s_1, \ldots, s_{\ell+n}$, it sets the jth output bit o_j (for $j \in \{N-m+1, \ldots, N\}$) to be the (unique) bit for which $s_j = s_j^{o_j}$.

Figure 1: A $\mathcal{C}$-PFE protocol for semi-honest adversaries.

values $\left[\mathsf{Enc}_{pk}(s_{N-m+1}^0), \mathsf{Enc}_{pk}(s_{N-m+1}^1)\right], \ldots, \left[\mathsf{Enc}_{pk}(s_N^0), \mathsf{Enc}_{pk}(s_N^1)\right]$ in round 1 (since these outgoing wires do not connect to any ingoing wires). Moreover, P_1 can simply set $s_{N-m+1}^0 = \cdots = s_N^0 = 0$ and $s_{N-m+1}^1 = \cdots = s_N^1 = 1$ (and then there is no need to send the output-wires message in the third round); that is, for the gates whose outgoing wires are the output of the circuit, P_1 can encrypt the wire value itself rather than encrypting a key that encodes the wire value.

Security against a semi-honest P_1 is easy to see. In fact, security in that case holds in a *statistical* sense. Indeed, with all but negligible probability it holds that $s_i^0 \neq s_i^1$ for all $i \in \{1, \ldots, N\}$. Assuming this to be the case, the top two rows of each encGG_i sent by P_2 to P_1 in round 2 consist only of (random) encryptions of the four independent, uniform values

$$a_i \cdot s_j^0 + b_i, \quad a_i \cdot s_j^1 + b_i, \quad a_i' \cdot s_k^0 + b_i', \quad a_i' \cdot s_k^1 + b_i'.$$

In particular, these values are independent of the interconnections between gates of C_f, and thus the view of P_1 is independent of the circuit held by P_2.

Security against a semi-honest P_2 holds *computationally*, assuming semantic security of the homomorphic encryption scheme and security against linear related-key attacks for the symmetric-key encryption scheme. Roughly, the initial encryptions sent to P_2 in round 1 do not reveal anything about the values s_i^0, s_i^1 that P_1 assigns to each outgoing wire in the circuit. Thus, the information sent to P_2 in round 3 is essentially equivalent to the information sent to P_2 in a standard application of Yao's garbled-circuit methodology, with the only difference being that here ingoing wires and outgoing wires have different keys, and P_2 must compute a key L_i on some ingoing wire by "translating" a key s_j on the outgoing wire connected to this ingoing wire.

3.2 Proof of Security

Theorem 4 *Assume the homomorphic encryption scheme used is semantically secure, and the symmetric-key encryption scheme used is secure against linear related-key attacks (as in Definition 3) and has elusive and efficiently verifiable range. Then the protocol of Figure 1 is a secure $\mathcal{C}$-PFE protocol for semi-honest adversaries.*

Proof: We start by proving security for a semi-honest P_1, which is the easier case. The simulator $\mathcal{S}_1$ in this case chooses a uniform random tape for P_1; this defines pk, sk, and values $\{s_i^0, s_i^1\}_{i=1}^N$. If there exits an i for which $s_i^0 = s_i^1$ then the simulator aborts. Otherwise, for $i = 1$ to n the simulator chooses random values r_0, r_1, r_2, r_3 and computes

$$\mathsf{encGG}_i = \begin{pmatrix} [\mathsf{Enc}_{pk}(r_0),\ \mathsf{Enc}_{pk}(r_1)] \\ [\mathsf{Enc}_{pk}(r_2),\ \mathsf{Enc}_{pk}(r_3)] \\ \left[\mathsf{Enc}_{pk}(s_{\ell+i}^0),\ \mathsf{Enc}_{pk}(s_{\ell+i}^1)\right] \end{pmatrix}; \tag{1}$$

it gives $\mathsf{encGG}_1, \ldots, \mathsf{encGG}_n$ to P_1 as the 2nd-round message of P_2. This completes the simulation.

Note that the simulator aborts with only negligible probability. We claim that conditioned on the simulator's not aborting, the simulation is perfect. Indeed, when the simulator does not abort we have $s_i^0 \neq s_i^1$ for all i. In a real execution of the protocol, each encrypted garbled gate would be computed by P_2 as

$$\mathsf{encGG}_i = \begin{pmatrix} \left[\mathsf{Enc}_{pk}(a_i \cdot s_j^0 + b_i),\ \mathsf{Enc}_{pk}(a_i \cdot s_j^1 + b_i)\right] \\ \left[\mathsf{Enc}_{pk}(a_i' \cdot s_k^0 + b_i'),\ \mathsf{Enc}_{pk}(a_i' \cdot s_k^1 + b_i')\right] \\ \left[\mathsf{Enc}_{pk}(s_{\ell+i}^0),\ \mathsf{Enc}_{pk}(s_{\ell+i}^1)\right] \end{pmatrix}, \tag{2}$$

where a_i, b_i, a_i', b_i' are chosen uniformly at random and j, k depend on the specific circuit C_f held by P_2. Pairwise independence of the mapping $f_{s,b}(s) = as + b$, along with distinctness of s_j^0, s_j^1 and s_k^0, s_k^1, imply that the encrypted garbled gates computed as in (2) are distributed *identically* to encrypted garbled gates computed as in (1). Taking into account the negligible probability with which $\mathcal{S}_1$ aborts, we see that the output of $\mathcal{S}_1$ is statistically close to P_1's view in a real execution.

We next describe a simulator $\mathcal{S}_2$ who is to simulate the view of a semi-honest P_2. The simulator is given $1^k, C_f$, and $y = C_f(x)$, and does as follows. First, it chooses a uniform random tape for P_2; this defines a_i, b_i, a_i', b_i' for $i \in \{1, \ldots, n\}$. The simulator runs $\mathsf{Gen}(1^k)$ to obtain (pk, sk), and then sets $C_i^0, C_i^1 \leftarrow \mathsf{Enc}_{pk}(\mathbf{0})$ for $i \in \{1, \ldots, N\}$. (The $\mathbf{0}$ here denotes some "all-0 message" of the appropriate length in the underlying plaintext space. It does not really matter what plaintext is encrypted here, as long as its length matches the key length of the symmetric-key scheme.) $\mathcal{S}_2$ gives pk and $\left[C_1^0, C_1^1\right], \ldots, \left[C_N^0, C_N^1\right]$ to P_2 as the first-round message of the protocol.

For the third-round message of the protocol, $\mathcal{S}_2$ must simulate a garbling of each gate in C_f. This is done similarly to the approach taken in [21], modifying their simulation appropriately for our setting. For each outgoing wire $i \in \{1, \ldots, N\}$ in the circuit, $\mathcal{S}_2$ chooses two random keys s_i, s'_i. Then for each gate $i \in \{1, \ldots, n\}$ the simulator does the following. Say the left ingoing wire of gate i is connected to outgoing wire j, and the right ingoing wire of gate i is connected to outgoing wire k. (As always, the outgoing wire from gate i is numbered $\ell + i$.) Then $\mathcal{S}_2$ computes

$$L_i = a_i s_j + b_i, \quad L'_i = a_i s'_j + b_i, \quad R_i = a'_i s_k + b'_i, \quad R'_i = a'_i s'_k + b'_i, \tag{3}$$

followed by

$$\begin{aligned} &\widetilde{\mathsf{GG}}_i \\ &= \left\{ \mathsf{sEnc}_{L_i}(\mathsf{sEnc}_{R_i}(s_{\ell+i})),\ \mathsf{sEnc}_{L'_i}(\mathsf{sEnc}_{R_i}(s_{\ell+i})),\ \mathsf{sEnc}_{L_i}(\mathsf{sEnc}_{R'_i}(s_{\ell+i})),\ \mathsf{sEnc}_{L'_i}(\mathsf{sEnc}_{R'_i}(s_{\ell+i})) \right\}. \end{aligned} \tag{4}$$

(The four ciphertexts are in random permuted order. Note that the same outgoing-wire key $s_{\ell+i}$ is encrypted each time.) $\mathcal{S}_2$ gives $\widetilde{\mathsf{GG}}_1, \ldots, \widetilde{\mathsf{GG}}_n$ to P_2. It also gives $s_1, \ldots, s_\ell$ to P_2 as the input-wire keys. Finally, for $i \in \{1, \ldots, m\}$ it gives (s_{N-m+i}, s'_{N-m+i}) to P_2 if $y_i = 0$, or (s'_{N-m+i}, s_{N-m+i}) if $y_i = 1$ as the output-wire keys. This completes the simulation.

The proof that the simulated view of P_2 is computationally indistinguishable from the view of P_2 in a real execution of the protocol is very similar, at least at a high level, to the proof given in [21]. Fix some x and C_f for the remainder of the discussion. Let H denote the distribution of P_2's view in a real execution of the protocol on those inputs. Define a distribution H' as follows: in round 1, random values $\{s_i^0, s_i^1\}_{i=1}^N$ are chosen as in the real protocol, but the ciphertexts sent in the first round are all encryptions of $\mathbf{0}$. The third-round message, however, is constructed honestly using the $\{s_i^0, s_i^1\}_{i=1}^N$ chosen in round 1 along with the $\{a_i, b_i, a'_i, b'_i\}_{i=1}^n$ values chosen by the honest P_2. That is, for each gate i whose left ingoing wire is connected to outgoing wire j, and whose right ingoing wire is connected to outgoing wire k, the keys $L_i^0, L_i^1, R_i^0, R_i^1$ are computed as

$$L_i^0 = a_i s_j^0 + b_i, \quad L_i^1 = a_i s_j^1 + b_i, \quad R_i^0 = a'_i s_k^0 + b'_i, \quad R_i^1 = a'_i s_k^1 + b'_i, \tag{5}$$

and then the garbled gate

$$\mathsf{GG}_i \leftarrow \mathsf{encYao}\left([L_i^0, L_i^1], [R_i^0, R_i^1], [s_{\ell+i}^0, s_{\ell+i}^1]\right)$$

is computed. The third-round message is $\mathsf{GG}_1, \ldots, \mathsf{GG}_n$, the input-wire keys $s_1^{x_1}, \ldots, s_\ell^{x_\ell}$, and the output-wire keys $\left(s_{N-m+1}^0, s_{N-m+1}^1\right), \ldots, \left(s_N^0, s_N^1\right)$. It follows immediately from the semantic security of the public-key encryption scheme that H and H' are computationally indistinguishable.

Before continuing, we define a notion of active/inactive keys exactly as in [21]. Consider the (normal) evaluation of $C_f(x)$. If the value on a given wire i in this evaluation is the bit b, then we say the corresponding outgoing wire key s_i^b is *active* while s_i^{1-b} is *inactive*.

We now define a sequence of distributions $H_0, \ldots, H_n$. In each of these distributions random values $\{s_i^0, s_i^1\}_{i=1}^N$ and $\{a_i, b_i, a'_i, b'_i\}_{i=1}^n$ are chosen as in H, H', and the ciphertexts sent in the first round are all encryptions of $\mathbf{0}$ as in H'. In distribution H_i, the final $n-i$ garbled gates are computed "normally", as in H and H'. (Recall the gates are sorted in topological order.) The first i garbled gates, however, are computed as in (4), but encrypting the active key on the relevant outgoing wire in each case; i.e., for gate $j \in \{1, \ldots, i\}$ where key $s_{\ell+j}^b$ is active, compute

$$\begin{aligned} &\widetilde{\mathsf{GG}}_j \\ &= \left\{ \mathsf{sEnc}_{L_j^0}(\mathsf{sEnc}_{R_j^0}(s_{\ell+j}^b)),\ \mathsf{sEnc}_{L_j^1}(\mathsf{sEnc}_{R_j^0}(s_{\ell+j}^b)),\ \mathsf{sEnc}_{L_j^0}(\mathsf{sEnc}_{R_j^1}(s_{\ell+j}^b)),\ \mathsf{sEnc}_{L_j^1}(\mathsf{sEnc}_{R_j^1}(s_{\ell+j}^b)) \right\}. \end{aligned} \tag{6}$$

The third round message is thus $\widetilde{\mathsf{GG}}_1, \ldots, \widetilde{\mathsf{GG}}_i$, $\mathsf{GG}_{i+1}, \ldots, \mathsf{GG}_n$, along with the input-wire keys and output-wire keys as in H, H'.

Note that H_0 is identical to H'. Furthermore, as in [21], distribution H_n is identical to the simulated view output by $\mathcal{S}_2$. Thus, the proof is complete once we show that H_{i-1} is computationally indistinguishable from H_i for all i. Computational indistinguishability of H_{i-1} and H_i follows roughly as in [21], except that in our case we need to rely on the secure of the symmetric-key encryption scheme against linear related-key attacks. Details follow.

Fix some i^*, and say the left ingoing wire of gate i^* is connected to outgoing wire j, and the right ingoing wire of gate i^* is connected to outgoing wire j'. Furthermore, let d and d' denote the out-degree of wires j and j', respectively; i.e., d is the number of ingoing wires that are connected to outgoing wire j, and d' is the number of ingoing wires that are connected to outgoing wire j'. Consider an adversary $\mathcal{A}$ as in Definition 3, who is given two tuples $(\hat{a}_1, \hat{b}_1, \ldots, \hat{a}_d, \hat{b}_d)$ and $(\hat{a}'_1, \hat{b}'_1, \ldots, \hat{a}'_{d'}, \hat{b}'_{d'})$, and access to two sets of oracles $\{\mathsf{sEnc}^c_{\hat{a}_i s + \hat{b}_i}(\cdot,\cdot)\}_{i=1}^d$ and $\{\mathsf{sEnc}^c_{\hat{a}'_i s' + \hat{b}'_i}(\cdot,\cdot)\}_{i=1}^{d'}$. (Definition 3 only deals with the case where a single s is chosen and $\mathcal{A}$ is given access to a single set of linearly related oracles, but a standard hybrid argument shows that security against linear related-key attacks holds when two s's are chosen and $\mathcal{A}$ is given access to two sets of linearly related oracles.) Intuitively, $\mathcal{A}$ will (implicitly) use s as the inactive key on outgoing wire j, and s' as the inactive key on outgoing wire j'; also, $\mathcal{A}$ will (implicitly) use $\hat{a}_k, \hat{b}_k$ as the pairwise-independent hash for the kth ingoing wire to which outgoing wire j is connected, and will (implicitly) use $\hat{a}'_k, \hat{b}'_k$ as the pairwise-independent hash for the kth ingoing wire to which outgoing wire j' is connected. In this way $\mathcal{A}$ can generate a distribution that is identical to H_{i^*-1} when $c = 0$, and is identical to H_{i^*} when $c = 1$; computational indistinguishability of H_{i^*-1} and H_{i^*} follows.

The details are straightforward, though tedious. Fixing some i^*, let j, j' and d, d' be as above. Let b, b' be such that s_j^b and $s_{j'}^{b'}$ are the active keys on outgoing wires j and j', respectively. $\mathcal{A}$ begins by generating a first-round message consisting of ciphertexts that are all encryptions of $\mathbf{0}$. Next, $\mathcal{A}$ chooses random keys $\{s_i^0, s_i^1\}$ for $i \in \{1, \ldots, N\} \setminus \{j, j'\}$, as well as random s_j^b and $s_{j'}^{b'}$. Then, $\mathcal{A}$ generates and outputs a garbled gate for each gate g of the circuit, as described next.

Case 1: $g < i^*$. Here $\mathcal{A}$ constructs a garbled gate as in (6), encrypting only the active key on outgoing wire $\ell + g$. If both ingoing wires to g are connected to outgoing wires *other* than j or j', then $\mathcal{A}$ can easily compute the garbled gate by itself using the keys it knows. (Note in particular that this holds when g is the gate whose outgoing wire is j or j' itself, since $\mathcal{A}$ knows the active key on those wires. Recall that any such g must satisfy $g < i^*$ by our assumption that the gates are topologically ordered.) Namely, $\mathcal{A}$ chooses random a_g, b_g, a'_g, b'_g, computes $L_g^0, L_g^1, R_g^0, R_g^1$ as in (5), and computes $\widetilde{\mathsf{GG}}_g$ as in (6).

When one of the ingoing wires to g is connected to j or j', however, $\mathcal{A}$ must use one of its oracles to generate the garbled gate. By way of example, say the left ingoing wire to g is connected to outgoing wire j, and that g is the kth gate to which outgoing wire j is connected $(1 \leq k \leq d)$. Then $\mathcal{A}$ computes R_g^0, R_g^1 as before (it can do this since it knows both keys on the outgoing wire that is connected to the right ingoing wire of g), and computes $L_g^b = \hat{a}_k s_j^b + \hat{b}_k$. $\mathcal{A}$ does not explicitly compute L_g^{1-b}, but will instead define L_g^{1-b} implicitly using its access to oracle $\mathsf{sEnc}^c_{\hat{a}_k s + \hat{b}_k}(\cdot,\cdot)$. (This also implicitly defines $s_j^{1-b} = s$.) In detail, letting s^* denote the active key on the outgoing wire $\ell + g$ from gate g, we have $\mathcal{A}$ compute $C_1 \leftarrow \mathsf{sEnc}_{L_g^b}(\mathsf{sEnc}_{R_g^0}(s^*))$ and $C_2 \leftarrow \mathsf{sEnc}_{L_g^b}(\mathsf{sEnc}_{R_g^1}(s^*))$, followed by $C'_3 \leftarrow \mathsf{sEnc}_{R_g^0}(s^*)$ and $C'_4 \leftarrow \mathsf{sEnc}_{R_g^1}(s^*)$. It then uses its oracle to compute $C_3 \leftarrow \mathsf{sEnc}^c_{\hat{a}_k s+\hat{b}_k}(C'_3, C'_3)$ and $C_4 \leftarrow \mathsf{sEnc}^c_{\hat{a}_k s+\hat{b}_k}(C'_4, C'_4)$. Finally, it sets $\widetilde{\mathsf{GG}}_g = \{C_1, C_2, C_3, C_4\}$, where the ciphertexts are in random permuted order.

The other three possible sub-cases cases are handled analogously. Briefly:

1. Say the right ingoing wire of g is connected to outgoing wire j, and let k be such that g is the kth gate to which outgoing wire j is connected ($1 \leq k \leq d$). Now L^0_g, L^1_g are computed normally (that is, as in (5)), and R^b_g is computed as $R^b_g = \hat{a}_k s^b_j + \hat{b}_k$. Encryption using R^{1-b}_g, however, is done using oracle access to $\mathsf{sEnc}^c_{\hat{a}_k s+\hat{b}_k}(\cdot,\cdot)$.

2. Say the left ingoing wire of g is connected to outgoing wire j'. Let k be such that g is the kth gate to which outgoing wire j' is connected ($1 \leq k \leq d'$). Then R^0_g, R^1_g are computed as in (5), $L^{b'}_g$ is computed as $L^{b'}_g = \hat{a}'_k s^{b'}_{j'} + \hat{b}'_k$, and encryption using $L^{1-b'}_g$ is done using oracle access to $\mathsf{sEnc}^c_{\hat{a}'_k s'+\hat{b}'_k}(\cdot,\cdot)$ (thus implicitly setting $s^{1-b'}_{j'} = s'$).

3. If the right ingoing wire of g is connected to outgoing wire j', and k is such that g is the kth gate to which outgoing wire j' is connected, then L^0_g, L^1_g are computed as in (5), $R^{b'}_g$ is computed as $R^{b'}_g = \hat{a}'_k s^{b'}_{j'} + \hat{b}'_k$, and encryption using $R^{1-b'}_g$ is done using oracle access to $\mathsf{sEnc}^c_{\hat{a}'_k s'+\hat{b}'_k}(\cdot,\cdot)$.

We assume for simplicity (and without loss of generality) that only gate i^* has its incoming wires connected to both j and j', though in fact this assumption is not needed for the proof.

Case 2: $g = i^*$. Let k, k' be such that i^* is the kth gate to which outgoing wire j is connected, and the k'th gate to which outgoing wire j' is connected. $\mathcal{A}$ computes $L \stackrel{\text{def}}{=} L^b_{i^*} = \hat{a}_k s^b_j + \hat{b}_k$ and $R \stackrel{\text{def}}{=} R^{b'}_{i^*} = \hat{a}'_{k'} s^{b'}_{j'} + \hat{b}'_{k'}$. (Recall that s^b_j [resp., $s^{b'}_{j'}$] is the active key on outgoing wire j [resp., j']. Thus, L is the active key on the left ingoing wire to gate i^*, and R is the active key on the right ingoing wire to gate i^*.) Let $s^0_{\ell+i^*}$ and $s^1_{\ell+i^*}$ be the keys on the outgoing wire ℓ_{i^*} that exits gate i^*, and let $s^* \stackrel{\text{def}}{=} s^{\text{NAND}(b,b')}_{\ell+i^*}$ denote the active key on that wire. Next, $\mathcal{A}$ computes four ciphertexts C_1, C_2, C_3, C_4 as follows:

- $\mathcal{A}$ computes $C_1 \leftarrow \mathsf{sEnc}_L\left(\mathsf{sEnc}_R\left(s^*\right)\right)$.
- $\mathcal{A}$ first computes $C'_2 \leftarrow \mathsf{sEnc}_R\left(s^{\text{NAND}(1-b,\,b')}_{\ell+i^*}\right)$ and $C^*_2 \leftarrow \mathsf{sEnc}_R\left(s^*\right)$. It then uses one of its oracles to obtain $C_2 \leftarrow \mathsf{sEnc}^c_{\hat{a}_k s+\hat{b}_k}(C'_2, C^*_2)$.
- $\mathcal{A}$ uses one of its oracles to obtain $C'_3 \leftarrow \mathsf{sEnc}^c_{\hat{a}'_{k'} s'+\hat{b}'_{k'}}(s^{\text{NAND}(b,\,1-b')}_{\ell+i^*}, s^*)$, and then computes $C_3 \leftarrow \mathsf{sEnc}_L\left(C'_3\right)$.
- $\mathcal{A}$ uses one of its oracles to obtain $C'_4 \leftarrow \mathsf{sEnc}^c_{\hat{a}'_{k'} s'+\hat{b}'_{k'}}(s^{\text{NAND}(1-b,\,1-b')}_{\ell+i^*}, s^*)$, and then uses another one of its oracles to obtain $C_4 \leftarrow \mathsf{sEnc}^c_{\hat{a}_k s+\hat{b}_k}(C'_4, C'_4)$.

That is, $\mathcal{A}$ implicitly uses $\hat{a}_k s + \hat{b}_k$ as the inactive key on the left ingoing wire to i^*, and implicitly uses $\hat{a}'_{k'} s' + \hat{b}'_{k'}$ as the inactive key on the right ingoing wire to i^*. Finally, $\mathcal{A}$ outputs $\mathsf{GG}_{i^*} = \{C_1, C_2, C_3, C_4\}$, where the ciphertexts are in random permuted order.

Case 3: $g > i^*$. This case is conceptually similar to case 1, with the difference being that here $\mathcal{A}$ always constructs garbled gates as in the real protocol. Once again, if both ingoing wires to g are connected to outgoing wires *other* than j or j', then $\mathcal{A}$ can easily compute the garbled gate by itself using the keys it knows. (Note that here it *cannot* be the case that j or j' is the outgoing wire from g, since the gates are topologically ordered.)

When one of the ingoing wires to g is connected to j or j', however, $\mathcal{A}$ must use one of its oracles to generate the garbled gate. By way of example, say the left ingoing wire to g is connected to outgoing wire j, and that g is the kth gate to which outgoing wire j is connected ($1 \leq k \leq d$). Then $\mathcal{A}$ computes R_g^0, R_g^1 as usual (it can do this since it knows both keys on the outgoing wire that is connected to the right ingoing wire of g), and computes $L_g^b = \hat{a}_k s_j^b + \hat{b}_k$. Letting $s_{\ell+g}^0, s_{\ell+g}^1$ be the keys on outgoing wire ℓ_g, adversary $\mathcal{A}$ constructs four ciphertexts as follows. It sets $C_1 \leftarrow \mathsf{sEnc}_{L_g^b}(\mathsf{sEnc}_{R_g^0}(s_{\ell+g}^{\text{NAND}(b,0)}))$ and $C_2 \leftarrow \mathsf{sEnc}_{L_g^b}(\mathsf{sEnc}_{R_g^1}(s_{\ell+g}^{\text{NAND}(b,1)}))$, followed by $C_3' \leftarrow \mathsf{sEnc}_{R_g^0}(s_{\ell+g}^{\text{NAND}(1-b,0)})$ and $C_4' \leftarrow \mathsf{sEnc}_{R_g^1}(s_{\ell+g}^{\text{NAND}(1-b,1)})$. It then uses its oracle to compute $C_3 \leftarrow \mathsf{sEnc}^c_{\hat{a}_k s+\hat{b}_k}(C_3', C_3')$ and $C_4 \leftarrow \mathsf{sEnc}^c_{\hat{a}_k s+\hat{b}_k}(C_4', C_4')$. Finally, it sets $\mathsf{GG}_g = \{C_1, C_2, C_3, C_4\}$, where the ciphertexts are randomly permuted. The other sub-cases are handled as in case 1, above.

The only dependence on c in the above is in the construction of garbled gate i^*. Indeed, the first $i^* - 1$ garbled gates are constructed as in (6), with four encryptions of the active key on the outgoing wire of the gate, and the last $n - i^* + 1$ garbled gates are constructed as in the real protocol. As for garbled gate i^*, if $c = 0$ then this is constructed as in the real protocol whereas if $c = 1$ then it is constructed as in (6). As claimed, then, if $c = 0$ then the output of $\mathcal{A}$ is distributed identically to H_{i^*-1} while if $c = 1$ then the output of $\mathcal{A}$ is distributed as in H_{i^*}. Computational indistinguishability of H_{i^*-1} and H_{i^*} follows, and this concludes the proof. ■

3.3 A More Efficient Variant

In this section we describe a more efficient variant of our protocol in which the wire labels are chosen in a coordinated fashion, as in [18]. Unfortunately, we are only able to prove security of the variant protocol in the random oracle model; see further discussion at the end of this section.

We merely sketch the basic idea. Now, in round 1, P_1 chooses a random shift r and $\ell + n$ outgoing-wire keys s_i^0; it then sets $s_i^1 = s_i^0 + r$ for all i. The first-round message from P_1 now contains pk and the $\ell + n$ ciphertexts $\mathsf{Enc}_{pk}(s_1^0), \ldots, \mathsf{Enc}_{pk}(s_{\ell+n}^0)$.

For each ingoing wire of the circuit, P_2 does as follows. Say the ingoing wire is connected to outgoing wire j. Then P_2 chooses random a and defines the (encrypted) 0-key for this ingoing wire to be (a re-randomization of) $\mathsf{Enc}_{pk}(s_j^0 + a)$, where this is computed using the homomorphic properties of the encryption scheme. Thus, if gate i of the circuit has its left ingoing wire connected to outgoing wire j and right ingoing wire connected to outgoing wire k, party P_2 defines the ith encrypted "garbled gate" via

$$\mathsf{encGG}_i = \begin{pmatrix} \mathsf{Enc}_{pk}(s_j^0 + a_i) \\ \mathsf{Enc}_{pk}(s_k^0 + a_i') \\ \mathsf{Enc}_{pk}(s_{\ell+i}^0) \end{pmatrix},$$

where a_i, a_i' are chosen uniformly at random. Finally, P_2 sends $\mathsf{encGG}_1, \ldots, \mathsf{encGG}_n$ to P_1.

Upon receiving this message, P_1 decrypts each encGG to obtain, for each gate i, the keys $\left(L_i^0, R_i^0, s_{\ell+i}^0,\right)$. It defines $L_i^1 = L_i^0 + r$ and $R_i^1 = R_i^0 + r$, and then prepares a garbled version GG_i of this gate as in the previous sections. P_2 can then compute the result as usual. The entire protocol is roughly twice as efficient as the original.

As we have mentioned, however, we are only able to prove security of this modified protocol in the (non-programmable) random oracle model. It seems that it should be possible to prove security in the standard model if the symmetric-key encryption scheme satisfies a strong enough definition of security. Correlation robustness [14] alone appears not to suffice; the problem is that there is also some circularity when, e.g., keys $s, s+r, s', s'+r$ are used to encrypt keys s'' and $s''+r$. Thus, some combination of correlation robustness and circular security seems necessary. It appear that the same issue is present in the works of [18, 25] as well.

4 Handling a Malicious P_1

In this section we briefly sketch extensions of our main protocol that achieve security against a malicious P_1. As in the previous section, our goal here is not to optimize the efficiency of the resulting protocol but rather to show a protocol whose complexity is still linear in n, the size of the circuit C_f. We remark that while it would not be difficult to further extend the protocol so as to be secure against a malicious P_2 (without using generic zero-knowledge proofs), we do not know how to do so while maintaining linear complexity and therefore omit further discussion of that case.

4.1 Protocol Modifications

To achieve security against a malicious P_1, we introduce the following changes to the protocol described in Section 3.1:

Proof of well-formedness of pk. We require P_1 to prove that the public key pk it sends in round 1 was output by the specified key-generation algorithm Gen. (This step is not necessary if it is possible to efficiently verify whether a given pk could have been output by Gen, as is the case with, e.g., El Gamal encryption.) We remark further that it suffices for the proof to be honest-verifier zero knowledge (since we only require security against a semi-honest P_2), and we do not require it to be a proof of knowledge.

The complexity of this step is independent of n.

Validity of outgoing-wire keys. Let $\left[C_1^0, C_1^1\right], \ldots, \left[C_N^0, C_N^1\right]$ denote the ciphertexts sent by P_1 in round 1. (Recall that it is supposed to be the case that $C_i^b = \mathsf{Enc}_{pk}(s_i^b)$.) We now require P_1 to prove that (1) each C_i^b is a well-formed ciphertext with respect to the public key pk (once again, this step is unnecessary if it is possible to efficiently verify validity of ciphertexts, as is the case with El Gamal encryption), and (2) for each i, the ciphertexts C_i^0, C_i^1 are encryptions of *distinct* values. If the encryption scheme is additively homomorphic, and we let s_i^0 (resp., s_i^1) denote the plaintext corresponding to C_i^0 (resp., C_i^1), then P_2 can compute $\mathsf{Enc}_{pk}(s_i^0 - s_i^1)$ and the latter step is equivalent to proving that this is not an encryption of 0. Once again, it suffices for these proofs to be honest-verifier zero knowledge and they are not required to be proofs of knowledge.

The complexity of this step is linear in n since the statement being proved can be written as a conjunction of n statements, each of which has size independent of n.

Correctness of garbled circuit construction. We require P_1 to prove correctness of the garbled gates it sends to P_2 in the final round. This amounts to proving, for each $i \in \{1, \ldots, n\}$, that GG_i

was correctly constructed from encGG_i. As before, it suffices for these proofs to be honest-verifier zero knowledge and they are not required to be proofs of knowledge.

Once again, the complexity of this step is linear in n since the statement being proved is a conjunction of n statements, each of which has size independent of n. We additionally note that by using an appropriate homomorphic encryption scheme and symmetric-key encryption scheme, these proofs can be made (reasonably) efficient using the techniques of Jarecki and Shmatikov [17] (who show efficient proofs for exactly this purpose when using a variant of the Camenisch-Shoup encryption scheme [7]).

Correctness of input-wire and output-wire keys. Finally, P_1 is required to prove that the input-wire and output-wire keys it sends in the final round are correct. Let $\left[C_1^0, C_1^1\right], \ldots, \left[C_N^0, C_N^1\right]$ denote the ciphertexts sent by P_1 in round 1 (recall it is supposed to be the case that $C_i^b = \mathsf{Enc}_{pk}(s_i^b)$), and let

$$\text{input-wires: } s_1, \ldots, s_\ell \quad \text{and} \quad \text{output-wires: } \left(s_{N-m+1}^0, s_{N-m+1}^1\right), \ldots, \left(s_N^0, s_N^1\right)$$

be the values sent by P_1 in the last round. Then P_1 must prove: (1) that for each index $i \in \{1, \ldots, \ell\}$, one of the ciphertexts C_i^0, C_i^1 is an encryption of the plaintext s_i, and (2) that for each index $i \in \{N-m+1, \ldots, N\}$, the ciphertext C_i^0 (resp., C_i^1) is an encryption of s_i^0 (resp., s_i^1). It suffices for each of these proofs to be honest-verifier zero knowledge; the first set of proofs (proving correctness of the input-wire keys) must be proofs of knowledge to allow for input extraction. (Alternately, if the proof of well-formedness of the public key is a proof of knowledge then proofs of knowledge are not needed here.)

The complexity of this step is linear in $\ell + m$.

We remark that most of the above proofs can be implemented efficiently for any homomorphic encryption scheme. The main exception is the proof of correctness of the garbled circuit construction; however, as noted already, there exists at least one specific homomorphic encryption scheme for which this step can be done efficiently [17].

4.2 Proof of Security

Theorem 5 *With the modifications described in the previous section, and under the same assumptions as in Theorem 4, the protocol of Figure 1 is a secure $\mathcal{C}$-PFE protocol for a malicious P_1.*

Proof: Security for a semi-honest P_2 follows from Theorem 4, since all the proofs added to the protocol are honest-verifier zero knowledge.

We briefly sketch the proof of security for a malicious P_1. Consider the following simulator $\mathcal{S}_1$ that is given black-box to P_1 and acts as follows: it receives the round 1 message from P_1, and verifies the relevant proofs given by P_1 showing that pk is valid and that each pair C_i^0, C_i^1 encrypts distinct plaintexts. If any of these proofs fails, $\mathcal{S}_1$ simply aborts. Otherwise, $\mathcal{S}_1$ generates a second-round message as in the proof of Theorem 4; i.e., for $i = 1$ to n the simulator chooses random r_0, r_1, r_2, r_3, computes

$$\mathsf{encGG}_i = \begin{pmatrix} [\mathsf{Enc}_{pk}(r_0),\ \mathsf{Enc}_{pk}(r_1)] \\ [\mathsf{Enc}_{pk}(r_2),\ \mathsf{Enc}_{pk}(r_3)] \\ \left[C_{\ell+i}^0,\ C_{\ell+i}^1\right] \end{pmatrix}, \tag{7}$$

and gives $\mathsf{encGG}_1, \ldots, \mathsf{encGG}_n$ to P_1.

$\mathcal{S}_1$ receives the third-round message from P_1 and verifies the relevant proofs given by P_1. If any of these proofs fails, $\mathcal{S}_1$ aborts. Otherwise, for each $i \in \{1, \ldots, \ell\}$ the simulator extracts (using the fact that the relevant proof here is a proof of knowledge) a bit x_i such that $C_i^{x_i}$ is an encryption of s_i, the ith input-wire value sent by P_1. (If extraction of any of these bits fails, $\mathcal{S}_1$ aborts.) It sets $x = x_1 \cdots x_\ell$ and sends x to the trusted party, and outputs whatever P_1 outputs. This completes the simulation.

Computationally indistinguishability of the simulation just described and the view of P_1 in a real execution of the protocol follows from the following observations:

- Assuming the proofs given by P_1 after the first round all succeed, with all but negligible probability it holds that (1) pk is a valid public key and (2) each pair C_i^0, C_i^1 encrypts distinct plaintexts. Assuming these to be the case then, as in the proof of Theorem 4, the second-round message generated by $\mathcal{S}_1$ is *identically distributed* to the second-round message that would be sent in an honest execution of the protocol. (As in the proof of that theorem, this follows from pairwise independence of the mapping used by an honest P_2.)

- Assuming the proofs given by P_1 after the third round all succeed, with all but negligible probability it holds that (1) each garbled gate was computed correctly; (2) P_1 sent valid input-wire labels; and (3) P_1 sent valid output-wire labels. Moreover, with all but negligible probability, for each x_i extracted by $\mathcal{S}_1$ it holds that $C_i^{x_i}$ is an encryption of s_i and so s_i is the key corresponding to bit x_i on input wire i. (Note further that it cannot be that $C_i^{1-x_i}$ is also an encryption of s_i, since P_1 proved is round 1 that C_i^0, C_i^1 encrypt distinct plaintexts.) It follows that the real P_2 would output $C_f(x)$, which is identical to the output of P_2 in the ideal world.

This completes the proof. ∎

5 Conclusions and Future Work

In this paper we have shown what is, to the best of our knowledge, the first protocol for PFE with complexity *linear* in the size of the circuit being computed (without relying on fully homomorphic encryption). Our results leave several interesting open questions:

- In addition to its theoretical importance, we believe our work is of practical relevance also: specifically, we expect that our approach to PFE will be both easier to implement and more efficient (for moderate circuit sizes) than approaches that rely on universal circuits. In future work we plan to experimentally validate this claim by implementing (an optimized version of) our protocol and comparing it to the protocol of [19, 30].

- From a theoretical perspective, our work leaves open the question of whether *fully secure* PFE (i.e., PFE with security against a malicious P_1 and a malicious P_2) is possible with linear complexity, without relying on fully homomorphic public-key encryption. Even security in the *covert* setting [2] (with linear complexity) would be interesting.

- It would also be interesting to further improve on the cryptographic assumptions needed: e.g., to construct a protocol based on semantically secure symmetric-key encryption (without requiring related-key security), or to avoid the use of homomorphic public-key encryption.

References

[1] B. Applebaum, D. Harnik, and Y. Ishai. Semantic security under related-key attacks and applications. Available at `http://eprint.iacr.org/2010/544`.

[2] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *4th Theory of Cryptography Conference — TCC 2007*, volume 4392 of *LNCS*, pages 137–156. Springer, 2007.

[3] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider. Secure evaluation of private linear branching programs with medical applications. In *14th European Symposium on Research in Computer Security (ESORICS)*, volume 5789 of *LNCS*, pages 424–439. Springer, 2009.

[4] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *2nd Theory of Cryptography Conference — TCC 2005*, volume 3378 of *LNCS*, pages 325–341. Springer, 2005.

[5] J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel. Privacy-preserving remote diagnostics. In *14th ACM Conf. on Computer and Communications Security (CCCS)*, pages 498–507. ACM Press, 2007.

[6] C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-round secure computation and secure autonomous mobile agents. In *27th Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, volume 1853 of *LNCS*, pages 512–523. Springer, 2000.

[7] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology — Crypto 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, 2003.

[8] R. Canetti, Y. Ishai, R. Kumar, M. K. Reiter, R. Rubinfeld, and R. N. Wright. Selective private function evaluation with applications to private statistics. In *20th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 293–304. ACM Press, 2001.

[9] Y.-C. Chang and C.-J. Lu. Oblivious polynomial evaluation and oblivious neural learning. In *Advances in Cryptology — Asiacrypt 2001*, volume 2248 of *LNCS*, pages 369–384. Springer, 2001.

[10] K. Frikken, M. Atallah, and J. Li. Hidden access control policies with hidden credentials. In *Proc. ACM Workshop on Privacy in the Electronic Society (WPES)*, page 27. ACM, 2004.

[11] K. Frikken, M. Attallah, and C. Zhang. Privacy-preserving credit checking. In *ACM Conf. on Electronic Commerce (EC)*, pages 147–154. ACM, 2005.

[12] K. B. Frikken, J. Li, and M. J. Atallah. Trust negotiation with hidden credentials, hidden policies, and policy cycles. In *Network and Distributed System Security Symposium (NDSS)*, pages 157–172. The Internet Society, 2006.

[13] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications.* Cambridge University Press, Cambridge, UK, 2004.

[14] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology — Crypto 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[15] Y. Ishai and A. Paskin. Evaluating branching programs on encrypted data. In *4th Theory of Cryptography Conference — TCC 2007*, volume 4392 of *LNCS*, pages 575–594. Springer, 2007.

[16] T. Jakobsen, M. Makkes, and J. Nielsen. Efficient implementation of the Orlandi protocol. In *8th Intl. Conference on Applied Cryptography and Network Security (ACNS)*, volume 6123 of *LNCS*, pages 255–272. Springer, 2010.

[17] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *LNCS*, pages 97–114. Springer, 2007.

[18] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *35th Intl. Colloquium on Automata, Languages, and Programming (ICALP), Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[19] V. Kolesnikov and T. Schneider. A practical universal circuit construction and secure evaluation of private functions. In *Financial Cryptography and Data Security*, volume 5143 of *LNCS*, pages 83–97. Springer, 2008.

[20] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, 2007.

[21] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[22] Y. Lindell, B. Pinkas, and N. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *6th Intl. Conf. on Security and Cryptography for Networks (SCN '08)*, volume 5229 of *LNCS*, pages 2–20. Springer, 2008.

[23] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *Proc. 13th USENIX Security Symposium*, pages 287–302. USENIX Association, 2004.

[24] M. Naor and B. Pinkas. Oblivious polynomial evaluation. *SIAM Journal on Computing*, 35(5):1254–1281, 2006.

[25] J. B. Nielsen and C. Orlandi. LEGO for two-party secure computation. In *6th Theory of Cryptography Conference — TCC 2009*, volume 5444 of *LNCS*, pages 368–386. Springer, 2009.

[26] A. Paus, A.-R. Sadeghi, and T. Schneider. Practical secure evaluation of semi-private functions. In *7th Intl. Conference on Applied Cryptography and Network Security (ACNS)*, volume 5536 of *LNCS*, pages 89–106. Springer, 2009.

[27] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *Advances in Cryptology — Asiacrypt 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.

[28] A.-R. Sadeghi and T. Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In *11th Intl. Conf. on Information Security and Cryptology (ICISC)*, volume 5461 of *LNCS*, pages 336–353. Springer, 2008.

[29] T. Sander, A. Young, and M. Yung. Non-interactive cryptocomputing for NC^1. In *40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 554–567. IEEE, 1999.

[30] T. Schneider. Practical secure function evaluation. Master's thesis, University Erlangen-Nürnberg, 2008. Available from `http://thomaschneider.de/FairplayPF`.

[31] L. Valiant. Universal circuits. In *8th Annual ACM Symposium on Theory of Computing (STOC)*, pages 196–203. ACM Press, 1976.

[32] A. C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE, 1986.