How to Hide Circuits in MPC an Efficient Framework for Private Function Evaluation

Payman Mohassel and Saeed Sadeghian

University of Calgary

Abstract. We revisit the problem of general-purpose *private function evaluation* (PFE) wherein a single party P_1 holds a circuit $\mathcal{C}$, while each P_i for $1 \leq i \leq n$ holds a private input x_i, and the goal is for a subset (or all) of the parties to learn $\mathcal{C}(x_1, \ldots, x_n)$ but nothing else. We put forth a general framework for designing PFE where the task of hiding the circuit and securely evaluating its gates are addressed independently: First, we reduce the task of hiding the circuit topology to oblivious evaluation of a mapping that encodes the topology of the circuit, which we refer to as *oblivious extended permutation* (OEP) since the mapping is a generalization of the permutation mapping. Second, we design a subprotocol for private evaluation of a single gate (PFE for one gate), which we refer to as *private gate evaluation* (PGE). Finally, we show how to naturally combine the two components to obtain efficient and secure PFE.

We apply our framework to several well-known general-purpose MPC constructions, in each case, obtaining the most efficient PFE construction to date, for the considered setting. Similar to the previous work we only consider semi-honest adversaries in this paper.

1 Introduction

In a *private function evaluation* (PFE) protocol, a party P_1 holds a function f, and its corresponding circuit C_f, while every party P_i holds a private input x_i; their goal is for a subset (or all) of the parties to learn $f(x_1, \ldots, x_n)$ without learning any information beyond this. In particular, besides the size of the circuit, and the length of P_1's inputs and outputs, P_i ($i \geq 2$) should not learn anything else about the circuit. This is in contrast to the standard setting for secure multi-party computation where the function f and the corresponding circuit C_f are publicly known to all the participants. PFE is particularly useful in scenarios where learning the function compromises privacy, reveals security vulnerabilities, or when service providers need to hide the function or a specific implementation of it to protect their Intellectual Property. A number of papers in the literature have considered the design of efficient general-purpose private function evaluation protocols [1,2,3,4].

Solutions Based on Universal Circuits. Most general-purpose PFE solutions reduce the problem to secure computation of a *universal circuit* U_g that

T. Johansson and P. Nguyen (Eds.): EUROCRYPT 2013, LNCS 7881, pp. 557–574, 2013.

takes as input the circuit C_f (with at most g gates), and the parties' private inputs $x_1, \ldots, x_n$, and outputs $f(x_1, \ldots, x_n)$. The main objective of this line of work is to design smaller size universal circuits, and to optimize their implementation using existing MPC constructions such as Yao's garbled circuit protocol [2,5,3].

The Universal circuit approach works with any secure MPC protocol for evaluating boolean circuits and is applicable to both the two-party and the multi-party settings. Its main disadvantage, and the main motivation for other alternatives is the additional overhead in efficiency due to the size of universal circuits and the complexity of designing and implementing such circuits. Valiant [6] showed a construction of a boolean universal circuit achieving an optimal circuit size of $|U_g| \approx 19g \log g$. Kolesnikov and Schneider [2] gave an alternative construction of universal circuits. They obtain a worse asymptotic bound of $|U_g| \approx 1.5g \log^2 g$, but their techniques lead to smaller constant factors and seem to yield smaller universal circuits than Valiant's construction for circuit sizes less than 5000. Furthermore, the universal circuit approach does not provide a satisfactory solution in case of arithmetic circuits. While universal arithmetic circuits exist (e.g. see [7] and [8]), their sizes are too large for any practical purpose (e.g. as high as $O(g^5)$).

Solutions Based on Homomorphic Encryption. It is relatively easy to design a PFE based on a fully homomorphic encryption scheme [9]. While asymptotically optimal, this solution is not practical due to its high computational cost. Recently, Katz and Malka [4] designed a novel two-party PFE protocol based on a *singly homomorphic encryption*. Complexity of the resulting protocol is linear in the size of the circuit but the number of public-key operations is also linear in the size of the circuit. Standard techniques for reducing public-key operations (e.g. OT extension) do not seem applicable either. Given the significant gap between the efficiency of public- vs. symmetric-key operations, this new approach improves over the universal circuit only when dealing with large circuits. Finally, this solution only works in the two-party setting.

Our Contribution. Practical design and implementation of MPC has been the subject of active research in the last few years. As discussed above, however, when it comes to PFE the situations is not the same. The existing solutions are considerably less scalable and more expensive compared to their MPC counterparts, and *no good solution exists for the multiparty case, or when considering arithmetic circuits*. We revisit private function evaluation with the intention of designing more practical two-party and multi-party constructions. In particular, we put forth a general framework for designing PFE and show how it enables us to construct more efficient PFE variants of the well-known MPC protocols.

Our Framework for Designing PFE. In order to fully hide a circuit $\mathcal{C}$, one needs to hide two types of information about it: (i) the *topology* of the circuit, and (ii) the *function of the gates* in the circuit (AND, OR, XOR). Note that these are in addition to what is already hidden in a MPC setting. Following this observation we divide the task of private function evaluation into two different functionalities:

(1) the Circuit Topology Hiding (CTH) functionality, and (2) the Private Gate Evaluation (PGE) functionality. Next, we describe these two functionalities in more detail:

CTH *Functionality.* We observe that the topology of a circuit $\mathcal{C}$ can be fully described using a mapping $\pi_{\mathcal{C}} : \{1 \dots |\mathsf{OW}|\} \to \{1 \dots |\mathsf{IW}|\}$ where OW (outgoing wires) is the union of the set of input wires $\{\mathsf{ow}_1 = x_1, \dots, \mathsf{ow}_n = x_n\}$, and the output wires for each non-output gate in the circuit $\{\mathsf{ow}_{n+1}, \dots, \mathsf{ow}_{n+g-o}\}$ (g is the circuit size and o is the number of output gates), and IW (incoming wires) is the set of input wires to all the gates in the circuit $\{\mathsf{iw}_1, \dots, \mathsf{iw}_{2g}\}$. $\pi_{\mathcal{C}}$ maps i to j ($\pi_{\mathcal{C}}(i) = j$) if and only if wire $\mathsf{ow}_i \in \mathsf{OW}$ is connected to $\mathsf{iw}_j \in \mathsf{IW}$, in the circuit $\mathcal{C}$. Note that since the fan-out for each gate can be more than one, $\pi_{\mathcal{C}}$ is not always a function, but it is easy to check that its inverse $\pi_{\mathcal{C}}^{-1}$ is. *Note that the party who knows the function f and the corresponding circuit $\mathcal{C}$ can efficiently compute $\pi_{\mathcal{C}}$.* Figure 1 demonstrates an example circuit and its corresponding mapping. Intuitively the $\mathcal{F}_{\mathcal{CTH}}$ functionality provides a mechanism for obliviously applying the mapping π_C to the n input values and the $(g - o)$ values for intermediate outgoing wires (i.e. mapping them to incoming wires) in an on-demand fashion, and as the MPC protocol proceeds.

PGE *Functionality.* The PGE functionality can be seen as a PFE protocol where the function is a single gate. P_1 provides the gate's functionality, while all parties including P_1 provide their shares of the two inputs to the gate. The functionality returns to each party, his share of the gate's output.

These two functionalities can be naturally composed to obtain a complete PFE protocol as described in Figure 4. A visual demonstration of the steps appears in Figure 2.

Efficient Realizations of $\mathcal{F}_{\mathcal{CTH}}$. We refer to the mapping $\pi_{\mathcal{C}} : \{1 \dots |\mathsf{OW}|\} \to \{1 \dots |\mathsf{IW}|\}$ discussed above as an *extended permutation* (EP) since it not only permutes the elements in $\{1 \dots |\mathsf{OW}|\}$, but also can replicate them as many times as needed. A main component of our $\mathcal{F}_{\mathcal{CTH}}$ realization is a protocol for *oblivious evaluation* of this extended permutation (OEP) on a vector of inputs:

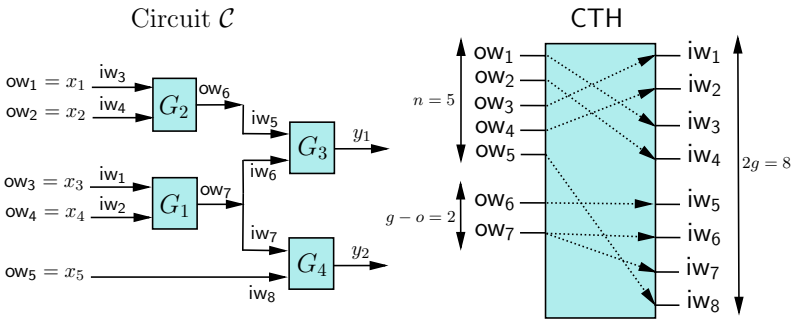

Fig. 1. An example circuit and the corresponding mapping

the first party holds π_C and a blinding[1] vector $\boldsymbol{t}$ of size $|\mathsf{IW}|$, while the second party holds an input vector $\boldsymbol{x}$ of size $|\mathsf{OW}|$. Their goal is to let the second party learn the output of π_C applied to $\boldsymbol{x}$, blinded by $\boldsymbol{t}$. Neither party should learn anything else. OEP can be instantiated using a singly homomorphic encryption, or any general-purpose 2PC. As discussed in the Full version, however, neither solution is efficient enough for use in practice. We introduce a new and efficient construction for OEP based on *generalized switching networks* and oblivious transfer.

OEP via Generalized Switching Networks. First, we show how to efficiently implement an extended permutation using a generalized switching network SN. Once the EP is represented using a SN, we solve the OEP problem by designing a new OT-based protocol for *Oblivious Switching Network evaluation* (OSN) where one party P_1 holds the selection bits to SN, and a blinding vector $\boldsymbol{t}$, while the other party P_2 holds the input vector $\boldsymbol{x}$ to the SN. The goal is for P_2 to learn the output of SN applied to the input vector $\boldsymbol{x}$, blinded by $\boldsymbol{t}$. Our OSN protocol runs in a *constant number of rounds* and requires $O(g)$ oblivious transfers where g is the number of switches in the network. We also need a *multiparty variant of our OEP protocol* where the mapping is known to a single party while the input vector $\boldsymbol{x}$ and the blinding vector $\boldsymbol{t}$ are shared among the players. We show how to construct such an m-party OEP protocol via m invocations of the two-party version.

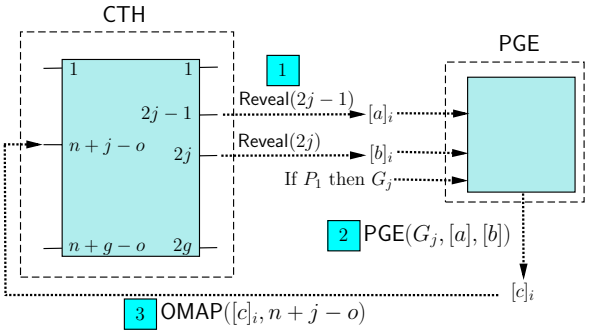

Fig. 2. Steps of framework for party i and the jth gate in a topological order

Improved Oblivious Shuffling. Digressing from the main topic of this paper, we note that OSN is a generalization of the previously studied problems such as oblivious shuffling [10] (a subprotocol used for private set intersection), or secure two-party permutation [11,12]. Our new construction yields more efficient solutions to these problems as well, improving on the previous proposals based

[1] The nature of blinding is intentionally left unspecified as different protocols may use different blinding functions. Our constructions use XOR or addition in a finite Ring for this purpose.

Table 1. Comparison of oblivious shuffling protocols. N is number of shuffled elements, ℓ is the length of each, and k is the security paremeter.

Oblivious Shuffling Protocols	Asymptotic Complexity
HE-Based	$O(N)$ Asym.
Garbled Circuit-Based [10]	$(\frac{4\ell(N\log N-N+1)}{3} + 2N\ell)$ Sym. + $O(k)$ Asym.
OSN-Based (our paper)	$(2N\log N - 2N + 2)$ Sym. + $O(k)$ Asym.

on garbled circuit implementation of sorting networks, permutation networks, or randomize shell sort [10,11,12]. See Table 1 for efficiency comparison with previous work.

Applying our Framework to Existing MPC. We apply the above framework to the GMW protocol [13], Yao's garbled circuit protocol [14], and secure computation of arithmetic circuits via homomorphic encryption [15]. In each case we obtain the most efficient PFE construction to date, for the considered setting.

Linear Multi-party PFE. We apply our framework to the seminal GMW protocol [13] to obtain a multiparty PFE against a dishonest majority. The CTH component can be instantiated using either the HE-based or the SN-based OEP discussed above. We also design a simple and efficient multiparty PGE functionality given a multiparty OT as in [16]. To the best of our knowledge, this is the first multiparty PFE besides the generic solutions of applying MPC to universal circuits. When instantiated using a HE-based OEP, it yields the *first multiparty PFE with linear complexity* (in the circuit size) and when instantiated using our new SN-based OEP, it yields a black-box construction based solely on OT. What makes the second instantiation desirable from a practical point of view, as demonstrated in some recent GMW implementations [17,18], is that it only uses *oblivious transfers*. As a result, one can use OT extension [19] and pre-processing techniques [20] to significantly reduce the number of public-key operations, and to shift the bulk of the computation to an *offline phase*. Table 2 compares the efficiency of these two constructions with the only other alternative, i.e. using GMW with universal circuits.

Table 2. Comparison of m-party PFE protocols. g denotes the number of gates.

Multi-Party PFE	Complexity
[2] Universal Circuits	$O(m^2 g \log^2 g)$ Sym. + $O(k)$ Asym.
[6] Universal Circuits	$O(m^2 g \log g)$ Sym. + $O(k)$ Asym.
GMW-PFE (SN-OEP)	$O(m^2 g + mg \log g)$ Sym. + $O(k)$ Asym.
GMW-PFE (HE-OEP)	$O(m^2 g)$ Sym. + $O(mg)$ HE. + $O(k)$ Asym.

More Efficient Two-party PFE. We also design a constant round two-party PFE based on Yao's garbled circuit protocol [14]. Once again, the $\mathcal{F}_{\mathcal{CTH}}$

functionality is realized using our OEP constructions and for the $\mathcal{F}_{\mathcal{PGE}}$ functionality we use Yao's garbling/ungarbling algorithms. To ensure that functions of the gates are hidden, we build the circuit entirely out of NAND gates. As we will see in Section 5.3, multiple subtleties need to be addressed for this work and in particular to guarantee that the circuit evaluator can unblind garbled keys during the evaluation of the garbled circuit without learning the values for the intermediate wires.

We note that the construction of [4] also fits in the general framework described above (though not presented in this way). However, our new abstraction helps us gain more efficiency improvements. When using our HE-OEP, we obtain a two-party PFE with linear complexity that is simpler and more efficient than that of [4] (see Full Version for details), and when implemented using our SN-OEP, the resulting protocol is concretely more efficient for most circuit sizes, since the number of public-key operations can be made independent of the circuit size (via OT extension). Our construction is both asymptotically and concretely more efficient than the previous work of [2] based on universal circuits. It is concretely more efficient than Valiant's construction [6]. Table 3 summarizes efficiency comparison of our two-party PFE with all previous constructions. In the full version of this paper [21], we show (thorough operations counting) that our construction concretely improves over previous constructions for benchmark circuits such as AES, RSA and Edit-distance.

Linear 2PC for Arithmetic Circuits. We also apply our framework to the construction for secure computation of arithmetic circuits based on a homomorphic encryption [15], and obtain the *first two-party PFE for arithmetic circuits with linear complexity*. Besides utilizing our $\mathcal{F}_{\mathcal{CTH}}$ realizations, we instantiate the $\mathcal{F}_{\mathcal{PGE}}$ functionality by designing a secure gate evaluation protocol wherein only one party knows/learns the functionality (multiplication or addition) but both parties learn their share of the output (product or sum).

Table 3. Comparison of 2-party PFE protocols. (HM: Homomorphic Multiplication, HA: Homomorphic Addition, HE: Homomorphic Encryption). Last column shows concrete gain over universal circuit approaches for benchmark circuits, AES, RSA and Edit-distance (refer to Full version for detailed discussion). g denotes the number of gates.

2-Party PFE	Complexity	Gain
[2]	$1.5g \log^2 g$ sym. + $O(k)$ Asym.	3-6
[6]	$19g \log g$ sym. + $O(k)$ Asym.	2
[4]	$O(g)$ Sym. + $O(g)$ (HE+HM+HA) + $O(k)$ Asym.	-
Yao-PFE (HE-OEP)	$O(g)$ Sym. + $O(g)$ (HE+HA) + $O(k)$ Asym.	-
Yao-PFE (SN-OEP)	$O(g \log g)$ Sym. + $O(k)$ Asym.	1

2 Preliminaries

Notations. For a set D, we denote its size by $|D|$. We use the same notation to show the size (number of gates) of a circuit C. We denote a vector by $\boldsymbol{v}$. We use $[a]$ to denote secret sharing of a value a among multiple parties. We intentionally do not specify the sharing scheme used. In our constructions we use a number of different schemes such as XOR sharing, and additive sharing over a finite ring. We denote the ith party's shared by $[a]_i$. We use $\{1...n\}$ to denote the set of positive integers less than equal to n.

Generalized Switching Networks. A switching network SN is a set of interconnected switches that takes N inputs and a set of selection bits, and outputs N values. Each *switch* in the network accepts two ℓ-bits strings as input and outputs two ℓ-bit strings. In our generalized notion of a switch, each of the two output strings can take the value of each of the two input strings. Therefore, assuming input values (x_0, x_1), and output values (y_0, y_1), four different switch types are possible. The two selection bits s_0 and s_1 determine the switch type. In particular, the output of the switch will be $y_1 = x_{s_1}$,and $y_0 = x_{s_0}$. In the rest of the paper, we drop the term generalized and simply refer to these networks as switching networks.

Definition 1 (Mapping for a Switching Network). *The* mapping $\pi : \{1...N\} \rightarrow \{1...N\}$ *corresponding to a switching network SN is defined such that* $\pi(i) = j$ *if and only if after evaluation of SN on the N inputs, the value of the input wire i is assigned to the output wire j (assuming a standard numbering of the input/output wires).*

Note that the mapping π need not be a function since the value for each input wire maybe mapped to multiple output wires in the network. On the other hand, π^{-1} is always a function.

Permutation Networks. A permutation network PN is a switching network for which the mapping is a permutation. In constructing a permutation network, one only needs to use two of the four switch types described above. Particularly, for each switch (also called a permutation cell) with inputs I_0 and I_1, one selection bit is sufficient to select between the two possible outputs (I_0, I_1) and (I_1, I_0).

An optimal construction for a permutation network was proposed by Waksman [22]. The main theorem of [22] states that for any N power of 2, there exists a permutation network with $N \log N - N + 1$ switches, and depth of $2 \log N - 1$. We refer the reader to [22] for the details of the construction which can be efficiently implemented with $O(N \log N)$ complexity.

In the remainder of the paper, if a switch takes two selection bits, we refer to it as a *2-switch*, and otherwise we use the term *1-switch*.

Security Definitions. Security definitions are the standard notions of security against semi-honest adversaries (see Full version).

3 Our Framework for Designing PFE Protocols

Similar to the previous work on private function evaluation, we assume that the following information about the circuit is publicly known: the number of gates in the circuit, the number of each party's input wires, and the number of output wires. Everything else about the circuit is considered private information. We aim to hide the circuit through the CTH and PGE functionalities discussed earlier. In this section we formally describe these functionalities and explain how they can be combined to obtain a PFE.

Our interpretation of sharing (denote using []) in the following discussion is very general. In the GMW-based PFE we use XOR sharing, for arithmetic circuits we use additive shares over a finite ring, and in Yao's garbled circuit, one party holds one random key (in a key pair) while the other party holds the mapping of each key to its actual bit value.

The $\mathcal{F}_{\mathcal{CTH}}$ functionality with circuit parameters n (number of input wires), g (number of gates), o (number of output wires), and internal variables $\mathsf{Out}[i,j]$ for $1 \leq i \leq m$ and $1 \leq j \leq 2g$ where m is the number of parties, and $\mathsf{Out}[i,j]$ denote P_i's share for the value of the j-th incoming wire in the circuit.

Parties Setup: P_1 computes the mapping $\pi_{\mathcal{C}}$ corresponding to circuit $\mathcal{C}$. He also generates m random vectors $\boldsymbol{t}_i$, $1 \leq i \leq m$, where $\boldsymbol{t}_i =< t_i[1], \ldots, t_i[2g] >$. P_i for $2 \leq i \leq m$ generates a random key vector $\boldsymbol{k}_i =< k_i[1], \ldots, k_i[2g] >$.

On Queries:

$\mathsf{OMAP}([x], j)$:

- P_1**'s Input:** $\pi_{\mathcal{C}}$, $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_m$.
- P_i**'s** $(1 \leq i \leq m)$ **Input:** $[x]_i$, $\boldsymbol{k}_i$, index j for outgoing wire ow_j.

It sends to P_1, $\mathsf{Out}[i,l] = [x]_i \otimes k_i[l] \otimes t_i[l]$ for all l where $\pi_{\mathcal{C}}(j) = l$. Other parties do not receive any output.

$\mathsf{Reveal}(j)$:

- P_i**'s** $(1 \leq i \leq m)$ **Input:** index j for the incoming wire iw_j.

It reveals $\mathsf{Out}[i,j]$ to P_i for $i \geq 2$. (Note that P_i can unblinds $\mathsf{Out}[i,j]$ using $k_i[j]$ and recover his fresh random share of $[x]_i \otimes t_i[j]$.)

Fig. 3. The Circuit-Topology Hiding Functionality ($\mathcal{F}_{\mathcal{CTH}}$)

CTH *Functionality.* As described in the introduction, the interconnection of wires in the circuit can be represented by a mapping $\pi_{\mathcal{C}}$. The CTH functionality is responsible for obliviously applying this mapping to the values of the input wires and the intermediate wires in the circuit, in an *on-demand* fashion. Our definition of the CTH functionality captures this useful property refered to as *on-demand mapping* via use of the OMAP/Reveal queries. The OMAP queries allow the participants in the CTH to feed their shares of the values for each

outgoing wire to the mapping (individually) and obtain the mapped/blinded outcomes for each incoming wire through the Reveal queries. Our new realization of the CTH functionality as well as the existing constructions all possess the on-demand property (see full version). Figure 3 describes the CTH functionality more formally.

The role of vectors $\boldsymbol{k}$ is to prevent P_1 from learning the other parties' shares and the role of vectors $\boldsymbol{t}$ is to hide P_1's mapping $\pi_{\mathcal{C}}$ from the other parties. The operator $\otimes$ is used to denote a blinding operation. Depending on the CTH realization, the blinding operation can be XORing, modular addition, or homomorphic addition using an additively homomorphic encryption.

The PGE *Functionality.*The PGE functionality can be seen as a PFE protocol where the function is a single gate. A formal description is as follows.

Inputs: P_1's input is G, $[a]_1$, $[b]_1$. P_i's input ($i \geq 2$) is $[a]_i$, $[b]_i$.
Output: P_i's output is fresh random shares of $G(a, b)$, i.e. $[c]_i = [G(a, b)]_i$

Our PFE Framework. These two functionalities can be naturally composed to obtain a complete PFE protocol as described in Figure 4. Our framework can be seen as a way to extend a PFE protocol for one gate (PGE) to a PFE protocol for the complete circuit (by employing the CTH functionality). We give an overview next. In the initialization phase, P_1, knowing the circuit $\mathcal{C}$, sorts the gates topologically and computes the mapping $\pi_{\mathcal{C}}$ corresponding to it. Next, each party distributes shares of its input to all parties. The idea is for the parties to send the value of each outgoing wire to the CTH functionality as soon as it is ready. Hence, at the start of the protocol they send shares of their input values to $\mathcal{F}_{\mathcal{CTH}}$ (the input wires are the first set of outgoing wires in the circuit). The $\mathcal{F}_{\mathcal{CTH}}$ maps these values to the corresponding incoming wires (through OMAP queries). This ends the initialization phase. Parties then individually evaluate the gates. For the current gate being evaluated, parties obtain their shares for the two input values using two Reveal queries to the $\mathcal{F}_{\mathcal{CTH}}$. Next, parties invoke the PGE functionality to receive fresh random shares for the output of the current gate. Parties send these newly learnt shares to the CTH functionality and repeat the process until all gates are evaluated.A visual demonstration of the steps appears in Figure 2.

Theorem 1. *Given secure realizations of $\mathcal{F}_{\mathcal{CTH}}$ and $\mathcal{F}_{\mathcal{PGE}}$ against semi-honest adversaries, the above PFE framework is secure against semi-honest adversaries.*

4 Realizing the CTH Functionality via OEP

What is an Extended Permutation? Before describing our construction in more detail, we need to explain the notion of an *extended permutation*. Recall that a mapping $\pi : \{1...N\} \rightarrow \{1...N\}$ is a permutation if it is a bijection (i.e. one-to-one and onto). An extended permutation generalizes this notion as follows:

Definition 2 (Extended Permutation). *For positive integers M and N, we call a mapping $\pi : \{1...M\} \rightarrow \{1...N\}$ an extended permutation (EP) if for*

every $y \in \{1...N\}$ *there is exactly one* $x \in \{1...M\}$ *such that* $\pi(x) = y$. *We often denote* x *by* $\pi^{-1}(y)$.

Note that in an extended permutation, unlike a standard permutation mapping, the mapping can also replicate/omit elements (as many times as needed) hence allowing the range to be larger or smaller than the domain.

CTH and the OEP Problem. To realize the CTH functionality we have to implement $n+g-o$ OMAP queries, one for each outgoing wire, and $2g$ Reveal queries, one for each incoming wire. When combined, these OMAP/Reveal queries naturally form a problem we refer to as *oblivious evaluation of the extended permutation* (OEP). We define the two-party OEP problem here. In the full version, we describe a natural generalization of the problem to the m-party case and show how to efficiently realize it using m invocations of the two-party variant (wee need the multiparty variant for our GMW-based PFE).

P_1**'s Inputs:** The circuit $\mathcal{C}$ with g gates, n input wires, and o output gates. Denote the corresponding mapping by $\pi_{\mathcal{C}}$.
P_i**'s Input** $(1 \leq i \leq m)$**:** x_j for all input wires j in the circuit belonging to P_i.
Outputs: For $1 \leq i \leq m$, P_i learns his share of the values for the output wires.

Initialization:

1. P_1 sort the gates in the circuit, topologically. Denote the ordered gates by $G_1, \ldots, G_g$.
2. For $1 \leq i \leq m$, P_i distributes shares of his inputs among all parties.
3. For $1 \leq j \leq n$, parties make the query $\mathsf{OMAP}([x_j], j)$ to the $\mathcal{F}_{\mathcal{CTH}}$.

Private Function Evaluation:
For $1 \leq j \leq g$:

1. Parties make the queries $\mathsf{Reveal}(2j-1)$, and $\mathsf{Reveal}(2j)$ to the $\mathcal{F}_{\mathcal{CTH}}$. Denote the output P_i receives by $[a]_i$ and $[b]_i$, respectively.
2. Parties invoke the $\mathcal{F}_{\mathcal{PGE}}$ where P_i's input is $([a]_i, [b]_i)$, while P_1's input also includes the gate functionality (G_j). Each party P_i receives its share of the gate's output, i.e. $[G_j(a,b)]_i$.
3. If $j < g - o$, parties send the query $\mathsf{OMAP}([G_j(a,b)], n+j)$ to $\mathcal{F}_{\mathcal{CTH}}$.

For $g - o < j \leq g$, parties reveal their shares of $[G_j(a,b)]$, and everyone reconstructs the value of the o output wires.

Fig. 4. A General Framework For m-Party PFE of Circuits

Definition 3 (The Two-party OEP Problem: $\mathsf{2\text{-}OEP}(\pi, \boldsymbol{x}, \boldsymbol{t})$**).** *In this problem, the first party* P_1 *holds an extended permutation* $\pi : \{1...M\} \rightarrow \{1...N\}$ *for two positive integers* M *and* N, *and a blinding vector* $\boldsymbol{t} = (t_1, \ldots, t_N)$ *while the second party* P_2 *holds a vector of inputs* $\boldsymbol{x} = (x_1, \ldots, x_M)$. *Both the* x_i*s and*

t_is are ℓ-bit strings where ℓ is a positive integer. At the end of the protocol, P_2 learns $(x_{\pi^{-1}(1)} \oplus t_1, \ldots, x_{\pi^{-1}(N)} \oplus t_N)$[2], while P_1 does not learn anything.

4.1 A New OEP Protocol

Next, we design a novel OEP protocol that improves on the efficiency of the above constructions. First, we show how to efficiently implement any extended permutation using a switching network. Then, we design a new and efficient protocol for oblivious evaluation of a switching network (OSN).

Building EPs out of Switching Networks. We first show how to construct an extended permutation using a switching network. Note that in a switching network, the number of inputs and outputs are the same which is in contrast to an extended permutation. Since for circuits we only deal with the case of $N \geq M$, the switching network we build for simulating an extended permutation $\pi : \{1...M\} \rightarrow \{1...N\}$, takes M real inputs of the EP and $N-M$ additional *dummy inputs*.

We divide the switching network into three components: (i) dummy-value placement, (ii) replication, and (iii) permutation (See Figure 5). Each component takes the output of the previous one as input.

Dummy-value Placement Component. takes the real and dummy values as input and for each real input that is mapped to k different outputs according to π, outputs the real value followed by $k-1$ dummy values. This is repeated for each real value. This process can be efficiently implemented using a Waksman permutation network.

Replication Component. takes the output of the previous component as input. It directly outputs each real value but replaces each dummy input with the real input that precedes it. Each replacement can be implemented using a 1-switch (with a single selection bit) choosing between rows 1 and 3 of Figure 5 (a), as discussed in Section 2. The entire replication phase can be implemented using $N-1$ such switches. At the end of this step, we have the necessary copies for each real input and the dummy inputs are eliminated.

Permutation Component. takes the output of the replication component as input and permutes each element to its final location as prescribed by π. Once again, this can be efficiently implemented using a Waksman permutation network.

Size of the Switching Network for an EP. Adding up the three components, the total number of 1-switches needed to implement the extended permutation described above is $2(N \log N - N + 1) + N - 1 = 2N \log N - N + 1$.

Oblivious Evaluation of Switching Networks (OSN). Next, we design a new and efficient protocol for oblivious evaluation of a generalized switching network. In this problem, P_2 holds the input vector $\boldsymbol{x}$ while P_1 holds the selection bits

[2] For simplicity we use XOR as the blinding function but one can replace XOR with any other natural blinding function.

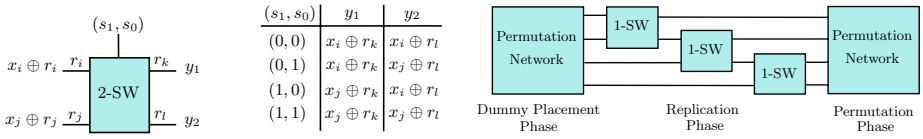

Fig. 5. (a) A 2-Switch (Left), (b) A Switching Network for an EP (Right)

into the switching network, and a blinding vector $\boldsymbol{t}$. P_2 learns the output of the network on his vector $\boldsymbol{x}$ blinded using vector $\boldsymbol{t}$. We start with a high level overview. A complete description appears in the full version.

Secure Evaluation of a Single 2-Switch. The idea can be best explained by describing the procedure for secure evaluation of a single 2-switch u in the network. Consider a 2-switch with input wires w_i and w_j and output wires w_k and w_l. P_2 assigns four uniformly random values r_i, r_j, r_k, r_l to the four wires. P_1 holds the blinded values $x_i \oplus r_i$ and $x_j \oplus r_j$ for the two input wires. The goal is to let P_1 learn the blinded values for the output wires (see Figure 5). Particularly, depending on the value of his two selection bits $s_0(u)$ and $s_1(u)$, P_1 learns one of the four possible output pairs: $(x_i \oplus r_k, x_j \oplus r_l)$, $(x_i \oplus r_k, x_i \oplus r_l)$, $(x_j \oplus r_k, x_i \oplus r_l)$, or $(x_j \oplus r_k, x_j \oplus r_l)$.

To implement this step, P_2 creates a table with four rows: $(r_i \oplus r_k, r_j \oplus r_l)$, $(r_i \oplus r_k, r_i \oplus r_l)$, $(r_j \oplus r_k, r_i \oplus r_l)$, and $(r_j \oplus r_k, r_j \oplus r_l)$. Then, P_1 and P_2 engage in a 1-out-of-4 oblivious transfer in which P_2's input is the four rows of the table he just created, and P_1's input is his two selection bits for the switch u. Without loss of generality suppose that P_1's selection bits are 0, and 0. Hence, P_1 retrieves the first row in the table, i.e. $(r_i \oplus r_k, r_j \oplus r_l)$. He then XORs $x_i \oplus r_i$ and $r_i \oplus r_k$ to recover $x_i \oplus r_k$ and XORs $x_j \oplus r_j$ and $r_j \oplus r_l$ to recover $x_j \oplus r_l$, i.e. the blinded values for the output wires.

Evaluating the Entire Switching Network. The above protocol can be extended to securely evaluate the entire switching network in constant round. In an *offline stage*, P_2 generates a set of random values for every wire in the network, and computes a table for each as described above. Then, P_1 and P_2 engage in a series of parallel 1-out-of-4 oblivious transfers, one for each switch, where P_1 learns a single row of each table according to his selection bits.

In the *online stage*, P_2 blinds his input vector using the randomness for the input wires, and sends them to P_1. P_1 now has all the information necessary to evaluate the switches in the network in a topological order, and recover the blinded values for the output wires (at this stage, P_1 locally performs a sequence of XORs discussed above). He then applies an additional layer of blinding using his random vector $\boldsymbol{t}$, and returns the result to P_2. P_2 can remove his own blinding (i.e. the randomness he generated for the output wires in the network) to learn the output of the switching network blinded only with P_1's vector $\boldsymbol{t}$.

The above OSN protocol runs in a constant number of rounds and requires one invocation of an oblivious transfer per switch in the network. We omit the proof of the following theorem.

Theorem 2. *In the OT-hybrid model, the above OSN protocol (and the resulting OEP) is secure against semi-honest adversaries.*

Efficiency of the New OEP. We can now evaluate the efficiency of the OEP protocol that results from applying our OSN construction to the switching network corresponding to an EP. As discussed earlier, the total number of switches needed to implement an extended permutation $\pi : \{1...M\} \rightarrow \{1...N\}$ is $2N \log N - N + 1$. Furthermore, we only need to use 1-switches to implement an EP which means we only need 1-out-of-2 OT as opposed 1-out-of-4 OT. This yields an OEP protocol with $O(k)$ public-key operations and $4N \log N - 2N + 2$ symmetric-key operations. The communication of the protocol is dominated by $O(N \log N)$ hash values.

How OSN Realizes $\mathcal{F}_{\mathcal{CTH}}$ Queries. It remains to show how our OSN implementation of OEP realizes the queries in $\mathcal{F}_{\mathcal{CTH}}$. While it is obvious that our OSN protocol securely performs all the OMAP/Reveal queries combined, for it to fully satisfy the CTH, we need the ability to make these queries on-demand (see full version for details).

5 Efficient PFEs from MPC

5.1 Multi-Party Private Function Evaluation

In this section we apply our framework to the seminal GMW protocol to obtain a multi-party PFE variant. In particular, we need to describe how the CTH and the PGE functionalities are designed and then plug them into the framework to obtain the desired multiparty PFE. We implement the PGE functionality by means of a *multi-party private gate evaluation* (m-XOR-PGE(G, a, b)) protocol. In such a protocol, only P_1 knows the functionality of the gate G while each party holds his XOR share of the input bits a and b and obtains his XOR share of the output bit $G(a, b)$. See full version, for an efficient instantiation based on oblivious transfer. The protocol requires the same number of OTs as a single gate evaluation in the standard GMW. Hence, making the gate functionality private comes for free in terms of computation or communication.

For the CTH functionality, we can use the multiparty variant of either the HE-OEP or the SN-OEP constructions discussed earlier, where each party uses his XOR shares of the outgoing wires as input to the OEP and obtains his share of the value for the incoming wires.

The following theorem is implied by the security of our framework (Theorem 1), secure instantiations of the OEP and the PGE functionalities and a standard sequential composition theorem [23].

Theorem 3. *Given that the OEP and* m-XOR-PGE *protocols are secure against semi-honest adversaries, the Multi-Party PFE protocol based on our framework is also secure against semi-honest adversaries.*

Efficiency. The resulting protocol requires a single invocation of the m-OEP protocol (even though the protocol is executed in an on-demand fashion), and one invocation of the m-XOR-PGE per gate. Using the HE-OPE instantiation, we obtain a protocol with linear complexity (linear number of exponentiations), and using the SN-OPE, we obtain a protocol that uses $O(m^2g+mg\log g)$ invocations of OT ($O(m^2g)$ for the PGE and $O(mg\log g)$ for the OEP). The number of rounds is equal to the number of gates since they are evaluated sequentially.

5.2 Private Function Evaluation for Arithmetic Circuits

PGE *for Arithmetic Circuits.* Let $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a semantically secure and additively homomorphic encryption scheme. Suppose $a = [a]_1 + [a]_2$ and $b = [b]_1 + [b]_2$ are the inputs to the gate, and $c = [c]_1 + [c]_2$ is the output of the gate (where the addition occurs over the domain of plaintexts for the encryption scheme). $[a]_i, [b]_i, [c]_i$ are the shares of P_i. In order to hide the functionality of the gate, we design a PGE protocol in which P_2's actions are independent of the functionality of the gate (i.e. addition or multiplication). To achieve this, P_2 sends to P_1 encryption of $[a]_2, [b]_2$, and $[a]_2[b]_2$. Given these three ciphertexts, P_1 can compute an encryption of both the sum and the product of a and b using homomorphic properties of the scheme. He then sends an encrypted random shares of the outcome to P_1 to decrypt (See Full version for details). It is easy to see that the protocol is secure again semi-honest adversaries if the encryption scheme is semantically secure. We omit the proof of the following theorem.

Theorem 4. *Given* $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *a semantically secure encryption scheme,* 2-Arith-PGE *protocol is secure against semi-honest adversaries.*

We plug in the above PGE and our HE-OEP protocols in our general framework to obtain an efficient and secure 2PC for arithmetic circuits with *linear complexity.* The following theorem is implied by the security of our framework (Theorem 1), secure instantiations of the OEP and the PGE functionalities and a standard sequential composition theorem [23].

Theorem 5. *Given that the OEP and* 2-Arith-PGE *protocols are secure against semi-honest adversaries, the 2-Party Arithmetic PFE protocol based on our framework is secure against semi-honest adversaries.*

Efficiency. Each PGE invocation requires a constant number of public-key operations adding up to a total of $O(g)$ public-key operations. The HE-OEP has a linear complexity leading to a PFE protocol with similar complexity. The number of rounds is equal to the number of gates since they are evaluated sequentially.

5.3 A Constant-Round Two-Party PFE

In this section we apply the PFE framework to Yao's garbled circuit protocol. We only describe the high level ideas here. A full description of the protocol (2-PFE) appears in the Full version of the paper [21]. At first sight, it may not

be obvious how to interpret the sharing mechanism in Yao's protocol. But a closer look at the garbling and evaluation steps reveals that the bit value a for a wire in the circuit is shared by having P_2 (garbler) hold the mapping of a pair of random keys to their bit value ($k^0 \rightarrow [a]_2$,$k^1 \rightarrow \overline{[a]_2}$), and P_1 (the evaluator) holding one of the two keys ($k^{[a]_1}$). Note that one may wonder why we do not simplify the sharing scheme by always letting $[a]_2 = 0$. But such a sharing would indeed be insecure in our PFE framework, and more specifically would allow the evaluator to learn values for the intermediate wires as he evaluates the circuit (since he creates and knows the mapping of keys). Making the CTH component work with this sharing scheme turns out to be the main technical difficulty in designing an efficient Yao-base PFE.

General Idea. Recall Yao's garbled circuit protocol in the semi-honest case. In our construction, the evaluator is the party who holds the circuit, while we intend to hide the circuit from the garbler. We need to hide the topology of the circuit from him using the CTH functionality: first, the Garbler generates his own random shares for the output wires of all the gates in the circuit (i.e. the permuted garbled key pairs for all those wires). Next, he sends all his shares to the CTH functionality, and receives his output which are his shares for the input wires to all the gates in the circuit (i.e. garbled key pairs for all those wires). The garbler now has all garbled keys he needs to garble the circuit. If we assume that all the gates are NAND, there would be no need to hide the gates functionalities. Therefore, our $\mathcal{F}_{\mathcal{PGE}}$ functionality realization consists of the normal garbling of the gates by the garbler and the standard evaluation of the gates by the evaluator. Next, we go into the details of each component and address some of the subtleties that arise.

PGE *Realization.* Realization of the $\mathcal{F}_{\mathcal{PGE}}$ functionality is simple. Lets assume that the inputs are shared using the above sharing scheme. P_2 first randomly generates his own share of the output wire for the current gate, which is basically generating two random keys and assigning them to bits zero and one. He then sends his share to CTH functionality. Upon receiving his shares for input wires to the gates, from CTH functionality, P_2 garbles each gate using his shares for the input and output wires of the gate. He then sends the garbled gates to P_1 who can use his own share of the input wires to ungarble a single row and learn his own share of the output wire.

We now need to integrate our CTH realization with the above PGE construction. For this to work, we need to modify our standard CTH realization, particularly to make sure that its outputs are fresh shares based on the sharing scheme above (i.e. $[a]_1$ and $[a]_2$, and the key pair are fresh and random).

CTH *Realization.* During the evaluation, P_1 needs to XOR his share with its corresponding blinding value(s) to obtain his correct input share for evaluating the next garbled gate. But observing which blinding value enables correct decryption of the next garbled gate (potentially) reveals the value of that intermediate wire. To avoid this issue, we need to ensure that the shares generated by the CTH are truly random. In particular, we need to ensure that P_1 cannot associate the first blinding

with key 0 and the second blinding with key 1. As a first solution, P_2 randomly swaps the key pairs to prevent such association by P_1.

P_2 Swaps Each Key Pair Randomly. We solve this problem by having P_2 swap each key pair randomly and independently (using a random bit-vector $\boldsymbol{v}$) before using them in the OMAP queries (for the CTH). Each pair should be swapped using a different bit since using the same bit would reveal whether the bit values for certain intermediate wires are the same or not. If the first(second) blinding is used for two or more wires we learn that their value is the same, though we don't know if it zero or one. This solves the issue above, but undermines correctness of the protocol. When P_2 sends the swapped key pairs to P_1, he gets back an extended permuted (and blinded) set of key pairs. As a result, P_2 does not know the correct order for each pair, and will not be able to perform the garbling of the gates without knowing which key is for 0 and which is for 1.

P_1 and P_2 Jointly Swap Each Key Pair Into Its Original Form. A naive fix would be to attach each "swapping bit" to its corresponding key pair as it goes through the CTH, and reveal the bit to P_2 as part of the output of the CTH, who then uses it to swap the key pair back to its original order. But this would allow P_2 to learn some information about $\pi_{\mathcal{C}}$ (and the topology of the circuit) by comparing the swapping bits in the input and output key pairs for the CTH.

To address this issue, P_1 and P_2 perform this step together, each holding an XOR share of swapping bits. In particular, the random bit vector $\boldsymbol{v}$ will be fed to the CTH, but P_2 only learns a blinded version, i.e. $v''_i = v_{\pi^{-1}(i)} \oplus v'_i$ for $1 \leq i \leq 2g$, where the blinding vector $\boldsymbol{v'} = (v'_1, \ldots, v'_{2g})$ is only known to P_1. To swap each key pair back to its original order, P_1 first swaps the pair using v'_i, and sends it to P_2. P_2 then swaps it one more time using v''_i which puts the key pair back in its original order. Of course, at this point, the key shares are fresh and random.

If we use a homomorphic-based OEP, this solution is sufficient, but when using the CTH functionality in a black-box way, and particularly when using our SN-OEP construction, there is one more issue to address. The described solution does not use the OEP in a black-box fashion, since P_1 needs to swap the outcome using $\boldsymbol{v'}$, before sending it to P_2. But if the pair is swapped using a random bit vector not known to P_2, he cannot use the appropriate random values to unblind the result (recall the final step of the OEP where P_2 removes his blinding from the output).

P_1 Does his Swapping Using an OSN Protocol. To handle this problem, we require that P_1's swapping procedure based on the bit-vector $\boldsymbol{v'}$ takes place as part of an oblivious switching network evaluation where the v'_is are P_1's selection bits to the network. This requires the use of an additional layer of switches attached to the original switching network for the OEPs. This also has the advantage of making the usage of the OEP and the OSN protocols black-box.

When using our SN-OEP in the above construction, the total number of symmetric operations required for the protocol is $8g \log 2g + 5g + 2$. We discuss our efficiency in detail in the Full version [21] where we also prove the following theorem.

Theorem 6. *Given that the OSN and the OEP protocols are secure against semi-honest adversaries, and that Yao's protocol uses a symmetric-key encryption with related-key security, the* 2-PFE *protocol is secure against semi-honest adversaries.*

References

1. Abadi, M., Feigenbaum, J.: Secure circuit evaluation. Journal of Cryptology 2, 1–12 (1990)
2. Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008)
3. Sadeghi, A.R., Schneider, T.: Generalized universal circuits for secure evaluation of private functions with application to data classification. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 336–353. Springer, Heidelberg (2009)
4. Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 556–571. Springer, Heidelberg (2011)
5. Schneider, T.: Practical secure function evaluation (2008)
6. Valiant, L.: Universal circuits (preliminary report). In: Proceedings of the Eighth Annual ACM STOC, pp. 196–203 (1976)
7. Shpilka, A., Yehudayoff, A.: Arithmetic circuits: A survey of recent results and open questions (2010)
8. Raz, R.: Elusive functions and lower bounds for arithmetic circuits. In: Proceedings of the 40th Annual ACM STOC (2008)
9. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM, STOC 2009, pp. 169–178. ACM (2009)
10. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: Proceedings of 19th NDSS Conference (2012)
11. Wang, G., Luo, T., Goodrich, M.T., Du, W., Zhu, Z.: Bureaucratic protocols for secure two-party sorting, selection, and permuting. In: Proceedings of the 5th ACM ASIACCS, pp. 226–237 (2010)
12. Du, W.: A Study of Several Specific Secure Two-party Computation Problems. PhD thesis, Department of Computer Sciences, Purdue University (2001)
13. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth Annual ACM, STOC 1987, pp. 218–229. ACM (1987)
14. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science, pp. 162–167 (October 1986)
15. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–299. Springer, Heidelberg (2001)
16. Franklin, M., Gondree, M., Mohassel, P.: Multi-party indirect indexing and applications. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 283–297. Springer, Heidelberg (2007)
17. Choi, S.G., Hwang, K.-W., Katz, J., Malkin, T., Rubenstein, D.: Secure multiparty computation of boolean circuits with applications to privacy in on-line marketplaces. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 416–432. Springer, Heidelberg (2012)

18. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (2012)
19. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
20. Beaver, D.: Precomputing oblivious transfer. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 97–109. Springer, Heidelberg (1995)
21. Mohassel, P., Sadeghian, S.: How to hide circuits in mpc: An efficient framework for private function evaluation (2013), http://eprint.iacr.org/
22. Waksman, A.: A permutation network. J. ACM 15, 159–163 (1968)
23. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology 13(1), 143–202 (2000)