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 $\mathcal{C}(x_1, \dots, x_n)$ 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.

- In the *multiparty* case with dishonest majority, we apply our techniques to the seminal GMW protocol [GMW87] and obtain the first general-purpose PFE with *linear complexity* in the circuit size.
- In the *two-party* case, we transform Yao's garbled circuit protocol [Yao86] into a constant-round two-party PFE. Depending on the instantiation of the underlying subprotocol, we either obtain a two-party PFE with linear complexity that improves on the only other work with similar asymptotic efficiency (Katz and Malka, ASIACRYPT 2011 [KM11]), or a two-party PFE that provides the best concrete efficiency to date despite not being linear.
- The above two constructions are for boolean circuits. In case of *arithmetic circuits*, we obtain the first PFE with linear complexity based on any additively homomorphic encryption scheme.

Though each construction uses different techniques, a common feature in all three is that the overhead of hiding the circuit $\mathcal{C}$ is essentially equal to the cost of running the OEP protocol on a vector of size $|\mathcal{C}|$. As a result, to improve efficiency, one can focus on lowering the cost of the underlying OEP protocol. OEP can be instantiated using a singly homomorphic encryption or any general-purpose MPC but we introduce a new construction that we show is significantly more efficient than these alternatives, in practice. The main building block in our OEP construction is an efficient protocol for *oblivious switching network evaluation* (OSN), a generalization of the previously studied oblivious shuffling problem which is of independent interest. Our results noticeably improve efficiency of the previous solutions to oblivious shuffling, yielding a factor of 25 or more gain in computation and communication.

[*] email address: `pmohasse@cpsc.ucalgary.ca`

[†] email address: `sadeghis@ucalgary.ca`

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.[1] 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 special and general-purpose private function evaluation protocols [AF90, KM11, GHS10, SS09, PSS09, BFK$^+$09, KS08a, IP07, BPSW07].

Solutions Based on Universal Circuits. Most general-purpose PFE solutions reduce the problem to secure computation of a *universal circuit* U_g that 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 [KS08a, Sch08, SS09].

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 [Val76] showed a construction of a boolean universal circuit achieving an optimal circuit size of $|U_g| \approx 19g \log g$. Kolesnikov and Schneider [KS08a] 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 [SY10] and [Raz08]), 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 [Gen09]. While asymptotically optimal, this solution is not practical due to its high computational cost. Recently, Katz and Malka [KM11] 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.

1.1 Our Contribution

Practical design and implementation of MPC has been the subject of active research in the last few years. These efforts have, in part, lead to the introduction of several software implementations and MPC frameworks [MNPS04, PSSW09, HEKM11, KSS12]. As discussed above, however, when it comes to PFE the

[1] The traditional definition of PFE assumes that only P_i $(i \geq 2)$ holds an input to the function, but this can naturally be generalized to the case where P_1 also holds a private input.

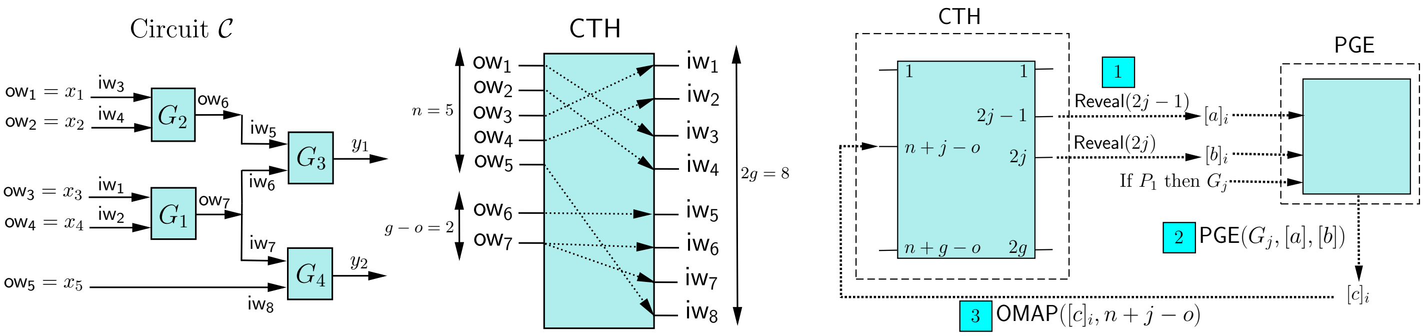


Figure 1: (a) An example circuit and the corresponding mapping (b) Steps of framework for party i and the jth gate in a topological order.

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 $\pi_{\mathcal{C}} : \{1 \dots |\mathsf{OW}|\} \rightarrow \{1 \dots |\mathsf{IW}|\}$ 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(a) 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 1(b).

Oblivious Shuffling Protocols	Asymptotic Complexity	Concrete Efficiency Gain
HE-Based	$O(N)$ Asym.	175
Garbled Circuit-Based [HEK12]	$(\frac{4\ell(N\log N-N+1)}{3}+2N\ell)$ Sym. + $O(k)$ Asym.	25
OSN-Based (our paper)	$(2N\log N-2N+2)$ Sym. + $O(k)$ Asym.	1

Table 1: Comparison of our OSN-based oblivious shuffling vs. HE-based and garbled circuit-based constructions. Let N denote the number of elements being permuted, ℓ be the length of each element and k is the security paremeter. The last column shows some concrete efficiency gain of our OSN construction over the corresponding construction for $128 \leq N \leq 8192$, $\ell = 32$. These numbers are based on experiments in [HEK12]. We expect similar gains for larger values of N too.

Efficient Realizations of $\mathcal{F}_{\mathcal{CTH}}$. We refer to the mapping $\pi_{\mathcal{C}} : \{1 \ldots |\mathsf{OW}|\} \to \{1 \ldots |\mathsf{IW}|\}$ discussed above as an *extended permutation* (EP) since it not only permutes the elements in $\{1 \ldots |\mathsf{OW}|\}$, 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: the first party holds π_C and a blinding[2] vector $\vec{t}$ of size $|\mathsf{IW}|$, while the second party holds an input vector $\vec{x}$ of size $|\mathsf{OW}|$. Their goal is to let the second party learn the output of π_C applied to $\vec{x}$, blinded by $\vec{t}$. Neither party should learn anything else. OEP can be instantiated using a singly homomorphic encryption, or any general-purpose 2PC. As discussed in Section 4, 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. Such a network is a set of interconnected switches where each switch maps two inputs to two outputs, using at most two selection bits. This is a generalization of the well-known permutation networks since the network uses four different switch types (as opposed to two), as specified by its two selection bits. 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 $\vec{t}$, while the other party P_2 holds the input vector $\vec{x}$ to the SN. The goal is for P_2 to learn the output of SN applied to the input vector $\vec{x}$, blinded by $\vec{t}$. 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.

The resulting OEP is more efficient than the homomorphic-based solutions since the number of exponentiations can be made independent of the size of the network (using OT extension), and is more efficient than the Yao-based one since the complexity of our construction does not grow with the number of bits needed to represent each input element. We also need a *multiparty variant of our OEP protocol* where the mapping is known to a single party while the input vector $\vec{x}$ and the blinding vector $\vec{t}$ are shared among the players. We show how to construct such an m-party OEP protocol via m invocations of the two-party version.

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 [HEK12] (a subprotocol used for private set intersection), or secure two-party permutation [WLG+10, Du01]. Our new construction yields more efficient solutions to these problems as well, improving on the previous proposals based on garbled circuit implementation of sorting networks, permutation networks, or randomize shell sort [HEK12, WLG+10, Du01]. See Table 1 for concrete (a factor of 25 or more improvement over best previous solutions) and asymptotic efficiency comparisons with previous work.

Applying our Framework to Existing MPC. We apply the above framework to the GMW protocol [GMW87], Yao's garbled circuit protocol [Yao86], and secure computation of arithmetic circuits via

[2] 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.

Multi-Party PFE	Complexity
[KS08a] Universal Circuits	$O(m^2 g \log^2 g)$ Sym. + $O(k)$ Asym.
[Val76] 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.

Table 2: Comparing our m-party PFE with the generic solution of applying GMW to the universal circuit constructions of [KS08a] and [Val76]. g denotes the number of gates, k denotes the security parameter.

homomorphic encryption [CDN01]. 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 [GMW87] 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 [FGM07]. 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 [CHK^{+}12, NNOB21], is that it only uses *oblivious transfers*. As a result, one can use OT extension [IKNP03] and pre-processing techniques [Bea95] 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.

More Efficient Two-party PFE. We also design a constant round two-party PFE based on Yao's garbled circuit protocol [Yao86]. 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 [KM11] 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 [KM11] (see Section 5.4 for details on the efficiency gain), 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 [KS08a] based on universal circuits. It is concretely more efficient than Valiant's construction [Val76]. Table 3 summarizes concrete efficiency comparison of our two-party PFE with all previous constructions. In appendix J.1 we show that our construction concretely improves over all previous construction 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 [CDN01], 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).

2-Party PFE	Complexity	Concrete Efficiency Gain
[KS08a]	$1.5g \log^2 g$ sym. + $O(k)$ Asym.	3-6
[Val76]	$19g \log g$ sym. + $O(k)$ Asym.	2
[KM11]	$O(g)$ 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

Table 3: Comparison of our 2-party PFE protocols with previous works. (HM: Homomorphic Multiplication, HA: Homomorphic Addition, HE: Homomorphic Encryption). Last column shows concrete gain of our 2-PFE over universal circuit approaches for benchmark circuits, AES, RSA and Edit-distance (refer to section J.1 for more detailed discussion). g denotes the number of gates, and k is the security parameter.

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 $\vec{v}$. 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.

Homomorphic Encryption. We use a semantically-secure public-key encryption scheme $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with public key pk that allows for simple computations on encrypted data. We require the scheme to be additively homomorphic and denote the addition operation by $+_h$ such that $\mathsf{Enc}_{pk}(m_1 + m_2) = \mathsf{Enc}_{pk}(m_1) +_h \mathsf{Enc}_{pk}(m_2)$. Furthermore, given $\mathsf{Enc}_{pk}(m)$ and a plaintext c,there exist an efficient operations denoted by $\times_h$ such that $\mathsf{Enc}_{pk}(cm) = c \times_h \mathsf{Enc}_{pk}(m)$.

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 2.1 (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 [Wak68]. The main theorem of [Wak68] 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 [Wak68] 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 given in Appendix A.

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.

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 as discussed in Appendix G. Figure 2 describes the CTH functionality more formally.

The role of vectors $\vec{k}$ is to prevent P_1 from learning the other parties' shares and the role of vectors $\vec{t}$ 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 given in Figure 3.

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 $\mathcal{F}_{\mathcal{CTH}}$ 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 $\vec{t_i}$, $1 \leq i \leq m$, where $\vec{t_i} =< t_i[1], \ldots, t_i[2g] >$. P_i for $2 \leq i \leq m$ generates a random key vector $\vec{k_i} =< k_i[1], \ldots, k_i[2g] >$.

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

- **P_1's Input:** $\pi_{\mathcal{C}}$, $\vec{t}_1, \ldots, \vec{t}_m$.
- **P_i's ($1 \leq i \leq m$) Input:** $[x]_i$, $\vec{k}_i$, 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 $[x]_i \otimes t_i[j]$.)

Figure 2: The Circuit-Topology Hiding Functionality ($\mathcal{F}_{\mathcal{CTH}}$)

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. $[c]_i = [G(a, b)]_i$

Figure 3: The Private Gate Evaluation Functionality ($\mathcal{F}_{\mathcal{PGE}}$)

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.

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

the CTH functionality and repeat the process until all gates are evaluated.A visual demonstration of the steps appears in Figure 1(b). Proof of the following theorem is given in Appendix B.

Theorem 3.1. *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 4.1 (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 Appendix F, 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).

Definition 4.2 (The Two-party OEP Problem: $2\text{-}\mathsf{OEP}(\pi, \vec{x}, \vec{t})$)**.** *In this problem, the first party P_1 holds an extended permutation $\pi : \{1...M\} \to \{1...N\}$ for two positive integers M and N, and a blinding vector $\vec{t} = (t_1, \ldots, t_N)$ while the second party P_2 holds a vector of inputs $\vec{x} = (x_1, \ldots, x_M)$. Both the x_is and t_is are ℓ-bit strings where ℓ is a positive integer. At the end of the protocol, P_2 learns $(x_{\pi^{-1}(1)} \oplus t_1, \ldots, x_{\pi^{-1}(N)} \oplus t_N)$*[3]*, while P_1 does not learn anything.*

Existing Solutions and Their Efficiency. The two existing solutions for OEP are based on general-purpose MPC (e.g. Yao's garbled circuit) and based on any additively homomorphic encryption. We refer the reader to Appendix C for more detail on these solutions and their efficiency. Asymptotically speaking, the construction based on homomorphic encryption has linear complexity and hence is superior to the Yao-based one. Interestingly, however, the Yao-based solution seems to be the more efficient of the two in practice, for many values of interest for N and ℓ. In particular, the experiments in [HEK12] show that despite the asymptotic efficiency of the solution based on homomorphic encryption, for values of $128 < N < 8192$, and with $\ell = 32$, the garbled circuit implementation of the Waksman network is more than 7 times faster than the homomorphic-based shuffling (See Section 7.2 of [HEK12]). Though the above experiments were done for the special case of oblivious shuffling, the same efficiency analysis holds in the more general case of OEP.

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 $\pi : \{1...M\} \to \{1...N\}$, 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.

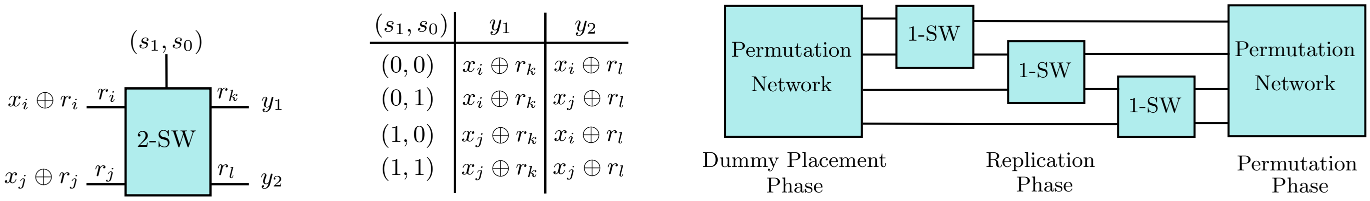
- **Dummy-value placement component.** takes the real and dummy values as input and for each real input that is mapped to k different outputs according to π, outputs the real value followed by $k-1$ dummy values. This is repeated for each real value. This process can be efficiently implemented using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

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

- **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 of 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 $\vec{x}$ while P_1 holds the selection bits into the switching network, and a blinding vector $\vec{t}$. P_2 learns the output of the network on his vector $\vec{x}$ blinded using vector $\vec{t}$. We start with a high level overview. A complete description appears in Figure 8 of Appendix D.



(s_1, s_0)	y_1	y_2
$(0,0)$	$x_i \oplus r_k$	$x_i \oplus r_l$
$(0,1)$	$x_i \oplus r_k$	$x_j \oplus r_l$
$(1,0)$	$x_j \oplus r_k$	$x_i \oplus r_l$
$(1,1)$	$x_j \oplus r_k$	$x_j \oplus r_l$

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

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 $(r_j \oplus r_k, r_j \oplus r_l)$ as shown in step 4 of Figure 8. 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 $\vec{t}$, 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 $\vec{t}$.

The above OSN protocol runs in a constant number of rounds and requires one invocation of an oblivious transfer per switch in the network. See Appendix D for a more detailed discussion of its efficiency. We also prove the following theorem in Appendix E.

Theorem 4.3. *In the OT-hybrid model, the OSN protocol of Figure 8 (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 $\pi : \{1...M\} \to \{1...N\}$ 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.

Both our protocol and the Yao-based solution are constant round, and when OT extension is used, require the same number of public-key operations $O(k)$.[4] However, the number of symmetric-key operations in our construction for a single Waksman network is $2N \log N - 2N + 2$ compared to the garbled circuit approach of [HEK12] which requires $4\ell(N \log N - N + 1)/3 + 2N\ell$ symmetric-key operations (assuming each non-free gate requires four symmetric-key operations)[5]. This is a significant improvement specially as one considers larger values of ℓ. For example for $\ell = 32$ (which is used for comparison in [HEK12]) our OSN protocol implementing the Waksman network, is approximately **25** times faster. Considering that the Yao-based solution is 7 times faster than HE-based shuffling, we obtain a factor of **175** efficiency gain over the HE-based approach. The bandwidth usage of our construction is also better by the same factors. An additional advantage of our construction is a *very cheap online phase* that does not require any cryptographic operations and only consists of XOR operations. For the same parameters, our SN-OEP construction is **161** times faster than HE-based OEP. For a detailed discussion refer to Appendix D.

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. The On-demand property of the OSN protocol is discussed in Appendix G.

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 $G(a, b)$. In Appendix H we show a simple construction for this problem based on any multiparty OT (e.g. see [FGM07]).

[4] The Yao-based solution requires $O(N\ell)$ OTs. Hence, as long as $N\ell > k$, the number of public-key operations in the Yao-based solutions and our protocol are the same.

[5] The numbers are derived from the size of optimized circuits for Waksman networks given in [HEK12]

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 3.1), secure instantiations of the OEP and the PGE functionalities and a standard sequential composition theorem [Can00].

Theorem 5.1. *Given that the OEP and* $\mathsf{m\text{-}XOR\text{-}PGE}$ *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 $\mathsf{m\text{-}OEP}$ protocol (even though the protocol is executed in an on-demand fashion), and one invocation of the $\mathsf{m\text{-}XOR\text{-}PGE}$ 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

In this section we apply the same framework to secure 2PC 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. A detailed description of the protocol appears in Appendix I. 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 5.2. *Given $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ a semantically secure encryption scheme,* $\mathsf{2\text{-}Arith\text{-}PGE}$ *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 3.1), secure instantiations of the OEP and the PGE functionalities and a standard sequential composition theorem [Can00].

Theorem 5.3. *Given that the OEP and* $\mathsf{2\text{-}Arith\text{-}PGE}$ *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 Figure 10. 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 ($k^0 \to [a]_2$,$k^1 \to \overline{[a]_2}$), 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 holding 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 garbled 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 gate 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).

Achieving constant round protocol. Given a correct CTH realization, it is possible to design a *constant-round* version of the protocol. Note that P_2's shares for all the outgoing wires can be generated by him at random and in the beginning of the protocol (without interaction with P_1). At this stage, parties can indeed execute the CTH functionality in its entirety for P_2 to learn his fresh random shares for the $2g$ incoming wires to the circuit. The only difference comparing to our framework is that P_1 does not learn his shares for the intermediate wires until later (in the evaluation). Also, note that P_2's portion of PGE is only based on his own randomly generated shares and thus can be done at once and without interaction with P_1 for each gate. In particular, he garbles all the gates using the keys he obtained from the CTH functionality and sends the garbled gates to P_1. P_1 then performs his portion of the gate evaluation, evaluating the gates in topological order to obtain his share of each incoming wire and eventually the output wires. During the evaluation (ungarbling) phase, when P_1 retrieves a garbled key for an output of a gate in the circuit, he needs to XOR it with its corresponding blinding value(s) to obtain his correct input share for evaluating

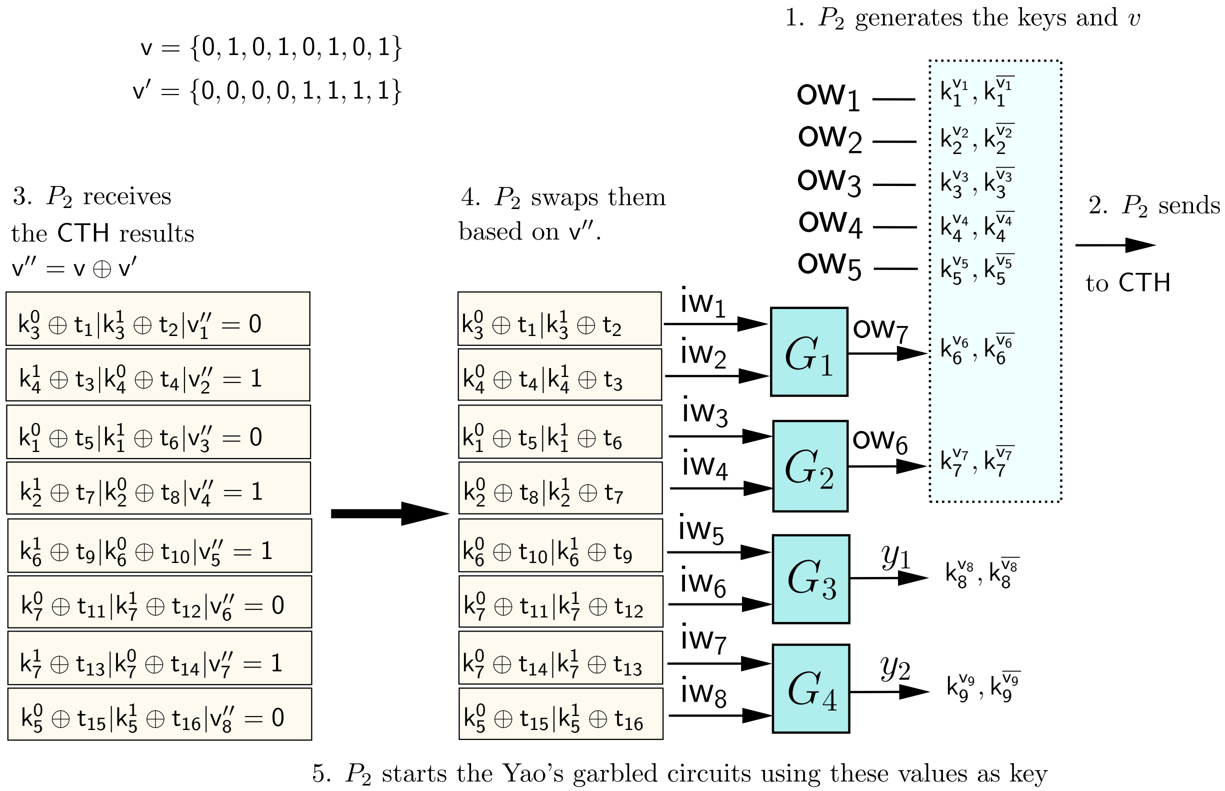

1. P_2 generates the keys and v
$\mathsf{v} = \{0,1,0,1,0,1,0,1\}$
$\mathsf{v}' = \{0,0,0,0,1,1,1,1\}$
ow_1
ow_2
ow_3
ow_4
ow_5
$\mathsf{k}_1^{\mathsf{v}_1}, \mathsf{k}_1^{\overline{\mathsf{v}_1}}$
$\mathsf{k}_2^{\mathsf{v}_2}, \mathsf{k}_2^{\overline{\mathsf{v}_2}}$
$\mathsf{k}_3^{\mathsf{v}_3}, \mathsf{k}_3^{\overline{\mathsf{v}_3}}$
$\mathsf{k}_4^{\mathsf{v}_4}, \mathsf{k}_4^{\overline{\mathsf{v}_4}}$
$\mathsf{k}_5^{\mathsf{v}_5}, \mathsf{k}_5^{\overline{\mathsf{v}_5}}$
2. P_2 sends
to CTH
3. P_2 receives the CTH results $\mathsf{v}'' = \mathsf{v} \oplus \mathsf{v}'$
4. P_2 swaps them based on v''.
$\mathsf{k}_3^0 \oplus \mathsf{t}_1 | \mathsf{k}_3^1 \oplus \mathsf{t}_2 | \mathsf{v}_1'' = 0$
$\mathsf{k}_4^1 \oplus \mathsf{t}_3 | \mathsf{k}_4^0 \oplus \mathsf{t}_4 | \mathsf{v}_2'' = 1$
$\mathsf{k}_1^0 \oplus \mathsf{t}_5 | \mathsf{k}_1^1 \oplus \mathsf{t}_6 | \mathsf{v}_3'' = 0$
$\mathsf{k}_2^1 \oplus \mathsf{t}_7 | \mathsf{k}_2^0 \oplus \mathsf{t}_8 | \mathsf{v}_4'' = 1$
$\mathsf{k}_6^1 \oplus \mathsf{t}_9 | \mathsf{k}_6^0 \oplus \mathsf{t}_{10} | \mathsf{v}_5'' = 1$
$\mathsf{k}_7^0 \oplus \mathsf{t}_{11} | \mathsf{k}_7^1 \oplus \mathsf{t}_{12} | \mathsf{v}_6'' = 0$
$\mathsf{k}_7^1 \oplus \mathsf{t}_{13} | \mathsf{k}_7^0 \oplus \mathsf{t}_{14} | \mathsf{v}_7'' = 1$
$\mathsf{k}_5^0 \oplus \mathsf{t}_{15} | \mathsf{k}_5^1 \oplus \mathsf{t}_{16} | \mathsf{v}_8'' = 0$
$\mathsf{k}_3^0 \oplus \mathsf{t}_1 | \mathsf{k}_3^1 \oplus \mathsf{t}_2$
$\mathsf{k}_4^0 \oplus \mathsf{t}_4 | \mathsf{k}_4^1 \oplus \mathsf{t}_3$
$\mathsf{k}_1^0 \oplus \mathsf{t}_5 | \mathsf{k}_1^1 \oplus \mathsf{t}_6$
$\mathsf{k}_2^0 \oplus \mathsf{t}_8 | \mathsf{k}_2^1 \oplus \mathsf{t}_7$
$\mathsf{k}_6^0 \oplus \mathsf{t}_{10} | \mathsf{k}_6^1 \oplus \mathsf{t}_9$
$\mathsf{k}_7^0 \oplus \mathsf{t}_{11} | \mathsf{k}_7^1 \oplus \mathsf{t}_{12}$
$\mathsf{k}_7^0 \oplus \mathsf{t}_{14} | \mathsf{k}_7^1 \oplus \mathsf{t}_{13}$
$\mathsf{k}_5^0 \oplus \mathsf{t}_{15} | \mathsf{k}_5^1 \oplus \mathsf{t}_{16}$
iw_1
iw_2
iw_3
iw_4
iw_5
iw_6
iw_7
iw_8
G_1
G_2
G_3
G_4
ow_7
ow_6
$\mathsf{k}_6^{\mathsf{v}_6}, \mathsf{k}_6^{\overline{\mathsf{v}_6}}$
$\mathsf{k}_7^{\mathsf{v}_7}, \mathsf{k}_7^{\overline{\mathsf{v}_7}}$
y_1
$\mathsf{k}_8^{\mathsf{v}_8}, \mathsf{k}_8^{\overline{\mathsf{v}_8}}$
y_2
$\mathsf{k}_9^{\mathsf{v}_9}, \mathsf{k}_9^{\overline{\mathsf{v}_9}}$
5. P_2 starts the Yao's garbled circuits using these values as key

Figure 6: CTH realization of our 2-PFE protocol, applied to the example of figure 1.

the next garbled gate. Since P_1 knows $\pi_{\mathcal{C}}$ and all the blinding values, he has the necessary information to perform this step. He then asks P_2 for his share of the output wires of the circuit (i.e. the translation table in standard Yao) to determine his actual output. P_1's shares of his inputs wires are delivered to him using one OT per wire (as done in standard Yao).

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 the first blinding to the 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 $\vec{v}$) before using them in the OMAP queries (for the CTH). Steps 1 and 2 of figure 6 demonstrate this fix. 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 $\vec{v}$ will be fed to the CTH (see step 4a of the protocol), 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 $\vec{v'} = (v'_1, \ldots, v'_{2g})$ 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 (step 3 of figure 6), 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 (step 4 of figure 6). Of course, at this point, the key shares are fresh and random.

  If we use a homomorphic-based OEP, this solution is sufficient (see section 5.4), 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 $\vec{v'}$, 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 $\vec{v'}$ 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 (see Figure 11 of Appendix J). This also has the advantage of making the usage of the OEP and the OSN protocols black-box.

We prove the following theorem in Appendix J.2.

Theorem 5.4. *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.*

5.4 Further Optimizations and Efficiency

Combining the switching networks into one. For ease of composition, and to make the constructions conceptually easier to understand, in Figure 10 we describe four separate switching networks, one for the OEP, and three for SN_0,SN_1, and SN_2. But since SN_0 and SN_1 implement π_c, we can use a single switching network that concatenates their inputs. As a result, the computation (number of oblivious transfers) reduces by a factor of two. The switching network SN_2 still needs to be evaluated separately.

A simpler protocol based on homomorphic encryption. As mentioned earlier, the switching network SN_2 is not needed when we implement the OSN using a homomorphic encryption scheme. The intuition is that P_2 can still decrypt the outcome of the OSN protocol without knowing the correct swapping bit used by P_1[6]. The resulting 2-PFE is more efficient than the protocol of Katz and Malka [KM11]. The main reason is that we blind each key by homomorphically adding a random value to it, while they need to apply a universal hashing procedure that requires both a homomorphic addition and a multiplication to blind each key. This is a noticeable improvement since homomorphic multiplication is often more

[6] Since the final unblinding step in the homomorphic-based OEP is simply to decrypt the obtained ciphertexts.

expensive as it requires exponentiation. The reason for the universal hashing is to allow P_1 to evaluate the garbled circuit without learning the values for the intermediate wires (the same hashing is applied to both keys in the pair). But, we solve this problem via use of swapping bit vectors $(\vec{v}, \vec{v'})$, which can be concatenated to the end of each key pair as it is being encrypted, hence avoiding any additional cost.

Efficiency of the SN-OEP instantiation. 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 the efficiency in detail in Appendix J.1 and show that our protocol is a factor of 142 more efficient than the PFE protocol of [KM11], , for wide range of circuit sizes. We also show that it is 2 times more efficient than applying Yao's garbled circuits to Valiant's universal circuit construction, and 3-6 times faster than applying Yao's garbled circuits to universal circuit construction of [KS08a], for the benchmark circuits such as AES, RSA and Edit-distance. For the latter two constructions, we assume that they take full advantage of free XOR gates and in particular that 3/4 of the gates in the Universal circuit are XOR gates that computed for free.

Acknowledgements. We would like to thank an Anonymous EUROCRYPT reviewer for very helpful discussions on efficiency of our two-party PFE and the comparison of its efficiency with the Valiant-based universal circuit approach.

References

[AF90] Martin Abadi and Joan Feigenbaum. Secure circuit evaluation. *Journal of Cryptology*, 2:1–12, 1990.

[AHI10] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. 2nd Symp. on Innovations in Computer Science (ICS), 2010. Available at `http://eprint.iacr.org/`.

[Bea95] Donald Beaver. Precomputing oblivious transfer. In Don Coppersmith, editor, *Advances in Cryptology CRYPT0 95*, volume 963 of *Lecture Notes in Computer Science*, pages 97–109. Springer Berlin / Heidelberg, 1995.

[BFK$^+$09] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 424–439, Berlin, Heidelberg, 2009. Springer-Verlag.

[BPSW07] J. Brickell, D.E. Porter, V. Shmatikov, and E. Witchel. Privacy-preserving remote diagnostics. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 498–507. ACM, 2007.

[Can00] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '01, pages 280–299, London, UK, UK, 2001. Springer-Verlag.

[CHK$^+$12] Seung Geol Choi, Kyung-Wook Hwang, Jonathan Katz, Tal Malkin, and Dan Rubenstein. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In *Proceedings of RSA 2012, Cryptographers' Track (CT-RSA)*, 2012.

[Du01] Wenliang Du. *A Study of Several Specific Secure Two-party Computation Problems*. PhD thesis, Department of Computer Sciences, Purdue University, 2001.

[FGM07] M. Franklin, M. Gondree, and P. Mohassel. Multi-party indirect indexing and applications. *Advances in Cryptology–ASIACRYPT 2007*, pages 283–297, 2007.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[GHS10] R. Gennaro, C. Hazay, and J. Sorensen. Text search protocols with simulation based security. *Public Key Cryptography–PKC 2010*, pages 332–350, 2010.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[Gol04] O. Goldreich. *Foundations of cryptography: Basic applications*. Cambridge Univ Pr, 2004.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Proceedings of 19th Network and Distributed Security Symposium (NDSS)*, 2012.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 35–35, Berkeley, CA, USA, 2011. USENIX Association.

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. *Advances in Cryptology-CRYPTO 2003*, pages 145–161, 2003.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, pages 575–594, Berlin, Heidelberg, 2007. Springer-Verlag.

[KM11] Jonathan Katz and Lior Malka. Constant-round private function evaluation with linear complexity. In Dong Lee and Xiaoyun Wang, editors, *Advances in Cryptology ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 556–571. Springer Berlin / Heidelberg, 2011.

[KS08a] Vladimir Kolesnikov and Thomas Schneider. Financial cryptography and data security. chapter A Practical Universal Circuit Construction and Secure Evaluation of Private Functions, pages 83–97. Springer-Verlag, Berlin, Heidelberg, 2008.

[KS08b] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *Automata, Languages and Programming*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer Berlin / Heidelberg, 2008.

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX conference on Security symposium*, Security'12, pages 14–14, Berkeley, CA, USA, 2012. USENIX Association.

[Lip05] H. Lipmaa. An oblivious transfer protocol with log-squared communication. In *Proceedings of the 8th Information Security Conference (ISC' 05)*, volume 3650, pages 314–328. Springer, 2005.

[LP07] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. pages 52–78. Springer, 2007.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22:161–188, 2009.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - a secure two-party computation system. In *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 20–20, Berkeley, CA, USA, 2004. USENIX Association.

[NNOB21] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. Advances in Cryptology–CRYPTO 2012, 2021.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, SODA '01, pages 448–457, 2001.

[PSS09] Annika Paus, Ahmad-Reza Sadeghi, and Thomas Schneider. Practical secure evaluation of semi-private functions. In *Proceedings of the 7th International Conference on Applied Cryptography and Network Security*, ACNS '09, pages 89–106, Berlin, Heidelberg, 2009. Springer-Verlag.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel Smart, and Stephen Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *Advances in Cryptology ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer Berlin / Heidelberg, 2009.

[PVW08] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. *Advances in Cryptology–CRYPTO 2008*, pages 554–571, 2008.

[Raz08] R. Raz. Elusive functions and lower bounds for arithmetic circuits. In *Proceedings of the 40th annual ACM symposium on Theory of computing.* ACM, 2008.

[Sch08] Thomas Schneider. Practical secure function evaluation, 2008.

[SS09] Ahmad-Reza Sadeghi and Thomas Schneider. Information security and cryptology — icisc 2008. chapter Generalized Universal Circuits for Secure Evaluation of Private Functions with Application to Data Classification, pages 336–353. Springer-Verlag, Berlin, Heidelberg, 2009.

[SY10] A. Shpilka and A. Yehudayoff. *Arithmetic circuits: A survey of recent results and open questions.* 2010.

[Val76] L.G. Valiant. Universal circuits (preliminary report). In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 196–203. ACM, 1976.

[Wak68] Abraham Waksman. A permutation network. *J. ACM*, 15:159–163, January 1968.

[WLG+10] Guan Wang, Tongbo Luo, Michael T. Goodrich, Wenliang Du, and Zutao Zhu. Bureaucratic protocols for secure two-party sorting, selection, and permuting. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 226–237, New York, NY, USA, 2010. ACM.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162 –167, oct. 1986.

A Security Definitions

A.1 Oblivious Transfer

Our protocols use Oblivious Transfer (OT) as a building block. Since we focus on protocols that run in constant rounds, we describe an abstraction for a one-round OT protocol here. A One-round OT involves a server holding a list of t secrets $(s_1, s_2, \ldots, s_t)$, and a client holding a selection index i. The client sends a query q to the server who responds with an answer a. Using a and its local secret, the client is able to recover s_i.

More formally, a one-round 1-out-of-t oblivious transfer (OT_1^t) protocol is defined by a tuple of PPT algorithms $OT_1^t = (\mathrm{G_{OT}}, \mathrm{Q_{OT}}, \mathrm{A_{OT}}, \mathrm{D_{OT}})$. The protocol involves two parties, a client and a server where the server's input is a t-tuple of strings $(s_1, \ldots, s_t)$ of length τ each, and the client's input is an index $i \in [t]$. The parameters t and τ are given as inputs to both parties. The protocol proceeds as follows:

1. The client generates $(pk, sk) \leftarrow \mathrm{G_{OT}}(1^\mathrm{k})$, computes a query $q \leftarrow \mathrm{Q_{OT}}(\mathrm{pk}, 1^\mathrm{t}, 1^\tau, \mathrm{i})$, and sends (pk, q) to the server.

2. The server computes $a \leftarrow \mathrm{A_{OT}}(\mathrm{pk}, \mathrm{q}, \mathrm{s_1}, \ldots, \mathrm{s_t})$ and sends a to the client.

3. The client computes and outputs $\mathrm{D_{OT}}(\mathrm{sk}, \mathrm{a})$.

In case of semi-honest adversaries, many of the OT protocols in the literature are one-round, and secure against parallel compositions (e.g. see [NP01, Lip05, PVW08]).

A.2 Secure Two-party Computation

Let $f = (f_1, f_2)$ of the form $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$ be a two party computation and Π be a two-party protocol for computing f between the parties p_1 and p_2. The input of p_1 is x and the input of p_2 is y. We briefly review the simulation-based notion of security for secure two-party computation here, focusing on the case where the adversary is semi-honest.

A.2.1 Security Against Semi-Honest Adversaries

Readers can refer to [Gol04] for a detailed discussion. Security for a two-party computation is defined by requiring indistinguishability (either perfect, statistical or computational) between a real execution of the protocol and an ideal execution in which there is a TTP (trusted third party) who receives the parties input, evaluates the function and outputs the results to them. Consider a semi-honest and *admissible* adversary $\mathcal{A}$. An admissible adversary is one that corrupts exactly one of the two party. $\mathcal{A}$ also knows an auxiliary input z. Without loss of generality we assume the $\mathcal{A}$ corrupts the first party.

In the real world, the honest party follows the description of protocol Π as instructed and responds to messages sent by $\mathcal{A}$ on behalf of the other party. Let $view_{\Pi,\mathcal{A}}(x, y)$ denote $\mathcal{A}$'s view through this interaction, and let $out_\Pi(x, y)$ denote the output of the honest party. The execution of Π in the real model on input pair (x, y) is defined as follows:

$$\mathrm{REAL}_{\Pi,\mathcal{A}(z)}(x,y) \overset{\text{def}}{=} (\mathrm{view}_{\Pi,\mathcal{A}}(x,y), \mathrm{out}_{\Pi}(x,y))$$

In the ideal model, in which there is a TTP, both parties send their inputs to TTP. The trusted party replies with $f_1(x,y)$ to the first party and with $f_2(x,y)$ to the second party. The honest party outputs whatever is sent by the trusted party, and $\mathcal{A}$ outputs an arbitrary function of its view. Let $out_{f,\mathcal{A}}(x,y)$ and $out_f(x,y)$ denote the output of $\mathcal{A}$ and the honest party respectively in the ideal model. The execution of Π in the ideal model on input pair (x,y) is defined as follow:

$$\mathrm{IDEAL}_{f,\mathcal{A}(z)}(x,y) \overset{\text{def}}{=} (\mathrm{out}_{f,\mathcal{A}}(x,y), \mathrm{out}_f(x,y))$$

Definition A.1. *We say that Π securely computes f in the presence of static malicious adversaries if for every pair of admissible non-uniform probabilistic polynomial-time machines $\bar{A} = (A_1, A_2)$ in the real model, there exists a pair of admissible nonuniform probabilistic expected polynomial-time machines $\bar{B} = (B_1, B_2)$ in the ideal model, such that*

$$\left\{\mathrm{IDEAL}_{f,\bar{B}}(x,y)\right\} \equiv \left\{\mathrm{REAL}_{\Pi,\bar{A}}(x,y)\right\}$$

Namely the two distributions are indistinguishable.

A.2.2 The OT hybrid model

We use the OT hybrid model to prove the security of our proposed protocols. In the OT hybrid model (e.g. see [Can00, LP07]), it suffices to analyze the security of a protocol in a hybrid model in which the parties interact with each other and have access to a trusted party (ideal functionality) that computes the oblivious transfer protocol for them. This model is a hybrid of the real and ideal models: on the one hand, the parties send regular messages to each other, similar to the real model; on the other hand, the parties have access to a trusted party, similar to the ideal model.

B Proof of Security for the PFE Framework

We note that to make our PFE Framework conceptually simpler, we described it as a black-box construction based on the $\mathcal{F}_{\mathcal{CTH}}$, and the $\mathcal{F}_{\mathcal{PGE}}$ functionalities. However, for the purpose of the security proof, it is useful to break down the $\mathcal{F}_{\mathcal{CTH}}$ functionality into a sequence of calls to the OMAP functionality, and replace calls to the Reveal queries with a message from P_1 to other parties with their blinded shares (which they can unblind). With this new interpretation, the whole framework can be seen as a series of sequential calls to the OMAP functionality and the PGE functionality. As a result, given the security of the framework with access to these ideal functionalities, and secure instantiation of each functionality, we can invoke the existing sequential composition theorems [Can00] to obtain security for the resulting PFE constructions. In light of this discussion, we prove security of our framework given access to the ideal OMAP and PGE functionalities.

Proof. **Case 1: $P_1, \ldots, P_{m-1}$ are corrupted.**

Simulation. For any PPT adversary $\mathcal{A}_1$, controlling $P_1, \ldots, P_{m-1}$ in the real world, we describe a simulator $\mathcal{S}_1$ who simulates $\mathcal{A}_1$'s view in the ideal world. $\mathcal{S}_1$ runs $\mathcal{A}_1$ on input shares of $P_1, \ldots, P_{m-1}$ and circuit $\mathcal{C}$ (with mapping $\pi_{\mathcal{C}}$).

1. $\mathcal{S}_1$ sends the inputs of $P_1, \ldots, P_{m-1}$ and the circuit $\mathcal{C}$ (part of P_1's input to the TTP. It receives the o output values $out_1, \ldots, out_o$, as the outcome of evaluating the circuit $\mathcal{C}$ on all the parties' input (including honest P_m).

2. $\mathcal{A}_1$ distributes shares of inputs for $P_1, \ldots, P_{m-1}$ to $\mathcal{S}_1$. $\mathcal{S}_1$ then generates a fake input on behalf of P_m and distributes the shares of all parties by sending their shares to $\mathcal{A}_1$.

3. $\mathcal{S}_1$ and $\mathcal{A}_1$ send shares of the inputs they just distributed to the OMAP functionality. $\mathcal{S}_1$ also sends a blinding vector $\vec{k}_m$ on behalf of P_m while $\mathcal{A}_1$ sends a vector of blinding vectors $\vec{t}_1, \ldots, \vec{t}_m$. As a result, $\mathcal{A}_1$ receives blinded shares of a subset of the incoming wires in the circuit.

4. For $1 \leq j \leq g$:

   - $\mathcal{A}_1$ sends the blinded shares he holds for the two incoming wires of gate G_j to $\mathcal{S}_1$, which $\mathcal{S}_1$ unblinds using $\vec{k}_m$ to retrieve $[a']^j_m$ and $[b']^j_m$.
   - $\mathcal{S}_1$ sends $[a']^j_m$ and $[b']^j_m$ to $\mathcal{F}_{\mathcal{PGE}}$ ideal functionality while $\mathcal{A}_1$ sends $[a']^j_i, [b']^j_i$ for $1 \leq i < m$ and G_j to the PGE. As a result, $\mathcal{S}_1$ gets back $[c']^j_m$.
   - $\mathcal{S}_1$ sends $[c']^j_m$ to the OMAP functionality while $\mathcal{A}_1$ sends all other parties shares. Once again $\mathcal{A}_1$ receives blinded shares of a subset of the incoming wires in the circuit (those connected to outputs of G_j).

5. For $g - o < j \leq g$: $\mathcal{A}_1$ reveals the shares $[c']^j_1, \ldots, [c']^j_{m-1}$ of the output of G_j. $\mathcal{S}_1$ who has obtained the output out_{g-j} corresponding to this gate earlier in the simulations, computes an appropriate shares $[c'']^j_m$, such that a reconstruction of the shares returns out_{g-i} and sends $[c'']^j_m$ to $\mathcal{A}_1$.

This ends the description of the simulator $\mathcal{S}_1$.

Indistinguishability of the views. It is easy to see that the ideal and real distributions defined in Definitions of Appendix A.2.1 are indistinguishable. In particular, in both the real and the ideal worlds, all messages seen by $\mathcal{A}_1$ are uniformly random shares except in the final step where shares of the output of the circuit are revealed. In the latter case, however, $\mathcal{A}_1$'s view in both worlds consist of random shares of the correct outputs values i.e. $out_1, \ldots, out_o$. This completes the security argument for this corruption case.

All other corruption scenarios where P_1 is one of the corrupted parties, can be simulated in an identical fashion. It remains to show that efficient simulation is possible in the case where $P_2, \ldots, P_m$ are corrupted by the adversary.

Case 2: $P_2, \ldots, P_m$ **are corrupted.** the simulation in this case is essentially the same as the previous case. Note that in this case the simulator needs to play the role of honest P_1, and consequently not only choose a fake input for P_1 but also create and use a fake circuit with n inputs, g gates and o outputs. But this does not effect the adversaries view as one again, he only sees random shares for all the intermediate wires, and the o outputs of the circuit. We omit the details of the proof for this case.

■

C Existing OEP Solutions and Their Complexity

We briefly review the existing methods for designing an OEP and compare their efficiency.

Using General-Purpose 2PC. Obviously, the OEP problem can be solved using any general 2PC protocol. For instance, one can first implement the functionality of an extended permutation using a switching network (we show how to do so in Section 4.1). Then, P_1 and P_2 can engage in a Yao's garbled circuit protocol where P_1's inputs are the selection bits to the network and his blinding vector $\vec{t}$, and P_2's input is the input vector $\vec{x}$. The boolean circuit being computed consists of the circuit for the switching network, and some additional gates to blind the network's output using $\vec{t}$.

For an EP from M inputs to N outputs, the number of switches required in the switching network will be in the order of $O(N \log N)$ which leads to a circuit with $O(N \log N\ell)$ gates; P_1 has $O(N \log N + N\ell)$ input bits in the circuit ($O(N \log N)$ for the selection bits and $O(N\ell)$ for the blinding vector), and P_2 has $O(M\ell)$ input bits in the circuit.

Kolesnikov and Schneider [KS08a], describe a construction for a similar circuit (called expanded permutation) as part of designing an efficient universal circuit. They also show how one can take advantage of the free-XOR optimization techniques [KS08b] in Yao's garbled circuit to reduce the number of non-free gates for each switch by 75%.

Katz et al. [HEK12] proposed a similar protocol for the special case of oblivious permutation network evaluation (referred to as a shuffling network in their paper). They use the Waksman [Wak68] switching network which consists of $O(N \log N)$ switches, and employ *oblivious swappers* to implement each switch. Several optimization techniques are used to improve the implementation of these swappers in a garbled circuit construction. According to their estimates, the total number of non-free gates (i.e. total number of gates minus the number of XOR gates) for the whole circuit is $\ell(N \log N - N + 1)/3$ (See Table 1 of [HEK12]).

Using Homomorphic Encryption (HE-OEP). It is also possible to use a semantically secure singly homomorphic encryption scheme to design an OEP protocol. In this protocol P_2, the party holding the input vector $\vec{x}$ generates and sends a public key pk for the scheme $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ to P_1. P_2 then encrypts his inputs $(x_1, \ldots, x_M)$ and sends them to P_1:

$$P_2 \text{ sends } \mathsf{Enc}_{pk}(x_1), \ldots, \mathsf{Enc}_{pk}(x_{|M|}) \text{ to } P_1$$

P_1 in return applies the EP by replicating and permuting the received ciphertexts as prescribed by π_C. He then uses the homomorphic properties of encryption scheme to add/multiply his blinding values and sends the result to P_2.

$$P_1 \text{ sends } \mathsf{Enc}_{pk}(x_{\pi_C^{-1}(1)} + t_1), \ldots, \mathsf{Enc}_{pk}(x_{\pi_C^{-1}(N)} + t_N) \text{ to } P_2$$

P_2 decrypts the ciphertexts and recovers the final result. This approach requires $O(N)$ encryption/decryption and $O(N)$ homomorphic operations on the ciphertexts. Note that in the above construction, the blinding function is not necessarily an XOR operation. The exact operation depends on the particular homomorphic encryption scheme being used.

Efficiency Discussion. Asymptotically speaking, the construction based on homomorphic encryption has linear complexity and hence is superior to the Yao-based one. Interestingly, however, the Yao-based solution seems to be the more efficient of the two in practice, for many values of interest for N and ℓ. The latter conclusion is based on the recent experimental results provided in [HEK12]. In this paper, the authors implement an oblivious shuffling protocol using both approaches, and compare their efficiency. In particular, the authors show that despite the asymptotic efficiency of the solution based on homomorphic encryption, for values of $128 < N < 8192$, and with $\ell = 32$, the garbled circuit implementation of the Waksman network is more than 7 times faster than the homomorphic-based shuffling (See Section 7.2 of [HEK12]). The homomorphic-based solution does not lead to any savings in the bandwidth either. Though the above analysis is done for permutation mappings, we note that the same efficiency analysis holds in the more general case of extended permutations.

D Complete Description of OSN Protocol

Efficiency of the OSN Protocol. We analyze, round, computation and communication complexity of the above protocol.
Round complexity. Given that we can run the oblivious transfers in parallel, our protocol runs in one and a half rounds (three messages). P_1 sends his OT queries to P_2. P_2 replies with the OT responses and his blinded inputs to P_1. P_1 then evaluates the switching network and sends the blinded outputs back.
Computation. The only cryptographic operation in the above OSN protocol is oblivious transfer. The number of 1-out-of-4 OTs is equal to the number of switches in the network, i.e. q. By employing the OT extension techniques [IKNP03], this only requires $O(k)$ public-key operations where k is a statistical security parameter, and $O(q)$ symmetric-key operations ($4q$ hash evaluations to be precise). Using the OT preprocessing techniques of [Bea95], the above computation can be performed in an offline stage, and the online stage will only consist of $O(q)$ XOR operations.

Efficiency of SN-OEP. The number of symmetric operations in our SN-OEP protocol is $4N \log N - 2N + 2$. To get a number of symmetric operations in Yao-OEP we use the number of gates required for a WN from [HEK12] and extend it to the full OEP, which would be $\ell(2N \log N - N + 1)/3$. This translates to $4\ell(N \log N - N + 1)/3 + 2N\ell$ symmetric operations. By plugging in the parameters from [HEK12] $\ell = 32, N = 128$ and $\ell = 32, N = 8192$, and comparing the number of symmetric operations we get an average of 23 time improvement over Yao-OEP. Considering that Yao-OEP is 7 times faster than HE-OEP, we get factor of 161 efficiency gain over HE-OEP.

E Security Proof for the OSN and the OEP Protocols

We show that as long the oblivious transfer protocol used is secure, so is our protocol. Particularly, if the OT is secure against semi-honest adversaries when executed in parallel, the OSN protocol described above is also secure against semi-honest adversaries.

The proof also implies the security of our OEP protocol since our OEP construction simply runs the OSN protocol on the switching network for the EP. The following Theorem formalizes this statement.

Theorem E.1. *In the OT-hybrid model, the OSN protocol is secure against semi-honest adversaries.*

Proof. Our proof follows the ideal/real world simulation paradigm.

Case 1: P_2 is corrupted.

Simulation. For any PPT adversary $\mathcal{A}_2$, controlling P_2 in the real world, we describe a simulator $\mathcal{S}_2$ who simulates $\mathcal{A}_2$'s view in the ideal world. $\mathcal{S}_2$ runs $\mathcal{A}_2$ on $\vec{x}$.

1. $\mathcal{A}_2$ builds a table $T(u)$, for each switch u according to the protocol.

2. $\mathcal{A}_2$ sends the tables as his input to the OT ideal functionality.

3. $\mathcal{S}_2$ randomly generates the selection bits (corresponding to a random mapping π') and a uniformly random vector $\vec{t'}$.

4. $\mathcal{A}_2$ sends his blinded inputs to $\mathcal{S}_2$.

5. $\mathcal{S}_2$, having the tables and the blinded inputs, evaluates the switching network on them, using the selection bits he generated earlier.

6. $\mathcal{S}_2$ sends back the blinded outputs $O'_i = x_{\pi'^{-1}(i)} \oplus t'_i$ for $1 \le i \le N$ to $\mathcal{A}_2$.

7. $\mathcal{A}_2$ outputs $\vec{O'} = (O_1, \ldots, O_N)$. $\mathcal{S}_2$ outputs the same and halts.

This ends the description of the simulator $\mathcal{S}_2$.

Indistinguishability of the views. We now show that the ideal and real distributions defined in Definitions of Appendix A.2.1 are indistinguishable. Since P_1 (the honest party) has no outputs in the OSN protocol, for the ideal distribution we have that,

$$\mathrm{IDEAL}_{\mathrm{f},\mathcal{S}_2(\mathrm{z})}(\mathrm{x},\mathrm{y}) \overset{\mathrm{def}}{=} (\mathrm{view}_{\pi,\mathcal{A}}(\mathrm{x},\mathrm{y}), \mathrm{out}_{\pi}(\mathrm{x},\mathrm{y})) = ((\mathrm{x}_{\pi'^{-1}(1)} \oplus \mathrm{t}'_1), \ldots, (\mathrm{x}_{\pi'^{-1}(\mathrm{N})} \oplus \mathrm{t}'_{\mathrm{N}}))$$

In the real protocol, on the other hand, the actual selection bits of P_1 corresponding to the mapping π, and his actual blinding vector $\vec{t}$ is used, hence, the output $\mathcal{A}_2$ receives is of the form $x_{\pi^{-1}(i)} \oplus t_i$ for $1 \leq i \leq N$:

$$\mathrm{REAL}_{\Pi,\mathcal{A}_2(\mathrm{z})}(\mathrm{x},\mathrm{y}) = ((\mathrm{x}_{\pi^{-1}(1)} \oplus \mathrm{t}_1), \ldots, (\mathrm{x}_{\pi^{-1}(\mathrm{N})} \oplus \mathrm{t}_{\mathrm{N}}))$$

Since both the ideal and the real distribution consist of the blinded outputs (one blinded using P_1's random blinding vector $\vec{t}$ and the other using $\mathcal{S}_2$'s randomly generated blinding vector $\vec{t'}$), the distributions are uniformly random, and identical.

Case 2: P_1 is corrupted.

Simulation. For any PPT adversary $\mathcal{A}_1$, controlling P_1 in the real world, we describe a simulator $\mathcal{S}_1$ who simulates $\mathcal{A}_1$'s view in the ideal world. $\mathcal{S}_1$ runs $\mathcal{A}_1$ on π and the blinding vector $\vec{t}$.

1. $\mathcal{S}_1$ sends π and $\vec{t}$ to the TTP as his input. Honest P_2 will receive $(x_{\pi^{-1}(1)} \oplus t_1), \ldots, (x_{\pi^{-1}(N)} \oplus t_N)$ as his output from the TTP.

2. $\mathcal{S}_1$ generates random values for every wire in the switching network as an honest P_2 would.

3. $\mathcal{S}_1$ uses them to build a table $T(u)$, for each switch u in the network.

4. $\mathcal{A}_1$ sends his selection bits as input to the OT ideal functionality. $\mathcal{S}_1$ uses these selection bits to choose one of the four rows in each table.

5. $\mathcal{S}_1$ randomly generates his inputs to the switching network $(x'_1, \ldots, x'_N)$. He then blinds them using the random values he generated earlier for the input wires. Let's assume that the switching network is of depth d. We denote the list of the blinded inputs to the switching network by Tr'_0, and the selected rows for each switch at layer i of the network by Tr_i for $1 \leq i \leq d$. We also denote the entries in Tr'_i by $(Tr'_i(1), \ldots, Tr'_i(h_i))$, where h_i denotes the number of switches at depth i. $\mathcal{S}_1$ sends $(Tr'_0, \ldots, Tr'_d)$ to $\mathcal{A}_1$.

6. $\mathcal{A}_1$ evaluates the switching network, and recovers $x'_{\pi'^{-1}(i)} \oplus r'_i$ for $1 \leq i \leq N$ where r_is are the random values $\mathcal{S}_1$ generated for the output wires of the network. $\mathcal{A}_1$ blinds the output using his blinding vector $\vec{t}$ and sends $O'_i = x'_{\pi'^{-1}(i)} \oplus r'_i \oplus t_i$ for $1 \leq i \leq N$ back to $\mathcal{S}_1$.

7. $\mathcal{A}_1$ outputs his view, namely $(\pi, \vec{t}, (Tr'_0, \ldots, Tr'_d))$. $\mathcal{S}_1$ does the same and halts.

This ends the description of the simulator $\mathcal{S}_1$.

Indistinguishability of the views. We now show that the ideal and real distributions defined in Definitions of Appendix A.2.1 are indistinguishable. For the ideal execution of protocol we have:

$$\mathrm{IDEAL}_{f,\mathcal{S}_1(z)}(x,y) = ((\pi, \vec{t}, (Tr'_0, \ldots, Tr'_d)), (x_{\pi^{-1}(1)} \oplus t_1), \ldots, (x_{\pi^{-1}(N)} \oplus t_N))$$

The second part is P_2's output in the OSN protocol which is the same in the real and ideal execution.

The real distribution is as follows

$$\text{REAL}_{\Pi,\mathcal{A}_1(z)}(x,y) = (\pi, \vec{t}, (Tr_0,\ldots,Tr_d), (x_{\pi^{-1}(1)} \oplus t_1), \ldots, (x_{\pi^{-1}(N)} \oplus t_N))$$

where the Tr_is are created using the actual input of P_2 as opposed to fake inputs.

We denote the IDEAL distribution by D', and the REAL distribution by D. We sequentially modify the REAL distribution until it is identical to the IDEAL one. We let D_d be the distribution of the REAL execution, with the exception that Tr_d (the randomly encoded rows for the last layer) is substituted by a random set of values Tr'_d. Since the values in Tr_d are *one-time pad encryptions* of the values from the previous layer, Tr'_d and Tr_d will both have a uniform distribution, and hence $D_d = D$. We can use a similar argument to prove that $D_{i-1} = D_i$ for $1 \le i \le d$. Finally, it is easy to see that D_0 where all the Tr_is are replaced with uniformly random Tr'_is is identically distributed to IDEAL in which uniformly random input vector $\vec{x}$ was used. This completes the argument that the IDEAL and REAL distributions are identical.

■

F A Multiparty OEP Protocol

Definition F.1. ***The n-Party OEP Problem (*$\mathsf{m\text{-}OEP}(\pi,\vec{x})$*).*** *In this problem, the first party P_1 holds an extended permutation $\pi : \{1...M\} \to \{1...N\}$; and all the parties (including P_1) hold their XOR share of the vector of inputs $\vec{x} = (x_1,\ldots,x_M)$ where x_is are ℓ-bit strings. At the end of the protocol, each P_i learns his XOR share of the output vector $(x_{\pi^{-1}(1)},\ldots,x_{\pi^{-1}(N)})$ but nothing else.*

Next, we describe a protocol for multi-party OEP. The protocol consists of $n-1$ two-party OEPs, between P_1 and $P_i (1 \le i \le n)$. Details follow:

Multi-Party Oblivious Extended Permutation ($\mathsf{m\text{-}OEP}(\pi,\vec{x})$)

P_1's Inputs: An extended permutation $\pi : \{1...M\} \to \{1...N\}$.
Shared Inputs: For $1 \le i \le n$, P_i holds $\vec{x^i}$, his XOR share of the input vector $\vec{x} = (x_1,\ldots,x_M)$.
Shared Output: For $1 \le i \le n$, P_i holds his XOR share of the output vector $\vec{O} = (x_{\pi^{-1}(1)},\ldots,x_{\pi^{-1}(N)})$.

1. P_1 applies π to his share of the input vector $\vec{x^1}$, to retrieve a vector $\vec{y^1}$ of N elements.
2. For $2 \le j \le n$
   (a) P_1 generates a uniformly random blinding vector $\vec{t^i} = (t^i_1,\ldots,t^i_N)$.
   (b) P_1 and P_i engage in a $\mathsf{2\text{-}OEP}(\pi,\vec{x^i},\vec{t^i})$, as a result of which P_i learns the vector $\vec{y^i} = (x^i_{\pi^{-1}(1)} \oplus t^i_{\pi^{-1}(1)},\ldots,x^i_{\pi^{-1}(N)} \oplus t^i_{\pi^{-1}(N)})$.
3. P_1 computes his share of the output vector: $\vec{y^1} \oplus \bigoplus_{2\le j\le n} \vec{t^j}$.
4. For $2 \le j \le n$, P_j already holds his share of the output vector, i.e. $\vec{y^j}$.

For correctness, it is easy to verify that XORing the shares of everyone's output yields the desired output $\vec{O}$. The complexity of the protocol is exactly a factor of $n-1$ more than the two-party variant. This yields a protocol with exactly $(n-1)(4N\log N - 2N + 4)$ oblivious transfers. Furthermore, using standard sequential composition theorems for MPC, the security of the $\mathsf{m\text{-}OEP}$ protocol follows in a straightforward manner from the security of our $\mathsf{2\text{-}OEP}$ protocol (discussed in Appendix E). We only state the theorem.

Theorem F.2. *The* $\mathsf{m\text{-}OEP}$ *protocol is secure against a semi-honest adversary who may corrupt upto $n-1$ parties, if the* $\mathsf{2\text{-}OEP}$ *is secure against semi-honest adversaries.*

G On-Demand Property of CTH Realizations

It is easy to see that our OSN protocol provides this feature. Once the offline phase of the protocol is completed, we can consider an alternative way of evaluating the switching network by P_1. For each input/output pair of the network, when P_1 receives the blinded value for the input, he can evaluate a single path in the switching network to compute the blinded output value. In order to do so, P_1 needs to follow the path that the input takes as it goes through the switches in the network (P_1 knows the selection bit vector and hence knows the path).

SN-OEP:

- $\mathsf{OMAP}([x], j)$ query: Receive the jth input from ith party and for each j such that $\pi_{\mathcal{C}}(j) = l$, does the on-demand evaluation between j and l and sets $\mathsf{Out}[i, l] = [x] \oplus k_i(l) \oplus t_i(l)$.
- $\mathsf{Reveal}(j)$ query: Sends $\mathsf{Out}[i, j]$ to P_i.

An OEP protocol based on homomorphic encryption also has the on-demand evaluation property.

Security of SN-OEP Realization of OMAP Query. As mentioned each OMAP query is equivalent to an on-demand evaluation of one input. In other words, we are able to realize each OMAP query individually using on-demand evaluation property of SN-OEP construction.

The security of each OMAP query follows from our proof of OSN protocol (See Appendix E). Particularly, for the case where P_2 is corrupted, since he only observes the final values which are blinded by P_1, it is the same as the case where we are doing all OMAP queries together and both Ideal and Real distributions are uniformly random and therefore indistinguishable. For the case where P_1 is corrupted, we can follow the same indistinguishability argument in the proof. In fact, if we only follow the path from outputs (all l such that $\pi_{\mathcal{C}}(j) = l$) to input (j) in the network, we can make the same argument with the difference that in each layer, only the values which belong to these paths are in the views. Therefore, we can see our SN-OEP protocol as a combination of independent OMAP queries implemented together, while maintaining their security individually.

HE-OEP:

- $\mathsf{OMAP}([x], j)$ query: Receive the jth input from ith party and for each j such that $\pi_{\mathcal{C}}(j) = l$, sets $\mathsf{Out}[i, l] = \mathsf{Enc}_{pk}([x] + t_i(l))$.
- $\mathsf{Reveal}(j)$ query: Sends $\mathsf{Out}[i, j]$ to P_i.

The more efficient Yao-based solution does not provide this capability since all the inputs to the garbled circuit need to be available to the circuit evaluator to be able to compute even a single output.

H Multiparty Private Gate Evaluation

We now describe a multiparty protocol for secure evaluation of a single gate G, where only P_1 knows the functionality of G, and everyone holds their share of the two inputs bits a and b to the gate. The goal is to have everyone learn their share of $G(a, b)$. To design such a protocol, we take advantage of a natural generalization of oblivious transfer to the multiparty case, where both the inputs and the outputs are shared. Such a generalization has already been considered, e.g. in [FGM07] where a simple and black-box construction based on any two-party OT is also presented. Roughly speaking, their protocol requires the execution of a two-party OT between P_1 and each party $P_i (i > 1)$. We only define multiparty OT here, and use it in a black-box manner. We refer the readers to [FGM07] for the details of the construction and its security.

Definition H.1 (multiparty oblivious transfer: $\mathsf{m\text{-}OT}_m(\sigma, \Delta)$)**.** *A multiparty oblivious transfer (*$\mathsf{m\text{-}OT}$*) is a protocol involving n parties where: at the beginning of the protocol, each party holds a share of a secret index σ and one distinguished party (P_1) holds a table of m bits, the database $\Delta = (\delta_0, \ldots, \delta_{m-1})$; at the end of the protocol, each party holds a share of the database element δ_σ. In the terminology of oblivious transfer, every party is a receiver and one party is also the sender.*

Our private gate evaluation protocol easily follows from such multiparty OT.

Multi-Party Private Gate Evaluation ($\mathsf{m\text{-}XOR\text{-}PGE}(G, a, b)$)

P_1**'s Inputs:** A gate G.
Shared Inputs: For $1 \le i \le n$, P_i holds a_i, and b_i, his XOR shares of the two input bits a and b.
Shared Output: For $1 \le i \le n$, P_i holds his XOR share of the output bit $G(a, b)$.

1. P_1 creates the truth table Δ for the gate G:

a	b	
0	0	$G(0,0)$
0	1	$G(0,1)$
1	0	$G(1,0)$
1	1	$G(1,1)$

2. Parties engage in a multiparty OT $\mathsf{m\text{-}OT}_4(2a + b, \Delta)$ to learn their XOR share of the output bit.

Our $\mathsf{m\text{-}XOR\text{-}PGE}$ protocol requires a single invocation of a multiparty OT. Each multiparty OT itself requires the invocation of $n - 1$ OTs. This is equal to the number of OTs needed to evaluate a single gate in the standard GMW. Hence, making the gates private does not require additional computation or communication.

The correctness and security of the $\mathsf{m\text{-}XOR\text{-}PGE}$ protocol directly follows from that of the multiparty OT.

Theorem H.2. *The* $\mathsf{m\text{-}XOR\text{-}PGE}$ *protocol is secure against a semi-honest adversary who may corrupt upto $n - 1$ parties.*

I 2-Arith-PGE

J Description and Efficiency of 2-PFE Protocol and Proof of Security

J.1 Efficiency of the Yao-based PFE

Overall efficiency. In addition to the cost of Yao's garbled circuit protocol which requires $O(g)$ symmetric-key operations and $O(\ell)$ oblivious transfers, our protocol needs to execute a single OEP protocol, and a single OSN protocol. Using the optimizations mentioned above (for combining three switching networks), the total number of switches are $4g \log 2g - 2g + 1 + 2g = 4g \log 2g + 1$, which leads to the same number of oblivious transfers. With the use of OT extension, this requires $8g \log 2g + 2$ symmetric-key operations (two for each OT), and yields a two-party PFE with $8g \log 2g + 5g + 2$ symmetric-key operations (assuming that each gate in Yao's protocol requires four encryptions).

Comparison to construction of [KM11]. We now show a concrete (but rough) comparison of our two-party PFE with the protocol of [KM11]. Note that for input sizes ranging form 128 to 8192, their

	# of gates [KSS12]	Our 2-PFE	[Val76] UC	Ratio 1	[KS08a] UC	Ratio 2
AES	49912	6830798.24	14864610.03	2.17	21336628.72	3.12
RSA-256	266150119	62785164329	1.4153E+11	2.25	3.37727E+11	5.37
4095 Edit-Distance	5901194475	1.60316E+12	3.63932E+12	2.27	9.9792E+12	6.22

Table 4: Comparison of number of symmetric operations for our 2-PFE protocol vs. applying Yao's garbled circuits to Valiant's universal circuit [Val76] (Ratio 1) and [KS08a] (Ratio 2) while employing Free-XOR technique.

homomorphic-based OEP requires at least a factor of 7 more work than the Yao-based solutions (as discussed in Appendix C, and in [HEK12]), and as discussed in Appendix our solution improves on the Yao-based solutions by a factor of 23. This yields a factor of 161 improvement for the OEP component. The garbled circuit evaluation in both our protocol and theirs has the same cost and requires at most $5g$ ($4g$ for garbling, g for evaluation using point and permute) symmetric-key operations. Note that since the cost both our solution and HE-based solution does not grow with size of input to 80-bits which is the minimum security required for Yao, we still use 161 in our analysis here. To compute the efficiency gain we have to compute the following equation: (averaged for $g = 128$ and $g = 8192$)

$$\frac{\text{HE-OEP} + \text{Yao}}{\text{SN-OEP} + 4g + \text{Yao}} = \frac{161(\text{SN-OEP}) + \text{Yao}}{\text{SN-OEP} + 4g + \text{Yao}} = \frac{161(8g\log 2g - 4g + 2) + 5g}{8g\log 2g - 4g + 2 + 4g + 5g} \approx 142$$

This shows a factor of 142 or more improvement in the total work of our two-party PFE compared to [KM11]. Note that the OT extension can be used to evaluate both the OTs needed in Yao's protocol and our OEPs, and hence no additional public-key operations are needed.

Comparison to universal circuit constructions. We analyze the approach of applying Yao's garbled circuits to the universal circuit construction of Valiant [Val76] and [KS08a]. According to [KS08a], the number of gates for the Valiant's universal circuit that can handle circuits with g gates is, $(19g + 9.5n + 9.5v\log(g + n)$ (Circuits from n inputs to v outputs). The number of gates for the universal circuit of [KS08a] is: $1.5g\log^2 g + (u + 2g)\log n + (g + 3v)\log v$. It is possible to use free xor technique on $3/4$ of the overhead gates [KS08b]. For a more meaningful comparison we use the parameters for benchmark circuits [KSS12], such as AES, RSA and Edit-Distance to give concrete numbers on the improvement ratio of our scheme (see Table 4).

J.2 Proof of Security for the 2-PFE Protocol

Theorem J.1. *Given that the OSN and the OEP protocols are secure against semi-honest adversaries, and that the Yao's protocol uses a symmetric-key encryption with related-key security, the* 2-PFE *protocol is secure against semi-honest adversaries.*

Proof Sketch. What follows is a proof sketch. A complete proof will appear in the full version of the paper. Our 2-PFE protocol can be divided into two stages: (i) generating the random wire keys necessary to create a garbled circuit, and (ii) executing the Yao's garbled circuit protocol using those keys, where P_2 is the circuit garbler and P_1 is the circuit evaluator.

In the 2-PFE protocol, the OEP and the OSN are utilized in the first stage, in a black-box manner. The second stage, simply uses Yao's garbled circuit protocol. In [KM11] it is shown that if the symmetric-key encryption scheme used is semantically secure against linear related-key attacks [AHI10], and the following two conditions are met, the Yao's garbled circuit protocol is secure against semi-honest adversaries:

1. P_2 cannot distinguish the distribution of the $6g$ k-bit keys used in garbling ($4g$ for input wires, and $2g$ for output wires) from a uniformly random distribution U_{6gk}.

2. P_1 only knows a specific set of linear constraints between a subset of the keys but nothings else, where the set of constraints cannot be used to solve for any key values. More formally, there exist a simulator S_1 that takes P_1's inputs, and generates P_1's view in the key generation phase (S_1 need not take the keys as input). In our case, the linear constraints are all of the form $\bigoplus_{i \in T} k_i = c$, for values of c known to P_1, and for specific subsets T of the keys where $|T| \geq 2$.

As a result, our proof will only focus on showing that the first stage of the construction guarantees the above two properties for the set of keys generated. The security of the overall protocol then follows based on the existing proof of security for Yao's garbled circuit protocol implemented with a related-key secure encryption, as provided in [KM11].

Lets consider the steps of the 2-PFE that constitute the first stage, i.e. steps 1 to 7. We assume the security of the OEP protocol and the OSN protocol and hence consider the real 2-PFE protocol executed in a OPE- and OSN-hybrid model:

- **OEP Ideal functionality:** P_1 and P_2 use the OEP as the ideal functionality. P_1 sends $\vec{v'}$ (his uniformly random blinding vector) and π as his inputs, and P_2 sends v as his input to the OEP ideal functionality. As a result, P_2 will learn $v''_i = v_{\pi^{-1}(i)} \oplus v'_i$ for $1 \leq i \leq 2g$ and P_1 learns nothing.
- **OSN Ideal functionality:** P_1 produces the mapping π' by combining π and v'_i for $1 \leq i \leq 2g$. P_1 sends π' and t_i for $1 \leq i \leq 2g$ as his input to OSN ideal functionality. P_2 also sends $X_0(i), X_1(i)$ for $1 \leq i \leq 2g$, to the OSN ideal functionality. OSN ideal functionality returns $(X_{v'_i}(i) \oplus t_{2i-1}, X_{\bar{v}'_i}(i) \oplus t_{2i})$ for $1 \leq i \leq 2g$ to P_2, while P_1 does not learn anything.
- The list of key pairs $(X_{v'_i}(i) \oplus t_{2i-1}, X_{\bar{v}'_i}(i) \oplus t_{2i})$ for $1 \leq i \leq 2g$ returned to P_2 by the OSN ideal functionality and the additional g key pairs for the output wire generated by P_2 at random, constitute the list of keys P_2 needs to use in Yao's garbled circuit.

The first property we need to prove is that P_2 can not distinguish the keys retrieved from the OSN protocol from keys sampled uniformly at random. Consider P_2's view in the above OSN-, OEP-hybrid real execution. In the OEP ideal execution, P_2 gets to see $v''_i = v_{\pi^{-1}(i)} \oplus v'_i$ for $1 \leq i \leq 2g$, where $\vec{v'}$ is chosen uniformly at random by P_1, and in the OSN ideal execution he gets to see $(X_{v'_i}(i) \oplus t_{2i-1}, X_{\bar{v}'_i}(i) \oplus t_{2i})$ for $1 \leq i \leq 2g$, where $\vec{t}$ is chosen uniformly at random by P_1. Based on the security of one-time pad encryption, his whole view in the above execution can be replace by a uniformly random string. This essentially provides us with our first requirement.

Now lets consider P_1's view in the above execution. P_1 does not receive any outputs from either ideal functionalities (OEP, OSN) and hence his view in the (hybrid) real protocol is simply his inputs, i.e. $\pi, \pi', \vec{v'}$, and $\vec{t}$. A simulator S_1 can trivially generate this view since he has the same inputs. This provides us with our second requirement.

According to the above discussion, this completes our proof sketch of security for the 2-PFE protocol against semi-honest adversaries.

■

Oblivious Switching Network Evaluation (OSN)

Common Inputs: A switching network SN with q switches, and N inputs/outputs.
P_1**'s Inputs:** The selection bit vector $\vec{s} = ((s_0(0), s_1(0)), \ldots, (s_0(q), s_1(q)))$ for the switching network SN and a random blinding vector $\vec{t} = (t_1, \ldots, t_N)$, where $t_i \in \{0,1\}^\ell$. We denote the mapping associated with SN by π.
P_2**'s Inputs:** The vector $\vec{x} = (x_1, \ldots, x_N)$, where $x_i \in \{0,1\}^\ell$.
Outputs: P_2 learns the output $z_i = x_{\pi^{-1}(i)} \oplus t_i$ for $1 \le i \le N$. P_1 does not learn anything.

P_1 **learns a random encoding of the SN with the selection bit vector** $\vec{s}$**.**

1. We denote the $2q+n$ wires of the SN with $w_1, \ldots, w_{2q+n}$. For every $1 \le i \le 2q+n$:
2. P_2 generates a uniformly random bitstring $r_i \in \{0,1\}^\ell$
3. For each switch u with input wires w_i and w_j, and output wires w_k and w_l:
4. P_2 computes the following table:

$s_1(u)$	$s_0(u)$	
0	0	$T_0 : T_0^0 = (r_i \oplus r_k)$, $T_0^1 = (r_i \oplus r_l)$
0	1	$T_1 : T_1^0 = (r_i \oplus r_k)$, $T_1^1 = (r_j \oplus r_l)$
1	0	$T_2 : T_2^0 = (r_j \oplus r_k)$, $T_2^1 = (r_i \oplus r_l)$
1	1	$T_3 : T_3^0 = (r_j \oplus r_k)$, $T_3^1 = (r_j \oplus r_l)$

Figure 7: Table for each switch

5. P_1 and P_2 engage in a 1-out-of-4 oblivious transfer where P_1's (the receiver) input is $s(u) = 2s_1(u) + s_0(u)$ and P_2's (the sender) input is (T_0, T_1, T_2, T_3). P_1 obtains $(T^0_{s(u)}, T^1_{s(u)})$, a *random encoding* of the switch u with selection bits $s_0(u)$ and $s_1(u)$.

P_2 **blinds his inputs.**

1. For each input wire w_i P_2 sends his blinded input $y_i = x_i \oplus r_i$ to P_1.

P_1 **evaluates the random encoding of SN on** P_2**'s blinded inputs.**

1. In topological order, for each switch u with input wires i, j and output wires k, l, P_1 does the following:

   If $s_0(u) = 0$ then $y_k = y_i \oplus T^0_{s(u)}$ else $y_k = y_j \oplus T^0_{s(u)}$
   If $s_1(u) = 0$ then $y_l = y_i \oplus T^1_{s(u)}$ else $y_l = y_j \oplus T^1_{s(u)}$

2. P_1 eventually obtains blinded values for all the output wires of SN. He will further blind these values using his random values and send to P_2. In particular, for each output switch h ($1 \le h \le N/2$) with output wires w_k, w_l, P_2 computes:

$$z'_k = y_k \oplus t_{2(h-1)+1}, z'_l = y_l \oplus t_{2h}$$

P_1 **unblinds and retrieves his output.**

1. For each output switch with output wires w_k, w_l, P_1 unblinds its outputs using the random strings for those wires:

$$z_k = z'_k \oplus r_k, z_l = z'_l \oplus r_l$$

Figure 8: Oblivious Switching Network Evaluation.

2-Party PGE Protocol for Arithmetic Gate

Let $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an additively homomorphic encryption scheme. Arithmetic gate G with inputs $a = [a]_1 + [a]_2$ and $b = [b]_1 + [b]_2$ and output $G(a, b) = c = [c]_1 + [c]_2$ and G can be one of these two functions only known to P_1: $G(a, b) = a.b$ or $G(a, b) = a + b$.
P_1**'s Inputs:** $[a]_1, [b]_1$ and G.
P_2**'s Inputs:** $[a]_2, [b]_2$.
Output: P_1 learns $[G(a, b)]_1$ and P_2 learns $[G(a, b)]_2$ such that $G(a, b) = [G(a, b)]_1 + [G(a, b)]_2$.
P_2 generates a key pair $\mathsf{Gen}(1^k) \rightarrow (pk, sk)$, and sends pk to P_1 (this step is performed once for all gates.

1. P_2 computes the encryptions $\mathsf{Enc}_{pk}(a_2), \mathsf{Enc}_{pk}(b_2), \mathsf{Enc}_{pk}(a_2b_2)$ and sends them to P_1.

2. P_1 computes his output share and encrypted output share of P_2 as follows:

   - **If** $G(a, b) = a + b$**:** P_1 generates a random value r and locally computes $c_1 = a_1 + b_1 - r$ as his share of output and computes the $\mathsf{Enc}_{pk}(c_2) = \mathsf{Enc}_{pk}(a_2) +_h \mathsf{Enc}_{pk}(b_2) +_h \mathsf{Enc}_{pk}(r)$ as the encrypted output share of P_2 and sends it back to P_2.

   - **If** $G(a, b) = a.b$**:** P_1 randomly generates c_1 as his output share. He then computes $\mathsf{Enc}_{pk}(c_2) = \mathsf{Enc}_{pk}(a_1b_1 - c_1) +_h \mathsf{Enc}_{pk}(a_2b_2) +_h (a_1 \times_h \mathsf{Enc}_{pk}(b_2)) +_h (b_1 \times_h \mathsf{Enc}_{pk}(a_2))$, and sends it to P_2.

3. P_2 decrypts $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(c_2))$ and recovers his share of gate's output c_2.

Figure 9: 2-Party PGE Protocol for Arithmetic Gate ($\mathsf{2\text{-}Arith\text{-}PGE}$)

Two-Party Private Function Evaluation (2-$\mathsf{PFE}(C, x, y)$)

P_1**'s Inputs:** $x \in \{0,1\}^n$, a boolean circuit $\mathcal{C}$ with g gates, and o output gates.
P_2**'s Inputs:** $y \in \{0,1\}^n$.
Output: $C(x, y)$.

1. P_1 sorts the gates topologically and computes the extended permutation $\pi_{\mathcal{C}}$ corresponding to circuit $\mathcal{C}$ as described in Section 3.

2. P_2 randomly generates a key pair for each of his input bits, and for the output bit of each *non-output* gate in the circuit. This yields a total of $M = n + g - o$ key pairs. We denote these key pairs by k_i^0, k_i^1 for $1 \leq i \leq M$. Each key is k bits long, where k is the security parameter.

3. P_2 generates a uniformly random vector $\vec{v} = (v_1, \ldots, v_M)$ and swaps the ith key pair according to the bit v_i: $(X_0(i) = k_i^{v_i}, X_1(i) = k_i^{\bar{v_i}})$.

4. CTH

   (a) P_1 and P_2 engage in an OEP protocol where P_1's input is the extended permutation π_C and a random blinding bit-vector $\vec{v'}$ with $2g$ elements, while P_2's input is his vector $\vec{v}$. As a result, P_2 learns $v''_i = v_{\pi^{-1}(i)} \oplus v'_i$ for $1 \leq i \leq 2g$ (Figure 11 shows the input/output of this OEP protocol).

   (b) Consider two switching networks SN_0 and SN_1 where P_1's selection bits for both are chosen based on π_C, and P_2's input vector to each is $\vec{X_0} = (X_0(i))_{1 \leq i \leq M}$, and $\vec{X_1} = (X_1(i))_{1 \leq i \leq M}$, respectively. The outcome of the EP would be $X_0(\pi^{-1}(i))$, $X_1(\pi^{-1}(i))$ for $1 \leq i \leq 2g$. We attach a third switching network SN_2 to the end of SN_0, SN_1. SN_2 contains $2g$ parallel switches where the ith switch ($1 \leq i \leq 2g$) takes the ith output of SN_0 and SN_1 as its input and takes v'_i as its selection bit. This extra layer is used to let P_1 swap the final key pairs according to the bit vector v' while still enabling P_2 to unblind the results. The collection of these three switching networks form a larger switching network SN with $2M$ inputs and $4g$ outputs (Figure 11 shows the structure of this switching network).

   (c) P_1 and P_2 engage in an OSN protocol to evaluate SN, where P_1's selection bits and P_2's input vector are already described above. P_1 also uses a fresh blinding vector $\vec{t}$ with $4g$ k-bit elements. As a result, P_2 obtains $2g$ key pairs $(X_{v'_i}(i) \oplus t_{2i-1}, X_{\bar{v'_i}}(i) \oplus t_{2i})$ for $1 \leq i \leq 2g$.

   (d) P_2 swaps each blinded key pair one more time according to the bit v''_i for $1 \leq i \leq 2g$. Note that this essentially cancels out the previous two swappings (one by P_1 using $\vec{v}$ and the other by P_2 using $\vec{v'}$), and returns each key to its original location (but blinded). We denote these $2g$ key pairs with $(k_i'^0, k_i'^1)$ for $1 \leq i \leq 2g$.

5. P_1 and P_2 are now ready to engage in a Yao's garbled circuit protocol. We do not go into all the details of the protocol and refer the reader to [LP09], for a complete description. However, we point out a few point regarding the garbling and the evaluation process.

   - **Garbling.** For the ith gate in the circuit, P_2 creates the garbled truth table following Yao's garbled circuit protocol, where the key pairs used for the two input wires are $(k_{2i-1}'^0, k_{2i-1}'^1)$ and $(k_{2i}'^0, k_{2i}'^1)$ and the key pair used for the output wire is (k_i^0, k_i^1). P_2 also generates o additional key pairs for the output wire of the output-gates, and uses them in the garbling process.
   - **Evaluation.** P_1 evaluates the gates in the topological order he sorted them, using the Yao's garbled circuit protocol with a minor modification: Suppose P_1 is evaluating the ith gate, and he has already recovered the output keys for the two gates that feed the two inputs. P_1 first blinds these keys using appropriate components of the blinding vector $\vec{t}$ and uses these blinded values to decrypt one of the four ciphertexts in the garbled truth table.

Figure 10: Two-Party Private Function Evaluation

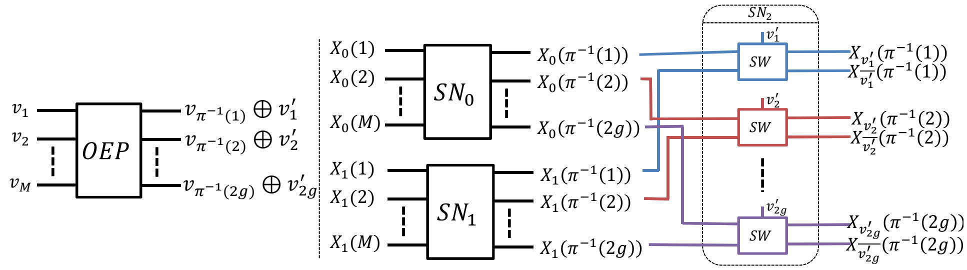


Figure 11: The OEP for $\vec{v}$ (Left), The Switching Networks SN_0, SN_1, SN_2 (Right)