1 Introduction

Assume that you invent an algorithm that is able to diagnose different diseases very efficiently based on some general inputs about a person’s health. Clearly, this would be valuable to people and different healthcare institutions would like to apply this in their everyday work. On the other hand, you do not want to hand the algorithm over as it could clearly bring you a lot of money if everyone has to buy this service from you. However, the medical institutions can not just give you the data about their patients. Hence, in this situation you would like to keep your algorithm secret and they also need to keep their inputs secret, but you are both interested in still evaluating this algorithm with the given inputs. This is exactly the problem solved by private function evaluation.

Circuit hiding multi-party computation (MPC) or private function evaluation (PFE) is a special additional case of multi-party computation where a private function \( f \) needs to be evaluated on private inputs \( x_i \) to obtain \( f(x_1, \ldots, x_m) \). In the concrete setting, we consider the case where one party inputs the function and all parties can provide the inputs. This paper introduces a framework for this kind of computation based on [MS13].

The main body of this overview is divided to two parts. First main part in Sec. 3 introduces the ideas put forth by [MS13] whereas the second part in Sec. 4 writes out the concrete protocols required for secure multi-party boolean circuit evaluation. These are supported by the preliminaries in Sec. 2 and summary of related work in Sec. 5.

2 Preliminaries

2.1 Circuits

Circuits are a model of computation that contain gates and wires. Each gate computes some functionality and the wires transport the gate outputs to inputs of other gates. In addition, some wires will be the outputs of the circuit where we can find also the output of the functionality of the circuit. A circuit has \( g \) gates, \( n \) input wires and \( o \) output
Circuit evaluation can be performed gate-by-gate. It is always possible to find a topological ordering of the gates so that for each gate $G$ the gates that use its outputs as inputs are after $G$ in this ordering. We number the gates as $G_i$ for $i \in \{1, \ldots, g\}$ in topological order. The wires in the circuit can be labelled based on the gates that they give input to. For a gate $G_i$ the input wires are $iw_{2i}$ and $iw_{2i-1}$.

**Example 1.** Fig. 1 pictures the circuit from \cite{MS13} that we will also use as an example. As pictured it has four gates with unspecified functionality. The gates $G_i$ are ordered in one of the valid topological orders and the circuit can be evaluated in this order. The parameters for this circuit are $g = 4$ gates, $n = 5$ inputs denoted as $x$, and $o = 2$ outputs denoted as $y$. The incoming wires are numbered according to the gate numbers following the previous rule. For example, the inputs to gate $G_3$ are $ow_5$ and $iw_6$ where $2 \cdot 3 - 1 = 5$ gives the number of the first input from the gate number 3.

Depending on the type of the circuit the wires can carry different values and the gates are allowed to have different functionalities. For example, boolean circuits operate on single bit values and each gate is a boolean function, such as conjunction or disjunction. On the other hand, arithmetic circuits operate on numerical values and the gates compute operations such as addition or multiplication.

To ensure the consistency with the proposed numbering we only consider fan-in 2 gates, meaning that each gate operation has two inputs. However, this is not a significant restriction on the theoretical side as all boolean functionalities can for example be built from a NAND gates or a combination of XOR and AND gates. Similarly, all arithmetic circuits can be built from addition and multiplication. Also, fan-in 1 gates can be modelled as fan-in 2 gates with a dummy input. Hence, in practice the restriction on fan-in means that the circuits can have more gates than optimal unbounded circuits, but the differences between each concrete basis are related by a constant factor.

### 2.2 Oblivious Transfer

An oblivious transfer (OT) protocol (originating from \cite{Rab81}) is a two-party protocol where the sender has two values $x_1$ and $x_2$ and the receiver has an index $i$ (2-OT). After the protocol the receiver should learn $x_i$ and nothing else. In addition, the sender should learn nothing about the index $i$. However, it can be extended to more values, where sender has $x_1$ to $x_m$ and the receiver still needs to learn $x_i$ ($m$-OT, 1-out-of-$m$ OT).
An information theoretic reduction from 1-out-of-($m - 1$) oblivious transfer to 1-
out-of-2 OT that requires $m - 1$ 2-OT invocations can be found in [BCR86]. The core
idea is simple. The sender picks random values $r_i$ and computes message pairs $(x_1, r_1)$,
$(x_2 ⊕ r_1, r_2 ⊕ r_1)$ to $(x_{m - 2} ⊕ r_{m - 3}, r_{m - 3} ⊕ r_{m - 2})$ and the final pair $(x_{m - 1} ⊕ r_{m - 2}, x_m ⊕ r_{m - 2})$.
For each of these pairs, a 2-OT protocol is used. If the receiver needs $x_i$ from $m$-OT
then it will ask for $x_i ⊕ r_j$ from the $i$'th pair and for the elements $r_1$ to $r_j ⊕ r_{j-1}$ from
the first pairs to compute $r_j$. From the pairs after $i$ it can ask for any element, but will
not receive any useful information, because it does not know how to remove the blinding
given by $r_{j+1}$. Note that in practice more efficient computationally secure approaches
for building 1-out-of-$m$ OT for large $m$ from 2-OT also exist.

In addition, OT can be extended to work in a setting with more than two parties. A
multi-party oblivious transfer is a protocol where one party holds the values $x_1, \ldots, x_m$,
but the choice index $i$ is secret shared between multiple parties. In the end of the
protocol, instead of learning the value $x_i$, the parties learn shares of this value. In
previous terminology, all parties are the receivers, but one party who holds the initial
values is the sender. As this requires the knowledge of secret sharing, we introduce this
protocol in the following subsection as Alg. [1]

### 2.3 Secure Multi-party Computation

Secure multi-party computation (MPC) is a tool for many parties to jointly compute
functions on their private inputs. In a common setting, there are $m$ parties each having
some secret inputs $x_i$ who compute a known function $f$ to learn $f(x_1, \ldots, x_m)$. The only
thing that is allowed to leak about the private inputs is the final output of the function.

There are different means of doing secure computations. However, an important tool
in many of those is secret sharing [Sha79]. A sharing scheme distributes a secret value $x$
into shares $[x]$, where each party $P_i$ learns exactly one of the shares $[x]_i$. A single share
does not give any information about the secret $x$, but there exists a way to combine the
shares to restore the value $x$. An important example of secret sharing required for this
paper is called XOR sharing, where $x = [x]_1 ⊕ \ldots ⊕ [x]_m$ and the secret $x \in \{0, 1\}$.

**GMW protocols** One of the secure computation schemes that uses XOR sharing is
the GMW protocol [GMW87]. This protocol proposes secure computation of boolean
circuits with gates AND and XOR in the presence of passive adversary. The XOR gates
can be computed locally, for example to compute $[a ⊕ b]$ from $[a]$ and $[b]$ each party
only uses its shares $[a ⊕ b]_i = [a]_i ⊕ [b]_i$. However, computing the AND gate requires
communication for doing 1-out-of-4 OT. To compute $[a \land b]$ from the shares we need
$a \land b = ([a]_1 \lor \ldots \lor [a]_m) \land ([b]_1 \lor \ldots \lor [b]_m)$. Each party can locally compute $[a]_i \land [b]_i$, but computing $[a]_i \land [b]_j \lor [a]_j \land [b]_i$ for $i \neq j$ requires collaboration of $P_i$ and $P_j$. Based on its own values $P_i$ builds the evaluation table for the two inputs of $P_j$ and they
engage in a 1-out-of-4 OT where $P_j$ uses its inputs as the choice index. This version
requires $m\choose 2$ runs of the OT protocol, because it has to be executed between any pair.
However, evaluation of AND gate can also be seen as one call to a multi-party 1-out-of-4
OT protocol with choice indices \([a]\) and \([b]\).

**Multi-party oblivious transfer** The following introduces a protocol from \cite{FGM07,CK88} that uses two-party 2-OTs as a building block as described in Alg. 1 to build multi-party 2-OT. The idea of the algorithm is that we have XOR sharing \(\bigoplus_{i=1}^{n} r^i_{[x]_i} = x_{I}\) and the \(r^i_{[x]_i}\) is what party \(i\) learns as output, giving \([x]_{I}\).

Consider the correctness of the ideas presented in Alg. 1. Firstly, consider the case where \(x_1 = x_0 = \bigoplus_{i=1}^{n} r^i_0\), but by definition \(r^i_0 = r^i_1\) and therefore any combination of the shares defined by the choices of \(P_i\) gives a valid XOR sharing of \(x_I\). When \(x_1 \neq x_0 = \bigoplus_{i=1}^{m} r^i_0\), then \(r^i_0 \neq r^i_1\) for all \(i\). Based on the values the parties have, there always exists \(k\) such that we can reorder the shares to give

\[
x_I = \bigoplus_{i=1}^{m} r^i_{[x]_i} = \bigoplus_{i=1}^{k} r^i_1 \oplus \bigoplus_{i=k+1}^{m} r^i_0 = \bigoplus_{i=1}^{k} (r^i_0 \oplus 1) \oplus \bigoplus_{i=k+1}^{m} r^i_0 = \bigoplus_{i=1}^{m} r^i_0 \oplus \bigoplus_{i=1}^{k} 1 = x_0 \oplus \bigoplus_{i=1}^{k} 1.
\]

If \(I = 1\) then \(k\) is odd, meaning that \(x_I = x_0 \oplus 1 = x_1\) because \(x_1 \neq x_0\), analogously, for \(I = 0\) we get \(x_I = x_0\).

**Algorithm 1 Multi-party 2-OT**

**Data:** \(P_1\) has \(x_0\) and \(x_1\), shared index \([I]\)

**Result:** Shared result \([x]_{I}\)

1. \(P_1\) randomly generates \(r^i_0 \leftarrow \{0, 1\} \text{ for } i \in \{2, \ldots, m\}\)
2. \(P_1\) computes \(r^i_1 = x_1 \oplus x_0 \oplus r^i_0\) for \(i \in \{2, \ldots, m\}\)
3. for \(i \in \{2, \ldots, m\}\) do
4. \(\text{OT protocol } P_i \text{ is a sender with } (r^i_0, r^i_1) \text{ and } P_i \text{ is a receiver with input } [I]_i\)
5. \(P_i\) fixes \(r^i_{[x]_i}\) as its output
6. end for
7. \(r^0_1 = x_0 \oplus \bigoplus_{i=2}^{m} r^i_0\)
8. \(r^1_1 = x_1 \oplus x_0 \oplus r^0_1\)
9. \(P_1\) outputs \(r^1_{[x]_i}\)

The multi-party 2-OT protocol requires \(m - 1\) OT protocol executions. Given the previous construction for 4-OT from 2-OT, the protocol we need in the GMW setting needs 3 multi-party 2-OT protocols. In total, the gate evaluation takes \(3m - 3\) two-party 2-OT protocols and is more efficient than the original gate evaluation with pairwise OT protocols in GMW protocol.

### 2.4 Permutation Network

Switching permutation networks are circuits of switches that can produce any permutation of an input. A construction for these networks was proposed by Waksman \cite{Wak68}. A switch is a logic gate with two inputs \(x_1\) and \(x_2\) and two outputs \(y_1\) and \(y_2\) and a
special selection input \( s \). The selection bit specifies whether the output is an identity \( y_1 = x_1 \) and \( y_2 = x_2 \) or the wires are swapped to give \( y_1 = x_2 \) and \( y_2 = x_1 \).

Waksman proposed a construction where for \( N = 2^k \) inputs \( N \log N - N + 1 \) switches are needed. Other constructions exist also for general values of \( N \), but introduce additional complexity \([\text{WHH}+12]\). The construction works recursively, by reducing the computation of the whole permutation \( \{1, \ldots, N\} \rightarrow \{1, \ldots, N\} \) to \( N - 1 \) switches and two computations of \( \{1, \ldots, N/2\} \rightarrow \{1, \ldots, N/2\} \). The base case is the permutation of two elements that can be computed by one switch. One layer of the construction can be seen on Fig. 2 where \( O_1 \) can always be corresponding to identity and can be left out. Independently of the definition of the permutation, the network will always have the same topology, but based on the permutation definition, the switches need to be fixed with a concrete selection bit. A concrete network with functionalities of the switches can be built from the permutation matrix.

By adding the outer layer \( O_1 \) switch to the network despite being redundant, the network can always have a shape where all switches either give both wires as outputs or both wires feed other switches in the network. Hence, instead of talking about output wires, it is reasonable to consider output switches.

3 General Framework

The beauty of the work \([\text{MS}13]\) is that it provides a very modular framework for private function evaluation illustrated on Fig. 3. Essentially, their idea follows from an observation, that a circuit has two important aspects that the construction should hide: the topology and the function of each gate. Especially, they show that the topology can be hidden using a tool that they call extended permutation and they also provide a construction for securely evaluating the extended permutation using switching networks. However, due to the modularity their idea can be extended with other concrete construc-
Figure 3: Private function evaluation components as considered in this work.

<table>
<thead>
<tr>
<th>Circuit Hiding MPC (CH) (Sec 3.3)</th>
<th>Circuit Topology Hiding (CTH) (Sec. 3.1.4)</th>
<th>Extended Permutation (OEP) (Sec. 3.1.2)</th>
<th>Private Gate Evaluation (PGE) (Sec. 3.2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple CH</td>
<td>Switching Networks (Sec 3.1.3)</td>
<td>MPC techniques</td>
<td></td>
</tr>
</tbody>
</table>

tions for the two main building blocks. For example, they also consider homomorphic encryption based extended permutations. Secondly, the private gate evaluation is in itself a private function evaluation problem, however, as it is a special case, it can be solved by cleverly using traditional MPC ideas.

There are three things that this framework has to reveal about the circuit. Firstly, the number of inputs has to be known in order for the other parties to correctly give their private values as inputs. Secondly, the number of outputs is not hidden as all parties must jointly work to reveal the outcomes. Thirdly, also the number of gates is not hidden as the circuit is still evaluated gate-by-gate collaboratively by all parties.

This section gives a general overview of the constructions from [MS13] as well as clearly marked examples based on the circuit from Example 1. The examples are intended to make it easier to grasp new ideas, but can freely be skipped if the previous theoretical explanation was sufficient for the reader.

3.1 Circuit Topology Hiding

The topology of the circuit is defined by the connections between separate gates. Private evaluation however needs to hide this as it can give information about what the function does. For a concrete example, it would be easy to see, if some inputs are not used at all, or which inputs contribute to which outputs. For an experienced reader, the Sec. 3.1.4 gives a short summary of the key aspects to hiding the topology. Otherwise Sec. 3.1.1 provides the general viewpoint about topology and Sec. 3.1.2 discusses oblivious evaluation of extended permutations and together they build up the intuition for Sec. 3.1.4.

3.1.1 General idea of topology hiding

Firstly, consider a labelling of outgoing wires to better characterize the circuit. They are numbered based on the gate numbers, where each gate $G_i$ has output $ow_{i+n}$ for circuit with $n$ inputs. All together, there are $2g$ incoming wires and $n$ outgoing wires for the circuit inputs and $g-o$ outgoing wires for intermediate gate outputs.

**Example 2.** The example circuit with separate labelling of the incoming and outgoing wires is illustrated on Figure 4. Respectively, five inputs of the circuit are labelled as $ow_1$ to $ow_5$ for outgoing wires. This can be thought of as outputs of special input gates. They are connected with $iw_i$ for $i \in \{1, 2, 3, 4, 8\}$ to the incoming wires that feed the gates. Following the introduced rule, the outgoing wire of $G_2$ is $ow_{5+2}$ because we have
Figure 4: Example circuit with incoming and outgoing wire labelling

\[
\begin{align*}
\text{ow}_1 &= x_1 & \text{iw}_3 \\
\text{ow}_2 &= x_2 & \text{iw}_4 \\
\text{ow}_3 &= x_3 & \text{iw}_1 \\
\text{ow}_4 &= x_4 & \text{iw}_2 \\
\text{ow}_5 &= x_5 & \text{iw}_8 \\
\text{ow}_7 &= & \text{iw}_5 \\
\text{ow}_6 &= & \text{iw}_6 \\
\text{G}_2 & & \text{G}_3 \\
\text{G}_1 & & \text{y}_1 \ 	ext{y}_2
\end{align*}
\]

Figure 5: Connections \(\pi_C\) for the example circuit, (a) Alignment of the labels from Fig. 4 (b) reverse function \(\pi_C^{-1}\) used in circuit evaluation

\[
\begin{array}{c|c}
 x & \pi_C^{-1}(x) \\
 1 & 3 \\
 2 & 4 \\
 3 & 1 \\
 4 & 2 \\
 5 & 7 \\
 6 & 6 \\
 7 & 6 \\
 8 & 5 \\
\end{array}
\]

\(n = 5\). The outputs of the circuit however play a different role and are not included in the numbering as they are not mapped with any incoming wires.

Circuit topology can be defined as a relation \(\pi_C\) from the outgoing wires to the incoming wires of the gates. Especially, because the rules in the wire numbering connect incoming and outgoing wires to concrete gates, the whole topology can be captured. To evaluate \(G_i\), we can apply the inverse of \(\pi_C\) to find which inputs to use.

**Example 3.** Fig. 5 describes the connections corresponding to our example. Essentially, the (a) part gives a realignment of Fig. 4 where the gates have been removed. For example, the outgoing wire \(\text{ow}_7\) of gate \(G_1\) is connected to two incoming wires \(\text{iw}_6\) and \(\text{iw}_7\) of gates \(G_3\) and \(G_4\) respectively. Also, the input wires are included, as for example, \(\text{ow}_5\) matches to \(\text{iw}_8\). The part (b) gives an evaluation table for the function \(\pi_C^{-1}(x)\) that for the input \(x\) meaning the number of the wire \(\text{iw}_x\) gives the number of the corresponding outgoing wire \(\text{ow}_{\pi_C^{-1}(x)}\).

In summary, the key to hiding the topology is in secure evaluation of the \(\pi_C^{-1}\) mapping. Note that, if each output wire would be an input to only one gate, then \(\pi_C\) is a permutation. However, due to the possible cases where one wire gives input to many gates we call this an extended permutation.
3.1.2 Oblivious Extended Permutation

**Definition 1** (Extended permutation). A relation \( \pi : \{1, \ldots, M\} \to \{1, \ldots, N\} \) is an extended permutation, if for each \( y \in \{1, \ldots, N\} \) there exists exactly one \( x \in \{1, \ldots, M\} \) such that \( \pi(x) = y \). Hence, \( \pi^{-1} = \{1, \ldots, N\} \to \{1, \ldots, M\} \) is a surjection.

To securely evaluate \( \pi_C \), we can not use it in a straightforward way as this would reveal the whole topology. To use \( \pi_C \) we in fact only need the previously computed value corresponding to this wire as this would be the input value to the new gate and not the label. However, the values only become known during the evaluation of the circuit, therefore this mapping must to be reactive. Firstly, each time an output of a gate is computed, it is inserted into the map \( \pi_C^{-1} \). Secondly, it must be possible to retrieve this input later using the label of the input wire that carries this value.

**Example 4.** In our example, we evaluate gate \( G_1 \) and store the output value as the value of wire \( ow_7 \) in the mapping, then we evaluate gate \( G_2 \) and store the value for \( ow_6 \). After that we come to gate \( G_3 \) that requires the previously computed \( ow_6 \) as an input on wire \( iw_5 \). The reveal query with input 5 to the mapping \( \pi_C^{-1} \) gives us this value. Analogously, reveal query with input 6 gives the value stored for \( ow_7 \).

Even this does not yet ensure security because the whole values on the wires leak both the secret inputs as well as the topology. However, this is the problem solved by traditional secure multi-party computation as it only remains to ensure the privacy of the function inputs and intermediate values. To that end, secret sharing is used so instead of each party seeing the value of the wire, they only learn a fresh share. Using secret sharing we obtain the oblivious evaluation of the extended permutation.

**Example 5.** For example, assume that two parties jointly evaluate a gate \( G_2 \) in our example and both of them respectively get a private output \([x]_1 \) and \([x]_2 \) where these values code the output \( x \). They store this in the mapping as corresponding to outgoing wire \( ow_6 \). Next, they start to evaluate \( G_3 \) and by querying the mapping with input 5 for \( iw_5 \) they should learn the value they just stored. However, in this case, the first party obtains \([x’]_1 \) and the second party obtains \([x’]_2 \) where \( x = x’ \), but the share \([x’]_i \), is independent of the respective shares \([x]_i \), that either party gave to the functionality.

3.1.3 Oblivious Extended Permutation from Switching Networks

One general way how to build an extended permutation is by using a switching network with more general switches than considered for Waksman permutation networks in 2.4. A generalised 2-switch has two selection bits and correspondingly four options for outputs - it either behaves as a common switch or copies one of the inputs to both outputs. Fig. 6 illustrates the two types of switches using their evaluation tables.

The input to the extended permutation is \( M \) actual values followed by \( N - M \) dummy values. The total switching network for extended permutation has three components. Firstly, the dummy values are placed to follow the outputs that are replicated. Secondly,
the dummy inputs are replaced by the actual values that need to be replicated in the final output. Finally, the actual permutation takes place as illustrated on Fig. 7.

Both the final permutation and dummy placement phase can be implemented using Waksman permutation networks. The basic 1-switches are special cases of the 2-switches required for the extended permutation network. The replication can be computed by a series of \( N - 1 \) 2-switches where the only possible inputs are \((0, 0)\) to replicate the first input value and \((1, 0)\) to keep both outputs the equal to the respective inputs. In total this means that the network has \( 2(N \log N - N + 1) + N - 1 \) switches for \( N \) inputs.

**Example 6.** This switching network for our example is illustrated by Fig. 7. The only replicated output is \( \text{ow}_6 \). Hence, the dummy placement outputs the order \( \text{ow}_1, \text{ow}_2, \text{ow}_3, \text{ow}_4, \text{ow}_5, \text{ow}_6 \), dummy value and \( \text{ow}_7 \). The output of the dummy placement phase is changed into \( \text{ow}_1, \text{ow}_2, \text{ow}_3, \text{ow}_4, \text{ow}_5, \text{ow}_6, \text{ow}_7 \) and \( \text{ow}_6 \) during the replication. Finally the actual permutation is applied that outputs the values in order given by Fig. 5.
3.1.4 Requirements for Circuit Topology Hiding

The previous exploration of what it means to hide a circuit topology amounts to securely evaluating the extended permutation (Def. 1) in an on-demand manner. The setting of securely evaluating an extended permutation with multiple parties is the following. The first party $P_1$ knows the extended permutation $\pi$. All parties $P_i$ know a sharing of the input vector $\vec{x} = (x_1, \ldots, x_M)$ where each element $x_i$ has a fixed length $\ell$. After the secure evaluation, the parties hold the shares of $(x_{\pi^{-1}(1)}, \ldots, x_{\pi^{-1}(N)})$ and learn nothing else.

Any protocol that securely corresponds to this functionality is an oblivious permutation. The definition of the circuit topology hiding functionality captures the on-demand requirement. At first, the $P_1$ inputs $\pi_C$. On each mapping query, all parties $P_i$ input their shares and the label $i$ for $\omega_i$. On each reveal query with input $j$ the value for $\omega_{\pi^{-1}(j)}$ is given out as a fresh share for each parties. It is easy to see that this description coincides with the description of the oblivious extended permutation functionality with addition of the iterations with mapping and revealing queries.

3.2 Private Gate Evaluation

Somewhat recursively, privately evaluating one gate is actually an important special case of private function evaluation. However, it is easier than evaluation of general functions, since each gate has fixed number of inputs and outputs and only limited amount of possible functionalities. For our case, each gate has two inputs and one output and the number of functionalities depends on the size of the inputs and outputs.

For example, for boolean gates one oblivious transfer protocol (as considered in Sec. 4) or one garbled gate evaluation would be sufficient. On the other hand, for arithmetic circuits we can employ fully homomorphic encryption. However, how the gate is evaluated depends on the concrete secure multi-party framework used for the computations.

3.3 Private Function Evaluation

Finally, it remains to combine the topology hiding property with secure evaluation of a single gate to obtain private function evaluation. However, this is straightforward, as the parties go through the gates in the topological order defined by $P_1$, each time using the circuit topology hiding functionality to learn the input shares, then invoking the private gate evaluation where $P_1$ always knows the gate and then including their private outputs to the map. This cycle of evaluating one gate is also pictured by Fig. 8.

Inputs and outputs require special handling. During the input phase parties can secret share their inputs. Each party then includes these in the circuit topology hiding functionality. After the evaluation, there is no need to give the shares corresponding to the outputs back to the CTH functionality. Rather, these shares can either be opened following the traditional MPC functionality or used in subsequent computations.

Example 7. Hence, the circuit hiding secure evaluation would proceed as follows.
Figure 8: Steps of evaluating gate $G_j$

- The party $P_1$ has the circuit. It sorts the gates in the topological order $G_1$ to $G_4$, labels the wires and fixes the mapping $\pi_C$.
- Each party $P_i$ secret shares its inputs between all parties.
- Each party $P_i$ inputs its shares to the map.
- For each gate $G_j$ the gate is evaluated:
  - The parties follow the procedure on Fig. 8
  - If $j \in \{3, 4\}$ meaning it is an output gate then the mapping is omitted
- For $G_3$ and $G_4$ the parties reveal the output $c$ using $[c]$. 

4 Concrete Initialisation

The general framework can be applied to different secure multi-party computation techniques. The paper explores the GMW protocol and Yao's garbled circuits [Yao82] in case of boolean circuits and two-party arithmetic circuits based on fully and additively homomorphic encryption. In this section we explore the case where multiple parties are evaluating a boolean circuit using the GMW style. Hence, we go over the general ideas presented in Sec. 3 and give concrete protocols for these ideas. We assume that there are $m$ parties $P_1$ to $P_m$ where the first party $P_1$ is the one that inputs the function and all parties can provide inputs to the function. The secret sharing is bitwise XOR sharing and also the rest of the setting is the same as the GMW explained in Sec. 2.3. To achieve a more complete overview of the original paper, we discuss the switching network based protocol for oblivious extended permutations.

4.1 Private Gate Evaluation

The main part of private function evaluation that was left unclear by the general exposition of the idea is how exactly can parties compute one gate privately. The setting
Figure 9: Truth table for gate $G$

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>$G(0, 0)$</td>
<td>$G(0, 1)$</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>$G(1, 0)$</td>
<td>$G(1, 1)$</td>
</tr>
</tbody>
</table>

is that party $P_1$ gives the input that determines the functionality of the gate $G$ and all parties provide the input shares for the two inputs $a$ and $b$. As an outcome all parties learn the shares of $G(a, b)$.

In fact, the core idea is to use something similar to the basic GMW type gate evaluation, namely oblivious transfer. However, here we clearly require multi-party OT for four inputs. Using these tools, the protocol becomes straightforward as given in Alg. 2.

**Algorithm 2** Multi-party private gate evaluation

**Data:** Shared inputs $[a]$ and $[b]$, $P_1$ knows $G$

**Result:** Shared result $[G(a, b)]$

1: $P_1$ generates the truth table for the functionality $G$ as given in Fig. 9.
2: multi-party OT protocol messages from Fig. 9 selection indices $[a]$ and $[b]$

We can use the multi-party OT protocol from Alg. 1 to build the case with 4 inputs as discussed for basic GMW protocol. This means that for PGE each gate has to be evaluated as an AND gate in the traditional GMW case. The added overhead of private gate evaluation depends on the concrete circuit. For traditional GMW one would focus on circuits with few AND gates and not caring about XOR gates whereas private gate evaluation is most efficient for circuits with the smallest total number of gates. For $g$ gates and $m$ parties private gate evaluation requires $g \cdot (3m - 3)$ traditional two-party 2-OT protocols. For fixed number of parties the complexity is linear in the circuit size.

### 4.2 Circuit Topology Hiding

The topology hiding functionality can be achieved in many ways, for example using fully homomorphic encryption or switching networks. In here, we consider the version with switching networks. Essentially, as discussed in the general overview, the required protocol must achieve the multi-party oblivious extended permutation property. Similarly to the PGE functionality, also the multi-party OEP can be obtained from 2-party OEP. The construction is given in Alg. 3. The $m$-party OEP protocol extends the complexity of the two-party protocol by a factor of $m - 1$.

#### 4.2.1 Two-party secure switching networks

In the beginning, $P_2$ holds the input vector $\vec{x}$ and $P_1$ knows the selection bits that define the switches. In addition, $P_1$ has a blinding vector $\vec{t}$. After the protocol, $P_2$ has the output $\vec{y}$, but blinded with $\vec{t}$ where $\vec{y}$ and $\vec{t}$ give the secret shared output of the network.
**Algorithm 3 Multi-party Oblivious Extended Permutation**

**Data:** \( P_1 \) has extended permutation \( \pi, [\vec{x}] = ([x_1], \ldots, [x_M]) \)

**Result:** Shared permuted vector \([\vec{\pi}] = ([x_{\pi^{-1}(1)}], \ldots, [x_{\pi^{-1}(N)}])\)

1. \( P_1 \) fixes \( \vec{y} = \pi([\vec{x}]) \)
2. for \( i \in \{2, \ldots, n\} \) do
3. \( P_1 \) generates a random vector \( \vec{t} = (t_1, \ldots, t_N) \)
4. \( P_1 \) and \( P_i \) engage in 2-party OEP with inputs \( \pi, [\vec{x}], \vec{t} \)
5. \( P_i \) obtains \( \vec{\pi} = ([x_{\pi^{-1}(1)}] \oplus t_1, \ldots, [x_{\pi^{-1}(N)}] \oplus t_N) \)
6. end for
7. \( P_1 \) fixes \( [\vec{\pi}]_1 = \vec{y} \oplus \bigoplus_{i=2}^{n} \vec{t} \)

**Figure 10:** Secure evaluation table for a 2-switch

<table>
<thead>
<tr>
<th>( (s_0, s_1) )</th>
<th>( y_1 )</th>
<th>( y_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0)</td>
<td>( x_i \oplus r_k )</td>
<td>( x_i \oplus r_l )</td>
</tr>
<tr>
<td>(1, 0)</td>
<td>( x_i \oplus r_k )</td>
<td>( x_j \oplus r_l )</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>( x_j \oplus r_k )</td>
<td>( x_i \oplus r_l )</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>( x_j \oplus r_k )</td>
<td>( x_j \oplus r_l )</td>
</tr>
</tbody>
</table>

**Evaluation of a single 2-switch** Consider a switch \( u \) with input wires \( w_i \) and \( w_j \) and output wires \( w_k \) and \( w_l \). \( P_2 \) chooses randomisers \( r_i, r_j, r_k \) and \( r_l \) for each of the wires and gives \( P_1 \) the blinded input \( x_i \oplus r_i \) and \( x_j \oplus r_j \). Finally, \( P_1 \) should learn the blinded outputs as in Fig. [10] based on the selection bits and \( P_2 \) keeps \( r_k \) and \( r_l \) as its outputs.

A switch can be implemented using a 1-out-of-4 oblivious transfer. \( P_2 \) uses the randomisers to create the table on Fig. [11] and \( P_1 \) uses its selection bits as inputs to the OT. Finally, \( P_1 \) uses the outputs from the OT to compute the blinded outputs. For example, for selection bits \( (0, 0) \) \( P_1 \) learns \( y_1 = r_i \oplus r_k \) and \( y_2 = r_i \oplus r_l \) and it knows that this switch replicates \( x_i \) to both outputs. Hence, it computes \( x_i \oplus r_i \oplus y_1 = x_i \oplus r_k \) for the first output and \( x_i \oplus r_i \oplus y_2 = x_i \oplus r_l \) for the second output. However, for selection \( (0, 1) \) it uses \( x_j \oplus r_j \oplus y_1 = x_j \oplus r_k \) for the first output and \( x_i \oplus r_i \oplus y_2 = x_i \oplus r_l \).

**Two-party secure switching networks** The algorithm for evaluating the whole network using the ideas from single switch evaluation is given in Alg. [4] Note that all parties know the layout of the switching network with \( q \) switches and \( N \) inputs.
and outputs, however only \( P_1 \) knows the functionalities of the switches. The switching network computes the functionality of the extended permutation \( \pi \). This network has \( 2q + N \) wires that we denote by \( w_i \) for \( i \in \{1, \ldots, 2q + N\} \). We number the switches using index \( u \) and assume that a switch \( u \) has input wires \( w_i \) and \( w_j \) and outputs \( w_k \) and \( w_l \) as in the previous example of single switch evaluation.

**Algorithm 4** Secure evaluation of a 2-party switching network

**Data:** \( P_1 \) has the selection bits \( \vec{s} = ((s_0(1), s_1(1)), \ldots, (s_0(q), s_1(q))) \) and a blinding vector \( \vec{t} = (t_1, \ldots, t_N) \)

**Data:** \( P_2 \) has \( \vec{x} = (x_1, \ldots, x_N) \)

**Result:** \( P_2 \) learns \( z_i = x_{\pi^{-1}(i)} \oplus t_i \) for \( i \in \{1, \ldots, N\} \), \( P_1 \) outputs \( \vec{t} \).

1. \( P_2 \) generates \( r_i \) for every wire \( w_i \)
2. **for every switch \( u \) do**
3. \( P_2 \) creates a table like Fig. 11
4. Parties run 1-out-of-4 OT, where \( s(u) = 2s_1(u) + s_0(u) \), \( P_1 \) stores \((T^0_{s(u)}, T^1_{s(u)})\).
5. **end for**
6. **for** Each input wire \( w_i \) where \( i \in \{1, \ldots, N\} \) **do**
7. \( P_2 \) sends \( y_i = x_i \oplus r_i \) to \( P_1 \)
8. **end for**
9. **for every switch \( u \) in topological order do**
10. \( P_1 \) computes using the results \((T^0_{s(u)}, T^1_{s(u)})\) for \( u \):
11. **if** \( s_0(u) = 0 \) **then**
12. \( y_k = y_i \oplus T^0_{s(u)} \)
13. **else**
14. \( y_k = y_j \oplus T^0_{s(u)} \)
15. **end if**
16. **if** \( s_1(u) = 0 \) **then**
17. \( y_l = y_i \oplus T^1_{s(u)} \)
18. **else**
19. \( y_l = y_j \oplus T^1_{s(u)} \)
20. **end if**
21. **end for**
22. **for every output switch \( u \) do**
23. \( P_1 \) computes \( z_k' = y_k \oplus t_{2(u-1)+1} \), \( z_l' = y_l \oplus t_{2u} \) and sends to \( P_2 \)
24. \( P_2 \) removes the blinding \( z_k = sk' \oplus r_k \) and \( z_l = z'_l \oplus r_l \).
25. **end for**

Basically, Alg. 4 applies the single switch evaluation procedure to all switches. However, for efficiency reasons instead of going in a large loop switch by switch, some operations are grouped together. Especially, all the oblivious transfers can be done together because they do not depend on the actual inputs. In addition, \( P_2 \) only supplies the initial inputs but does not have to know anything about the intermediate values computed in the network, hence \( P_1 \) can do the evaluation locally after learning the blinded inputs from
This way, instead of doing an interactive OT for each switch, the resulting protocol has constant number of rounds independently of the size of the switching network.

**On-demand evaluation**  This protocol also allows for on-demand evaluation where not all inputs are known in the beginning. The output of the switching network can be found if the input moving to this output is fixed. Hence, $P_1$ could evaluate it in paths, provide outputs and learn new inputs after private gate evaluation. $P_1$ has to keep track of all inputs and intermediate states, $P_2$ has to keep all the blinding values $r_i$.

### 4.3 Complexity of GMW style circuit hiding

The switching networks for the extended permutation have size $2N \log N - N + 1$ for $N$ inputs. Each of the switches can be evaluated using one 1-out-of-4 OT. Actually, although it uses general switches for the description, each phase of the computation (dummy placement, replication and permutation) still uses a special flavour of the 2-switch with only two possible outcomes. Hence all necessary switches can be evaluated using a multi-party 2-OT. If we bound input size based on the circuit size $N < g$ then it requires $O(g \log g)$ 1-out-of-2 OT protocols for 2-parties. Given the construction in Alg. 3 it results in $O((m - 1)g \log g)$ traditional OT executions for the $m$-party case.

The PGE has $g \cdot (3m - 3)$ basic 2-party 2-OT calls as noted before. In total, the complexity of this approach is $O(mg + mg \log g)$ OTs for $m$ parties and $g$ gates.

## 5 Related Work

Besides the GMW-based protocol, [MS13] also considers Yao’s garbled circuits and two-party arithmetic case. For the multi-party case, the proposed GMW construction with homomorphic encryption based OEP is the first multi-party PFE achieving linear complexity for the circuit size. Also the two-party cases with homomorphic encryption improve the linear complexity of previous constructions. However, given current achievements regarding precomputing OT and OT-extensions, it is likely that the introduced switching network based GMW style PFE is at present the most practical approach.

The first specific PFE problem that was considered focused on two parties where one has the function and the other gives the input [AF90]. Secure evaluation of branching programs has also been one two-party PFE approach that follows the same client-server setting [IP07, BPSW07, BFK+09]. Another special case where the class of allowed functions is limited is discussed in [PSS09]. Some protocols have also been considered for specific functionalities. The examples include two-party private polynomial evaluation [CL01], secure verification against unknown access policies [FAL04], privacy preserving credit checking [FAZ05, FLA06], and intrusion detection [NSMS14].

The main idea for general PFE has been to use universal circuits and traditional MPC techniques. A universal circuit has one input that encodes the desired function and any functions up to a fixed size can be evaluated by the circuit. This approach demonstrates that it is feasible to have private functions as inputs to secure computations. However,
the current optimal sizes of the universal circuits to evaluate \( g \) gates are \( O(g \log g) \) gates for boolean circuits [Val76] and \( O(g^5) \) gates for arithmetic circuits [Raz08, SY10] making it likely that special protocols for PFE could outperform this approach. Special design of universal circuits for PFE has been considered in [KS08, SS08].

Another general solution is to use fully homomorphic encryption that in theory can provide efficient results when the encryption schemes become practical. For example, [KM11] achieves linear complexity in the circuit size using singly homomorphic encryption in the two-party setting. The paper [MS13] offers a more efficient alternative for two-party arithmetic circuits as it can combine additively homomorphic and singly homomorphic encryption. Limiting the rounds of communication for PFE was also studied before fully homomorphic encryption was shown to exist. For example, two-party evaluation of logarithmic depth circuits in a non-interactive setting in [SYY99].

In addition, [LW15] proposes an extended permutation evaluation for multi-party arithmetic circuits. They consider the setting where the party that knows the function does not have to directly participate in the computations, but can just give the function as an input. In addition, the function could also be a result of previous computations.

**Malicious security model** Clearly, the circuit hiding property can also be obtained in the case of malicious adversary. For universal circuits malicious security is achievable using actively secure MPC techniques to evaluate the circuit. In addition, usage of fully homomorphic encryption and zero knowledge proofs are likely to provide a linear complexity malicious private function evaluation. On the more practical side, the introduced framework has also been adapted to the case of malicious adversary [MSS14]. They propose a framework that can be achieved using actively secure secret sharing based MPC that allows the computation of reactive functionalities. They show both the possibility to achieve linear complexity with a special OEP protocol and more practical approach with constant rounds and \( O(g \log g) \) complexity using the switching networks.

Overall, [MSS14] follows the same pattern as introduced in this overview to enable arithmetic circuit evaluation over a finite field. Similarly to recent actively secure MPC protocols they work in the preprocessing model. Their offline phase depends on the private function, but not on the inputs of other parties. The offline phase precomputes random shared and authenticated vectors and the extended permutation on one of the vectors and \( P_1 \) commits to all gate types. This phase deploys ideas from MPC. The linear complexity also requires a zero-knowledge proof to show that the used relation is an extended permutation. On the other hand, \( O(g \log g) \) complexity uses a switching network protocol quite like introduced in this overview, but deploying MPC instead of OT. The online computation turns the ideas of secure computation somewhat upside down, by keeping one-time-pad passwords shared, but performing computations on the public ciphertexts. Analogously, the MAC values that protect the integrity are public and the keys remain secret shared. \( P_1 \) uses the shared outcome of the offline phase to evaluate the circuit on publicly known values whereas other parties verify this work by using the underlying MPC protocol and MACs. Hence obtaining specific protocols for the online phase that partially use the protocols from the underlying MPC scheme.
6 Conclusion

The paper [MS13] provided an interesting and promising framework for hiding the functions computed in secure multi-party computation. The general idea is that a function is given as a boolean or arithmetic circuit where it is necessary to hide the functionality of each gate as well as the connections between the gates. The latter is achieved via the use of oblivious evaluation of extended permutations. Unfortunately, there is no good guidelines for achieving private gate evaluation and the specifics of the underlying secure computation scheme have to be taken into account.

References


