Highly Efficient and Reusable Private Function Evaluation with Linear Complexity

Osman Biçer\textsuperscript{1}, Muhammed Ali Bingöl\textsuperscript{2,3}, Mehmet Sabır Kiraz\textsuperscript{3}

\textsuperscript{1} Koç University, Graduate School of Sciences and Engineering, İstanbul, Turkey
\textsuperscript{2} Sabancı University, Faculty of Engineering and Natural Sciences, İstanbul, Turkey
\textsuperscript{3} TÜBİTAK BİLGEM Kocaeli, Turkey

Abstract. Private function evaluation aims to securely compute a function $f(x_1, \ldots, x_n)$ without leaking any information other than what is revealed by the output, where $f$ is a private input of one of the parties (say Party$_1$) and $x_i$ is a private input of the $i$-th party Party$_i$. In this work, we propose a novel and secure two-party private function evaluation (2PFE) scheme based on DDH assumption. Our scheme introduces a reusability feature that significantly improves the state-of-the-art. Accordingly, our scheme has two variants, one is utilized in the first evaluation of the function $f$, and the other is utilized in its subsequent evaluations. To the best of our knowledge, this is the first and most efficient special purpose PFE scheme that enjoys a reusability feature. Our protocols achieve linear communication and computation complexities and a constant number of rounds which is at most three.

Keywords: Private function evaluation, Secure 2-party computation, Communication complexity, Cryptographic protocol.

1 Introduction

Private function evaluation (PFE) is a special case of secure multi-party computation where the function to be evaluated is also a private input of one of the parties. In this paper, we consider the two-party PFE (2PFE) setting where the first party (say Party$_1$) has a function input $f$ (compiled into a boolean circuit $C_f$) and optionally a private input bit string $x_1$, whereas the other party (say Party$_2$) has an input bit string $x_2$. The parties aim to evaluate $f$ on $x_1$ and $x_2$ so that at least one of them would obtain the resulting $f(x_1, x_2)$ without any of them deducing any information about the other one’s private input beyond what $f(x_1, x_2)$ itself reveals.

Efficient and practical PFE schemes are becoming increasingly important as many real-world applications require protection of their valuable assets. For example, many software companies targeting the global market are extremely concerned about illegal reproduction of their software products. Software obfuscation methods usually prevent reverse engineering, but still allow direct copying

\textsuperscript{4} Note that PFE also comprises the setting where $x_1 = \bot$. 
of programs. Another solution could be providing the software-as-a-service in the cloud to eliminate the risk of exposure. However, this solution also causes another issue, i.e., threatening the privacy of customer data, since computations need to take place at the hands of software vendors. Fully homomorphic encryption (FHE) can also be a potential solution to such problems [Gen09], but, unfortunately, it is still far from being practical [HS15]. Another decent approach targeting those problems falls into the category of PFE. Compared to FHE, PFE is currently much closer to practical use. Moreover, in many occasions such a PFE scheme is quite beneficial, including the ones where a service provider may opt keeping the functionality and/or its specific implementation confidential, and the ones where the disclosure of the function itself means revelation of sensitive information, or causes a security weakness.

The current research goal for secure computation protocols (including PFE) is efficient and practical solutions with low round, communication, and computation complexities. Among these three measures, as also pointed out by Beaver, Micali, and Rogaway, the number of rounds is the most valuable resource [BMR90]. The other important research goal in this area is the minimization of communication complexity. Since hardware trends show that computation power progresses more rapidly compared to communication channels, the main bottleneck for many applications will be the bandwidth usage.

1.1 Related Work

First proposed by Andrew Yao [Yao82, Yao86], secure two-party computation (2PC) comprises the techniques for joint evaluation of a function by two parties on their respective secret inputs. In recent years, there have been a promising progress over the original Yao’s protocol [BMR90, NPS99, KS08a, PSSW09, KMR14, ZRE15, KKS16]. As a consequence of these improvements, secure computation techniques now have promising results.

2PFE differs from the standard 2PC in that the latter involves both parties evaluating a publicly known function on their private inputs, whereas in the former, the function itself is also a private input. 2PC concept is first appeared in [AFK87, AF90]. So far, there are basically two main approaches that PFE solutions are built upon.

The first one is based on a universal circuit which takes a boolean circuit $C$ with circuit size less than $g$ and an input $(x_1, \ldots, x_n)$, and outputs $C(x_1, \ldots, x_n)$. The idea is that if the regular secure computation techniques can be applied on a universal circuit, then a PFE scheme can be obtained. Consequently, the efforts targeting the efficiency of universal circuit based PFEs have generally been towards reducing the size of universal circuits, and the cost of their secure computation [KS08b, SS09, KS16, GKS17].

The second approach is avoiding the use of universal circuits and designing special purpose PFE protocols. Following this line of work, several PFE schemes have been proposed, e.g., as [PSS09, KM11, MS13, Sad15, BBKL17]. A remarkable work embracing this approach is singly homomorphic encryption based 2PFE scheme of Katz and Malka applied on boolean circuits [KM11]. [KM11] utilizes
a singly homomorphic scheme (e.g., ElGamal \cite{ElGamal1985} or Paillier \cite{Paillier1999}) for the generation of the two random tokens on each wire, later utilized in the 2PC stage. They first propose a basic version of their protocol in \cite[Sect. 3.1]{Kim2011} and for the efficiency concerns they propose a more efficient variant in \cite[Sect. 3.2]{Kim2011}. Both schemes have only three rounds, and provides $O(g)$ asymptotic complexity in terms of communication and computation, where $g$ denotes the circuit size. The latter one reduces the communication and offline computation complexity.

In \cite{Mohassel2013}, Mohassel and Sadeghian proposed 2PFE schemes, for boolean circuits and arithmetic circuits. Considering boolean circuits, they propose two types of protocols: one is based on oblivious evaluation of switching networks (OSN) (what we call \cite{Mohassel2013}-OSN) and the other one is based on singly homomorphic encryption (what we call \cite{Mohassel2013}-HE). Even though \cite{Mohassel2013}-OSN is efficient for small sized circuits, it is still inefficient for large circuits due to its $O(g \log(g))$ communication and computation complexities. It fails to outperform asymptotically linear communication and computation complexities of \cite{Kim2011}. On the other hand, \cite{Mohassel2013}-HE provides linear communication and computation complexities and slightly outperforms \cite[Sect. 3.2]{Kim2011}. We remark that to the best of our knowledge, a reusability feature cannot be adapted to \cite{Kim2011}, \cite{Mohassel2013}-HE, or \cite{Mohassel2013}-OSN.

\cite{Mohassel2013} also proposes a protocol for arithmetic circuits based on partial (singly) homomorphic encryption. This protocol has equal number of rounds to its gates (see \cite[p. 570]{Mohassel2013}), whereas the other PFE protocols for boolean circuits have constant number of rounds. For large circuits the number of rounds will be a bottleneck. \cite{Mohassel2013} also proposes a multi-party PFE variant based on OSN that remains the most efficient one to date. Their proposals are essentially secure in the semi-honest model, and has later been extended to the malicious model by \cite{Mohassel2014}.

Recently \cite{Barak2017} improves the OSN based 2PFE protocol of \cite{Mohassel2013}. They show how to utilize the elegant \textit{half gates} technique \cite{Zhang2015} to their 2PFE scheme. Compared to \cite{Mohassel2013}-OSN protocol, the optimization of \cite{Barak2017} improves by more than 40% reduction in overall communication cost. However, it still has the inherited asymptotical complexities of \cite{Mohassel2013}-OSN.

---

5 Throughout this paper, the term “token” stands for a random bit string generated for a wire of the boolean circuit, and has hidden semantics of either 0 or 1.

6 This is due to the fact that the blinding operations in these protocols are one-time pads (XOR or cyclic addition), therefore, reusing the blinded values inevitably leaks information about the truth values of intermediate wires. On the other hand, our mechanism relies on DDH so that the blinding values would remain unknown to the respective parties.

7 We can intuitively say that as the latency between parties increases, so does the cost of each additional communication round (we refer to \cite{Song2013} that backs up this discussion). A similar analysis on trade-offs between boolean and arithmetic circuit based protocols has also been addressed in \cite[p. 527]{Choudhuri2014}.
1.2 Our Contributions

In this work, we propose a highly efficient 2PFE scheme for boolean circuits secure in the semi-honest model. Our scheme enjoys the cost reduction due to the reusability of tokens that will be used in the 2PC stage. This eliminates some of the computations and exchanged messages in the following re-executions for the same function. Therefore, one of the strongest aspects of our proposed protocol is the remarkable cost reduction if the same function is evaluated more than once (possibly on varying inputs). We highlight that such a cost reduction is not applicable to the protocols of [KM11] and [MS13] since they require running the whole protocol from scratch. In this respect, we present two variants of our scheme: (1) single execution protocol, (2) re-execution protocol. The former protocol is utilized in the first evaluation of the function, while the latter one is utilized in the second or later evaluations of the same function between the two parties. We note that the latter protocol is more efficient than the former one due to the fact that it benefits from the reusable tokens generated already in the first evaluation. The latter case is likely to be encountered more frequently in practice, compared to the cases where the function is evaluated just once between the two given parties.

In what follows, we summarize the contributions of our single and re-execution protocols in comparison with the state-of-the-art protocols.

– Regarding our single execution protocol, independent of the circuit size, our scheme provides 12.5% reduction in communication cost over [KM11, Sect. 3.2]. Compared to [MS13]-OSN, [BBKL17], and [GKS17], our protocol asymptotically reduces the communication cost. Namely, while the asymptotic communication costs of those protocols are equal to $O(g \log(g))$, our first protocol provides $O(g)$ communication complexity (similar to the ones in [KM11]) where $g$ is the number of gates. To illustrate the significance of this asymptotical improvement, our cost reduction is about 94% over [MS13]-OSN, about 88% over [BBKL17], and 68% over [GKS17] for a thousand-gate circuit. Moreover, for a billion-gate circuit, our cost reduction is about 98% over [MS13]-OSN, about 96% over [BBKL17], and about 89% over [GKS17]. Note that the number of protocol rounds of [MS13]-OSN is equal to 6, while that of our first protocol is equal to 3 (similar to the ones in [KM11]).

– Regarding our re-execution protocol, also independent of the circuit size, our scheme achieves about 50% reduction in communication cost over [KM11, Sect. 3.2]. Similar to our first protocol, the asymptotic communication complexity of our second protocol is also superior to [MS13]-OSN, [BBKL17], and [GKS17] (while being equal to those of [KM11]). To clarify, the cost reduction is about 96% over [MS13]-OSN, about 93% over [BBKL17], and about 81% over [GKS17] for a thousand-gate circuit. Moreover, for a billion-gate circuit, the cost reduction is about 99% over [MS13]-OSN, about 97% over [BBKL17], and about 94% over [GKS17]. Moreover, the number of rounds of our re-execution protocol is equal to 1, or 2, or 3 depending on the
Table 1. Comparison with existing constant-round 2PFE schemes (the OSN based [MS13], the HE based [MS13] protocol, both [KM11] protocols, [BBKL17], and [GKS17]) in terms of overall communication (in bits) and online computation costs (in terms of symmetric-key operations), offline computation costs (in terms of symmetric-key operations), and the number of rounds. $M$, $N$, $\lambda$, and $\rho$ denote the number of outgoing wires (i.e., equal to $n + g - m$), the number of incoming wires (i.e., equal to $2g$), the security parameter, and the computation cost ratio, respectively. The costs of [GKS17] are approximated.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Communication</th>
<th>Online Comp.</th>
<th>Offline Comp.</th>
<th>Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>[KM11] Sec.3.1</td>
<td>$(4M + 10N)\lambda$</td>
<td>$(\rho + 2.5)N$</td>
<td>$4(M + N)\rho$</td>
<td>3</td>
</tr>
<tr>
<td>[KM11] Sec.3.2</td>
<td>$(2M + 7N)\lambda$</td>
<td>$(\rho + 2.5)N$</td>
<td>$2(M + N)\rho$</td>
<td>3</td>
</tr>
<tr>
<td>[MS13]-OSN</td>
<td>$(10N \log_2 N + 4N + 5)\lambda$</td>
<td>$6N \log_2 N + 2.5N + 3$</td>
<td>$O(\lambda)$</td>
<td>6</td>
</tr>
<tr>
<td>[MS13]-HE</td>
<td>$(2M + 6N)\lambda$</td>
<td>$(\rho + 2.5)N$</td>
<td>$2(M + N)\rho$</td>
<td>3</td>
</tr>
<tr>
<td>[BBKL17]</td>
<td>$(6N \log_2 N + 0.5N + 3)\lambda$</td>
<td>$6N \log_2 N + N + 3$</td>
<td>$O(\lambda)$</td>
<td>6</td>
</tr>
<tr>
<td>[GKS17]</td>
<td>$(2N \log_2 N)\lambda$</td>
<td>$0.7N \log_2 N$</td>
<td>$2N \log_2 N$</td>
<td>3</td>
</tr>
<tr>
<td>Our 1st Pro.</td>
<td>$(2M + 6N)\lambda$</td>
<td>$(4\rho + 2.5)N$</td>
<td>$(3M - 1)\rho$</td>
<td>3</td>
</tr>
<tr>
<td>Our 2nd Pro.</td>
<td>$4N\lambda$</td>
<td>$(\rho + 0.5)N$</td>
<td>$2(M + N)\rho + 2 \frac{1}{2}$</td>
<td>3</td>
</tr>
</tbody>
</table>

input string length of $\text{Party}_1$ (i.e., owner of $f$). This reflects the improvement of our re-execution variant over other 2PFE protocols also in terms of round complexity.

- We also deal with the case that $\text{Party}_1$ runs the 2PFE protocol for the same private function with various $\text{Party}_2$s separately. This is a common scenario where $\text{Party}_1$ may run a business with many customers for her algorithm/software. Trivially, our re-execution protocol can be utilized between the same two parties in the second and subsequent evaluations after the first evaluation. In order to eliminate the requirement of running the single execution protocol with each $\text{Party}_2$, we propose a more efficient mechanism for the generation of the reusable tokens by employing a threshold based system.

Table 1 compares the existing 2PFE schemes in terms of overall communication and online computation costs, and the number of rounds for $\lambda$-bit security. [MS13] results in $O(N \log_2 N)$ whereas [KM11] and our protocols achieve linear complexity in terms of communication and computation. We also provide Table 2 that depicts a comparison in terms of overall communication costs for various circuit sizes. Note that although the complexity of [MS13]-HE is also comparable to our single execution protocol, in the later executions parties can enjoy the massive cost reduction of our reusability feature, which is not possible for [MS13]-HE.

8 More concretely, if $\text{Party}_1$ has $x_1 = \bot$, then the number of rounds is equal to 1. If $\text{Party}_1$ has a non-empty input $x_1$ in such that the OT extension is not applicable for its garbled input, then the number of rounds is equal to 2. Otherwise, the number of rounds is equal to 3.

9 Note that $M \leq N$, therefore $O(M + N) = O(N)$.
Table 2. Comparison with the existing constant-round 2PFE schemes OSN based protocol of [MS13], HE based [MS13] protocol, both [KM11] protocols, [BBKL17], and [GKS17] in terms of overall communication costs for various circuit sizes. Here we assume that $N = 2M$ (i.e., the number of output bits of the function equals the number of its input bits) and $\lambda = 128$. The values calculated for [GKS17] are approximated.

<table>
<thead>
<tr>
<th></th>
<th>Number of Gates</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$2^{10}$</td>
<td>$2^{15}$</td>
<td>$2^{20}$</td>
<td>$2^{25}$</td>
<td>$2^{30}$</td>
</tr>
<tr>
<td>KM11</td>
<td>0.38 MB</td>
<td>12.00 MB</td>
<td>0.38 GB</td>
<td>12.00 GB</td>
<td>384.00 GB</td>
</tr>
<tr>
<td>KM11</td>
<td>0.25 MB</td>
<td>8.00 MB</td>
<td>0.25 GB</td>
<td>8.00 GB</td>
<td>256.00 GB</td>
</tr>
<tr>
<td>MS13-OSN</td>
<td>3.56 MB</td>
<td>164.00 MB</td>
<td>6.69 GB</td>
<td>264.00 GB</td>
<td>10,048.00 GB</td>
</tr>
<tr>
<td>MS13-HE</td>
<td>0.22 MB</td>
<td>7.00 MB</td>
<td>0.22 GB</td>
<td>7.00 GB</td>
<td>224.00 GB</td>
</tr>
<tr>
<td>BBKL17</td>
<td>1.89 MB</td>
<td>90.50 MB</td>
<td>3.77 GB</td>
<td>151.00 GB</td>
<td>5,776.00 GB</td>
</tr>
<tr>
<td>GKS17</td>
<td>0.68 MB</td>
<td>32.00 MB</td>
<td>1.31 GB</td>
<td>52.00 GB</td>
<td>1,984.00 GB</td>
</tr>
<tr>
<td>Our 1st Pro.</td>
<td>0.22 MB</td>
<td>7.00 MB</td>
<td>0.22 GB</td>
<td>7.00 GB</td>
<td>224.00 GB</td>
</tr>
<tr>
<td>Our 2nd Pro.</td>
<td>0.13 MB</td>
<td>4.00 MB</td>
<td>0.13 GB</td>
<td>4.00 GB</td>
<td>128.00 GB</td>
</tr>
</tbody>
</table>

1.3 Organization

In Section 2, we give a preliminary background that is used throughout the paper. Section 3 presents our baseline mechanism for 2PFE and further precomputation optimization on it for resource efficiency. In Section 4, we further propose two new methods to achieve more efficient re-executions between the two parties and in the presence of multiple Party$s$. Section 5 provides the complexities of our resulting protocols, and compare them with the other state-of-the-art 2PFE protocols. Finally, in Section 6, we give the security proofs of our protocols in the semi-honest model.

2 Preliminaries

This section provides some background information on the DDH assumption and the state-of-the-art generic 2PFE framework.

2.1 Decisional Diffie-Hellman Assumption

The Decisional Diffie-Hellman (DDH) assumption for $G$ provides that the following two ensembles are computationally indistinguishable

$$\{(P_1, P_2, a \cdot P_1, a \cdot P_2) : P_i \in G, a \in_{R} \mathbb{Z}_q^*\} \approx_c$$

$$\{(P_1, P_2, a_1 \cdot P_1, a_2 \cdot P_2) : P_i \in G, a_1, a_2 \in_{R} \mathbb{Z}_q^*\}.$$
where $X \approx_c Y$ denotes that the sets $X$ and $Y$ are computationally indistinguishable. The security of our protocols is based on the following lemma of Naor and Reingold [NR04] providing a natural generalization of the DDH assumption for $m > 2$ generators.

**Lemma 1 ([NR04]).** Under the DDH assumption on $\mathbb{G}$, for any positive integer $m$,

$$\{(P_1, \ldots, P_m, a \cdot P_1, \ldots, a \cdot P_m) : P_i \in \mathbb{G}, a \in R Z_q^* \} \approx_c$$

$$\{(P_1, \ldots, P_m, a_1 \cdot P_1, \ldots, a_m \cdot P_m) : P_i \in \mathbb{G}, a_1, \ldots, a_m \in R Z_q^* \}.$$

There exist certain elliptic curve groups where the DDH assumption holds. We will not go through the details of these primitives and refer the reader to [Bon98, HMV03]. The main advantage of the elliptic curve DDH assumption over the discrete logarithm based DDH assumption is that the discrete logarithm DDH problem requires sub-exponential time [LV01] while the current best algorithms known for solving the elliptic curve DDH problem requires exponential time resulting the same security with smaller key sizes. Therefore, in general, the elliptic curve based systems are more practical than the classical discrete logarithm systems since smaller parameters may be chosen to ensure the same level of security. For example, for the 112-bit symmetric key security level, a 2048-bit large prime number is required for a discrete logarithm group, whereas only a 224-bit prime $p$ is sufficient for a NIST-elliptic curve over $\mathbb{F}_p$ [Gir16].

### 2.2 Notation and Concept of 2PFE Framework

In a two-party private function evaluation (2PFE) scheme, Party$_1$ has a function input $f$ (compiled into a boolean circuit $C_f$) and optionally a private input bit string $x_1$, whereas Party$_2$ has an input bit string $x_2$. The parties aim to evaluate $f$ on $x_1$ and $x_2$ so that at least one of them would obtain the resulting $f(x_1, x_2)$. The recent 2PFE schemes [KM11, MS13] conform to a generic 2PFE framework (formalized by [MS13]) that basically reduces the 2PFE problem to hiding both parties’ input strings and topology of the circuit. The framework is not concerned with hiding the gates since it allows only one type of gate in the circuit structure.

We briefly describe the generic 2PFE framework as follows. Before starting the 2PFE protocol, Party$_1$ compiles the function into a boolean circuit $C_f$ consisting of only one type of gates (e.g., NAND gates). Let $g$, $n$, and $m$ denote the number of gates (circuit size), the number of inputs, and the number of outputs of $C_f$, respectively. Let $\text{OW} = (ow_1, \ldots, ow_{n+g-m})$ denote the set of outgoing wires that is the union of the input wires of the circuit and the output wires of its non-output gates. Note that the total number of elements in $\text{OW}$ is $M = n + g - m$. Similarly, let $\text{IW} = (iw_1, \ldots, iw_{2g})$ denote the set of incoming wires that is the union of the input wires of each gate in the circuit. Note also that the total number of elements in $\text{IW}$ is $N = 2g$. Throughout this paper, $M$ and $N$ denote the numbers of outgoing and incoming wires, respectively. Let
\(\pi_f\) be a mapping such that \(j \leftarrow \pi_f(i)\) if and only if \(ow_i \in \text{OW}\) and \(iw_j \in \text{IW}\) correspond to the same wire in the circuit \(C_f\).

We define the public information of the circuit \(C_f\) as \(\text{PubInfo}_{C_f}\), which is composed of: (1) the number of each party’s input bits, (2) the number of output bits, (3) the total number of incoming wires \(N\) and that of outgoing wires \(M\), (4) the incoming and outgoing/output wire indices that belong to each gate, (5) the outgoing wire indices corresponding to each party’s input bits. Note that, it is a common assumption among PFE schemes \([\text{KM}11,\text{MS}13,\text{BBKL}17]\) that both parties have pre-agreement on the number of gates \((g)\), the number of input wires \((n)\), the number of output wires \((m)\), the number of input bits of Party_1 \((q)\). Both parties generate \(\text{PubInfo}_{C_f}\) at the beginning of the protocol execution (without an additional round of communication). Namely, each party runs the following deterministic procedure to obtain \(\text{PubInfo}_{C_f}\) on public input \((g, n, m, q)\):

- Set \(N := 2g\), \(M := n + g - m\).
- For \(i = 1, \ldots, g\) set \(iw_{2i-1}\) and \(iw_{2i}\) as the incoming wire of the gate \(G_i\).
- For \(i = 1, \ldots, g - m\) set \(ow_i\) as the outgoing wire of the gate \(G_i\).
- For \(i = 1, \ldots, q\) set \(ow_{g - m + i}\) as the outgoing wire corresponding to Party_1’s \(i\)-th input bit.
- For \(i = 1, \ldots, n - q\) set \(ow_{g - m + q + i}\) as the outgoing wire corresponding to Party_2’s \(i\)-th input bit.
- For \(i = 1, \ldots, m\) set the output wire \(y_i\) as the output of \(G_{g - m + i}\).
- Return \(\text{PubInfo}_{C_f} := (M, N, \text{OW}, \text{IW}, y)\).

Next, Party_1 generates \(\pi_f\) (i.e., the connection between incoming and outgoing wire indices) using the following randomized procedure on input \((C_f, \text{OW}, \text{IW})\):

- Randomly permute the indices \(1, \ldots, g - m\), and assign it to an ordered set \(A\).
- For \(i = 1, \ldots, g - m\) assign \(G_A[i]\) to the \(i\)-th non-output gate in topological order.
- For \(i = 1, \ldots, m\) assign \(G_{g - m + i}\) to \(i\)-th output gate.
- Extract \(\pi_f\) from \(C_f\) according to the connections between \(ow_s\) and \(iw_s\).
- Return \(\pi_f\).

During the protocol execution, Party_1 and Party_2 first engage in a mapping evaluation protocol where Party_2 obliviously obtains the tokens on gate inputs, and then they mutually run a 2PC protocol (i.e., conventional Yao’s scheme) where Party_2 garbles each gate separately using those tokens, and Party_1 evaluates the garbled circuit.

### 3 Our Primary Proposal

In this section, as a warm up, we first present our baseline scheme without any optimization. We then optimize it by offline/online decomposition, and obtain a more efficient mechanism in terms of online computation in Figure 1.
3.1 The Description of the Baseline Scheme

Prior to the protocol execution, both parties should have a pre-agreement on a cyclic group \( G \) of large prime order \( q \in O(\lambda) \) with a generator \( P \). In accordance with the generic 2PFE framework (see Section 2.2), our protocol follows the following procedures. In the preparation stage, \text{Party}_1 \) compiles the function into a boolean circuit \( C_f \) consisting of only one type of gates (e.g., NAND gates). Each party runs the deterministic procedure described in Section 2.2 to generate PubInfo\(_{C_f}\) on input \((g,\pi_1,m,q)\). \text{Party}_1 \) then extracts \( \pi_f \). \text{Party}_2 \) picks \( M \) other generators for the generator set \( \mathcal{P} := (P_1, \ldots, P_M) \), and then sends \( \mathcal{P} \) to \text{Party}_1. \text{Party}_1 \) assigns the random set \( T := (t_1, \ldots, t_j) \) such that \( t_j \in \mathbb{Z}_q^* \) for \( j = 1, \ldots, N \). Next, \text{Party}_1 \) computes \( Q_j := t_j \cdot P_{\pi_j^{-1}(j)} \) for \( j = 1, \ldots, N \) where \( \pi_j^{-1}(j) \) denotes the index of the outgoing wire connected to \( iw_j \), and sends the set \( \mathcal{Q} := (Q_1, \ldots, Q_N) \) to \text{Party}_2. Now, both parties have the knowledge of the set \( (\mathcal{P}, \mathcal{Q}) \), which we define as Reusable Mapping Template\(^{10} \) (ReuseTemp\(_f\)).

\textbf{Definition 1 (Reusable Mapping Template).} Let \( \pi_f \) and \( \pi_f^{-1} \) be defined as above. A Reusable Mapping Template is a set ReuseTemp\(_f\) := \((\mathcal{P}, \mathcal{Q})\) such that \( \mathcal{P} := (P_1, \ldots, P_M) \) where \( P_i \) is a generator of the group picked for \( ow_i \) by \text{Party}_2 \) and \( \mathcal{Q} := (Q_1, \ldots, Q_N) \) where \( Q_j := t_j \cdot P_{\pi_j^{-1}(j)} \) is a group element generated for \( iw_j \) by \text{Party}_1 \) for \( t_j \in \mathbb{Z}_q^* \), \( i = 1, \ldots, M \), and \( j = 1, \ldots, N \).

\text{Party}_2 \) then picks \( \alpha_0, \alpha_1 \in \mathbb{Z}_q^* \), keeps them private, and computes the following four ordered sets

\[
\begin{align*}
\mathcal{W}^0 &:= (W^0_0, \ldots, W^0_M : W^0_i \leftarrow \alpha_0 \cdot P_i, \ i = 1, \ldots, M), \\
\mathcal{W}^1 &:= (W^1_0, \ldots, W^1_M : W^1_i \leftarrow \alpha_1 \cdot P_i, \ i = 1, \ldots, M), \\
\mathcal{V}^0 &:= (V^0_0, \ldots, V^0_N : V^0_j \leftarrow \alpha_0 \cdot Q_j, \ j = 1, \ldots, N), \\
\mathcal{V}^1 &:= (V^1_0, \ldots, V^1_N : V^1_j \leftarrow \alpha_1 \cdot Q_j, \ j = 1, \ldots, N).
\end{align*}
\]

\text{Party}_2 \) next generates the following two randomly chosen ordered sets for output wires of the circuit

\[
\begin{align*}
Y^0 &:= (y^0_0, \ldots, y^0_{o} : y^0_i \leftarrow_R \{0,1\}^\ell, \ i = 1, \ldots, o), \\
Y^1 &:= (y^1_0, \ldots, y^1_{o} : y^1_i \leftarrow_R \{0,1\}^\ell, \ i = 1, \ldots, o)
\end{align*}
\]

where \( \ell \) is the bit length of a group element (i.e., \( \ell = \lceil \log_2(q) \rceil \)).

Both parties then engage in a 2PC protocol where \text{Party}_2 \) and \text{Party}_1 \) play the garbler and evaluator roles, respectively. \text{Party}_2 \) garbles the whole circuit by using \( \mathcal{W}^0, \mathcal{W}^1, \mathcal{V}^0, \mathcal{V}^1, Y^0, Y^1 \), and PubInfo\(_{C_f}\). Note that in contrast to

\(^{10}\) The efficiency of our scheme mostly results from the reusability of ReuseTemp\(_f\), during token generations in re-executions for the same private function. Although [KM11] also involves homomorphic encryption for token generation, it requires all protocol steps to be repeated in each re-execution.
the usual garbling in [KM11, MS13], in our garbling phase, Party₂ has group elements instead of random tokens. To use group elements as keys, we now define an instantiation of a dual-key cipher (DKC) notion of [BHR12] using a pseudorandom function as

\[
\text{Enc}_{P_1, P_2}(m) := [H(P_1, P_2, \text{gateID})]_\ell \oplus m
\]

where \( P_1 \) and \( P_2 \) are two group elements used as keys, \( m \) is the \( \ell \)-bit plaintext, \( \text{gateID} \) is the index number of the gate, \( H : G \times G \times \{0, 1\}^* \rightarrow \{0, 1\}^{\ell + \tau} \) is a hash-function (which we model as a random oracle), \( \tau \) is an integer such that \( \tau > 2 \log_2(4g) \) for preventing collisions in the \( \tau \) rightmost bits of hashes, and \([H(X)]_\ell \) denotes the truncated hash value (of the message \( X \)) which is cropped to the \( \ell \) leftmost bits of \( H(X) \) for some \( X \). Also, we denote \([H(X)]_\tau \) for the truncated hash value (of the message \( X \)) which is cropped to the \( \tau \) rightmost bits of \( H(X) \) for some \( X \). The former truncated hash value is used for encryption, while the latter is utilized for the point and permute optimization of Beaver et al. [BMR90]. Note that the encryption scheme \( \text{Enc} \) is based on the encryption scheme in [LPS08] and differs from it only by utilization of group elements as keys.

Let \( G_a \) be a non-output NAND gate for some \( a \in \{1, \ldots, g\} \). Let also \( i_w, i_w \) be the incoming wires and \( o_w \) be the outgoing wire of \( G_a \) where \( i, j \in \{1, \ldots, M\} \) and \( z \in \{1, \ldots, N\} \). To garble \( G_a \), \( \text{Party}_2 \) prepares the following four ciphertexts

\[
ct_i^1 := \text{Enc}_{V_{i,0}, V_{j,0}}(\overline{W}_z^0), \quad ct_i^2 := \text{Enc}_{V_{i,0}, V_{j,0}}(\overline{W}_z^1),
\]

\[
ct_i^3 := \text{Enc}_{V_{i,1}, V_{j,0}}(\overline{W}_z^0), \quad ct_i^4 := \text{Enc}_{V_{i,1}, V_{j,0}}(\overline{W}_z^1)
\]

where \( \overline{W}_z^0 \) and \( \overline{W}_z^1 \) are the \( \ell \)-bit string representations of the group elements. Similarly, let \( G_b \) be an output NAND gate for some \( b \in \{1, \ldots, g\} \). Let also \( i_w, i_w \) be the incoming wires and \( z \) be the output wire index of \( G_b \) where \( i, j \in \{1, \ldots, M\} \) and \( z \in \{1, \ldots, o\} \). To garble \( G_b \), \( \text{Party}_2 \) prepares the following four ciphertexts

\[
ct_b^1 := \text{Enc}_{V_{i,0}, V_{j,0}}(y_z^1), \quad ct_b^2 := \text{Enc}_{V_{i,0}, V_{j,0}}(y_z^2),
\]

\[
ct_b^3 := \text{Enc}_{V_{i,1}, V_{j,0}}(y_z^1), \quad ct_b^4 := \text{Enc}_{V_{i,1}, V_{j,0}}(y_z^2).
\]

For the point and permute optimization [BMR90] for each gate \( G_a \) in the circuit, \( \text{Party}_2 \) picks random indices \( I_a^1, I_a^2 \in \{1, \ldots, \tau\} \) such that

\[
\{(X[I_a^1], X[I_a^2]), (Y[I_a^1], Y[I_a^2]), (Z[I_a^1], Z[I_a^2]), (T[I_a^1], T[I_a^2])\} = \{(0, 0), (0, 1), (1, 0), (1, 1)\}
\]

where \( X = [H(V_{i,0}, V_{j,0}, \text{gateID})]_\tau \), \( Y = [H(V_{i,0}, V_{j,1}, \text{gateID})]_\tau \), \( Z = [H(V_{i,1}, V_{j,0}, \text{gateID})]_\tau \), \( T = [H(V_{i,1}, V_{j,1}, \text{gateID})]_\tau \), and \( S[I_a] \) denotes the \( I_a \)-th bit of the bit string \( S \). We denote each garbled gate \( G_{Ga} \), which is then composed of four \( \ell \)-bit ciphertexts, \( ct_a^1, ct_a^2, ct_a^3, \) and \( ct_a^4 \), and an index pair \( (I_a^1, I_a^2) \). Note that the set
In this section, we optimize our baseline scheme presented in Section 3 by utilizing the Reusable Mapping Template ReuseTemp\_j when the same private function of ciphertexts in the \(GG_a\) are ordered according to \(I_{a}^{1}\)-th and \(I_{a}^{2}\)-th bits of their corresponding \(X\), \(Y\), \(Z\), and \(T\) values. For example, let \(X = 011001 \ldots 1\), \(Y = 101111 \ldots 0\), \(Z = 110001 \ldots 0\), and \(T = 010111 \ldots 1\). If \((I_{a}^{1}, I_{a}^{2}) = (1, 5)\) then \((X[1], X[5]) = (0, 0), (Y[1], Y[5]) = (1, 1), (Z[1], Z[5]) = (1, 0), (T[1], T[5]) = (0, 1)\), and therefore, we have \(GG_a = (ct_{a}^{1}, ct_{a}^{2}, ct_{a}^{3}, ct_{a}^{4}, ct_{a}^{5}, ct_{a}^{6}, (I_{a}^{1}, I_{a}^{2}))\). A trivial method for finding such a pair \((I_{a}^{1}, I_{a}^{2})\) could be as follows. First, \(\text{Party}_2\) can find \(I_{a}^{1}\) such that \([[X[I_{a}^{1}], Y[I_{a}^{1}], Z[I_{a}^{1}], T[I_{a}^{1}]]] = \{0, 0, 1, 1\}\) with probability of 6/16 in each trial. Then, \(I_{a}^{2}\) could also be found with probability of 4/16 in each trial. Therefore, the expected number of trials to find a pair of \((I_{a}^{1}, I_{a}^{2})\) is 7.

\(\text{Party}_2\) garbles all the gates in the circuit in the above-mentioned way, and obtains the garbled circuit \(F\). \(\text{Party}_2\) then sends \(F\) and its garbled input \(X_2\) (i.e., the \(W_i\) group elements for outgoing wires corresponding to \(x_2\)) to \(\text{Party}_1\). As usual, \(\text{Party}_1\) gets its own garbled input \(X_1\) (i.e., the \(W_i\) group elements for outgoing wires corresponding to \(x_1\)) from \(\text{Party}_2\) using oblivious transfers (OT) (or one invocation of the OT extension schemes [IKNP03, KK13, ALSZ13]). Note that this does not increase the round complexity of our overall protocol, since the exchanges needed for OTs can be jointly run with the earlier exchanged messages.

Using \(F\), the garbled input \(X = (X_1, X_2), T\), and \(\pi_f\), \(\text{Party}_1\) evaluates the whole garbled circuit in topological order. If an outgoing wire \(ow_d\) is mapped to an incoming wire \(iw_e\), then the group element \(V_e\) of the \(e\)-th incoming wire is computed by the multiplication of the group element \(W_d\) of the \(d\)-th outgoing wire and the blinding value \(t_e\) (i.e., if \(\pi_f(d) = e\), then \(V_e = t_e \cdot W_d\)). Each garbled gate \(GG_a\) can be evaluated when both group elements \((V_i, V_j)\) on its incoming wires \((iw_i, iw_j)\) are computed. To evaluate each \(GG_a\), \(\text{Party}_1\) first computes \(H(V_i, V_j, \text{gatel})\), and then XORs the ciphertext in the \(GG_a\) pointed by \(I_{a}^{1}\)-th and \(I_{a}^{2}\)-th bits of the \(H(V_i, V_j, \text{gatel})\). At the end, \(\text{Party}_1\) obtains the token set \(Y = (y_1, \ldots, y_o)\) for the output bits \(f(x_1, x_2)\).

### 3.2 Single Execution Protocol

We now introduce an optimized implementation of our baseline scheme by carrying out some of the computations in the offline stage. In general, such precomputation techniques enhance real-time performance at the cost of extra preliminary computations and storage consumption. Besides, in today’s technological perspectives, memory consumption is rarely considered to be a serious drawback since storage units are abundant in many recent devices. We provide our optimized single execution protocol with a precomputation phase in Figure 4. The computations that can be carried out in the precomputation phase include the generation of \(P\), and the computation of the sets \(W^0\) and \(W^1\) by \(\text{Party}_2\).

### 4 Optimized Proposal with Re-execution Feature

In this section, we optimize our baseline scheme presented in Section 3 by utilizing the Reusable Mapping Template ReuseTemp\_j when the same private function
First Protocol: Our Single Execution 2PFE Scheme

Party’s Input: $x_1 \in \{0, 1\}^*$, a boolean circuit $C_f$ consisting of NAND gates (compiled from the function $f$) and a mapping $\pi_f$ (extracted from $C_f$).

Party’s Input: $x_2 \in \{0, 1\}^*$.

Pre-shared Information: A group $G$ of prime order $q$ with a generator $P$ and PubInfo$_{C_f}$.

Output: $f(x_1, x_2)$.

Precomputation phase

1. Party$_1$ runs generates the set $P$ of $M$ random generators. It also picks $\alpha_0, \alpha_1 \in \mathbb{Z}_q^*$, and prepares the group element sets $W_0 := (W_0^1, \ldots, W_0^M) : W_0^i \leftarrow \alpha_0 \cdot P_1, \; i = 1, \ldots, M$, for FALSEs and $W_1 := (W_1^1, \ldots, W_1^M) : W_1^i \leftarrow \alpha_1 \cdot P_1, \; i = 1, \ldots, M$, for TRUEs, where $P_1$ is the $i$-th element in $P$ and each $W_i^a$ is a token for $\omega_i \in \text{OW}$, $b \in \{0, 1\}$. Party$_2$ stores $P$, $W_0$, $W_1$, $\alpha_0$, and $\alpha_1$.

Online phase

Round 1:
2. Party$_2$ sends $P$ to Party$_1$.

Round 2:
3. Party$_2$ generates the blinding set $T := (t_1, \ldots, t_J) \in \mathbb{Z}_q^*$, computes the set $Q = \{Q_1, \ldots, Q_N \mid Q_i = t_j \cdot P_{\pi_f^{-1}(j)} \mid j = 1, \ldots, N\}$. If there is a possibility of re-execution of the protocol for the same function, then Party$_2$ stores ReuseTemp$_f := (P_1, \ldots, P_M, Q_1, \ldots, Q_N)$ (see Figure 2 for our re-execution protocol). Party$_1$ sends $Q$ to Party$_2$.

Round 3:
4. Party$_2$ prepares the group element sets $V_0 := (V_0^1, \ldots, V_0^M) : V_0^j \leftarrow \alpha_0 \cdot Q_j, \; j = 1, \ldots, N$ for FALSEs and $V_1 := (V_1^1, \ldots, V_1^M) : V_1^j \leftarrow \alpha_1 \cdot Q_j, \; j = 1, \ldots, N$ for TRUEs for $i_w \in \text{IW}$. If there is a possibility of re-execution of the protocol for the same function, then Party$_2$ stores ReuseTemp$_f := (P_1, \ldots, P_M, Q_1, \ldots, Q_N)$ (see Figure 2 for the re-execution protocol). Next, Party$_2$ generates two random token sets for output wires of the circuit $Y_0 := \{y_0^1, \ldots, y_0^a \mid y_0^i \leftarrow R \{0, 1\}^*, \; i = 1, \ldots, a\}$ and $Y_1 := \{y_1^1, \ldots, y_1^* \mid y_1^i \leftarrow R \{0, 1\}^*, \; i = 1, \ldots, a\}$.

5. The 2PC protocol now starts from this stage where Party$_2$ becomes the garbler and Party$_1$ becomes the evaluator. Using $W_0, W_1, Y_0, Y_1, \text{andPubInfo}_f$, Party$_2$ prepares the garbled circuit $F$ by garbling each gate as follows. Party$_2$ prepares the following four ciphertexts to garble a non-output NAND gate $G_a$ whose incoming wires $i_w$ and $i_w$, and outgoing wire is $\omega_2$: $\text{Enc}_{y_0^0} \cdot V_0^0 (\text{W}_1^0)$, $\text{Enc}_{y_0^0} \cdot V_0^1 (\text{W}_1^1)$, $\text{Enc}_{y_1^0} \cdot V_1^0 (\text{W}_1^0)$. Similarly, Party$_2$ prepares the following four ciphertexts to garble an output NAND gate $G_b$ whose incoming wires $i_w$ and $i_w$, and output wire index is $z$: $\text{Enc}_{y_0^0} \cdot V_0^0 (y_0^z)$, $\text{Enc}_{y_0^0} \cdot V_1^1 (y_1^z)$, $\text{Enc}_{y_1^0} \cdot V_1^0 (y_0^z)$, $\text{Enc}_{y_1^0} \cdot V_1^1 (y_1^z)$. Each garbled gate $GG_a$ is then composed of four $\ell$-bit ciphertexts and two $\log_2(\tau)$-bit indices, $I^a_\ell$ and $I^a_\tau$ (see Section 3.1 for garbling details). Party$_1$ sends $F$ and the garbled input $X$ for its own input $x_2$ to Party$_1$ Party$_1$ also obtains the garbled input $X_1$ for its own input $x_1$ from Party$_2$ using parallel 1-out-of-2 OTs (or a more efficient OT extension scheme).\(^a\)

6. Using $F$, the garbled input $X = (X_1, X_2)$, $T$, and $\pi_f$, Party$_1$ evaluates the whole garbled circuit in topological order. If an outgoing wire $\omega_d$ is mapped to an incoming wire $i_w$, then the group element $V_d$ of the $d$-th incoming wire is computed by the multiplication of the group element $W_d$ of the $d$-th outgoing wire with the blinding value $t_e$ (i.e., if $\pi_f(d) = e$, then $V_e = t_e \cdot W_d$). Each garbled gate $GG_a$ can be evaluated whenever both group elements $(V_1, V_e)$ on its incoming wires ($i_w$, $i_w$) are computed. To evaluate each $GG_a$, Party$_1$ first computes $H(V_1, V_e, \text{gatetol})$, and then XORs the ciphertext in the $GG_a$ pointed by $I^a_\ell$-th and $I^a_\tau$-th bits of $[H(V_1, V_e, \text{gatetol})]$. At the end, Party$_1$ obtains the token set $Y = (y_1, \ldots, y_a)$ for the output bits $f(x_1, x_2)$.

\(^a\) Note that the OT protocol rounds can be combined with the former protocol rounds for minimization of the overall rounds.

Fig. 1. Our Optimized Single Execution 2PFE Protocol via decomposition of offline/online computations
is evaluated more than once. This optimization result in a more efficient protocol of our protocol shown in Figure 2. We also propose an efficient method for executions with multiple Party$_2$.

### 4.1 Re-execution Protocol

One of the novelties of our scheme over the state-of-the-art is that our scheme results in a significant cost reduction when the same private function is evaluated more than once between the same or varying evaluating parties. This feature is quite beneficial in relevant real-life scenarios where individuals (or enterprises) can mutually and continuously have long-term business relationship instead of a single deal. Note that such a cost reduction is not available in the protocols of [KM11] and [MS13], since they require all token generation and 2PC procedures repeated in all executions. However, our scheme involves ReuseTemp$_f$ that is reusable for the generation of tokens on incoming and outgoing wires. The reusability of ReuseTemp$_f$ incurs a massive reduction in protocol overhead since a large part of costs in existing 2PFE protocols [KM11, MS13] results from the generation of these tokens.

With the re-execution optimization, for the $k$-th evaluation, Party$_2$ again picks $\alpha_{0,k}, \alpha_{1,k} \in R Z_q^*$ values during the precomputation phase. Party$_2$ then prepares the sets $W_0^k$, $W_1^k$, $V_0^k$, and $V_1^k$ as in the single execution protocol. The online phase then includes only the 2PC stage that also runs the same way as in Section 3. During the evaluation procedure of the 2PC stage, Party$_1$ always use the same $T$ in all protocol runs. We provide our optimized re-execution protocol in Figure 2.

### 4.2 Executing with Various Party$_2$s

In the previous section, we addressed the case where the same two parties would like to evaluate the same function multiple times. In this section, we deal with the case that Party$_1$ would like to run the 2PFE protocol for the same private function with various Party$_2$s separately. This is a relevant scenario where Party$_1$ may run a business with many customers for her algorithm/software. Suppose that a cryptological research institution invents a practical algorithm for breaking RSA. Since such an algorithm would clearly attract a substantial demand, the institution may prefer preserving the details of the algorithm selling only its use. On the other hand, in many cases the clients would not like to share the keys (i.e., private inputs) with the institution. This is one of the several scenarios that a 2PFE protocol for the same private function with various Party$_2$s is suitable for.

First of all, we recall that the execution of our second protocol in Figure 2 requires the preknowledge of ReuseTemp$_f := (P, Q)$ by Party$_2$ and the set $T$ by Party$_1$. Trivially, once ReuseTemp$_f$ and $T$ is produced in the first execution with any Party$_2$ as in our first protocol in Figure 1, then they can be stored, and our second protocol can be made use of in the subsequent executions with the same Party$_2$. We are here interested in a more efficient mechanism running
Second Protocol: Our $k$-th Re-execution 2PFE Scheme

Party’s Input: $x_{1,k} \in \{0,1\}^*$, a boolean circuit $C_f$ consisting of NAND gates (compiled from the function $f$), a mapping $\pi_f$ (extracted from $C_f$), and the blinding set $T := \{t_1, \ldots, t_j : t_j \in \mathbb{Z}_q^\ast, j = 1, \ldots, N\}$.

Party’s Input: $x_{2,k} \in \{0,1\}^*$.

Pre-shared Information: A cyclic group $G$ of prime order $q$ with a generator $P$, $\text{PubInfo}_{C_f}$, and $\text{ReuseTemp}$.

Output: $f(x_{1,k}, x_{2,k})$.

Precomputation phase of the $k$-th execution:

1. Party$_2$ picks $a_{0,k}, \alpha_{1,k} \in \mathbb{Z}_q^\ast$ and prepares the group element sets $W_0^0 := (W_{1,k}^0, \ldots, W_{M,k}^0)$ for FALSEs and $W_1^0 := (W_{1,k}^1, \ldots, W_{M,k}^1)$ for TRUEs while $\text{Precomp}$. $W_k$ and $\text{Precomp}$ is already computed in the first execution of the function $f$ as in Figure 1.

2. The 2PC protocol now starts from this stage where Party$_2$ becomes the garbler and Party$_1$ becomes the evaluator. Using $W_0^0$, $W_1^0$, $V_0^1$, $V_1^1$, $Y_{0}^1$, $Y_{1}^1$, and $\text{PubInfo}_{C_f}$. Party$_2$ prepares the garbled circuit $F_k$ by garbling each gate as follows. Party$_2$ prepares the following four ciphertexts to garble a non-output NAND gate $G_a$ whose incoming wires are $i_{w_i}$ and $i_{w_j}$, and outgoing wire is $o_w$: $\text{Enc}_{\alpha_{1,k}, V_0^1} (W_{1,k}^j, W_{1,k}^j, W_{1,k}^j)$, $\text{Enc}_{\alpha_{1,k}, V_1^1} (W_{1,k}^j, W_{1,k}^j, W_{1,k}^j)$, $\text{Enc}_{\alpha_{1,k}, V_0^1} (W_{1,k}^j, W_{1,k}^j, W_{1,k}^j)$, $\text{Enc}_{\alpha_{1,k}, V_1^1} (W_{1,k}^j, W_{1,k}^j, W_{1,k}^j)$.

3. Party$_2$ also prepares the following four ciphertexts to garble an output NAND gate $G_b$ whose incoming wires are $i_{w_i}$ and $i_{w_j}$ and output wire index is $z$: $\text{Enc}_{\alpha_{1,k}, V_0^1} (y_{0,k}^1), \text{Enc}_{\alpha_{1,k}, V_0^1} (y_{0,k}^1), \text{Enc}_{\alpha_{1,k}, V_1^1} (y_{1,k}^1), \text{Enc}_{\alpha_{1,k}, V_1^1} (y_{1,k}^1), \text{Enc}_{\alpha_{1,k}, V_1^1} (y_{1,k}^1), \text{Enc}_{\alpha_{1,k}, V_1^1} (y_{1,k}^1)$. Each garbled gate $GG_{a,b}$ is then composed of four $\ell$-bit ciphertexts and two $\log_2((\tau)-$bit indices, $I_{a,b}$ and $I_{b,b}$ (see Section 3.2 for garbling details). Party$_2$ stores $F_k$, $W_0^0$, $W_1^0$, $Y_{0}^1$, and $Y_{1}^1$.

Online phase of the $k$-th execution

Round 1:

3. Party$_2$ sends $F_k$ and the garbled input $X_{2,k}$ for its own input $x_{2,k}$ to Party$_1$.

4. Party$_1$ gets the garbled input $X_{1,k}$ for its own input $x_{1,k}$ from Party$_2$ using parallel 1-out-of-2 OTs (or a more efficient OT extension scheme).

5. Using $F_k$, the garbled input $X_k = (X_{1,k}, X_{2,k}), T$, and $\pi_f$, Party$_1$ evaluates the whole garbled circuit in topological order. If an outgoing wire $o_w$ is mapped to an incoming wire $i_{w_s}$, then the group element $V_s$ of the $s$-th outgoing wire is computed by the multiplication of the group element $W_q$ of the $d$-th outgoing wire and the blinding value $t_d$ (i.e., if $\pi_f(d) = e$, then $V_s := t_d \cdot W_q$). Each garbled gate $GG_{a,b}$ can be evaluated whenever both group elements $(V_{a,b}, V_{b,b})$ are computed. To evaluate each $GG_{a,b}$, Party$_1$ first computes $H(V_{a,b}, V_{b,b}, \text{gateID})$, and then XORs the ciphertext in the $GG_{a,b}$ pointed by $I_{a,b}$ and $I_{b,b}$-th bits of $[H(V_{a,b}, V_{b,b}, \text{gateID})]$. At the end, Party$_1$ obtains the token set $Y_k = (y_{1,k}^0, \ldots, y_{o,k}^0)$ for the output bits $f(x_{1,k}, x_{2,k})$.

Fig. 2. Our Optimized Re-Execution 2PFE Protocol via Reusable Mapping Template

\textit{ReuseTemp}_f is already computed in the first execution of the function $f$ as in Figure 1.
with various Party$_2$s by eliminating the costs of our first protocol for generating the preknowledge. The goal of this mechanism is to generate the generator set $\mathcal{P}$ in such a way that Party$_1$ does not know the relation between any two of its elements. $T$ and $Q$ can be subsequently computed, once the generator set $\mathcal{P}$ is given to Party$_1$. In order to do so, we utilize a distributed system based on a $t$-out-of-$n$ threshold mechanism (fault tolerant against arbitrary behaviour of up to $t$ malicious and colluding authorities) which takes $(G, q, P, M)$ as input and outputs $\mathcal{P}$.

In the offline stage of our new mechanism, the generator set $\mathcal{P}$ is generated by the distributed authorities, and given to Party$_1$. Next, Party$_1$ computes the sets $T$ and ReuseTemp$_f$. It then publishes PubInfo$_C$, and ReuseTemp$_f$ so that any prospective $k$-th party Party$_{2,k}$ can utilize them in a 2PFE protocol run. This offline stage is dealt with only once, and its outputs (i.e., $T$ and ReuseTemp$_f$) are used in later re-executions. Note that the flow of re-executions for all Party$_{2,k}$s is exactly the same as our second protocol in Figure 2. We would like to stress that the costs of any execution in our new mechanism with a distributed system does not differ from the second protocol.

5 Complexity Analysis

In this section, we first present the costs of our first and second protocols (given in Figure 1 and Figure 2 respectively) in terms of communication, online computation, and round complexities. We then compare these protocols with the existing boolean circuit based 2PFE schemes. For $\text{KM11}$ and homomorphic encryption based protocol of $\text{MS13}$, it is assumed that the elliptic curve ElGamal is used for the singly homomorphic encryption scheme (as suggested in their paper). Also, for $\text{KM11}$ and our protocols, we assume that each element of $G$ has a length $\ell = 2^\lambda$ bits for $\lambda$-bit security.

5.1 Complexity of Our Protocols

Communication cost. In our first protocol, the overall communication overhead is $(2M + 6N)\lambda$ bits, composed of (i) the set $\mathcal{P}$ ($M$ of $2\lambda$-bit strings) is sent by Party$_2$ in Round 1, (ii) the set $Q$ ($N$ of $2\lambda$-bit strings) is sent by Party$_1$ in Round 2, (iii) the garbled circuit ($2N$ of $2\lambda$-bit strings) is sent by Party$_2$ in Round 3, where $M$ is the number of outgoing wires and $N$ is the number of incoming wires ($N = 2g$). Considering our second protocol, the use of ReuseTemp$_f$ eliminates the transmission of $(2M + 2N)\lambda$ bits (required for token generation). Therefore, in total only $4N\lambda$ bits (required for the garbled circuit) are transmitted.

One can also suggest a single semi-trusted authority for generation of the generator set $\mathcal{P}$. However, the knowledge of the relations among the elements of $\mathcal{P}$ by a single party may violate the privacy of inputs, and therefore, it is better to distribute the trust among multiple authorities.
Computation cost. In terms of online computation complexity, our first protocol requires $4N$ elliptic curve point multiplications, composed of (i) $N$ operations by Party$_1$ in Round 2, (ii) $2N$ operations by Party$_2$ in Round 3, (iii) $N$ operations by Party$_1$ during the evaluation of the garbled circuit. There is also a relatively small cost of $2.5N$ symmetric-key operations during the 2PC stage (composed of $2N$ operations by Party$_2$ for garbling and $0.5N$ operations by Party$_1$ for evaluating). Our second protocol reduces the online computation costs to $N$ elliptic curve point multiplications and $0.5N$ symmetric-key operations (carried out only by Party$_1$). Note that Beaver’s OT pre-computation technique [Bea95] can be used for decomposing OT’s for Party$_1$’s input bits into online/offline stages. This eliminates online public-key operations of OT by carrying out them offline.

Number of rounds. Our first protocol has 3 rounds. The number of rounds in our second protocol is equal to 1, or 2, or 3 depending on the input string length of Party$_1$. Namely, if Party$_1$ has $x_1 = \perp$, then the number of rounds is equal to 1. If Party$_1$’s input bits are not many, it is more efficient to use separate OTs for Party$_1$’s input tokens in parallel instead of an OT extension scheme. There exists OT schemes with 2 rounds (e.g., [Bea95] and [NP01]). Hence, this choice results in a PFE scheme with overall 2 rounds. If Party$_1$’s input bits are many, then using an OT extension scheme is more efficient. Note that Ishai based OT extension schemes are composed of $O(\lambda)$ parallel OTs (again can be realized by Naor and Pinkas’s OT [NP01] and an additional round. Similarly, this choice results in a PFE scheme with overall 3 rounds.

5.2 Comparison

We now compare our 2PFE protocols with the state-of-the-art constant-round switching network (OSN) and homomorphic encryption (HE) based [MS13], singly homomorphic encryption based [KM11], switching network based [BBKL17], and universal circuit based [GKS17]. In our scheme, we utilize an EC cyclic group where the DDH assumption holds for state-of-the-art efficiency. For [KM11], we take into account both protocols: (1) their “C-PFE protocol” (see [KM11, Sect. 3.1]) and (2) their “A More Efficient Variant” (see [KM11, Sect. 3.2]). For a fair comparison, we assume that the point and permute optimization [BMR90] is directly applied to the both [MS13] and [KM11] during the 2PC protocol.

Note that each re-execution of a private function in any of [MS13] and [KM11] has the same cost as its first execution. Also for a fair comparison, we assume that EC-ElGamal is used as HE scheme in the related protocols.

Considering communication complexity, [MS13]-OSN costs $(10N \log_2 N + 4N + 5)\lambda$ bits, composed of $(10N \log_2 N + 2N + 5)\lambda$ in OSN phase and $2N\lambda$ in the 2PC phase. The communication cost of the [KM11] Sect. 3.2] is $(4M + 6N)\lambda$ bits, including $M$ of $4\lambda$-bit ciphertexts sent by Party$_2$ in Round 1, $N$ of $4\lambda$-bit ciphertexts sent by Party$_1$ in Round 2, and $2N$ of $\lambda$-bit ciphertexts sent by Party$_2$ in Round 3.

---

12 In [MS13] and [KM11], for the 2PC phases, the authors do not suggest any optimization. However, a point and permute optimization is available for both schemes.
Among all the state-of-the-art PFE protocols, our second protocol performs the best result in terms of round complexity. Namely, the number of rounds in our second protocol is equal to 1 if \( \text{Party}_1 \) has \( x_1 = \perp \), or 2 if \( \text{Party}_1 \) has a non-empty input \( x_1 \) in such that the OT extension is not applicable for its garbled input, or 3 otherwise. Note that the arithmetic circuit-based protocol of [MS13] provides a linear round complexity.

In terms of online computation overhead, [MS13]-OSN requires \( 6N \log_2 N + 2.5N + 3 \) symmetric-key operations, consisting of \( 6N \log_2 N + 3 \) operations that takes place in the OSN phase (due to the OT extension scheme on behalf of \( 2N \log_2 N + 1 \) OTs) and \( 2.5N \) operations for garbling and evaluating the garbled circuit. The total online computation cost of [KMI11] Sect. 3.2 is \( N \) online public key decryptions (roughly the same number of elliptic curve point multiplications) by \( \text{Party}_2 \) and relatively small additional cost of \( 2.5N \) symmetric-key operations for garbling and evaluating the garbled circuit. In order to compare the complexity of symmetric-key and asymmetric-key based operations, we define the computation cost ratio \( \rho \) as the cost of an elliptic curve point multiplication divided by the cost of a symmetric-key operation for the same security level. The value of \( \rho \) depends upon several factors, such as the software implementations, the symmetric-key encryption scheme, the availability of short-cut algorithms, the type of chosen elliptic curve, the hardware infrastructure, and the type of utilized processors. For example, according to [EIV18], in a setting where curve25519, and SHA256 are picked as the EC and the hash function, respectively, and the operations take place on an Intel Xeon Processor E3-1220 v6 (amd64, 4x3GHz), the value of \( \rho \) is roughly 130. Therefore, for circuits with more than \( 2^{20} \) gates, our scheme beats [MS13] also in terms of computation complexity.

For all given circuit sizes, the communication costs of our second protocol are significantly lower than that of existing 2PFE protocols. To illustrate, for a circuit with \( 2^{30} \) gates, compared to [MS13], our first and second protocols result in 97.8%, and 98.7% communication cost reduction, respectively. Moreover, compared to [KMI11], for any given circuit size, our first and second protocols achieve about 12%, and 50% reduction in communication cost, respectively.

### 6 Security of Our Protocols

In this section, we give simulation-based security proofs of our single execution protocol in Figure 1, re-execution protocol in Figure 2, and our mechanism with various \( \text{Party}_2 \)'s in Sect. 4.2 in accordance with the security proof of [KMI11].

**Theorem 1.** If the following three conditions hold then the 2PFE protocol proposed in Figure 1 is secure against semi-honest adversaries: (1) the DDH assumption is hard in the cyclic group \( G \), (2) the hash-function \( H : G \times G \times \{0,1\}^* \rightarrow \{0,1\}^{\ell + \tau} \) involved in the instantiation of DKC scheme is modeled as a random oracle, (3) the OT scheme securely realizes \( \mathcal{F}_{\text{OT}} \) functionality in the OT-hybrid model against semi-honest adversaries.
Proof. First, consider the case that $\text{Party}_1$ is corrupted. For any probabilistic polynomial time adversary $A_1$, controlling $\text{Party}_1$ in the real world, we construct a simulator $S_1$ that simulates $A_1$’s view in the ideal world. $S_1$ runs $A_1$ on $\text{Party}_1$’s inputs, $f$ and $x_1$, the function output token set $Y = (y_1, \ldots, y_0)$, the pre-shared group parameters, and $\text{PubInfo}_C$, as follows.

1. $S_1$ generates the generator set $\hat{P} := (\hat{P}_1, \ldots, \hat{P}_M)$. $S_1$ also prepares the group element sets $W^0 := (\hat{W}_M^0, \ldots, \hat{W}_i^0 : \hat{W}_i^0 \leftarrow \alpha_{0,i} \cdot P, \alpha_{0,i} \in \mathbb{Z}_q^*), i = 1, \ldots, M$ and $W^1 := (\hat{W}_M^1, \ldots, \hat{W}_i^1 : \hat{W}_i^1 \leftarrow \alpha_{1,i} \cdot P, \alpha_{1,i} \in \mathbb{Z}_q^*), i = 1, \ldots, M$. $S_1$ gives $\hat{P}$ to $A_1$.

2. $S_1$ receives the blinding set $T := (t_1, \ldots, t_j : t_j \in \mathbb{Z}_q^*, j = 1, \ldots, N)$ from $A_1$, and prepares the sets $\hat{Y}^0 := (\hat{V}_M^0, \ldots, \hat{V}_j^0 : \hat{V}_j^0 \leftarrow t_j \cdot \hat{W}_j^{\pi(j)}, j = 1, \ldots, N)$ and $\hat{Y}^1 := (\hat{V}_M^1, \ldots, \hat{V}_j^1 : \hat{V}_j^1 \leftarrow t_j \cdot \hat{W}_j^{\pi(j)}, j = 1, \ldots, N)$.

3. $S_1$ prepares the garbled circuit $\hat{F}$ by garbling each gate as follows. $S_1$ garbles each non-output NAND gate by encrypting only the group element for FALSE on its outgoing wire with all four possible input token combinations (i.e., for a gate whose incoming wires are $i_w$ and $i_w$, outgoing wire is $\omega_x$, $S_1$ prepares the following four ciphertexts: $\hat{c}_a^1 = \text{Enc}_{\hat{V}_i^0, \hat{V}_j^0}(\hat{W}_z^0), \hat{c}_a^2 = \text{Enc}_{\hat{V}_i^0, \hat{V}_j^1}(\hat{W}_z^0), \hat{c}_a^3 = \text{Enc}_{\hat{V}_i^1, \hat{V}_j^0}(\hat{W}_z^0), \hat{c}_a^4 = \text{Enc}_{\hat{V}_i^1, \hat{V}_j^1}(\hat{W}_z^0)$. To garble an output NAND gate whose incoming wires are $i_w$ and $i_w$, and output wire is $z$, $S_1$ prepares the four ciphertexts: $\hat{c}_b^1 = \text{Enc}_{\hat{V}_i^0, \hat{V}_j^0}(y_z), \hat{c}_b^2 = \text{Enc}_{\hat{V}_i^0, \hat{V}_j^1}(y_z), \hat{c}_b^3 = \text{Enc}_{\hat{V}_i^1, \hat{V}_j^0}(y_z), \hat{c}_b^4 = \text{Enc}_{\hat{V}_i^1, \hat{V}_j^1}(y_z)$. For each garbled gate $GG_a$, $S_1$ then permutes $\hat{c}_a^1$, $\hat{c}_a^2$, $\hat{c}_a^3$, $\hat{c}_a^4$, and places $\hat{I}_a^1, \hat{I}_a^2 \in \mathbb{R} \{1, \ldots, \tau\}, \hat{I}_a^3, \hat{I}_a^4 \in \mathbb{R} \{1, \ldots, \tau\}$, and places $\hat{c}_a^1$ in the order pointed by $\hat{I}_a^1$ and $\hat{I}_a^2$-th bits of $[H(\hat{V}_i^0, \hat{V}_j^0, \text{gateID})]$, among the other three ciphertexts. Each garbled gate $GG_a$ is then composed of four $\ell$-bit ciphertexts and two $\log_2(\tau)$-bit random values $\hat{I}_a^1$ and $\hat{I}_a^2$.

4. $S_1$ gives $\hat{F}$ to $A_1$ along with the simulated garbled input consisting of only the group elements for FALSEs on both parties’ input wires $X = (X_1, X_2)$. This completes our simulation.

In what follows, we prove that the information obtained by $\text{Party}_1$ in the real execution $(P, W, F)$ is identically distributed to $(\hat{P}, \hat{W}, \hat{F})$, where for outgoing wires, $\text{Party}_1$ obtains the group elements $W = (W_1, \ldots, W_M)$ while $A_1$ obtaining the group elements $\hat{W} = (\hat{W}_1, \ldots, \hat{W}_M)$. We now show the computational indistinguishability of $(P, W)$ and $(\hat{P}, \hat{W})$ by utilizing Lemma 4 which ultimately ties the security of our protocol to the DDH assumption. More concretely, we need to show

\[ \{(P_1, \ldots, P_M, W_1, \ldots, W_M) \} \approx_c \{(\hat{P}_1, \ldots, \hat{P}_M, \hat{W}_1, \ldots, \hat{W}_M) \} \]

\[ \{(r_1 \cdot P, \ldots, r_M \cdot P, \alpha_{b_1} \cdot (r_1 \cdot P), \ldots, \alpha_{b_0} \cdot (r_M \cdot P)) \} \approx_c \{(\hat{r}_1 \cdot P, \ldots, \hat{r}_M \cdot P, \hat{\alpha}_{b_1} \cdot P, \ldots, \hat{\alpha}_{b_0} \cdot P) \} \]

where $b_i \in \{0, 1\}$ is the semantic value on $\omega_i$ and $\hat{P}_i = \hat{r}_i \cdot P$. For the sake of a simpler representation, we replace $\alpha_{b_i} \cdot r_i$ with $r_{M+i}$, and $\hat{\alpha}_{b_i}$ with $\hat{r}_{M+i}$ for $i = 18$
while it is only sufficient to show that
\[(r_1, \ldots, r_{2M}) \approx_c (\tilde{r}_1, \ldots, \tilde{r}_{2M}).\]

For this purpose, we generate a new set \(R := (R_1, \ldots, R_{2M})\) by picking \(2M\) random generators. Hence, we now need to show
\[
\{(R_1, \ldots, R_{2M}, r_1, \ldots, r_{2M})\} \approx_c \{(R_1, \ldots, R_{2M}, \tilde{r}_1, \ldots, \tilde{r}_{2M})\}
\]

Thanks to Lemma \(\[\]\) and the underlying DDH assumption, we have both
\[
\{(R_1, \ldots, R_{2M}, \gamma \cdot R_1, \ldots, \gamma \cdot R_{2M})\} \approx_c \{(R_1, \ldots, R_{2M}, r_1, \ldots, r_{2M})\}
\]
and
\[
\{(R_1, \ldots, R_{2M}, \gamma \cdot R_1, \ldots, \gamma \cdot R_{2M})\} \approx_c \{(R_1, \ldots, R_{2M}, \tilde{r}_1, \ldots, \tilde{r}_{2M})\}
\]
where \(\gamma \in \mathbb{Z}_q^*\). Hence, the following sets are computationally indistinguishable
\[
\{(r_1, \ldots, r_{2M})\} \approx_c \{(\tilde{r}_1, \ldots, \tilde{r}_{2M})\}
\]
which effectively concludes the proof for \(\{(P, W)\} \approx_c \{\tilde{(P, \tilde{W})}\}\). Furthermore, since the same values in \(T\) are used among the outgoing wire tokens and incoming wire tokens in both the real and the ideal executions, we have \(\{(P, W, \mathcal{V})\} \approx_c \{(\tilde{P}, \tilde{W}, \tilde{\mathcal{V}})\}\) where for each incoming wire \(V = (V_1, \ldots, V_N)\) is the set of tokens obtained by \(\text{Party}_1\) and \(\tilde{V} = (\tilde{V}_1^0, \ldots, \tilde{V}_N^0)\) is the set of tokens obtained by \(\mathcal{A}_1\). In contrast to \([KM11]\), it is relatively simple to prove the computational indistinguishability of \(F\) and \(\tilde{F}\) in our scheme since we use a hash function modeled as random oracle during garbling. Once the distribution of four hash outputs for each gate (in the real and ideal executions) are proven to be computationally indistinguishable random values, outputs of our instantiation of DKC is also proven to be computationally indistinguishable. This results in the computational indistinguishability of each garbled gate \(GG_a\) and \(GG_c\), and eventually computational indistinguishability of \(F\) and \(\tilde{F}\). For a gate whose incoming wires are \(i\mathcal{W}_i\) and \(i\mathcal{W}_j\), in the real execution, we have four hash outputs involved in the garbling
\[
H(V_i^0, V_j^0, \text{gateD}), H(V_i^0, V_j^1, \text{gateD}), H(V_i^1, V_j^0, \text{gateD}), H(V_i^1, V_j^1, \text{gateD}).
\]
Similarly, for each gate, in the ideal execution, we have the following four hash outputs in the garbling
\[
H(\tilde{V}_i^0, \tilde{V}_j^0, \text{gateD}), H(\tilde{V}_i^0, \tilde{V}_j^1, \text{gateD}), H(\tilde{V}_i^1, \tilde{V}_j^0, \text{gateD}), H(\tilde{V}_i^1, \tilde{V}_j^1, \text{gateD}).
\]
Since in \(\text{Party}_1\)'s view, resulting from the indistinguishability of \(V\) and \(\tilde{V}\), the hash inputs are computationally indistinguishable, and therefore, the hash outputs are computationally indistinguishable random values. This completes the proof for \(\{(P, W, F)\} \approx_c \{(\tilde{P}, \tilde{W}, \tilde{F})\}\).
We now consider the case that Party$_2$ is corrupted. For any probabilistic polynomial-time adversary $A_2$, controlling Party$_2$ during our first protocol in the real world, we construct a simulator $S_2$ that simulates $A_2$’s view in the ideal world. $S_2$ runs $A_2$ on Party$_2$’s input, and the pre-shared group parameters, and PubInfo$_C$, as follows.

1. $S_2$ asks $A_2$ to generate $\hat{P} \leftarrow \text{Init}(G, q, P, M)$ and receives $\hat{P}$.
2. $S_2$ then picks $t_j \in \mathbb{Z}_q^*$ for $j = 1, \ldots, N$, and computes $Q_j \leftarrow t_j \cdot P$ which are now random group elements in $G$. $S_2$ assigns $\hat{Q} = (Q_1, \ldots, Q_N)$, and gives $\hat{Q}$ to $A_2$. This completes our simulation.

In the real execution of our protocol, Party$_2$ receives only the message $Q := (Q_1, \ldots, Q_N : Q_j \leftarrow t_j \cdot P_{\pi_1^{-1}(j)}, j = 1, \ldots, N)$ in Round 2 (apart from the exchanged messages during the OT protocol for Party$_1$’s garbled input). However, the transcripts received by Party$_2$ during the OT do not leak any information to Party$_2$ due to the ideal execution $\mathcal{F}_{OT}$ in the OT-hybrid model. Due to DDH assumption, in Party$_2$’s view, the distributions of $\hat{Q}$ and $Q$ are identical (i.e., $\hat{Q} \approx Q$). This concludes the proof for the single execution protocol.

Theorem 2. If the 2PFE protocol proposed in Figure 4 is secure against semi-honest adversaries (i.e., the three conditions in Theorem 1 are satisfied), then the 2PFE protocol proposed in Figure 3 is also secure against semi-honest adversaries.

Proof (Sketch). The main difference of the re-execution protocol from the first one is the utilization of ReuseTemp$_f$. Therefore, the proof will be complete once we show that the utilization of the sets $W^0_k$, $W^1_k$, $V^0_k$, and $W^1_1$ computed from the same ReuseTemp$_f$ in the $k$-th execution gives Party$_1$ no advantage in deducing Party$_2$’s inputs.

We now show that in Party$_1$’s view, $(W_k, V_k, W_{k+1}, V_{k+1})$ in two consecutive real executions are computationally indistinguishable from $(\tilde{W}_1, \tilde{V}_1, \tilde{W}_2, \tilde{V}_2)$ where $\tilde{W}_1 := (W_{1,1}, \ldots, W_{M,1} : W_{i,1} = \tilde{q}_{i,1} \cdot P, \tilde{q}_{i,1} \in \mathbb{Z}_q^*, i = 1, \ldots, M), \tilde{V}_1 := (\tilde{V}_{1,1}, \ldots, \tilde{V}_{N,1} : \tilde{V}_{j,1} \leftarrow t_j \cdot \tilde{W}_{\pi_1^{-1}(j),1}, j = 1, \ldots, N), \tilde{W}_2 := (\tilde{W}_{1,2}, \ldots, \tilde{W}_{M,2} : \tilde{W}_{i,2} = \tilde{q}_{i,2} \cdot P, \tilde{q}_{i,2} \in \mathbb{Z}_q^*, i = 1, \ldots, M), \text{ and } \tilde{V}_2 := (\tilde{V}_{1,2}, \ldots, \tilde{V}_{N,2} : \tilde{V}_{j,2} \leftarrow t_j \cdot \tilde{W}_{\pi_1^{-1}(j),2}, j = 1, \ldots, N)$. More concretely, we have

\[
\{(\overline{1},k), \ldots, (\overline{M},k), t_1 \cdot (\pi_f^{-1}(1),k), \ldots, t_N \cdot (\pi_f^{-1}(N),k), (\overline{1},k+1), \ldots, (\overline{M},k+1),
\overline{t_1} \cdot (\pi_f^{-1}(1),k+1), \ldots, \overline{t_N} \cdot (\pi_f^{-1}(N),k+1)\} \approx_c \{\tilde{q}_{1,1} \cdot P, \ldots, \tilde{q}_{M,1} \cdot P, t_1 \cdot (\tilde{q}_{\pi_1^{-1}(1),1} \cdot P), \ldots, t_N \cdot (\tilde{q}_{\pi_1^{-1}(N),1} \cdot P), \tilde{q}_{1,2} \cdot P, \ldots, \tilde{q}_{M,2} \cdot P, t_1 \cdot (\tilde{q}_{\pi_1^{-1}(1),2} \cdot P), \ldots, t_N \cdot (\tilde{q}_{\pi_1^{-1}(N),2} \cdot P)\}
\]

where $(i, j)$ is the abbreviation for $a_{b_{i,j}, j} \cdot P_i$, and $b_{i,k} \in \{0, 1\}$ is the semantic value of $\omega_w$ in the $k$-th execution. The proof of their indistinguishability relies on the same flow as the proof of Theorem 1 which depends on Lemma 1 and ultimately on the DDH assumption.
Theorem 3. If the threshold system is secure against malicious adversaries at most \( t - 1 \) of whom are allowed to collude, and the 2PFE protocol proposed in Figure 2 is secure against semi-honest adversaries; then our mechanism with various Party\( n \)s in Sect. 4.2 is also secure against semi-honest adversaries.

Proof (Sketch). First, the Party\( 1 \)'s view in the 2PFE mechanism is equivalent to the one in the protocol in Figure 2. Observe that the generator set is generated by the distributed system and the tokens (that are used in preparation of the garbled input \( X_k \) and the garbled circuit \( F_k \)) are computed from \( \alpha_0,k \) or \( \alpha_1,k \) in each evaluation as in Figure 2. Therefore, the 2PFE mechanism prevents Party\( 1 \) from deducing any information about Party\( n \)'s input.

Second, Party\( n \)'s cannot obtain any information about Party\( 1 \)'s input in none of the executions since the OT outputs are only obtained by Party\( 1 \) due the \( \mathcal{F}_{OT} \) functionality in the OT-hybrid model. Also, due to Theorem 1, no one can obtain information about \( \pi_f \) from the ReuseTemp\( f \). Moreover, any Party\( n,k \) has a negligible advantage on distinguishing the exchanged messages in an evaluation between Party\( 1 \) and Party\( 2,l \) from a random string due to the underlying DDH assumption for \( l \neq k \). More concretely, the tokens (that are used in preparation of Party\( 1 \)'s garbled input \( X_{2,l} \) and the garbled circuit \( F_l \)) are computed by multiplying the elements of the ReuseTemp\( f \) with the private values \( \alpha_{0,l} \) or \( \alpha_{1,l} \) of Party\( 2,l \).

\( \Box \)

References


