Generic 2-Party PFE with Constant Rounds and Linear Active Security, and Efficient Instantiation

Hanyu Jia[1,2], Xiangxue Li[1,3(✉)], Qiang Li[4], Yue Bao[5], and Xintian Hou[5]

[1] School of Software Engineering, East China Normal University, Shanghai 200062, China
xxli@cs.ecnu.edu.cn
[2] Shanghai Key Laboratory of Privacy-Preserving Computation, MatrixElements Technologies, Shanghai 201204, China
[3] Shanghai Key Laboratory of Trustworthy Computing, Shanghai 200062, China
[4] Institute of Cyber Science and Technology, Shanghai Jiaotong University, Shanghai 200240, China
qiangl@sjtu.edu.cn
[5] CATARC Software Testing (Tianjin) Co., Ltd., Tianjin 300300, China
{baoyue,houxintian}@catarc.ac.cn

Abstract. The paper considers generic construction of 2-party private function evaluation (PFE) in the malicious adversary model. There is hitherto only one concrete design of actively secure 2-party PFE protocol (Liu et al. at PKC 2022, and LWY hereafter) with constant rounds and linear complexity. One interesting feature of LWY is its function reusability (i.e., the same function is involved in multiple executions of LWY) which makes its execution more efficiently from the second execution. Nevertheless, in its first execution (in particular for those settings where only one invocation of the function is required), LWY is quite involved and too inefficient to be of practical use. For these settings (of non-reusable private functions), we initiate a generic construction of 2-party PFE protocol with constant rounds and linear complexity in the malicious adversary model based on Yao's garbled circuit and singly homomorphic encryption. When instantiated with ElGamal encryption and Groth secret shuffle (J. Cryptology 2010), the generic construction effectuates a novel concrete design of 2-party PFE, which has better performance and reduces 51.2% communication bits and 52.4% computation costs, compared to LWY (in its first execution) at the same security level. It even outperforms several 2-party PFE protocols (Katz and Malka at AISACRYPT 2011, and Mohassel and Sadeghian at EUROCRYPT 2013) that are secure in the semi-honest adversary model from the communication perspective. The proposed PFE and LWY thus make optimal solutions available for non-reusable and reusable private functions, respectively.

Keywords: Private function evaluation · Active security · Two-party computation · Extended permutation

F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 390–410, 2023.
https://doi.org/10.1007/978-3-031-25538-0_21

1 Introduction

Secure multi-party computation (MPC) protocols allow a group of participants to perform a computation together without revealing the private input of any party [13,57]. Since its introduction by Yao in the 1980 s,s, MPC has evolved from a theoretical curiosity to an important tool for building large-scale privacy-preserving applications [3,10,12,15,16,25,43,51].

SFE AND PFE. There are two cases of MPC protocols, secure function evaluation (SFE) and private function evaluation (PFE), depending on whether the functions involved in the computation are public/private. In SFE, the function $f(\cdots)$ (multi-variable) is not private knowledge of any participant, but an open function that all the parties agree to compute. Suppose there are n parties $P_1, \cdots, P_n$ and P_i has private data x_i. At the end of SFE, n parties are given $f(x_1, \cdots, x_n)$, but any P_i's private data x_i is not disclosed to any other parties. Many SFE protocols rely on translating f into a Boolean or arithmetic circuit $\mathcal{C}_f$. For example, Yao's garbled circuit (GC) [37,58] and GESS [32] are applicable to 2-party computation. Multi-party computation protocols include GMW [20], BGW [7], and BMR [5], and computation rounds of GMW and BGW are linear with the circuit depth, and BMR is constant-round. In PFE, the function $f(\cdots)$ is private knowledge of one of the parties, who cooperates with data owners $P_1, \cdots, P_n$ (holding private x_i, respectively) to jointly compute $f(x_1, \cdots, x_n)$. At the end of PFE, the final result would be obtained while the private knowledge of each party is not disclosed. Existing PFE protocols are mainly based on translating private function f into circuit $\mathcal{C}_f$ with u input wires, g gates, and o output wires. Before executing PFE, function owner should open some parameters to data owners [9,27,29,30,42,46,48], such as the values of u, g, o and some auxiliary parameters, as long as these public parameters would not help the adversaries learn the function f.

MPC SECURITY PROFILE. MPC security is generally divided into a semi-honest adversary model (for low security requirements, a.k.a. passive security model) and a malicious adversary model (for high security requirements, a.k.a. active security model) [13]. In the semi-honest adversary model, each corrupt party follows the prescribed protocol steps, except that he is curious (honest but curious) and tries to infer as much as possible other parties' private data from the transcripts. Therein, corrupt parties might collude with each other in an attempt to learn more knowledge. In the malicious adversary model, corrupt parties can deviate from the protocol arbitrarily. Besides the ability in the semi-honest adversary model, these corrupt parties can also take any action during the protocol execution. An actively secure PFE can detect the behavior of corrupt parties and eventually abort the protocol or kick out them to continue the protocol. The paper concentrates on actively secure PFE. Actively secure SFE [14,28,35,36,39,54] has been well studied, and relatively less attention is paid to actively secure PFE [29,42,48] besides UC-based designs.

GENERIC PFE: UC-BASED. Valiant proposes universal circuit (UC) to achieve PFE. UC can be programmed (letting p_c be control bits) to encode any circuit

$\mathcal{C}_f$ that needs to be protected [55]. Then UC could be public and p_c is private to function owner. Now the problem of computing $\mathcal{C}_f(x)$ from private circuit $\mathcal{C}_f$ (of function owner) and private data x (of data owner) is transformed into computing $\mathrm{UC}(x, p_c)$ via SFE protocols. That is, we gain a technical routine of reducing PFE to SFE by securely computing publicly available universal circuits [1,26,31,34,40,41,55,60]. One may thereby attempt to optimize UC as the smaller the size of the constructed UC, the smaller the consumption required in SFE. The best asymptotic size of Valiant's UC is $12g{\cdot}\log g$ (g is the number of gates in $\mathcal{C}_f$) by Liu et al. [41] and seems to reach theoretical optimum. UC-based PFE designs (either passively or actively secure) would thereby comply with the logarithmic factor in the complexity of $\mathcal{O}(g{\cdot}\log g)$.

Generic PFE: Beyond UC-Based. Katz and Malka [30] present a specific 2-party PFE protocol (KM hereafter) with linear complexity $\mathcal{O}(g)$ from Yao's GC and singly homomorphic encryption (HE) in the semi-honest adversary model. Later, Mohassel and Sadeghian show a general framework (MS hereafter) of PFE in the semi-honest adversary model [46]. MS is divided into two subprotocols, oblivious extended permutation (OEP) and private gate evaluation (PGE). OEP hides the topology of the circuit and protects the relationship between individual gates. PGE is a private computation of individual gates, and the computation of the circuit $\mathcal{C}_f$ is completed after computing g gates one by one. OEP based on oblivious switched network design [47] can build PFE protocols with complexity $\mathcal{O}(g{\cdot}\log g)$, while OEP based on HE can build PFE protocols with complexity $\mathcal{O}(g)$. Mohassel et al. [48] further describe a generic framework of PFE with linear complexity in the malicious adversary model based on the framework MS [46]. Technical tools used therein include one-time MAC for checking data consistency and actively secure SFE for data distribution and reconstruction. In addition, the zero-knowledge proof protocol $\mathcal{ZK}_{EP}$ with $\mathcal{O}(g)$ complexity proves to data owner that function owner performs correct extended permutation (EP, see Definition 1). $\mathcal{ZK}_{EP}$ is the first zero-knowledge protocol with linear complexity proving correct EP operation, and it is a general framework but constructed with redundant computations. Jia and Li [29] propose a more succinct double shuffle protocol ($\mathcal{ZK}_{DS}$) to prove the correctness of EP operations, which is constructed more light-weight and retains the generic feature of $\mathcal{ZK}_{EP}$. In addition, they also construct [29] a general-purpose PFE framework with $\mathcal{O}(g)$ complexity based on $\mathcal{ZK}_{DS}$ for better efficiency in the malicious adversary model.

DDH-Based Concrete PFE with Function Reusability. Bicer et al. [9] propose a 2-party PFE protocol based on decisional Diffie-Hellman (DDH) assumption with linear complexity and function reusability in the semi-honest adversary model. Liu et al. [42] propose a specific zero-knowledge protocol for proving correct EP operation and transforming the 2-party PFE [9] with passive security to that with active security (LWY hereafter). Function reusability says that the same function could be invoked multiple times (for different data each time) so that protocol executions become more efficient from the second execution[1]. The more times of protocol being executed, the smaller of the aver-

[1] The overhead required due to the EP operation on the function is no longer needed from the second execution, which accounts for the major part of the total overhead.

age overhead will be. For detailed protocol design, we refer the readers to the original paper [9,42].

Table 1. *PFE protocols with linear complexity O(g).*

PFE	Security	Parties	Round	Reusable
[30]	Semi-honest	Two	Constant	No
[46]	Semi-honest	Multi & Two	Constant	No
[9]	Semi-honest	Two	Constant	Yes
[48]	Malicious	Multi	#Gates	No
[42]	Malicious	Two	Constant	Yes
[29]	Malicious	Multi	#Gates	No

For ease of understanding, we list in Table 1 main PFE protocols with linear complexity.

1.1 Motivations

Consider 2-party PFE with constant rounds, linear complexity, and active security. Constructing actively secure PFE can be viewed as two divisions, one is to construct a passively secure PFE and another is to construct an efficient primitive proving to data owners that function owner correctly performs EP operation. The latter could be solved using the approach of zero-knowledge protocols [29,42,48]. There is hitherto only one concrete design (i.e., LWY) of actively secure 2-party PFE with constant rounds and linear complexity. One interesting feature of LWY is its function reusability. Nevertheless, in its first execution (in particular for those settings where only one invocation of the function is required), LWY is quite involved and too inefficient to be of practical use.

We note that privacy-preserving machine learning is a good solution to protecting both private models and sensitive data [8,19,44,45,56]. In this regard, the concrete PFE construction [42] with function reusability property could find its position in privacy-preserving inference. For privacy-preserving training however, the model parameters are updated continuously during the training process and now we confront intrinsically a series of non-reusable private functions. It naturally raises the following question:

Can we pursue a generic 2-party PFE with constant rounds and linear active security for non-reusable private functions, whose instantiations might achieve less computation and communication consumption (compared to the first execution of LWY)?

In this paper, we present a positive answer to the question. Next we start with some preparatory knowledge which inspire our constructions.

Two-Party PFE Protocol KM [30]. We briefly describe how KM works in the semi-honest adversary model. KM is based on Yao's GC and singly homomorphic encryption and we assume that the readers are familiar with GC. First, function owner (P_1) translates his private function f into a circuit $\mathcal{C}_f$ which

has u input wires, g NAND gates and o output wires (see Fig. 1). Each of these NAND gates is two fan-in and any fan-out. Let the u input wires of $\mathcal{C}_f$ and the output wires of the $g-o$ non-output gates be called outgoing wires, and the input wires of the g gates be called incoming wires. Data owner (P_2) holds private data $x \in \{0,1\}^u$. P_2 generates public-private key pair (pk, sk) for singly homomorphic encryption (e.g., ElGamal encryption) and sends pk to P_1. In addition, P_2 generates $u+g$ pairs of random wire keys, (s_i^0, s_i^1) representing bits 0 and 1, respectively. P_2 encrypts the first $u+g-o$ pairs wire keys with pk

$$[Enc_{pk}(s_1^0), Enc_{pk}(s_1^1)], \cdots, [Enc_{pk}(s_{u+g-o}^0), Enc_{pk}(s_{u+g-o}^1)] \tag{1}$$

and sends the ciphertexts to P_1. In Eq. (1), the first u ciphertext pairs correspond to the wire keys of u input wires of $\mathcal{C}_f$ and the last $g-o$ ciphertext pairs correspond to the wire keys of output wires of non-output gates. P_1 knows the topology of $\mathcal{C}_f$ and can extend $u+g-o$ ciphertext pairs to $2g$ ciphertext pairs by EP. These $2g$ pairs represent the wire keys of input wires of all gates. P_1 chooses a_i, b_i, a_i', and b_i', $i \in \{1, \cdots, g\}$, uniformly at random from appropriate domain, and encrypts them under pk. Then the $2g$ ciphertext pairs could be re-randomized due to the homomorphic property. For each gate $i \in \{1, \cdots, g\}$ of $\mathcal{C}_f$, suppose that its left incoming wire is connected with some outgoing wire l (of some preceding gate) and right incoming wire is connected with some outgoing wire r (of some preceding gate). P_1 computes

$$\begin{aligned} &[Enc_{pk}(a_i \cdot s_l^0 + b_i), Enc_{pk}(a_i \cdot s_l^1 + b_i)] \\ &[Enc_{pk}(a_i' \cdot s_r^0 + b_i'), Enc_{pk}(a_i' \cdot s_r^1 + b_i')] \end{aligned} \tag{2}$$

Finally, the re-randomized ciphertexts are returned to P_2, who can decrypt them to recover new wire keys representing input wires of each gate. The wire keys for output wires of each gate are generated by himself, so P_2 can create garbled tables. At this point, P_2 has the ability to perform GC protocol with P_1. P_1 knows the topology of $\mathcal{C}_f$ and the random values in the re-randomization, so he has the ability to act as an evaluator and eventually gets the o wire keys representing the output wires of $\mathcal{C}_f$. P_2 receives these o wire keys and obtains the result of $f(x)$.

The Design of $\mathcal{ZK}_{EP}$ [48]. $\mathcal{ZK}_{EP}$ is a generic zero-knowledge protocol to prove that EP is correctly executed. It consists of three components, Dummy Placement Phase, Replication Phase, and Permutation Phase. The number of inputs (input size) in all three phases is $2g$. The first component is the shuffle operation, which takes as input $2g$ ElGamal ciphertexts, including $u+g-o$ wire keys on outgoing wires generated by P_2 and $g-u+o$ dummy values known to both. P_1 then performs shuffle and re-randomization, proving the correctness of his operation by using $\mathcal{ZK}_{shuffle}$ [17]. The second component is a copy operation whose inputs are the outputs of the first component. P_1 performs replication and re-randomization and uses $\mathcal{ZK}_{rep}$ [46] to prove the correctness of his operation. The third component takes as inputs the outputs of the second component, performs shuffle and re-randomization (not the same as the first one), and outputs

$2g$ ElGamal ciphertexts of the wire keys of the incoming wires. P_1 again proves to P_2 the correctness of his operation by $\mathcal{ZK}_{shuffle}$ [17]. P_2 believes that P_1 performs the correct EP if all three components are verified correctly.

The Design of $\mathcal{ZK}_{DS}$ [29]. $\mathcal{ZK}_{DS}$ optimizes the $\mathcal{ZK}_{EP}$ protocol and consists of two succinct components, randomness-generating & outgoing-wires-determining (RG&OWD) and randomness-reusing & incoming-wires-determining (RR&IWD). In $\mathcal{ZK}_{DS}$, the inputs of RG&OWD are $u+g-o$ ElGamal ciphertexts of random wire keys provided by P_2. P_1 executes RG&OWD to generate $u+g-o$ new ElGamal ciphertexts of wire keys of outgoing wires, and proves that his execution is correct by $\mathcal{ZK}_{shuffle}$. P_2 receives a set $\boldsymbol{C}$ of auxiliary parameters ($\boldsymbol{C}$ does not disclose $\mathcal{C}_f$) from P_1 before the protocol. Each element in $\boldsymbol{C}$ tells how many times each random wire key[2] will be copied, and [29] proves that this does not decrease security. Then P_1 executes RR&IWD with the inputs being the ciphertexts of $2g$ random wire keys (generated by copying the inputs of RG&OWD according to $\boldsymbol{C}$). Its outputs are $2g$ new ElGamal ciphertexts of the wire keys of incoming wires. P_1 uses $\mathcal{ZK}_{shuffle}$ to prove the correctness of his execution. Once these two components have been executed, P_2 gets the inputs and outputs of EP. Two $\mathcal{ZK}_{shuffle}$ executions suffice for P_1 to prove the correctness of EP.

1.2 Contributions

LWY is well suited for scenarios where private functions might be reused multiple times (for different private data). In its first execution (in particular for those settings where only one invocation of the function is required) however, LWY is quite involved and too inefficient to be of practical use. For these settings (non-reusable private functions), we initiate the first generic construction of 2-party PFE with constant rounds and linear complexity in the malicious adversary model.

By learning the above, one might perceive that $\mathcal{ZK}_{EP}$ (yet cumbersome) could be *directly* plugged into passively secure KM to produce an actively secure 2-party PFE. This will not work for $\mathcal{ZK}_{DS}$ due to its particular structure, however. We thus design a novel generic construction of 2-party PFE protocol that can exactly support the succinct structure of $\mathcal{ZK}_{DS}$ protocol. The resulting actively secure 2-party PFE is with constant rounds and linear complexity, and its instantiation has much less consumption of communication and computation. Any optimization of $\mathcal{ZK}_{shuffle}$ [4,11,17,21–24,50] would lead to performance improvement of $\mathcal{ZK}_{DS}$ (and surely of our PFE protocol). As an instantiation, we take the scheme in [22] as candidate $\mathcal{ZK}_{shuffle}$ in our 2-party PFE and $\mathcal{ZK}_{DS}$ would thereby require approximately $9g \cdot ||\mathbb{Z}_q||$[3] proof bits and $36g$ exponentiations to prove the EP of $\mathcal{C}_f$. Together with ElGamal encryption, our instantiation of the proposed generic PFE construction has better performance and reduces 51.2% communication costs and 52.4% computation costs, respectively, compared to LWY (in its first execution) at the same security level. It

[2] Note that these random wire keys do not represent outgoing wires, but rather a shuffle and re-randomization with the outgoing wires.

[3] Usually $||\mathbb{Z}_q|| = 160$.

even outperforms passively secure KM [30] and MS [46] from the communication perspective.The proposed PFE and LWY thus make optimal solutions available for non-reusable and reusable private functions, respectively.

We provide in Sect. 3 detailed description of our 2-party PFE and in Sect. 4 detailed performance comparison.

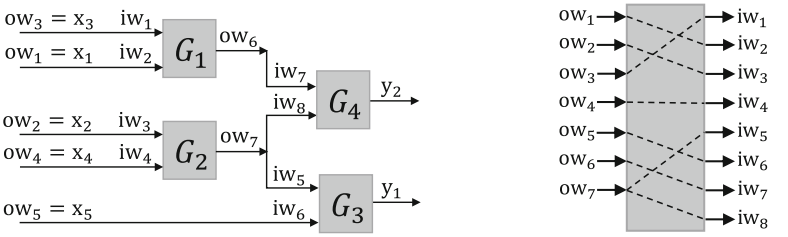
2 Preliminaries

We denote the security parameter by k. $r \leftarrow_R \{0,1\}^k$ means that r is a random number chosen uniformly at random from $\{0,1\}^k$. We use both bold and italic (lower-case or capital) letters to denote sets (e.g., $\boldsymbol{D}$), and italic for values or elements in a set (e.g., D_i). For a set $\boldsymbol{S}$, we denote the size of the set by $|\boldsymbol{S}|$, and we write $\boldsymbol{S} = \{S_i\}_{i=1}^{|\boldsymbol{S}|}$. We use $||\mathbb{G}||$ to denote the bit length of a group element in the group $\mathbb{G}$. We use $s := s + r$ to denote reassigning a new value to element s. We denote a mapping function by π, e.g., $j = \pi(i)$ where i is the preimage and j is the corresponding image. We use π_1 and π_2 to denote bijective functions and π_3 a surjective function.

We use a singly homomorphic encryption scheme (Gen,Enc,Dec), whose plaintext space is $\mathbb{G}$ of prime order q. We then use n to denote the security parameter of the homomorphic encryption scheme, i.e., $(pk, sk) \leftarrow \text{Gen}(1^n)$ denotes that a pair of public and private keys is generated. We want to encrypt the plaintext $m \in \{0,1\}^k$ and we can map m to a group element in $\mathbb{G}$. For the convenience of representation, we directly use $c = \text{Enc}_{pk}(m)$ to denote encryption of plaintext m into ciphertext c with the public key pk. We use $m = \text{Dec}_{sk}(c)$ to denote decryption of ciphertext c into plaintext m with private key sk. Singly homomorphic encryption suitable for our protocol includes Elliptic curve (EC) ElGamal [18] and Paillier [52], etc. It is believed [27] that EC ElGamal encryption is more efficient to implement some known 2-party PFE protocols from GC [30]. We use EC ElGamal encryption as well, and for a detailed homomorphic addition step we suggest to read [[27] Sect. 4.3].

We also need a symmetric encryption scheme (sEnc, sDec) whose plaintext space and key space are both k-bits random numbers. Given a ciphertext $c = \text{sEnc}_{sk}(m)$ (sk is the secret key in the symmetric cipher), we have $m = \text{sDec}_{sk}(c)$. The symmetric cipher would be used in our protocols to create standard Yao's GC for each garbled table (GT) and decrypt each GT. It is required [38] for (sEnc,sDec) that it has elusive and efficiently verifiable range. Additionally, we require that (sEnc,sDec) satisfies a weak form of related-key security, and the work of Applebaum et al. [2] can be used to construct a scheme that satisfies the security based on the decisional Diffie-Hellman assumption. Holz, Nissim et al. [27, Sect. 5.1] used an AES-128 encryption scheme [6] to construct the GC in the linear PFE protocol.

Definition 1 (EP [46]). *The inverse of π_3 is defined as an extended permutation function, i.e., π_3^{-1}. In the following, $|\boldsymbol{ow}|$ elements would be copied and permuted to $|\boldsymbol{iw}|$ elements, i.e., $ow_i = iw_{\pi_3^{-1}(i)}$.*



(a) A simple example circuit $\mathcal{C}_f$ where $u = 5$, $g = 4$ and $o = 2$.

(b) EP relationship between $\boldsymbol{ow}$ and $\boldsymbol{iw}$ in $\mathcal{C}_f$.

Fig. 1. A simple example of circuit $\mathcal{C}_f$, and its corresponding EP relationship.

There are two players in our 2-party PFE protocol, function owner P_1 holding f and data owner P_2 holding x. P_1 will privately translate the function f into a Boolean circuit $\mathcal{C}_f$ (see Fig. 1) with u input wires, g gates and o output wires (in general, $u \approx o, u \ll g$). Let $N = u + g$ and $M = 2g$. Note that all the g gates of $\mathcal{C}_f$ in our protocol are NAND gates with two fan-in and more than one fan-out, so there is no need to hide the gate function. We declare that this is different from the standard Yao's GC protocol. The $\mathcal{C}_f$ is like a directed acyclic graph, i.e., all the gates have topological order. We use $\boldsymbol{G} = \{G_i\}_{i=1}^{g}$ to denote the g gates that have been topologically sorted. We divide the g gates into output and non-output gates according to the destination of each gate output wire, i.e., $\{G_i\}_{i=1}^{g-o}$ are non-output gates and $\{G_i\}_{i=g-o+1}^{g}$ are output gates. We refer to both the input wires of $\mathcal{C}_f$ and the output wires of the non-output gates collectively as outgoing-wire (abbreviated as $\boldsymbol{ow}$), and in addition refer to the input wires of all gates collectively as incoming-wire (abbreviated as $\boldsymbol{iw}$). It is obvious that we can observe that $|\boldsymbol{ow}| = N - o$, $|\boldsymbol{iw}| = M$ and $|\boldsymbol{ow}| \leq |\boldsymbol{iw}|$. Since each gate of $\mathcal{C}_f$ is the NAND gate, the problem of protecting the private function f is transformed into the problem of protecting the EP relation from $\boldsymbol{ow}$ to $\boldsymbol{iw}$ ($iw_i = ow_{\pi_3(i)}$), i.e., P_1 hides the topological order of circuit $\mathcal{C}_f$ from P_2. We decompose the EP problem (π_3^{-1}) into two permutation problems (π_1 and π_2).

3 Two-Party PFE with Linear Active Security

3.1 High-Level Description

Next, we describe our generic construction of actively secure 2-party PFE. We suppose the readers are familiar with Yao's GC. Our PFE has two parties, P_1 the function owner and P_2 the data owner. P_1 has private function $f(\cdot)$ (one-variable), and P_2 has private data x. In the original Yao's GC, function owner acts as the garbler of the circuit and data owner acts as the evaluator of the circuit. In our PFE however, they are assumed opposite roles, i.e., P_2 acts as the garbler of the circuit and P_1 acts as the evaluator of the circuit. The goal is that P_1 and P_2 cooperate to compute $y = f(x)$ (only disclosed to P_2), while P_2 cannot

learn valid knowledge about f and P_1 cannot learn x. Our PFE uses a singly homomorphic encryption as one of the building blocks. EC ElGamal encryption is currently one of the most effective candidates applicable to our PFE [27,30].

The function f of P_1 is translated privately as a Boolean circuit $\mathcal{C}_f$ with u inputs, o outputs and g gates, and we let $N = u + g$. The g gates are all NAND gates. All gates are two fan-in and can be any fan-out. For each gate with the fan-out greater than 1, we view each of its output wires as a different wire. This differs from the gates in the universal circuit [41,55]. Highly optimized hardware synthesis tools already exist for translating the function to Boolean circuits with a small number of (or optimized) NAND gates [27]. Since the output of each non-output NAND gate and the input of the circuit ($\boldsymbol{ow}$) are used at least once in the circuit $\mathcal{C}_f$, P_1 can compute a set $\boldsymbol{C}$. P_2 has private data $x \in \{0,1\}^u$.

We view u, o, g and $\boldsymbol{C}$ as system parameters, which are not secret knowledge of f and x, i.e., others can know these parameters but cannot recover f and x effectively. The topology of $\mathcal{C}_f$ and x should be only known to P_1 and P_2, respectively. In our protocol, P_2 does not know the topology of the circuit, so he cannot construct the garbled gates directly. Instead, we can take advantage of the property that the ciphertexts can be directly summed up in a homomorphic encryption: let P_2 provide N - o wire keys, encrypt these wire keys, and send the ciphertexts to P_1. According to the topology of the circuit, P_1 can perform permutation (π_1) and re-randomization on N - o ciphertexts to obtain new N - o ciphertexts about the $\boldsymbol{ow}$, and then apply permutation (π_2) and re-randomization to the replication results of N - o ciphertexts to obtain new $2g$ ciphertexts about the $\boldsymbol{iw}$. Note that the protocol here is different from KM protocol, where P_2 generates the wire keys about $\boldsymbol{ow}$ directly. We say that P_1 constructs the encrypted garbled gate (abbreviated as $encGG_i$) for the i-th NAND gate in the circuit. By decrypting $encGG_i$, P_2 can obtain $3g$-o new wire keys which can be used to create garbled tables. After P_2 gets g $encGG$s, he can act as the garbler. As each wire key is re-randomized by P_1, P_2 cannot learn the topology of the circuit. We describe the protocol in detail below.

3.2 Specification

We decouple the PFE into three phases: the *offline* phase, the *initiation* phase, and the *evaluation* phase. Figure 2 shows the details.

1) Offline Phase. In this phase, P_1 translates the private function f into a circuit $\mathcal{C}_f$ and discloses the parameters u, g, o, and $\boldsymbol{C}$. P_2 generates a public/private key pair (pk,sk) for a singly homomorphic encryption and discloses the public key pk. In addition, P_2 chooses uniformly $r_i \leftarrow_R \{0,1\}^k$, $i \in \{1,2\}$, where k is the security parameter (say 128 [27]). These two random values would be used in creating the garbled tables.

Then, P_2 randomly generates a set of wire keys, $\boldsymbol{H} = \{h_i\}_{i=1}^N$. We assume that all the N wire keys represent the bit value 0, which are written as $\boldsymbol{H}^0 = \{h_i^0\}_{i=1}^N$. Similarly, we use $\boldsymbol{H}^1 = \{h_i^1\}_{i=1}^N$ to represent the N wire keys

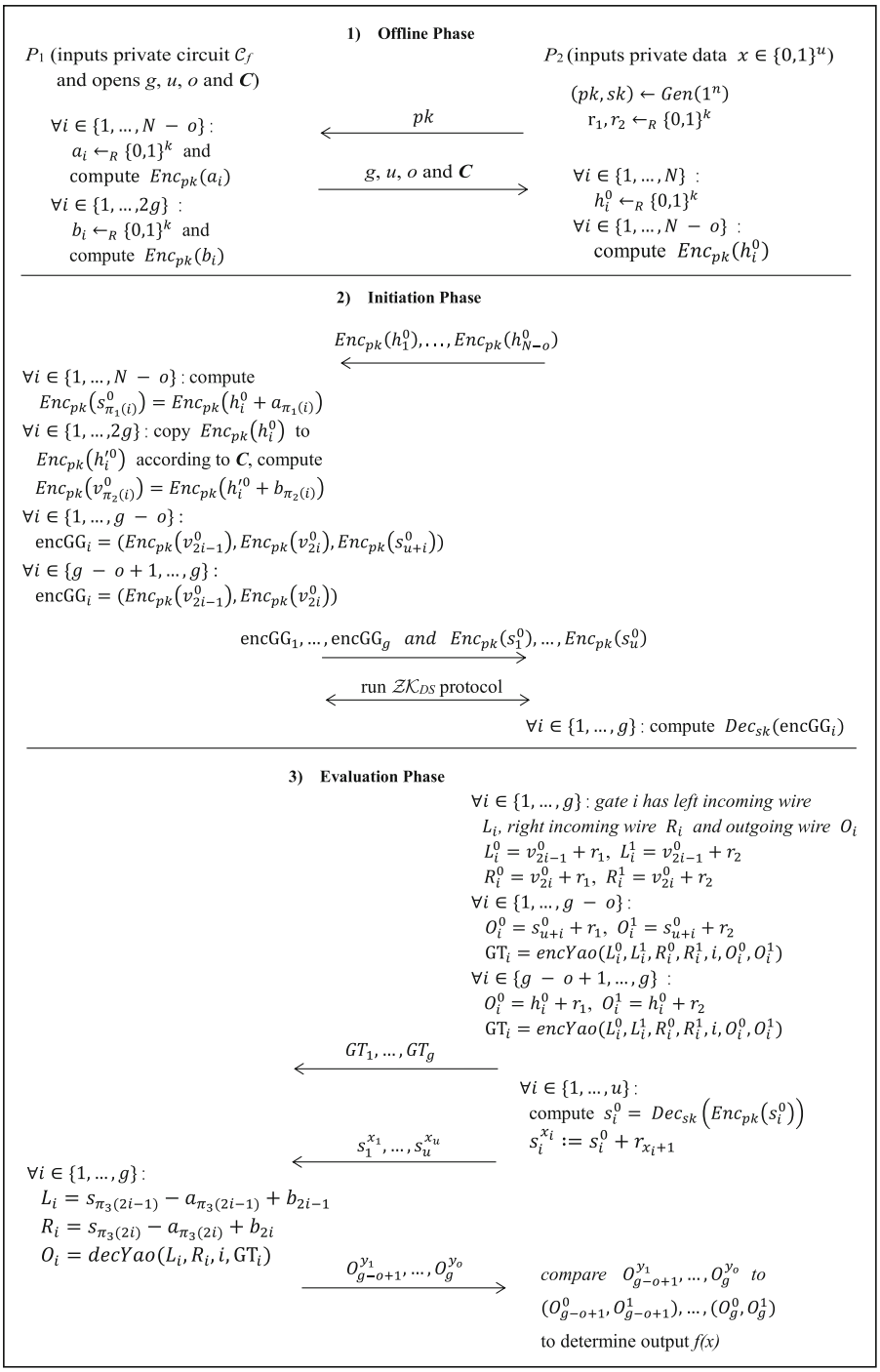


Fig. 2. Our 2-party PFE protocol. π_3 maps $2g$ incoming wires to N-o outgoing wires.

corresponding to the bit value 1. Rather than sampling the wire key h_i^1 as h_i^0, we produce it by adding a global shift r to h_i^0, i.e., $h_i^1 = h_i^0 + r$. This trick also appears in the application of *free-XOR* [33]. The security of Yao's protocol depends on the indistinguishability of the wire keys h_i^0 and h_i^1. Note that these N wire keys do not correspond one-to-one[4] to the N-o outgoing wires and o output wires of $\mathcal{C}_f$. In the semi-honest adversary model [30], the first u of these N wire keys correspond to the bits that represent the data x, the next g - o correspond to the output results that represent the non-output gates, and the last o correspond to the results that represent the output gates. We assume that P_2 selects the last o wire keys in $\boldsymbol{H}^0$, i.e., $\{h_i^0\}_{i=N-o+1}^{N}$, which correspond one-to-one to the output results used for the o output gates. P_2 encrypts the other N - o wire keys in $\boldsymbol{H}^0$. In addition, P_1 uniformly chooses N - o random values a_i and $2g$ random values b_i and encrypts them under pk.

The complexity of this phase is $\mathcal{O}(g)$ and we have about $||\mathbb{G}||$ communication bits and $8g$ exponentiation computations. This phase can be set up before the start of the protocol and thus one may freeze out the consumption of the phase.

2) Initiation Phase. In this phase, P_2 sends $\{\mathrm{Enc}_{pk}(h_i^0)\}_{i=1}^{N-o}$ to P_1. Upon receiving $N - o$ ciphertexts, P_1 first needs to determine the order of the wire keys used for $\boldsymbol{ow}$. He applies permutation (π_1) and re-randomization to $\{\mathrm{Enc}_{pk}(h_i^0)\}_{i=1}^{N-o}$ to obtain N - o new ciphertexts $\mathrm{Enc}_{pk}(s_i^0) = \mathrm{Enc}_{pk}(h^0_{\pi_1^{-1}(i)}) + \mathrm{Enc}_{pk}(a_i)$ based on the topology of the circuit and the property of homomorphic addition. The first u plaintexts in $\{\mathrm{Enc}_{pk}(s_i^0)\}_{i=1}^{N-o}$ correspond one-to-one to the wire keys that represent x. The next g - o plaintexts correspond one-to-one to the wire keys that represent the outputs of the non-output gates. Then P_1 extends $\{\mathrm{Enc}_{pk}(h_i^0)\}_{i=1}^{N-o}$ to $2g$ ciphertexts according to $\boldsymbol{C}$ and we denote the new set as $\{\mathrm{Enc}_{pk}(h_i^{'0})\}_{i=1}^{2g}$. This process is a $(c_i$ - $1)$-copy operation on $\mathrm{Enc}_{pk}(h_i^0)$, i.e., each ciphertext in $\{\mathrm{Enc}_{pk}(h^{'0}_{\sum_{j=1}^{i-1} c_j+1}), \cdots, \mathrm{Enc}_{pk}(h^{'0}_{\sum_{j=1}^{i} c_j})\}$ is equal to $\mathrm{Enc}_{pk}(h_i^0)$, where c_i is the i-th value in $\boldsymbol{C}$, $i \in \{1, \cdots, N-o\}$. Next, P_1 applies permutation (π_2) and re-randomization to $\mathrm{Enc}_{pk}(h'_j)$ to obtain $2g$ new ciphertexts $\mathrm{Enc}_{pk}(v_j^0) = \mathrm{Enc}_{pk}(h^{'0}_{\pi_2^{-1}(j)}) + \mathrm{Enc}_{pk}(b_j)$, $j \in \{1, \cdots, 2g\}$. The plaintexts of these $2g$ new ciphertexts correspond one-to-one to the wire keys of the incoming wires of the g gates that have been topologically sorted, i.e., the plaintexts of $\mathrm{Enc}_{pk}(v_{2i-1})$ and $\mathrm{Enc}_{pk}(v_{2i})$ correspond to the two wire keys of the i-th gate's two incoming wires, $i \in \{1, \cdots, g\}$. P_1 creates g encrypted garbled gates $encGG_i$, where the first g - o have different forms from the last o. Thus, the wire keys of two incoming wires used in the gate G_i and representing the bit value 0 are v_{2i-1}^0 and v_{2i}^0, and the wire keys of the outgoing wire are s_{u+i}^0, $i \in \{1, \cdots, g\}$, then the first g - o are $encGG_i = (\mathrm{Enc}_{pk}(v_{2i-1}^0), \mathrm{Enc}_{pk}(v_{2i}^0), \mathrm{Enc}_{pk}(s_{u+i}^0))$ $(1 \le i \le g - o)$ and the last o are $encGG_i = (\mathrm{Enc}_{pk}(v_{2i-1}^0), \mathrm{Enc}_{pk}(v_{2i}^0))$ $(g - o + 1 \le i \le g)$. P_1 then sends $\{encGG_i\}_{i=1}^{g}$ and $\{\mathrm{Enc}_{pk}(s_i^0)\}_{i=1}^{u}$ to P_2. P_1 also applies $\mathcal{ZK}_{DS}$ to prove the correctness of his execution. Then, P_2 decrypts each $encGG_i$ and recovers the new wire keys in preparation for

[4] The *one-to-one* here means that the i-th element of the former set $(\boldsymbol{A})$ corresponds to the i-th element of the latter set $(\boldsymbol{B})$, i.e., A_i corresponds to B_i.

constructing the GTs. From each of the first g - o $encGG_i$, P_2 obtains two input wire keys representing the bit value 0 of gate G_i, v^0_{2i-1} and v^0_{2i}, and also the output wire keys representing the bit value 0, s^0_{u+i}, $i \in \{1, \cdots, g-o\}$. From each of the last o $encGG_i$, P_2 gets only two input wire keys representing the bit value 0 of gates G_i, v^0_{2i-1} and v^0_{2i}, $i \in \{g-o+1, \cdots, g\}$. P_2 also decrypts $\{\text{Enc}_{pk}(s^0_i)\}^u_{i=1}$ and obtains $\{s^0_i\}^u_{i=1}$. Even with $\{s^0_i\}^{N-o}_{i=1}$ and $\{v^0_j\}^{2g}_{j=1}$, P_2 cannot learn about the topology of the circuit due to P_1's re-randomization on these values.

The complexity of this phase is $\mathcal{O}(g)$ and we have approximately $(4u + 8g - 4o) \cdot ||\mathbb{G}|| + 9g \cdot ||\mathbb{Z}_q||$ communication bits and $39g$ exponentiation computations.

3) Evaluation Phase. By decrypting each $encGG_i$, P_2 can act as the circuit garbler in Yao's protocol. As shown in Fig. 3, we let L_i, R_i and O_i denote the left incoming wire, right incoming wire and outgoing wire of the gate G_i. Next, we use the global random values r_1 and r_2 generated in the offline phase to define the corresponding wire keys: $L^0_i = v^0_{2i-1} + r_1$, $L^1_i = v^0_{2i-1} + r_2$, and similarly, $R^0_i = v^0_{2i} + r_1$ and $R^1_i = v^0_{2i} + r_2$. There will be a bit different for outgoing wires. The wire keys about the g - o non-output gates are $O^0_i = s^0_{u+i} + r_1$ and $O^1_i = s^0_{u+i} + r_2$, $i \in \{1, \cdots, g-o\}$; the wire keys of the o output gates are $O^0_i = h^0_i + r_1$ and $O^1_i = h^0_i + r_2$, $i \in \{g-o+1, \cdots, g\}$. Garbled tables GT_i can then be generated according to Yao's protocol, and the detailed implementation is shown in Fig. 4 where we use symmetric encipher sEnc. The secret keys of the symmetric encipher are the input keys (L^0_i, L^1_i) and (R^0_i, R^1_i) of G_i, and the plaintext is the output key (O^0_i, O^1_i), $i \in \{1, \cdots, g\}$. This yields a truth table which needs to be randomly permuted to become a garbled table. In [27], sEnc is instantiated with AES-128. We emphasize that the input and output wire keys of each gate in the circuit are preprocessed and additive homomorphic operations are applied. More discussions on optimization techniques (e.g., point-and-permute [5], garbled row reduction [49,53], or half-gates [59]) might be found in [27]. After that, P_2 sends $\{\text{GT}_i\}^g_{i=1}$ to P_1.

Now P_2 needs to compute the wire keys representing the private data x. It may be noted that we have not used the u wire keys about $\{s^0_i\}^u_{i=1}$. P_2 does not directly use these wire keys to represent the 0-bit in data x, but to perform the following computations. P_2 calculates $s^{x_i}_i := s^0_i + r_1$ to denote $x_i = 0$ and $s^{x_i}_i := s^0_i + r_2$ to denote $x_i = 1$, $i \in \{1, \cdots, u\}$, i.e., $s^{x_i}_i := s^0_i + r_{x_i+1}$. We use the wire keys $\{s^{x_i}_i\}^u_{i=1}$ of the circuit input wires to denote P_2's input bits $\{x_i\}^u_{i=1}$. P_2 sends these new wire keys to P_1.

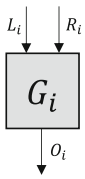


Fig. 3. L_i is the left incoming wire of G_i, R_i the right incoming wire and O_i the outgoing wire.

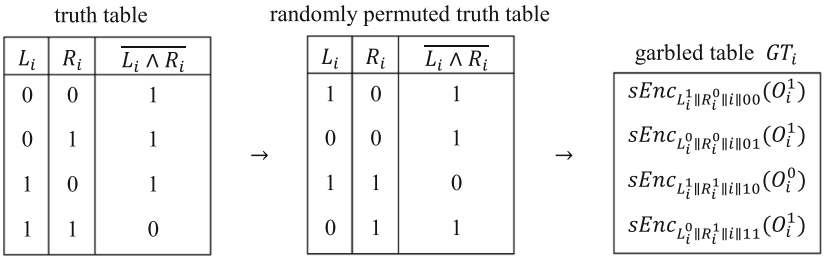


Fig. 4. Create a garbled table.

P_1 has the ability to act as the circuit evaluator after receiving the input bit keys sent to him by P_2, and he has enough information to decrypt the garbled tables and then recover the wire keys for the output wires in this circuit. This decryption process is similar to Yao's GC method, but not exactly the same. Next P_1 decrypts the garbled table corresponding to the g NAND gates one by one according to the topological order of the circuit. In order to decrypt the garbled table GT_i, P_1 recovers the keys used to encrypt the GT_i. Starting from GT_1, the two incoming wires of G_1 are the keys of the input bits about x, i.e., $s_1^{x_1}$ and $s_2^{x_2}$. P_1 knows the random values a_1 and a_2 used to re-randomize the two keys and also knows b_1 and b_2, so he can calculate $L_1^{x_1} = s_1^{x_1} - a_1 + b_1 = v_1^0 + r_{x_1+1}$ and $R_1^{x_2} = s_2^{x_2} - a_2 + b_2 = v_2^0 + r_{x_2+1}$. P_2 gets $L_1^{x_1}$ and $R_1^{x_2}$ to decrypt GT_1 and recovers the outgoing wire key $O_1^{NAND(x_1,x_2)}$, which is used to decrypt the garbled table(s) later. Thus P_1 decrypts the garbled tables one by one in topological order, and once all the garbled tables are computed, he obtains the output wire keys of the o output gates $\{O_{g-o+1}, \cdots, O_g\}$, and sends these wire keys to P_2. P_2 receives $\{O_{g-o+1}, \cdots, O_g\}$ to recover the value of *f(x)*.

The complexity of this phase is $\mathcal{O}(g)$ and we have $(u + 4g + o) \cdot k$ communication bits and u exponentiation computations. The costs of symmetric cipher are negligible.

To sum up, the computation complexity of the proposed protocol is $\mathcal{O}(g)$. We have the total communication overhead of $(4u + 8g - 4o + 1) \cdot ||\mathbb{G}|| + 9g \cdot ||\mathbb{Z}_q|| + (u + 4g + o) \cdot k$ bits and approximately $39g$ exponentiation computations[5].

3.3 Heuristic Analysis

We next check whether it can resist against malicious adversaries. To achieve actively secure PFE protocol, malicious P_1 should be prevented from learning the valid knowledge of P_2's private x, and malicious P_2 should be prevented from learning the valid knowledge of P_1's private f. If one party cheats in the protocol, it should be checkable by another party or this cheating is useless. In addition, the function f of P_1 should be irreversible, i.e., it should not be like $f(x) = x$.

[5] Exponentiation is the dominant computation in the protocol. We omit lightweight operations (e.g., symmetric cipher, addition, etc.).

We analyze above three phases (the values, the computation, and the interaction) one by one. Note that there is a takeaway in the protocol: we need to prove that P_1 uses the correct permutation maps π_1 and π_2. Fortunately, this can be resolved exactly by $\mathcal{ZK}_{DS}$ protocol. For the public key pk sent by P_2 to P_1 in the *offline phase*, one can simply and efficiently verify that it is generated by ElGamal encryption. P_2 generates random values r_1 and r_2 as part of the keys used to represent the wires, and for his own security he does not fudge, and even if he does he cannot learn the valid knowledge of f. In the *initiation phase*, the N - o wire keys sent by P_2 to P_1 can also be verified simply and efficiently that these ciphertexts are well-formed corresponding to the public key pk. Then P_1 does a permutation (π_1) and re-randomization operation on the N - o wire keys generated by P_2, and a permutation (π_2) and re-randomization operation on the $2g$ extended wire keys. At the end of this phase, P_2 knows the ciphertexts of $\boldsymbol{v}^0$ and $\boldsymbol{s}^0$. Along with the knowledge of $\boldsymbol{C}$, he knows the inputs and outputs of the two shuffles (permutation and re-randomization). P_1 can convince P_2 that he performs the correctly local work through the $\mathcal{ZK}_{DS}$ protocol. In the *evaluation phase* P_2 creates the garbled tables. If P_2 does not create these garbled tables correctly, it could be using fake wire keys or using the gates other than NAND. The former will be checked when P_1 decrypts the garbled table, and the latter is P_2 not getting the correct result. As long as P_1 does not tell P_2 in the process that he made an error in decrypting that one garbled table, P_2 cannot learn valid knowledge of f. The next P_2 sends P_1 the u bit keys representing the data x. These ciphertexts are well-formed corresponding to the public key pk and can be verified simply and efficiently, and P_2 cannot learn valid knowledge about f by falsifying these ciphertexts. At the end, P_1 decrypts the g garbled tables one by one, which are constructed using a symmetric encryption scheme. P_1 can only recover one row of the corresponding garbled table using the keys obtained in decrypting encrypted GT_1, and then calculates the keys to decrypt the subsequent garbled tables, and finally obtains o wire keys $O_{g-o+i}^{y_i}$ (where y_i is the i-th bit of $f(x)$), $i \in \{1, \cdots, o\}$. Since P_1 does not know the random values r_1 and r_2, the possibility of replacing $O_{g-o+i}^{y_i}$ with $O_{g-o+i}^{1-y_i}$ is negligible. If P_1 modifies $O_{g-o+i}^{y_i}$ privately, this can be checked by P_2 quite simply and effectively.

3.4 Security

Theorem 1. *The proposed 2-party PFE protocol has active security.*

Proof sketch. We can demonstrate the security of the proposed 2-party PFE protocol in the malicious adversary model by the real-ideal simulation paradigm. Intuitively, a protocol $\mathcal{P}$ is secure if anything a party sees can only be computed from that party's inputs and outputs.

We construct a simulator $\mathcal{S}_{2-party}$ that makes a poly-time environment $\mathcal{Z}$ unable to distinguish between the real protocol system and the ideal protocol system. We assume here that the corrupted adversary is malicious (active) and static. This simulator $\mathcal{S}_{2-party}$ runs a copy of the protocol given in Fig. 2, which relays messages between the parties and $\mathcal{Z}$ so that $\mathcal{Z}$ will see the same interface

as when the actual protocol is interacted with. The next detailed description of $\mathcal{S}_{2-party}$ is presented in Table 2.

Table 2. Simulator $\mathcal{S}_{2-party}$

Simulator $\mathcal{S}_{2-party}$
1) **Offline Phase**
P_2 generates (pk, sk).
– If P_2 is a corrupter. Simulator $\mathcal{S}_{2-party}$ verifies whether pk is a well-formed public key, and if not, simulator aborts.
– If P_2 is not a corrupter. Simulator $\mathcal{S}_{2-party}$ honestly follows the protocol.
2) **Initiation Phase**
P_2 sends $N-o$ ciphertexts, $\{Enc_{pk}(h_i)\}_{i=1}^{N-o}$.
– If P_2 is the corrupter. Simulator $\mathcal{S}_{2-party}$ verifies whether $\{Enc_{pk}(h_i)\}_{i=1}^{N-o}$ are well-formed ciphertexts, and if not, simulator aborts.
– If P_2 is not a corrupter. Simulator $\mathcal{S}_{2-party}$ honestly follows the protocol.
P_1 performs two different shuffling operations on $\{Enc_{pk}(h_i)\}_{i=1}^{N-o}$, as described in Sect. 3.
– If P_1 is the corrupter. Simulator $\mathcal{S}_{2-party}$ randomly generates two mapping functions (π_1 and π_2) and sends them to $\mathcal{ZK}_{DS}$. Then, simulator aborts.
– If P_1 is not the corrupter. Simulator $\mathcal{S}_{2-party}$ waits for two mapping functions (π_1 and π_2) sent by P_1 and sends them to $\mathcal{ZK}_{DS}$.
In both cases, if P_2 aborts, simulator aborts.
3) **Evaluation Phase**
After P_2 decrypts the encrypted garbled gates, he creates g garbled tables and calculates u wire keys representing his data x.
– If P_2 is the corrupter. Simulator $\mathcal{S}_{2-party}$ randomly generates g garbled tables and u wire keys and proceeds to decrypt the garbled tables.
– If P_2 is not the corrupter. Simulator $\mathcal{S}_{2-party}$ follows the protocol honestly.
In both cases, if P_1 aborts, simulator aborts.
P_1 decrypts the garbled tables and recovers the o wire keys
– If P_1 is the corrupter. Simulator $\mathcal{S}_{2-party}$ randomly generates o wire keys and proceeds with the protocol.
– If P_1 is not the corrupter. Simulator $\mathcal{S}_{2-party}$ follows the protocol honestly
In both cases, if P_2 aborts, simulator aborts.

In order to see that the simulated process is indistinguishable from the real process, we will show that the view of the environment in the ideal process is computationally indistinguishable from the view in the real process. This view includes the honest player's inputs and outputs as well as the corrupted player's view of the protocol execution.

The views of the adversaries P_1 include: the public key pk, random values $\{a_i\}_{i=1}^{N-o}$ and $\{b_i\}_{i=1}^{2g}$, $\{\mathrm{Enc}_{pk}(h_i^0)\}_{i=1}^{N-o}$, $\{\mathrm{Enc}_{pk}(s_i^0)\}_{i=1}^{N-o}$, $\{\mathrm{Enc}_{pk}(v_i^0)\}_{i=1}^{2g}$,

$\{GT_i\}_{i=1}^{g}$, $\{s_i^{x_i}\}_{i=1}^{u}$, $\{L_i\}_{i=1}^{g}$, $\{R_i\}_{i=1}^{g}$ and $\{O_i\}_{i=1}^{g}$. $\{s_i^{x_i}\}_{i=1}^{u}$, $\{L_i\}_{i=1}^{g}$, $\{R_i\}_{i=1}^{g}$ and $\{O_i\}_{i=1}^{g}$ are the results computed from the random values, which all look random and are therefore indistinguishable in real and ideal execution. Due to the randomness of r_1 and r_2, the probability that P_1 accurately computes $\{O_{g-o+i}^{1-y_i}\}_{i=1}^{o}$ based on $\{O_{g-o+i}^{y_i}\}_{i=1}^{o}$ is negligible, so in the evaluation phase, he must send the obtained o wire keys to P_2 correctly, and malicious falsification of wire keys is easily detected by P_2. The ElGamal scheme is based on the DDH difficulty assumption, and the probability of recovering sk according to pk is negligible. $\{Enc_{pk}(h_i^0)\}_{i=1}^{N-o}$ are the ciphertexts encrypted by the ElGamal scheme, sk is private to P_2 and therefore indistinguishable in the real and ideal execution. If the protocol is not aborted, $\{\mathrm{Enc}_{pk}(s_i^0)\}_{i=1}^{N-o}$ and $\{Enc_{pk}(v_i^0)\}_{i=1}^{2g}$ are valid permuted and re-randomized ElGamal ciphertexts due to the verification of $\mathcal{ZK}_{DS}$. $\{GT_i\}_{i=1}^{g}$ are obtained based on the symmetric encryption scheme and random garbled, and is therefore indistinguishable in real and ideal execution.

The views of the adversaries P_2 include: the pk and sk, random values r_1 and r_2, $\{\mathrm{Enc}_{pk}(h_i^0)\}_{i=1}^{N-o}$, $\{\mathrm{Enc}_{pk}(s_i^0)\}_{i=1}^{N-o}$, $\{\mathrm{Enc}_{pk}(v_i^0)\}_{i=1}^{2g}$, $\{h_i^0\}_{i=1}^{N}$, $\{s_i^0\}_{i=1}^{N-o}$, $\{v_i^0\}_{i=1}^{2g}$, $\{L_i\}_{i=1}^{g}$, $\{R_i\}_{i=1}^{g}$, $\{O_i\}_{i=1}^{g}$, $\{GT_i\}_{i=1}^{g}$ and $\{s_i^{x_i}\}_{i=1}^{u}$. $\{s_i^{x_i}\}_{i=1}^{u}$, $\{s_i^0\}_{i=1}^{N-o}$, $\{v_i^0\}_{i=1}^{2g}$, $\{L_i\}_{i=1}^{g}$, $\{R_i\}_{i=1}^{g}$ and $\{O_i\}_{i=1}^{g}$ are the results computed from the random values, which all look random and are therefore indistinguishable in real and ideal execution. ElGamal ciphertexts all are indistinguishable in real and ideal execution. P_2 must ensure that the $\{GT_i\}_{i=1}^{g}$ created is correct, otherwise they will be checked by P_1 when decrypting the garbled, or P_2 doesn't get the correct result. If P_2 wants to successfully cheat P_1 and learn valid knowledge of C_f , he must guess exactly every mapping relation of the function π_3, which is obviously a negligible probability. The random values and wire keys all have uniform distribution in ideal and real execution.

As a result, it is indistinguishable between ideal and real execution for environment $\mathcal{Z}$.

4 Performance

In this paper, we initiate the first generic construction of 2-party PFE protocol with constant rounds and linear complexity in the malicious adversary model. In the case where the function is invoked once, we compare it (after instantiated by ElGamal encryption and Groth's secret shuffle [22]) with the only 2-party PFE protocol LWY [42] in the malicious adversary model. See Table 3. We consider the total communication costs and the exponentiation of the protocol. We let $||\mathbb{G}|| = 1024$ and $||\mathbb{Z}_q|| = 160$. The communication bits and exponentiation computations of our protocol are about $10144g$ and $39g$, respectively. For the same parameters, LWY requires about $20800g$ communication bits and $82g$ exponentiation (including $8g$ exponentiation in constructing and decrypting GTs). Our protocol reduces approximately 51.2% communication costs and 52.4% computation costs, compared to the first execution of LWY. We mention that from the second execution, LWY requires at least a total of $4096g$ communication bits and $8g$ exponentiation computations in each execution.

We also analyze the communication costs of all passively secure 2-party PFE protocols. The communication costs in the original KM protocol Sect.3.1 [30], optimal KM protocol Sect.3.2 [30], MS [46], and Bicer et al.'s protocol [9] are approximately $16896g$ bits, $8704g$ bits, $10752g$ bits, and $7168g$ bits, respectively. Thus our protocol even outperforms the original KM and MS that are passively secure from the communication perspective.

Table 3. Communication costs and exponentiation consumption in LWY and ours. $||\mathbb{G}|| = 1024$, $||\mathbb{Z}_q|| = 160$ and $k = 128$.

PFE	Communication (bits)			Exponentiation		
	P_1	P_2	***Total***	P_1	P_2	***Total***
LWY [42]	$\sim 15520g$	$\sim 5280g$	$\sim 20800g$	$\sim 51g$	$\sim 31g$	$\sim 82g$
Ours	$\sim 7584g$	$\sim 2560g$	$\sim 10144g$	$\sim 18g$	$\sim 21g$	$\sim 39g$

5 Conclusion

Both our generic PFE and the concrete LWY are with constant rounds, linear complexity, and full security. They make optimal solutions available for non-reusable and reusable private functions, respectively. We believe that our constructions have practical relevance. In particular, we do expect our PFE could be both easier to implement and more efficient (for large circuits) than existing proposals (e.g., UC-based). We are also interested in constructing 2-party PFE with constant rounds and linear active security from other cryptographic primitives (to pursue better performance, e.g., without the usage of homomorphic encryption or reducing the number of exponentiations). We leave all above as future work.

Acknowledgement. Xiangxue Li is supported by National Natural Science Foundation of China (61971192), Shanghai Municipal Education Commission (2021-01-07–00-08-E00101), and Shanghai Trusted Industry Internet Software Collaborative Innovation Center.

References

1. Alhassan, M.Y., Günther, D., Kiss, Á., Schneider, T.: Efficient and scalable universal circuits. J. Cryptol. **33**(3), 1216–1271 (2020)
2. Applebaum, B., Harnik, D., Ishai, Y.: Semantic security under related-key attacks and applications. In: Innovations in Computer Science - ICS 2011, pp. 45–60 (2011)
3. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: ESORICS 2009

4. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 263–280. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_17
5. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: STOC (1990)
6. Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: 2013 IEEE Symposium on Security and Privacy, SP 2013, pp. 478–492 (2013)
7. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: STOC (1988)
8. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2–4, 1988, Chicago, Illinois, USA. pp. 1–10. ACM (1988). https://doi.org/10.1145/62212.62213
9. Bicer, O., Bingol, M.A., Kiraz, M.S., Levi, A.: Highly efficient and re-executable private function evaluation with linear complexity. IEEE Trans. Dependable Secure Comput. **19**(2), 835–847 (2020)
10. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: ACM CCS (2007)
11. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, pp. 315–334 (2018)
12. Demmler, D., Schneider, T., Zohner, M.: ABY - A framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
13. Evans, D., Kolesnikov, V., Rosulek, M.: A pragmatic introduction to secure multi-party computation. Found. Trends Priv. Secur. **2**(2–3), 70–246 (2018)
14. Frederiksen, T.K., Jakobsen, T.P., Nielsen, J.B., Nordholt, P.S., Orlandi, C.: MiniLEGO: Efficient Secure Two-Party Computation from General Assumptions. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 537–556. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_32
15. Frikken, K.B., Atallah, M.J., Li, J.: Attribute-based access control with hidden policies and hidden credentials. IEEE Trans. Comput. **55**(10), 1259–1270 (2006)
16. Frikken, K.B., Atallah, M.J., Zhang, C.: Privacy-preserving credit checking. In: EC (2005)
17. Furukawa, J., Sako, K.: An efficient scheme for proving a shuffle. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 368–387. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_22
18. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theor. **31**(4), 469–472 (1985)
19. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K.E., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: Balcan, M., Weinberger, K.Q. (eds.) Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19–24, 2016. JMLR Workshop and Conference Proceedings, vol. 48, pp. 201–210. JMLR.org (2016)
20. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: STOC, pp. 218–229 (1987)

21. Groth, J.: Linear algebra with sub-linear zero-knowledge arguments. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 192–208. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_12
22. Groth, J.: A verifiable secret shuffle of homomorphic encryptions. J. Cryptol. **23**(4), 546–579 (2010)
23. Groth, J., Ishai, Y.: Sub-linear zero-knowledge argument for correctness of a shuffle. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 379–396. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_22
24. Groth, J., Lu, S.: Verifiable shuffle of large size ciphertexts. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 377–392. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71677-8_25
25. Günther, D., Kiss, Á., Scheidel, L., Schneider, T.: Poster: Framework for semi-private function evaluation with application to secure insurance rate calculation. In: ACM CCS, pp. 2541–2543 (2019)
26. Günther, D., Kiss, Á., Schneider, T.: More efficient universal circuit constructions. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 443–470. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_16
27. Holz, M., Kiss, Á., Rathee, D., Schneider, T.: Linear-complexity private function evaluation is practical. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) ESORICS 2020. LNCS, vol. 12309, pp. 401–420. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59013-0_20
28. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: ACM CCS, pp.955–966 (2013)
29. Jia, H., Li, X.: Pfe: Linear active security, double-shuffle proofs, and low-complexity communication. Cryptology ePrint Archive, Report 2022/219 (2022)
30. Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 556–571. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25385-0_30
31. Kiss, Á., Schneider, T.: Valiant's universal circuit is practical. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 699–728. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_27
32. Kolesnikov, V.: Gate evaluation secret sharing and secure one-round two-party computation. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 136–155. Springer, Heidelberg (2005). https://doi.org/10.1007/11593447_8
33. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70583-3_40
34. Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85230-8_7
35. Lindell, Y.: Fast cut-and-choose-based protocols for malicious and covert adversaries. J. Cryptol. **29**(2), 456–490 (2015). https://doi.org/10.1007/s00145-015-9198-0
36. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_4

37. Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2008). https://doi.org/10.1007/s00145-008-9036-8
38. Lindell, Y., Pinkas, B.: A proof of security of yao's protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2009)
39. Lindell, Y., Riva, B.: Blazing fast 2pc in the offline/online setting with security for malicious adversaries. In: ACM CCS (2015)
40. Lipmaa, H., Mohassel, P., Sadeghian, S.: Valiant's universal circuit: Improvements, implementation, and applications, iACR Eprint 2016/017
41. Liu, H., Yu, Yu., Zhao, S., Zhang, J., Liu, W., Hu, Z.: Pushing the limits of valiant's universal circuits: simpler, tighter and more compact. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 365–394. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_13
42. Liu, Y., Wang, Q., Yiu, S.: Making private function evaluation safer, faster, and simpler. IACR Cryptol. ePrint Arch. p. 1682 (2021)
43. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: USENIX Security (2004)
44. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 19–38 (2017). https://doi.org/10.1109/SP.2017.12
45. Mohassel, P., Rindal, P.: Aby^3: A mixed protocol framework for machine learning. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15–19, 2018, pp. 35–52. ACM (2018). https://doi.org/10.1145/3243734.3243760
46. Mohassel, P., Sadeghian, S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 557–574. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_33
47. Mohassel, P., Sadeghian, S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 557–574. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_33
48. Mohassel, P., Sadeghian, S., Smart, N.P.: Actively secure private function evaluation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 486–505. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_26
49. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: EC (1999)
50. Neff, C.A.: A verifiable secret shuffle and its application to e-voting. In: CCS 2001, pp. 116–125 (2001)
51. Niksefat, S., Sadeghiyan, B., Mohassel, P., Sadeghian, S.S.: ZIDS: a privacy-preserving intrusion detection system using secure two-party computation protocols. Comput. J. **57**(4), 494–509 (2014)
52. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16
53. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_15

54. Shelat, A., Shen, C.: Two-output secure computation with malicious adversaries. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 386–405. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_22
55. Valiant, L.G.: Universal circuits (preliminary report). In: STOC (1976)
56. Wagh, S., Gupta, D., Chandran, N.: Securenn: 3-party secure computation for neural network training. Proc. Priv. Enhancing Technol. **2019**(3), 26–49 (2019)
57. Yao, A.C.: Protocols for secure computations. In: FOCS (1982)
58. Yao, A.C.C.: How to generate and exchange secrets. In: FOCS (1986)
59. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_8
60. Zhao, S., Yu, Yu., Zhang, J., Liu, H.: Valiant's universal circuits revisited: an overall improvement and a lower bound. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 401–425. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_15