

PFE: Linear Active Security, Double-Shuffle Proofs, and Low-Complexity Communication

Hanyu Jia, Xiangxue Li ^{***}

East China Normal University

Abstract. We consider private function evaluation (PFE) in malicious adversary model. Current state-of-the-art in PFE from Valiant’s universal circuits (Liu, Yu, et al., CRYPTO 2021) does not avoid the logarithmic factor in circuit size. In constructing linear active PFE, one essential building block is to prove the correctness of an extended permutation (EP, Mohassel and Sadeghian at EUROCRYPT 2013) by zero-knowledge protocols with linear complexity. The linear instantiation \mathcal{ZK}_{EP} by Mohassel, Sadeghian, and Smart (ASIACRYPT 2014) is a three-phase protocol, and each phase (dummy placement, replication, and permutation) is of size $2g$. Its overhead thus seems really outrageous, reducing its practicability. We present in this paper a novel and efficient framework \mathcal{ZK}_{DS} for proving the correct EP. We show that double shuffles suffice for EP (exponentiations and communication overheads are about 27% and 31% of \mathcal{ZK}_{EP} , respectively). Data owner(s) generates the randomness for the first shuffle whose outputs determine outgoing wires of the circuit defined by the function. Function owner reuses and extends the randomness in the second shuffle whose outputs determine the incoming wires.

From \mathcal{ZK}_{DS} , we build an online/offline PFE framework with linear active security. The online phase could be instantiated by any well-studied secure function evaluation (SFE) with linear active security (e.g., TinyOT at CRYPTO 2012). The offline phase depends only on the private function f and uses \mathcal{ZK}_{DS} to prove the EP relationship between outgoing wires and incoming wires in the circuit \mathcal{C}_f derived from f .

Private Function Evaluation, Active Security, Zero-Knowledge Proof, Extended Permutation, Shuffle.

1 Introduction

Secure Multi-Party Computation (MPC) protocols can provide privacy-preserving functionality [15, 54]. Through cryptographic methods, MPC protocols need no trusted third parties to ensure that remote computing can operate correctly while preserving the privacy of private data. One can find readily extensive and significant applications of MPC protocols in the era of increasing obsession with data insights [3, 8, 14, 17, 18, 26, 42, 49].

* Email: xxli@cs.ecnu.edu.cn

** Manuscript received April 19, 2021; revised August 16, 2021.

SFE AND PFE. There are two cases of MPC protocols according to the public/private function to be evaluated. Secure Function Evaluation (SFE) is well-established in MPC that allows two or more parties P_1, \dots, P_w to jointly compute a publicly known function f on private input data x_i of P_i ($i = 1, \dots, w$). SFE guarantees that each party can get the result $f(x_1, \dots, x_w)$ without revealing their private inputs x_i . Many SFE protocols have been proposed so far, and most of them rely on compiling the publicly known function into Boolean circuit, such as Yao’s Garbled Circuit [37,55], BGW [5], and GMW [21]. Private Function Evaluation (PFE) has a peculiar appearance where the function is a private input for one of the parties. In PFE, function owner compiles his private function into a circuit \mathcal{C}_f (boolean circuit or arithmetic circuit) and opens some parameters of the circuit to other parties [7,29,31,33,43,45], such as the number of gates (g), the number of input wires (u), the number of output wires (o), or some auxiliary parameters. These parameters of \mathcal{C}_f should not enable other parties to learn the function in polynomial time.

APPLICATION SCENARIOS OF PFE. There are various application scenarios for PFE, and we recall several examples here. In privacy-protected car insurance rate calculation [26], customer data and the details of rate calculation are kept private. In privacy-preserving intrusion detection systems [49], the server cannot learn the data of the client and the client cannot learn the signature on the server side. In credit check [18], it is required that neither private financial data of the customers nor private criteria of the lenders might be revealed. Attribute-based access control can be enhanced to protect both sensitive credentials and sensitive policies by using PFE [17]. There are also applications in medicine [3] and in software diagnostics [8].

GENERAL DESIGNS OF PFE. There are two main design approaches for PFE. One is to reduce PFE protocols to typical SFE protocols [1,27,32,33,39,40,52,56] by securely evaluating an open Universal Circuit (UC). This series of methods focus on optimizing the number of gates in the UC. The smaller the number of gates in the circuits, the smaller the total overhead of the protocols. Current state-of-the-art in PFE based on Valiant’s universal circuits [40] seems to reach a theoretical optimum $13g \cdot \log g$ (g is the number of the gates in the circuit), yet still does not avoid the logarithmic factor in circuit size. Despite the amazing versatility of universal circuits, independent techniques (free of the family of universal circuits) need to be investigated to create actively secure PFE with linear complexity. Mohassel and Sadeghian [43] propose a general framework under the semi-honest adversary model (see the next paragraph), which embraces the tasks of the hidden circuit (by *oblivious extended permutation of switching network* with complexity $\mathcal{O}(g \cdot \log g)$) and of securely evaluating its gates (by the *private gate evaluation*). Both tasks are handled independently and then combined naturally to form a PFE. In this line, we can also construct linear PFE by implementing extended permutation through *homomorphic encryption* in both two-party and multi-party settings [44]. For large circuits, this would be more efficient than UC-based methods. In addition to these two approaches for constructing general PFE, there are many purpose-specific PFE protocols, e.g.,

Katz and Malka [31] propose a two-party PFE combining Yao’s garbled circuit and a singly homomorphic encryption (HE).

SECURITY MODELS IN MPC. Security models in MPC include semi-honest adversary model and malicious adversary model. Semi-honest adversary (often referred to as honest-but-curious adversary) follows the specified steps, but tries to learn as much as possible from the messages sent by other parties. A malicious adversary (a.k.a. active adversary) can violate the specified steps at will. Namely, a malicious adversary has all the capabilities of semi-honest adversary and can take any action he wants during his execution in an attempt to learn more. In the paper, we are concerned particularly about the malicious adversary model. SFE protocols in the malicious model are well studied [16, 30, 34, 36, 38, 51], and they can be combined with UC to readily construct PFE in the malicious adversary model [1, 40]. We mention that there exists logarithmic factor in circuit size even in the optimal UC constructions, so do the PFE protocols derived in this way. Mohassel, Sadeghian and Smart [46] propose a novel framework (MSS framework) for PFE in the malicious adversary model. Therein the idea of switching network is used to construct extended permutation (EP) in [43] (see Definition 1), and other primitives include actively secure SFE, one-time MACs, and linear zero-knowledge proof (\mathcal{ZK}_{EP}) protocol of “correct extended permutation” of ElGamal ciphertexts. MSS framework has the advantage of linear complexity in circuit size (there are a host of exponentiations in the \mathcal{ZK}_{EP} , however). Although application scenarios of MPC protocols with active security are more realistic, their designing is more elaborate than those in the semi-honest adversary model. Fortunately, we do have several general PFE constructions against malicious adversaries (besides those based on UC).

1.1 Related Work

\mathcal{ZK}_{EP} WITH LINEAR COMPLEXITY IN CIRCUIT SIZE. \mathcal{ZK}_{EP} [46] was born to prove the correctness of the extended permutation of ElGamal ciphertexts, i.e., m ciphertexts are extended to n ciphertexts ($m \leq n$). Different with MSS framework characterized by its three-phase sequential shuffling, another line of proving EP is to design a direct construction of one module, and this is accomplished at PKC 2021 [41] where a specific honest-verifier zero-knowledge protocol is constructed. We mention that they are the only two solutions available to verify the correctness of EP. Our primary interest in the paper goes to the first line, i.e., MSS framework and the \mathcal{ZK}_{EP} protocol [46]. We thus minimize the structure of MSS framework by defining a variant of the framework (only double shuffles are used) and thereby present a zero-knowledge protocol for the variant that is much more efficient than \mathcal{ZK}_{EP} . Next, we recall \mathcal{ZK}_{EP} .

The solution by Mohassel, Sadeghian, and Smart [46] is to decouple \mathcal{ZK}_{EP} into three well-designed components. The original construction of EP is based on a switching network [33, 53]. In this case, to construct a scheme with linear overhead in circuit size, singly homomorphic encryption (instead of a switching network) is used to compute each component and re-randomize the ciphertexts.

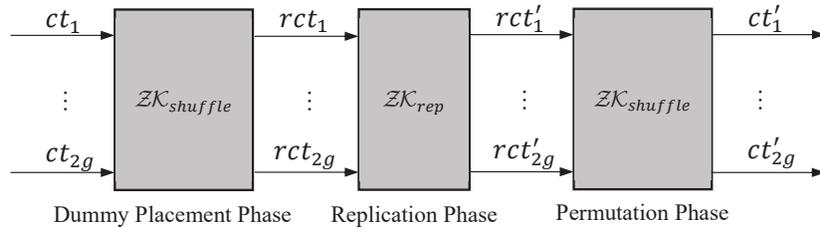


Fig. 1. \mathcal{ZK}_{EP} construction [46].

Both the first and the third components are shuffle operation. To hide the permutation relation, P_1 (prover) chooses uniformly at random a_i for each of the n ciphertexts provided by the verifiers for re-randomization (using HE property). Linear ZK proof of shuffle [19], $\mathcal{ZK}_{shuffle}$, is used to prove the correctness of these two components. The second component is a replication phase and it requires a separate ZK protocol (\mathcal{ZK}_{rep}) to justify the operation of the phase. The input ciphertexts ct_i of the first component include m real ciphertexts provided by the verifiers and $n - m$ identical dummy values encrypted with a homomorphic encryption scheme (where the private key is shared secretly among verifiers). The inputs of the second component are the outputs of the first component, and the inputs of the third component are the outputs of the second component. We illustrate \mathcal{ZK}_{EP} w.r.t. the parameters in the circuit \mathcal{C}_f , i.e., $m = g + u - o$ and $n = 2g$, usually $g \gg u$ and $u \approx o$. Fig. 1 shows \mathcal{ZK}_{EP} construction.

The first component (with n inputs and n outputs) is a shuffle operation. Its inputs contain n ElGamal ciphertexts fed by the verifiers, including m ciphertexts corresponding to real values and $n - m$ ciphertexts corresponding to the identical dummy values. Assume that some real ciphertext needs to be used k times in the circuit (i.e., it will be copied $k - 1$ times). Then in the outputs of the component, this real ciphertext would be immediately followed by $k - 1$ dummy ciphertexts. The output ciphertexts of the permutation are re-randomized, $rct_i = ct_{\pi(i)} + \text{Enc}_{pk}(a_i)$ (a_i is chosen uniformly at random), to hide the permuting relation in the component. This indeed gets a permutation and the component can be regarded as a shuffling of the input ciphertexts. Mohassel, Sadeghian, and Smart argue that the correctness of the first component can be verified by correct shuffle proof $\mathcal{ZK}_{shuffle}$ [19], which takes $18 * n$ exponentiations and the proof is of length about $5 * n * \|\mathbb{G}\|$ ($\|\mathbb{G}\|$ denotes the bit length of group element in \mathbb{G}) bits in proving the correctness.

The next one is a replication phase of the real ciphertexts, taking the outputs of the first component as its inputs. In the outputs of the component, each ciphertext corresponding to dummy value would be replaced directly by the last ciphertext (before it) corresponding to the real value. Naturally, all ciphertexts are re-randomized. To verify the correctness of the component, the protocol \mathcal{ZK}_{rep} is used. More precisely, \mathcal{ZK}_{rep} is equipped with three zero-knowledge protocols, two of which are \mathcal{ZK}_{eq} [10] and the third one is \mathcal{ZK}_{no} [28]. These two

\mathcal{ZK}_{eq} protocols are used to verify whether the i -th output of the component is equal to its i -th input ($2 \leq i \leq n$) or its $(i-1)$ -th output. For the first output result, we simply verify whether it is equal to the first input. \mathcal{ZK}_{no} is used to verify that each output ciphertext is not the dummy-valued ciphertext. We write $\mathcal{ZK}_{rep}(\{rct_i, rct'_i\})$ as below:

$$\mathcal{ZK}_{rep}^1 = \mathcal{ZK}_{no}(rct'_1) \wedge \mathcal{ZK}_{eq}(rct_1, rct'_1) : i = 1 \quad (1)$$

$$\mathcal{ZK}_{rep}^i = (\mathcal{ZK}_{eq}(rct_i, rct'_i) \vee \mathcal{ZK}_{eq}(rct'_{i-1}, rct'_i)) \wedge \mathcal{ZK}_{no}(rct'_i) : 2 \leq i \leq n \quad (2)$$

$$\mathcal{ZK}_{rep} = \wedge_{i=1, \dots, n} (\mathcal{ZK}_{rep}^i). \quad (3)$$

If Eq. (3) is correct, the component is justified. From [10] and [28], we have that the component takes about $69 * n$ exponentiations and the proof is of length about $16 * n * \|\mathbb{G}\|$ bits.

The last component is also a shuffle operation, taking the outputs of the second one as its inputs. It permutes the copied real ciphertexts to their appropriate positions. To hide the map, all input ciphertexts should be re-randomized. $\mathcal{ZK}_{shuffle}$ is used once more to verify the correctness of the component.

The protocol $\mathcal{ZK}_{shuffle}$ used in [45] was proposed by Furukawa and Sako [19], where the exponentiations required for shuffling n ciphertexts are about $18*n$ and the bit length of the proof is $5280 \cdot n + 13792$ (for $\|\mathbb{G}\| = 1024$ and $\|\mathbb{Z}_q\| = 160$). There exist some optimizations on $\mathcal{ZK}_{shuffle}$ so that the resulting proofs could be more efficient [4, 9, 19, 22–25, 47]. Zero-knowledge protocols in [19, 23, 25, 47] are available for verifying the correctness of the shuffles, all requiring proofs of length $\mathcal{O}(\eta)$ (η denotes the size of the shuffles, e.g., $\eta = 2g$ in MSS framework). The first sublinear shuffle protocol was proposed in [24] with a proof of length $\mathcal{O}(\eta^{2/3})$ which is further reduced to $\mathcal{O}(\eta^{1/2})$ in [4, 22]. It is showed [9] that the proof length would be $\mathcal{O}(\log \eta)$ if we want a protocol by sorting the circuits (based on discrete logarithm assumption). The numbers of the exponentiations required in above-mentioned work are all linear with η .

Remark. Replacing a better $\mathcal{ZK}_{shuffle}$ directly makes \mathcal{ZK}_{EP} less overhead. Similarly, replacing a better \mathcal{ZK}_{rep} could also make \mathcal{ZK}_{EP} more efficient. Rather than such trivial replacement, we attempt to optimize the proof framework of \mathcal{ZK}_{EP} . It is surely feasible that above-mentioned replacements can be taken in our proposed framework (if any).

1.2 Motivation and Contributions

This paper focuses on actively secure PFE. Our main observation is that there exist redundant components in the technique of [46] used to construct \mathcal{ZK}_{EP} , resulting in excessive exponentiations and communication overhead. In particular, \mathcal{ZK}_{EP} consists of a dummy placement phase, a replication phase, and a permutation phase, and each of the phases is of size $2g$. The overhead required by \mathcal{ZK}_{EP} thus seems really outrageous, reducing its practicability. If we could

remove the redundant components or reduce the size of the components, \mathcal{ZK}_{EP} would be improved in terms of less exponentiations and communication overhead, and the resulting actively secure PFE framework could thereby be more practical. We present in this paper a novel and efficient framework \mathcal{ZK}_{DS} to verify the correct EP of ElGamal ciphertexts. We show that double shuffles suffice in \mathcal{ZK}_{DS} : one shuffle is of size $u + g - o$, another is of size $2g$, and no replication phase is required. Note that the exponentiations and communications overhead of \mathcal{ZK}_{rep} in [43] account for 62% and 47% of \mathcal{ZK}_{EP} , respectively. Therein, u is the number of input wires in the circuit and o the number of output wires, and we have generally $u \approx o, g \gg u$ (e.g., a circuit might contain tens of thousands of or even millions of gates [2, 57]). Besides its succinct double-shuffle structure, \mathcal{ZK}_{DS} reduces nearly 69% of the communication overhead and 73% of the exponentiations (including the ciphertexts and corresponding proofs), compared to \mathcal{ZK}_{EP} when we use the same shuffle protocol as [46].

THE PROTOCOL \mathcal{ZK}_{DS} . In \mathcal{ZK}_{EP} , the essential trick is to verify the correctness of extending m ciphertexts into n ciphertexts using linear ZK protocols. It is already well-studied to prove the correct shuffle by a linear zero-knowledge protocol. Note that the input size of the permutation is equal to its output size in the shuffle. \mathcal{ZK}_{EP} takes extra $n - m$ dummy values to convert the EP problem into a permutation problem. This approach increases the input size of the permutation and requires a separate \mathcal{ZK}_{rep} protocol to justify the copy operation and the zero dummy value, which definitely introduces a lot of extra exponentiations. \mathcal{ZK}_{EP} is the first (and the only modularization-based) linear protocol that validates EP. If the input size of the permutation could be reduced and some ZK protocols could be removed, it will reduce significant computation and communication overhead, making the actively secure PFE more practical. For this purpose, we propose the design \mathcal{ZK}_{DS} .

One interesting trick behind \mathcal{ZK}_{DS} is that we let the prover initialize the protocol by sending a set $\mathbf{C} := \{c_1, \dots, c_m\}$ to the verifier. This tells the verifier that the i -th ciphertext (to be generated by the verifier and sent to the prover) would be used c_i times, $i = 1, \dots, m$. We argue that this special set \mathbf{C} does not leak the private knowledge of the prover. In fact, \mathbf{C} also appears in the second component of \mathcal{ZK}_{EP} (see Sect. 3 for more details). \mathcal{ZK}_{DS} consists of just two independent shuffles: one is of size m and another of size n . The prover only needs to prove to the verifier that the two shuffles are correct, i.e., he makes correct EPs. Succinct \mathcal{ZK}_{DS} of double-shuffle structure does not require the protocol \mathcal{ZK}_{rep} which is indispensable to \mathcal{ZK}_{EP} . Both \mathcal{ZK}_{DS} and \mathcal{ZK}_{EP} call shuffle operation twice. In the context of PFE from \mathcal{ZK}_{DS} and \mathcal{ZK}_{EP} : each of the two shuffles in \mathcal{ZK}_{EP} is of size $2g$; in \mathcal{ZK}_{DS} however, one shuffle is of size g and another is of size $2g$, which makes the double-shuffle structure of \mathcal{ZK}_{DS} 25% "smaller than" that in \mathcal{ZK}_{EP} . On the one hand, there are many ZK proofs (\mathcal{ZK}_{rep}) required in \mathcal{ZK}_{EP} but not required in \mathcal{ZK}_{DS} . On the other hand, the consumption of the ciphertexts communication in \mathcal{ZK}_{DS} is about a half of that in \mathcal{ZK}_{EP} . One might thereby obtain a more compact PFE from \mathcal{ZK}_{DS} (than from \mathcal{ZK}_{EP}). Sect. 3 presents detailed comparisons.

GENERAL FRAMEWORK OF LINEAR ACTIVELY SECURE PFE. Our linear-complexity framework for actively secure PFE is decoupled into two phases, online and offline, aiming at the players jointly computing $f(x_1, \dots, x_u)$. At the end of the framework, function owner cannot learn valid knowledge of the private input data, and data owner cannot learn valid knowledge of the private function. The offline phase is independent of the input data, but depends on the function. Function owner translates his private function f into a topological circuit \mathcal{C}_f before the protocol starts. There are $g + u - o$ outgoing wires and $2g$ incoming wires in \mathcal{C}_f . These two types of wires make an EP relationship and this is the exact knowledge that we should protect about the circuit (see Sect. 2 for details). The online phase could be instantiated by any well-studied SFE with linear active security [45]. The offline phase is a bit tricky, and it appears not easy to verify the EP relationship between outgoing wires and incoming wires through some primitive with less than or equal to linear complexity [45, 46]. If both online and offline phases have linear complexity in circuit size, we can then naturally get a linear PFE. \mathcal{ZK}_{EP} [45] seems to be a feasible solution to the tricky problem, but much involved and not practically efficient due to its relatively high overhead. Our \mathcal{ZK}_{DS} is a better candidate and the resulting PFE from \mathcal{ZK}_{DS} has the advantage of smaller communication and computation overhead. This will practically push actively secure PFE with linear complexity. Note that the correct EP protocol would be invoked two times in this general framework, meaning that more efficient ZK protocols for the correctness of EP are more desirable for practical PFE. We describe more details of the framework in Sect. 4.

2 Notations and Definitions

We use bold (lower-case or capital) letters to denote sets (e.g., \mathbf{ct} , \mathbf{C} , \mathbf{C}' , \mathbf{l} , etc) and standard letters (e.g., a_i , ct_i , c_i , ow_i , l_i , etc) for values or elements of a set. For a set \mathbf{D} , $|\mathbf{D}|$ denotes the size of the set, and we write $\mathbf{D} = \{D_i\}_{i=1}^{|\mathbf{D}|}$. $[a]$ (or $[\mathbf{a}]$) indicates that a (or \mathbf{a}) is shared by a secret sharing scheme. We use π to denote a map from a set to another set, e.g., i is a preimage and j is the corresponding image, then we have $j = \pi(i)$. The abstract diagram (see Fig. 2) could enable the readers to better understand the mapping relationship between incoming wires and outgoing wires. Hereafter we suppose P_1 to be the function owner (and P_1 can also keep input data x_1 in multi-party PFE).

We use a singly homomorphic encryption (Gen, Enc, Dec). We view its plaintext space as a cyclic group \mathbb{G} of prime order p . Let k be a system parameter and $r \leftarrow_R \{0, 1\}^k$ denote sampling r uniformly at random from $\{0, 1\}^k$. Given n the security parameter of the homomorphic encryption, Gen outputs a pair of public and private keys $(pk, sk) \leftarrow \text{Gen}(1^n)$. We have $\text{Dec}_{sk}(c_1 + c_2) = \text{Dec}_{sk}(c_1) + \text{Dec}_{sk}(c_2)$, given ciphertexts $c_1 = \text{Enc}_{pk}(m)$ and $c_2 = \text{Enc}_{pk}(r)$, m and $r \in \mathbb{G}$. For this kind of homomorphic property, some public key encryption schemes are available, such as ElGamal [20] and Paillier [50] etc. Therein, ElGamal encryption (based on the Diffie-Hellman assumption) can provide efficient implemen-

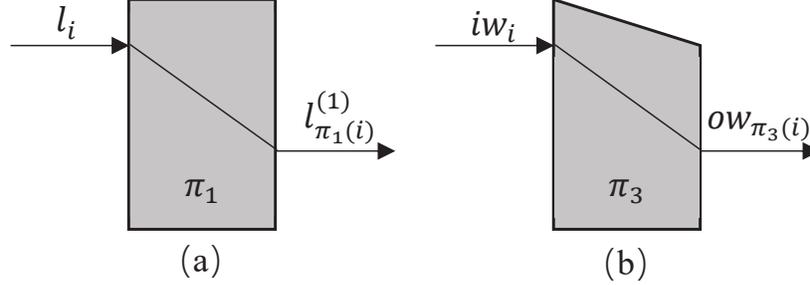


Fig. 2. l maps to $l^{(1)}$ and iw maps to ow . Two kinds of maps are used in our protocols: π_1 and π_3 . The former is a bijective function and the latter is surjective (not injective).

tation without decryption error and has been used in some known PFE [31] and GC designs [29]. Our protocols use ElGamal encryption as well.

In our protocols, function owner P_1 compiles his private function f into a Boolean or Arithmetic circuit \mathcal{C}_f with u inputs, g gates, and o outputs. We have usually $u \approx o$ and $g \gg u$. Let $N := u + g$. \mathcal{C}_f is like a directed acyclic graph in which the gates have topological order, namely, g gates has been numbered from smallest to largest by P_1 according to the topology of \mathcal{C}_f , denoted as $\{G_1, \dots, G_g\}$. If \mathcal{C}_f is an arithmetic circuit, it is composed of additive and multiplicative gates, with the bit value 0 for additive gate and 1 for multiplicative gate. If \mathcal{C}_f is a Boolean circuit, each of the gates is only a two fan-in NAND gate and its functionality does not need to be hidden. We divide all g gates in \mathcal{C}_f into output gates and non-output gates according to the destination of the output wires. We suppose w.l.o.g. that the last o gates $\{G_{g-o+1}, \dots, G_g\}$ in the gate sequence of the circuit are output gates and the first $g-o$ gates $\{G_1, \dots, G_{g-o}\}$ are non-output.

In this already topologically ordered Boolean circuit, we collect all the input wires of the circuit \mathcal{C}_f and all the outputs of the non-output gates and define the collection as a set of outgoing wires, denoted as ow . We have $|ow| = N - o$. Similarly, we get the set iw of incoming wires from all the input wires of all gates, and $|iw| = 2g$. $|ow| \leq |iw|$. Both ow and iw are topologically ordered sets for \mathcal{C}_f and correspond to the topologically ordered g gates. We suppose w.l.o.g. that $ow = \{ow_1, \dots, ow_{N-o}\}$ and $iw = \{iw_1, \dots, iw_{2g}\}$. We provide Fig. 3 for better reader-friendliness. To fully capture the topology of the circuit, we give each of the outgoing wires and incoming wires in the circuit a unique index. Each gate in \mathcal{C}_f is arbitrary fan-out and any outgoing wire ow_i could be used multiple times (and at least once), $i \in \{1, \dots, |ow|\}$. If ow_i is used c'_i times, we also say it has $c'_i - 1$ copies. As P_1 knows all the knowledge of \mathcal{C}_f , he can first generate a set $\mathcal{C}' = \{c'_1, \dots, c'_{|ow|}\}$, $\sum_{i=1}^{|ow|} c'_i = 2g$ and then perform a random permutation on \mathcal{C}' to get the set $\mathcal{C} = \{c_1, \dots, c_{|ow|}\}$. In other words, P_1 chooses a random map π such that $c'_i = c_{\pi(i)}$, $i = 1, \dots, |ow|$. π is P_1 's private knowledge. Data owner (say P_2) compiles his secret data x into binary form, i.e., $x = \{0,1\}^u$. P_1

sends P_2 u, g, o and C . This move does not leak the knowledge of C_f , which is hidden in the EP relationship between ow and iw in the circuit.

Definition [44] 1. We say the mapping $\pi_3^{-1} : \{1, \dots, |ow|\} \rightarrow \{1, \dots, |iw|\}$ is an extended permutation, since it can not only permute the elements in $\{1, \dots, |ow|\}$, but also replicate them as many times as needed (see Fig. 3).

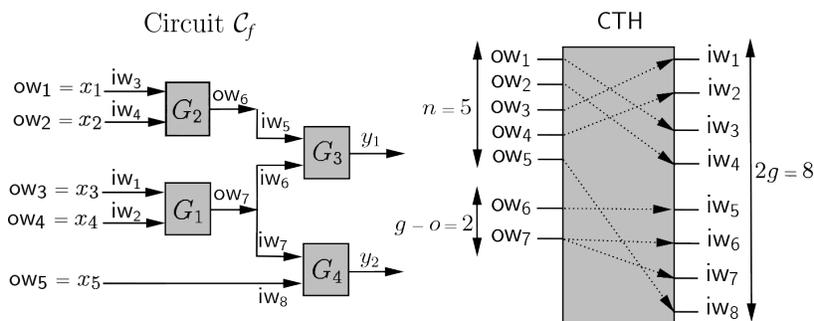


Fig. 3. An example circuit and the corresponding mapping π_3^{-1} [43]. The circuit topology hidden (CTH) indicates an extended permutation.

3 \mathcal{ZK}_{DS} : Double Shuffles Suffice for Correct EP

3.1 Observation

For the extended permutation relation of ow and iw in the circuit, Katz and Malka [31], and Mohassel and Sadeghian [43] present linear-complexity solutions in semi-honest adversary model by applying singly homomorphic encryption. Therein, function owner P_1 follows the protocol in an honest-but-curious manner to make a valid extended permutation of the $|ow|$ ciphertexts. In malicious adversary model however, P_1 may not follow the specification of the protocol when extending the permutation of the $|ow|$ ciphertexts, i.e., he does not perform the exact valid EP. Now we need some validation mechanism on P_1 's local computation (to make sure that P_1 has done the exact EP). For actively secure SFE, the cut-and-choose method is available for this kind of validation [35, 58]. For PFE however, it is not clear yet how to apply this method in checking EP's correctness [46]. The takeaway herein is that the EP is P_1 's private knowledge now and cannot be revealed to the verifier even in checking its correctness. As an open problem, more attempts need to be made in applying the cut-and-choose approach to PFE. In the literature, one (and the only modularization-based) feasible validation mechanism is the linear \mathcal{ZK}_{EP} proposed by Mohassel, Sadeghian, and Smart [46]. \mathcal{ZK}_{EP} consists of three components (see Fig. 1) that have been

well studied (with relatively high complexities of computation and communication however). We revisit \mathcal{ZK}_{EP} and propose a more efficient alternative \mathcal{ZK}_{DS} which could facilitate actively secure PFE as well.

In \mathcal{ZK}_{EP} , data owner (verifier) provides $|\mathbf{ow}|$ ciphertexts as inputs. Given these ciphertexts, function owner P_1 (prover) extends them to $|\mathbf{iw}|$ ciphertexts. In the first component (i.e., dummy placement phase) of \mathcal{ZK}_{EP} , P_1 promotes $|\mathbf{iw}| - |\mathbf{ow}| \approx g$ dummy-valued ciphertexts as a supplement and thus converts the EP into a permutation problem. This approach obviously increases the size of the input ciphertexts and requires an additional protocol (\mathcal{ZK}_{rep}) to ensure that P_1 has performed the correct copy work. The purpose of extending the $|\mathbf{ow}|$ ciphertexts to $|\mathbf{iw}|$ ciphertexts is to obtain the $2g$ input labels of topologically sorted g gates and maintain the relationship between the plaintexts corresponding to the $|\mathbf{iw}|$ ciphertexts and the plaintexts corresponding to the $|\mathbf{ow}|$ ciphertexts. To be more succinct, one heuristic motivation is to let P_1 tell data owner in advance how many times each of the $|\mathbf{ow}|$ ciphertexts would be copied, i.e., $\mathbf{C}' = \{c'_1, \dots, c'_{|\mathbf{ow}|}\}$. Now there is no need to verify the correctness of the copying phase, i.e., the \mathcal{ZK}_{rep} protocol could be removed. Unfortunately, this might also give away the valid knowledge of \mathcal{C}_f . For example, data owner knows how many times the $|\mathbf{ow}|$ -th ciphertext has been copied, and further knows where the output of the $(g - o)$ -th gate in the topological order goes, i.e., we cannot rule out the possibility of leaking the valid knowledge of \mathcal{C}_f . This is mainly due to the fact that the $|\mathbf{ow}|$ ciphertexts sent by data owner correspond one by one to the u inputs of the circuit and the $g - o$ outputs of the non-output gates (i.e., \mathbf{ow}). However, this gap can be fixed by making the requirement that there exist a random mapping relation between \mathbf{ow} and $|\mathbf{ow}|$ ciphertexts provided by data owner. I.e., we require that data owner provides $|\mathbf{ow}|$ ciphertexts according to $\mathbf{C} = \{c_1, \dots, c_{|\mathbf{ow}|}\}$, and this will remedy the flaw of leaking the knowledge of \mathcal{C}_f that occurs in the first try. Now there is also no need to introduce almost g additional dummy-valued ciphertexts to the first component of \mathcal{Z}_{EP} .

We mention that letting P_1 open the set \mathbf{C} and reveal to data owner does not sacrifice the security of \mathcal{ZK}_{DS} . A simple rethinking of \mathcal{ZK}_{EP} may find that it also reveals the set \mathbf{C} . In the replication phase of \mathcal{ZK}_{EP} , P_1 uses the \mathcal{ZK}_{rep} protocol to prove the correctness of his replication work. \mathcal{ZK}_{rep} contains three zero-knowledge protocols, two of which are \mathcal{ZK}_{eq} [10] and the last one is \mathcal{ZK}_{no} [28]. We have the detailed description in Sect. 1.1. In this component, data owner can learn how many times a particular ciphertext among the $g + u - o$ shuffled ciphertexts has been copied. This knowledge does not reveal the topology of the circuit, as data owner cannot learn the index of the ciphertext corresponding to the input ciphertext of the first component. In addition, data owner cannot know to which final location the ciphertext is mapped in the next component.

To ease the understanding, Table 1 summarizes some symbols representing ciphertexts and plaintexts/random values. For the last 6 lines in Table 1, similar to l_i in \mathbf{l} , we have $ctp_i^{(1)}$, $l_i^{(1)}$, $ct_i^{(2)}$, l'_i , $ctp_i^{(2)}$, and $l_i^{(2)}$ in $\mathbf{ctp}^{(1)}$, $\mathbf{l}^{(1)}$, $\mathbf{ct}^{(2)}$, \mathbf{l}' , $\mathbf{ctp}^{(2)}$, and $\mathbf{l}^{(2)}$ respectively, and we omit these symbols from the table. Other symbols can be defined in the same way and would be used in Sect. 4.

Table 1. Symbols.

<i>Symbols</i>	<i>Annotations</i>
l_i	random value generated by the verifier
\mathbf{l}	a set collected from random values l_i
$ct_i^{(1)}$	ciphertext of plaintext l_i
$\mathbf{ct}^{(1)}$	ciphertext set corresponding to plaintext set \mathbf{l}
$\mathbf{ctp}^{(1)}$	ciphertext set obtained from permutation & re-randomization on $\mathbf{ct}^{(1)}$
$\mathbf{l}^{(1)}$	plaintext set corresponding to ciphertext set $\mathbf{ctp}^{(1)}$
$\mathbf{ct}^{(2)}$	ciphertext set extended from $\mathbf{ct}^{(1)}$
\mathbf{l}	plaintext set corresponding to ciphertext set $\mathbf{ct}^{(2)}$
$\mathbf{ctp}^{(2)}$	ciphertext set obtained from permutation & re-randomization on $\mathbf{ct}^{(2)}$
$\mathbf{l}^{(2)}$	plaintext set corresponding to ciphertext set $\mathbf{ctp}^{(2)}$

3.2 Constructing \mathcal{ZK}_{DS}

Above-mentioned observations would lead to a particular design that differs from \mathcal{ZK}_{EP} . In \mathcal{ZK}_{EP} , P_1 privately compiles the function f into \mathcal{C}_f from which the mapping relation of the extended permutation is extracted. In our design, what P_1 needs to extract include the mapping relationship between the $|\mathbf{ow}|$ ciphertexts sent by the data owner and \mathbf{ow} , and that between the $|\mathbf{iw}|$ ciphertexts (after copying the $|\mathbf{ow}|$ ciphertexts) and \mathbf{iw} , represented by the maps π_1 and π_2 , respectively. Both π_1 and π_2 are bijective.

Our design \mathcal{ZK}_{DS} only includes one permutation of size $N - o$ and one permutation of size $2g$. There is no need to perform \mathcal{ZK}_{rep} or to generate the ciphertexts with almost g dummy values. P_1 should re-randomize the ciphertexts he received in the interactive session to hide the knowledge of maps (i.e., π_1 and π_2 below). We give an abstract description in Fig. 4. One can see from Fig. 4 that \mathcal{ZK}_{DS} is decoupled into two phases: the component of randomness-generating & outgoing-wires-determining (RG&OWD), and the component of randomness-reusing & incoming-wires-determining (RR&IWD). Both components can be proved correct by using $\mathcal{ZK}_{shuffle}$. We use π_1 to denote the permutation mapping in $\mathcal{ZK}_{shuffle_1}$, and π_2 to denote that in $\mathcal{ZK}_{shuffle_2}$. To facilitate the comparison with \mathcal{ZK}_{EP} , we also apply the scheme for proving a shuffle in [19]. We mention that the scheme [19] hereof is only for the purpose as an instantiation example of our building block and any efficient scheme (say, with linear computation complexity and logarithmic communication complexity [9]) of this kind is a good candidate for our protocol. Table 2 lists the zero-knowledge protocol for the black-boxes of Fig. 4 used in \mathcal{ZK}_{DS} .

Our aim is to turn the $|\mathbf{ow}|$ ciphertexts corresponding to the set of random values $\mathbf{l}^{(1)}$ into the $|\mathbf{iw}|$ ciphertexts corresponding to the set $\mathbf{l}^{(2)}$ by EP. In both components, these ciphertexts are then re-randomized to hide the knowledge of permutation relation, and the final ciphertexts are later open to the verifier.

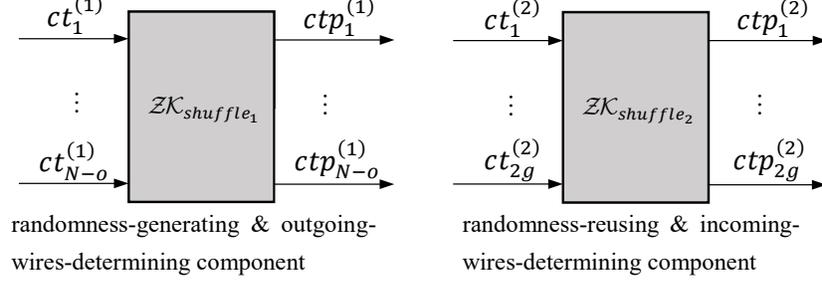


Fig. 4. \mathcal{ZK}_{DS} construction consisting of two $\mathcal{ZK}_{shuffle}$ protocols.

We use the set $\mathcal{C} = \{c_1, \dots, c_{|\mathbf{ow}|}\}$, $\sum_{i=1}^{|\mathbf{ow}|} c_i = 2g$. We take \mathcal{C} as one of system inputs of P_1 , i.e., P_1 sends \mathcal{C} to the verifier to initialize the interaction. Additional information about \mathcal{C}_f that can be disclosed is g , u and o . The verifier then generates a set \mathbf{l} of random values ($|\mathbf{l}| = |\mathbf{ow}|$) and a public/private key pair (pk, sk) of ElGamal encryption. The verifier encrypts each l_i in \mathbf{l} with pk and obtains the ciphertext set $\mathbf{ct}^{(1)}$, $ct_i^{(1)} = \text{Enc}_{pk}(l_i)$. \mathcal{ZK}_{DS} consists of two parts: $\mathcal{ZK}_{shuffle_1}$ with $|\mathbf{ow}|$ inputs and $\mathcal{ZK}_{shuffle_2}$ with $|\mathbf{iw}|$ inputs. The verifier sends the set of ciphertexts $\mathbf{ct}^{(1)}$ and pk to P_1 . The set $\mathbf{ct}^{(1)}$ is directly used as the input of the first component. Next, we describe the two components in detail.

3.2.1 RG&OWD

Once the verifier gets the sets $\mathbf{l}^{(1)}$ ($|\mathbf{l}^{(1)}| = N - o$) and $\mathbf{l}^{(2)}$ ($|\mathbf{l}^{(2)}| = 2g$) [46], he would easily compute on each of the g gates determined by incoming wires and outgoing wires. If $\mathbf{l}^{(1)}$ is generated by the verifier himself (as in [46]), then it seems that a replication phase is necessary. Unlike prior trick, we let $\mathbf{l}^{(1)}$ be jointly decided by the verifier and the prover. As the verifier wants P_1 to extend and permute the set $\mathbf{l}^{(1)}$ into the set $\mathbf{l}^{(2)}$ as well as prevents P_1 from learning $\mathbf{l}^{(1)}$, we equip the prover with the capability of generating the encryption $ctp_i^{(1)} = \text{Enc}_{pk}(l_i^{(1)})$ even without the exact knowledge $l_i^{(1)}$. Instead of generating the set $\mathbf{l}^{(1)}$ directly, the verifier generates a set of $\mathbf{l} = \{l_i\}_{i=1}^{N-o}$, and encrypts them (under pk) into a set of ciphertexts $\mathbf{ct}^{(1)} = \{ct_i^{(1)}\}_{i=1}^{N-o}$, where $ct_i^{(1)} = \text{Enc}_{pk}(l_i)$. The set \mathbf{l} has a shuffle (and re-randomization) relation with the set $\mathbf{l}^{(1)}$ (to be determined by P_1). The verifier sends the set $\mathbf{ct}^{(1)}$ to P_1 . P_1 takes the set of ciphertexts $\mathbf{ct}^{(1)}$ as the input to this component, and the size of the input is

Table 2. Zero-knowledge protocol in \mathcal{ZK}_{DS} . Generator g , public key $h = g^{sk}$, $ct_i = (\alpha_i, \beta_i)$ and $ctp_i = (\alpha'_i, \beta'_i)$.

ZK Protocol	Relation/Language	Ref.
$\mathcal{ZK}_{shuffle}(\mathbf{ct}, \mathbf{ctp})$	$\mathcal{R}_{shuffle} = \{(G, g, h, \mathbf{ct}, \mathbf{ctp}) \exists \pi, \text{ s.t.}$ $\alpha'_i = g^{r_i} \alpha_{\pi(i)} \wedge \beta'_i = h^{r_i} \beta_{\pi(i)} \wedge \pi \text{ is perm.}\}$	[19]

$|\mathbf{ow}|$. The i -th value c_i in the set \mathbf{C} says that the i -th ciphertext in the set $\mathbf{ct}^{(1)}$ would be used c_i times. We have thereby that there are c_i ciphertexts in the following set $\mathbf{ctp}^{(2)}$ corresponding to the same plaintext l_i , but due to the shuffle, the verifier does not know which specific plaintexts in $\mathbf{l}^{(1)}$ correspond to l_i . The probability that the verifier correctly guesses this shuffle relationship is negligible. We denote the permutation of this component by π_1 . P_1 knows the topology of the circuit and gets the set $\mathbf{ctp}^{(1)}$ by permuting the set $\mathbf{ct}^{(1)}$ according to π_1 . To hide the knowledge of π_1 , P_1 also does a re-randomization $\text{Enc}_{pk}(l_i^{(1)}) = \text{Enc}_{pk}(l_{\pi_1^{-1}(i)} + a_i)$ (a_i is chosen uniformly at random). All the steps of permutation and re-randomization are performed locally by P_1 , who needs to prove the correctness of his local operations to the verifier. The RG&OWD component can be seen as a shuffle of the input ciphertexts, and we use the $\mathcal{ZK}_{shuffle}$ protocol as a black box to justify this component [46].

3.2.2 RR&IWD

In the previous component, the verifier produces the set of ciphertexts $\mathbf{ct}^{(1)}$. In the current component, the ciphertexts in $\mathbf{ct}^{(1)}$ sent to P_1 in the first component are reused. In particular, P_1 copies $\mathbf{ct}^{(1)}$ to produce a new set $\mathbf{ct}^{(2)}$ of ciphertexts according to the values in the set \mathbf{C} . We take as an example the i -th ciphertext $ct_i^{(1)}$ in the set $\mathbf{ct}^{(1)}$ which corresponds to the value c_i in \mathbf{C} . Then in the inputs of this component, the subsequent $c_i - 1$ ciphertexts after this ciphertext are all set as the exact $ct_i^{(1)}$. Since the sum of the values in \mathbf{C} is $2g$, we have exactly $2g$ ciphertexts after the copy step. The verifier also knows \mathbf{C} and the set $\mathbf{ct}^{(1)}$ is generated by himself, thus one may view that the input of the $2g$ ciphertexts in the second component are also provided by the verifier (but without the cost of communication). We use \mathbf{l}' to denote the plaintext set corresponding to the ciphertext set $\mathbf{ct}^{(2)}$, i.e., $ct_i^{(2)} = \text{Enc}_{pk}(l'_i)$, $1 \leq i \leq 2g$. P_1 knows how many times each ciphertext in $\mathbf{ct}^{(1)}$ has been copied separately and knows the topology of the circuit, so he can permute them to get the ciphertext set corresponding to $\mathbf{l}^{(2)}$. We denote the permutation map of this component by π_2 . Naturally, all permuted ciphertexts are re-randomized: $\text{Enc}_{pk}(l_i^{(2)}) = \text{Enc}_{pk}(l'_{\pi_2^{-1}(i)} + b_i)$, where b_i s are chosen uniformly at random. We write the result of the second component by the permutation and re-randomization as $\mathbf{ctp}^{(2)} = \{ctp_i^{(2)}\}_{i=1}^{2g}$, where $ctp_i^{(2)} = \text{Enc}_{pk}(l_i^{(2)})$. Again, the permutation and re-randomization is performed locally by P_1 , who needs to prove the correctness of his local operations to the verifier. This RR&IWD component can be seen as a shuffle of $2g$ input ciphertexts and the correctness of this phase can be checked via $\mathcal{ZK}_{shuffle}$.

3.2.3 Performance Improvement

\mathcal{ZK}_{DS} needs only two independent shuffles in the two components of RG&OWD and RR&IWD. In contrast, \mathcal{ZK}_{EP} requires three components, i.e., dummy placement phase, replication phase, and permutation phase, the first and third of which are also shuffle structure. We first compare only the two shuffles in \mathcal{ZK}_{DS} and \mathcal{ZK}_{EP} . \mathcal{ZK}_{DS} has "smaller" shuffle than \mathcal{ZK}_{EP} . In fact, the sizes

of the shuffles in \mathcal{ZK}_{DS} are g and $2g$, respectively. However, each of the sizes is $2g$ in \mathcal{ZK}_{EP} . Therefore, \mathcal{ZK}_{DS} saves approximately 25% of the size over \mathcal{ZK}_{EP} in the double shuffles. In addition, \mathcal{ZK}_{DS} does not use the replication phase which is indispensable in \mathcal{ZK}_{EP} . Table 3 lists the input/output sizes for each component of \mathcal{ZK}_{EP} and \mathcal{ZK}_{DS} . For more explicit efficiency comparison with \mathcal{ZK}_{EP} , we also use the scheme of [19] here (same as that in [46]).

Table 3. Comparison of the size of each component between \mathcal{ZK}_{EP} and \mathcal{ZK}_{DS} . Usually $g \gg u$ and $u \approx o$.

ZK Protocol	MSS \mathcal{ZK}_{EP} [46]	Our \mathcal{ZK}_{DS}
$\mathcal{ZK}_{shuffle}$	$2g$	$u+g-o$
\mathcal{ZK}_{rep}	$2g$	\times
$\mathcal{ZK}_{shuffle}$	$2g$	$2g$

Consider the computation overhead. For the RG&OWD component in \mathcal{ZK}_{DS} , the $\mathcal{ZK}_{shuffle_1}$ protocol takes $|\mathbf{ow}| * 18$ (i.e., $(N - o) * 18$) exponentiations; for the RR&IWD component, the $\mathcal{ZK}_{shuffle_2}$ protocol takes $|\mathbf{iw}| * 18$ (i.e., $2g * 18$) exponentiations. Comparatively, each size of the shuffles in \mathcal{ZK}_{EP} is $2g$ and thus a total of $2 * 2g * 18$ exponentiations are required. In addition, the overhead of the replication phase includes $2g$ re-randomizations and the \mathcal{ZK}_{rep} protocol. The computation overhead of $2g$ re-randomizations consists of $2g$ encryptions of random values and $2g$ homomorphic addition operations. The \mathcal{ZK}_{rep} protocol is described in Eq. (3) and further includes the \mathcal{ZK}_{no} protocol [28] and the \mathcal{ZK}_{eq} protocol [10]. It turns out that at least $2g * 69$ exponentiations are required for Eq. (3). For total computational cost, the exponentiations required for \mathcal{ZK}_{EP} and \mathcal{ZK}_{DS} are about $222g$ [41] and $60g$, respectively. Thus \mathcal{ZK}_{DS} saves approximately 73% of exponentiations compared to \mathcal{ZK}_{EP} . We mention that there are different proposals for \mathcal{ZK}_{rep} with different overheads. It is surely not a surprise that more efficient $\mathcal{ZK}_{shuffle}$ will make \mathcal{ZK}_{DS} even better performance. E.g., the protocol of [4] is reported to be $3.4\times$ faster than that of [19]. To sum up, \mathcal{ZK}_{DS} shows great advantages over \mathcal{ZK}_{EP} in terms of computation overhead.

Consider the communication overhead. \mathcal{ZK}_{DS} gains communication advantages over \mathcal{ZK}_{EP} from three facets. The first discount is the proof communication about the $\mathcal{ZK}_{shuffle}$ protocol of approximate g ciphertexts. The second lessening is the proof communication of the \mathcal{ZK}_{rep} protocol. We also have the reduction of almost $4g$ ciphertexts communication (only 50% of \mathcal{ZK}_{EP}). Note that the proof bits are generally longer than the ciphertexts themselves. In a nutshell, \mathcal{ZK}_{DS} reduces the communication bits by almost 69%, compared to \mathcal{ZK}_{EP} . Take $|\mathbb{Z}_q| = 160$ and $|\mathbb{G}| = 1024$ [4, 19] as example. We have the bits (including the ciphertexts) of $24032 \cdot g$ in \mathcal{ZK}_{DS} and $76672 \cdot g$ in \mathcal{ZK}_{EP} .

Liu et al. [41] present a specific protocol for proving that ElGamal ciphertext list is derived from an extended permutation over a given list of elements, which

requires $\mathcal{O}(g)$ exponentiations and communication overhead of $\mathcal{O}(\log g)$. Bünz et al. [9] design a verifiable shuffle protocol with exponentiations and communication overheads of $\mathcal{O}(g)$ and $\mathcal{O}(\log g)$, respectively. As our \mathcal{ZK}_{DS} framework has only succinct double-shuffle, Bünz’s shuffle protocol will result in \mathcal{ZK}_{DS} having the same performance as Liu’s protocol. We emphasize that the \mathcal{ZK}_{DS} framework for proving the correctness of EP is more flexible.

3.2.4 Specification

The real inputs of the verifier in \mathcal{ZK}_{DS} are the ElGamal ciphertexts corresponding to \mathbf{l} of length $|\mathbf{ow}|$, and in addition he knows \mathbf{C} . The prover first applies one shuffle to the ciphertexts $\{ct_1^{(1)}, \dots, ct_{|\mathbf{ow}|}^{(1)}\}$ and another shuffle to the ciphertexts $\{ct_1^{(2)}, \dots, ct_{|\mathbf{iw}|}^{(2)}\}$, where $ct_i = (\alpha_i, \beta_i)$ (of ElGamal encryption). The prover obtains two sets of re-randomized ciphertexts as $\{ctp_1^{(1)}, \dots, ctp_{|\mathbf{ow}|}^{(1)}\}$ and $\{ctp_1^{(2)}, \dots, ctp_{|\mathbf{iw}|}^{(2)}\}$, where $ctp_i = (\alpha'_i, \beta'_i)$. The prover needs only two correct shuffle proofs to show the correctness of his local computations. Table 4 shows the full description of \mathcal{ZK}_{DS} .

Theorem 1. The set \mathbf{C} does not give away valid knowledge about the circuit \mathcal{C}_f .

Proof. Private function f is compiled by P_1 into circuit \mathcal{C}_f whose each outgoing wire would be used at least once so that $2g$ incoming wires could be obtained. c'_i denotes the times that i -th ciphertext about \mathbf{ow} (generated by verifier and sent to P_1) would be used in shuffles and \mathbf{C}' is the collection of all c'_i . Rather than sending out \mathbf{C}' itself, we require random permutation on \mathbf{C}' and thus get \mathbf{C} . Protocol interactions can still work well. The takeaway is that random permutation is P_1 ’s private knowledge and no valid knowledge of \mathcal{C}_f might be derived. This is the only role of \mathbf{C} in \mathcal{ZK}_{DS} . On other hand, both RG&OWD and RR&IWD count on permutation and re-randomization, making verifier incapable of inferring mapping relationship between $\mathbf{l}^{(1)}$ and $\mathbf{l}^{(2)}$. This completes the proof.

We employ the techniques of Cramer et al. [11], to combine honest verifier zero-knowledge (HVZK) proof systems corresponding to each component, at no extra cost, into HVZK proof systems of the same class [45].

Theorem 2. The protocol described in Table 4 is HVZK proof of the extended permutation.

Proof. Below is a description of the proof. We describe our \mathcal{ZK}_{DS} protocol as an HVZK proof of the extended permutation if our construction can achieve the extended permutation effect. In our framework, the verifier is given the \mathbf{C} . The outputs $\mathbf{ctp}^{(1)}$ of the RG&OWD component represent the knowledge about \mathbf{ow} and the outputs $\mathbf{ctp}^{(2)}$ of the RR&IWD component represent the knowledge about \mathbf{iw} . The verifier receives $\mathbf{ctp}^{(1)}$ and $\mathbf{ctp}^{(2)}$ and can decrypt them to recover their plaintexts ($\mathbf{l}^{(1)}$ and $\mathbf{l}^{(2)}$). The \mathbf{ow} to \mathbf{iw} conversion is the mapping knowledge of the extended permutation, i.e., $\mathbf{l}^{(1)}$ to $\mathbf{l}^{(2)}$ conversion is the extended permutation relationship. Due to the property of homomorphic

encryption, P_1 does not see $l^{(1)}$ and $l^{(2)}$. Due to two different permutations and re-randomization, the verifier does not know the mapping relationship between the set $l^{(1)}$ and the set $l^{(2)}$. To summarize, our solution achieves the EP effect. Once both the RG&OWD component and the RR&IWD component pass the verification, one can believe that the prover performs a valid EP, and also a valid circuit topology. The proofs of above two components make up the proof of \mathcal{ZK}_{DS} protocol. Finally we employ the techniques of Cramer et al. [11], to combine HVZK proof systems corresponding to each component, at no extra cost, into HVZK proof systems of the same class. Note that we make a black-box call to the underlying ZK proof system.

4 General PFE Framework with Linear Active Security

This section describes a general actively secure PFE framework with linear complexity in circuit size [45]. \mathcal{ZK}_{DS} can be of significance in this context. We suppose u parties whose joint task is to compute $f(x_1, \dots, x_u)$. In many PFE application scenarios, function owner is not the recipient of final computed result. In malicious adversary model, whether PFE is robust or with abort is related to underlying SFE [6, 12, 13, 48].

4.1 High-level Description

Our framework is not based on universal circuit, and consists of an offline phase and an online phase. The offline phase is independent of data owners' private data, but depends on function owner's private function. This framework can turn any actively secure SFE (with the following features) into an actively secure PFE. The features of the underlying SFE include: its construction is based on secret sharing, it is actively secure (either robust or with aborts), and it has the abilities of implementing reactive functionalities, of opening various shares securely, and of efficiently generating random values for sharing. Some candidate SFE protocols include BDOZ [6], SPDZ [13], Tiny-OT [48] or VIFF [12]. We do not specify which specific SFE protocol to use, and as long as above conditions are met, it can be turned into an actively secure PFE through our framework.

In our PFE, P_1 is the function owner who privately translates the function f as a topological circuit \mathcal{C}_f . Each data owner provides his private data as input. The online phase in our framework is linear in circuit size and it can be implemented by actively secure SFE that satisfies above conditions. If the offline phase is also linear in circuit size, then we obtain a linear-complexity, actively secure PFE. The tricky problem herein is how to extend and permute two sets of random values of length $|ow|$ ($[l^{(1)}]$ and $[t^{(1)}]$) into two sets of random values of length $|iw|$ ($[l^{(2)}]$ and $[t^{(2)}]$) using a less than or equal to linear method in the offline phase. $[l^{(1)}]$ and $[t^{(1)}]$ are two sets of shared random values generated by data owners through the underlying SFE, where those secret shares cannot be learned by P_1 . Data owners then jointly encrypt the two sets of shared values using a singly homomorphic encryption, and P_1 transforms the resulting ciphertext sets ($ctp^{(1)}$ and $ctp'^{(1)}$) into two sets of ciphertext ($ctp^{(2)}$ and $ctp'^{(2)}$)

Table 4. The protocol for zero-knowledge proof of \mathcal{ZK}_{DS}

Protocol \mathcal{ZK}_{DS}
<p>Input of the Verifier (P_2): Ciphertext set $\mathbf{ct}^{(1)} = \{ct_1^{(1)}, \dots, ct_{ \mathbf{ow} }^{(1)}\}$.</p> <p>Input of the Prover (P_1): Permutation maps π_1, π_2 and set \mathbf{C} (random permutation of \mathbf{C}').</p>
<p>P_1 sends \mathbf{C}, u, g and o to P_2.</p> <p>P_2 sends the ciphertext set $\mathbf{ct}^{(1)} = \{ct_1^{(1)}, \dots, ct_{ \mathbf{ow} }^{(1)}\}$ and \mathbf{pk} to P_1.</p> <p>P_1 evaluates the components.</p> <ul style="list-style-type: none"> – P_1 finds corresponding permutation π_1 for RG&OWD component and π_2 for RR&IWD component, respectively. – P_1 produces ciphertexts $\mathbf{ct}^{(2)} = \{ct_1^{(2)}, \dots, ct_{ \mathbf{iw} }^{(2)}\}$ from ciphertexts $\mathbf{ct}^{(1)} = \{ct_1^{(1)}, \dots, ct_{ \mathbf{ow} }^{(1)}\}$ according to \mathbf{C}. – P_1 applies the RG&OWD component to $\{ct_1^{(1)}, \dots, ct_{ \mathbf{ow} }^{(1)}\}$, and finds $\mathbf{ctp}^{(1)} = \{ctp_1^{(1)}, \dots, ctp_{ \mathbf{ow} }^{(1)}\}$. – P_1 applies the RR&IWD component to $\{ct_1^{(2)}, \dots, ct_{ \mathbf{iw} }^{(2)}\}$, and finds $\mathbf{ctp}^{(2)} = \{ctp_1^{(2)}, \dots, ctp_{ \mathbf{iw} }^{(2)}\}$. <p>$P_1$ computes the ZK proofs and sends out the outputs of two components and the proofs.</p> <ul style="list-style-type: none"> – $\mathcal{ZK}_{shuffl_{e_1}}(\mathbf{ct}^{(1)}, \mathbf{ctp}^{(1)})$, $\mathcal{ZK}_{shuffl_{e_2}}(\mathbf{ct}^{(2)}, \mathbf{ctp}^{(2)})$ are used by P_1 to produce proof of correctness for his evaluation in two components. – P_1 sends $\{ctp_1^{(1)}, \dots, ctp_{ \mathbf{ow} }^{(1)}\}$, $\{ctp_1^{(2)}, \dots, ctp_{ \mathbf{iw} }^{(2)}\}$ and all proofs to P_2. <p>P_2 verifies P_1's computations.</p> <ul style="list-style-type: none"> – P_2 checks the proofs sent by P_1.

by applying extended permutation and re-randomization. Data owners jointly decrypt $\mathbf{ctp}^{(2)}$ and $\mathbf{ctp}'^{(2)}$ to obtain shares of the resulting plaintexts.

In our scheme, instead of generating $[\mathbf{l}^{(1)}]$ and $[\mathbf{t}^{(1)}]$ directly, data owners generate two sets of shared random values $[\mathbf{l}]$ and $[\mathbf{t}]$ of length $|\mathbf{ow}|$ through the underlying SFE. P_1 helps data owners to evaluate $[\mathbf{l}]$ and $[\mathbf{t}]$ into $[\mathbf{l}^{(1)}]$ and $[\mathbf{t}^{(1)}]$, respectively. P_1 knows the topology of \mathcal{C}_f and can apply the extended permutation to turn $[\mathbf{l}^{(1)}]$ and $[\mathbf{t}^{(1)}]$ into $[\mathbf{l}^{(2)}]$ and $[\mathbf{t}^{(2)}]$, respectively. In order to obtain active security of the offline phase, the following three steps are required to be actively secure.

1. Two sets of random values $[\mathbf{l}]$ and $[\mathbf{t}]$ are shared among data owners, and they need to be encrypted jointly by the parties through the underlying SFE to obtain the ciphertext sets $\mathbf{ct}^{(1)}$ and $\mathbf{ct}'^{(1)}$, i.e., $ct_i^{(1)} = \text{Enc}_{pk}(l_i)$ and $ct_i'^{(1)} = \text{Enc}_{pk}(t_i)$. The encryption scheme herein needs to be singly homomorphic

(e.g., ElGamal encryption), where the secret key is secretly shared among the parties. The resulting ciphertexts are sent to P_1 .

2. P_1 applies \mathcal{ZK}_{DS} to convert the ciphertexts $\mathbf{ct}^{(1)}$ (resp., $\mathbf{ct}'^{(1)}$) into $\mathbf{ctp}^{(1)}$ (resp., $\mathbf{ctp}'^{(1)}$) and $\mathbf{ctp}^{(2)}$ (resp., $\mathbf{ctp}'^{(2)}$), respectively. P_1 sends them to data owners where $ctp_i^{(1)} = \text{Enc}_{pk}(l_i^{(1)})$, $ctp_i^{(2)} = \text{Enc}_{pk}(l_i^{(2)})$, $ctp_i'^{(1)} = \text{Enc}_{pk}(t_i^{(1)})$ and $ctp_i'^{(2)} = \text{Enc}_{pk}(t_i^{(2)})$. \mathcal{ZK}_{DS} proves that P_1 's local work is correct.
3. Data owners jointly decrypt the two sets of ciphertexts $\mathbf{ctp}^{(1)}$ (resp., $\mathbf{ctp}'^{(1)}$) and $\mathbf{ctp}^{(2)}$ (resp., $\mathbf{ctp}'^{(2)}$) by the underlying SFE and finally get the shares of plaintexts $[\mathbf{l}^{(1)}]$ (resp., $[\mathbf{t}^{(1)}]$) and $[\mathbf{l}^{(2)}]$ (resp., $[\mathbf{t}^{(2)}]$).

In our construction, we use ElGamal encryption as the singly homomorphic encryption scheme to generate public key and shared secret key through the underlying SFE [45, Fig.11]. We assume secret key is in the form of shared bits, i.e., $[sk] = \sum [sk_i] * 2^i$. In the first step, all parties jointly encrypt some plaintext γ (each one has a share in $[\gamma]$) and then output an ElGamal ciphertext (α_i, β_i) . The two sets of shared random values are jointly encrypted as $\mathbf{ct}^{(1)}$ and $\mathbf{ct}'^{(1)}$, which are used as inputs for the two extended permutations, respectively. In the second step, this tricky problem can be solved using \mathcal{ZK}_{DS} twice. At the end of the second step, data owners have the ciphertexts $\mathbf{ctp}^{(1)}$ (resp., $\mathbf{ctp}'^{(1)}$) and $\mathbf{ctp}^{(2)}$ (resp., $\mathbf{ctp}'^{(2)}$). In the third step, data owners jointly decrypt four sets of ciphertexts to obtain the shares of the plaintexts. With all these three steps, one can readily combine them into an actively secure PFE.

4.2 Offline Phase

Now, we describe the offline phase of our actively secure PFE. The function owner compiles his private function f as a topological circuit \mathcal{C}_f , which is like a directed acyclic graph with g gates, u inputs and o outputs. These g gates are only two fan-in and could be any fan-out. For gates with fan-out greater than 1, we count each of their output wires as a different wire. This differs from the gates in the universal circuit, where all gates are with fan-out smaller than or equal to two. Suppose we have an arithmetic circuit and g gates in \mathcal{C}_f are additive or multiplicative gates. Bit 0 represents that the gate is an additive gate ($G_i = 0$), and 1 a multiplicative gate ($G_i = 1$). P_1 secretly shares the bit string of the g gates with data owners. P_1 also broadcasts the set \mathcal{C} .

Each data owner has his private input data, say $x_i \in \mathbf{F}_{p^k}$. All data owners need to jointly prepare two pairs of random value sets ($[\mathbf{l}], [\mathbf{r}]$) and ($[\mathbf{t}], [\mathbf{s}]$) through the underlying SFE, $|\mathbf{l}| = |\mathbf{t}| = |\mathbf{ow}|$ and $|\mathbf{r}| = |\mathbf{s}| = |\mathbf{iw}|$. The two sets of shared random values $[\mathbf{l}]$ and $[\mathbf{t}]$ are jointly encrypted into $\mathbf{ct}^{(1)}$ and $\mathbf{ct}'^{(1)}$. Data owners send the public key pk , $\mathbf{ct}^{(1)}$ and $\mathbf{ct}'^{(1)}$ to P_1 . Through \mathcal{ZK}_{DS} , the two sets of ciphertexts are permuted and re-randomized by P_1 into $\mathbf{ctp}^{(1)}$ and $\mathbf{ctp}'^{(1)}$ ($|\mathbf{ctp}^{(1)}| = |\mathbf{ctp}'^{(1)}| = |\mathbf{ow}|$), and $\mathbf{ctp}^{(2)}$ and $\mathbf{ctp}'^{(2)}$ ($|\mathbf{ctp}^{(2)}| = |\mathbf{ctp}'^{(2)}| = |\mathbf{iw}|$), respectively. We take $[\mathbf{l}]$ as an example, which corresponds to the ciphertext set $\mathbf{ct}^{(1)}$ sent to P_1 by data owners. P_1 applies the first permutation π_1 and re-randomization and gets $ctp_i^{(1)} = \text{Enc}_{pk}(l_i^{(1)}) = \text{Enc}_{pk}(l_{\pi_1(i)} + a_i)$, with a_i being

a random value chosen uniformly by P_1 . P_1 copies the ciphertexts $\mathbf{ct}^{(1)}$ according to \mathbf{C} and obtains a set of ciphertexts $\mathbf{ct}^{(2)}$ ($|\mathbf{ct}^{(2)}| = |\mathbf{iw}|$). Then P_1 applies the second permutation π_2 and re-randomization and gets $ctp_i^{(2)} = \text{Enc}_{pk}(l_i^{(2)}) = \text{Enc}_{pk}(l_{\pi_2(i)}^{(1)} + b_i)$, with b_i being a random value chosen uniformly by P_1 . P_1 proves the correctness of his local work with \mathcal{ZK}_{DS} . The transformation of the set \mathbf{t} is similar to that of \mathbf{l} . Data owners jointly decrypt $\mathbf{ctp}^{(1)}$ and $\mathbf{ctp}^{(2)}$ to obtain $[\mathbf{l}^{(1)}]$ and $[\mathbf{l}^{(2)}]$, and decrypt $\mathbf{ctp}'^{(1)}$ and $\mathbf{ctp}'^{(2)}$ to obtain $[\mathbf{t}^{(1)}]$ and $[\mathbf{t}^{(2)}]$.

Each of the above-mentioned sets with $|\mathbf{iw}|$ entries is fresh for the inputs to the g gates. Each of the sets with $|\mathbf{ow}|$ entries is fresh for the outputs of the $g - o$ non-output gates and the u inputs of the circuit. Data owners have the shares of r_i and the shares of $l_i^{(2)}$, respectively, and then compute $[p_i] = [r_i - l_i^{(2)}]$ through the underlying SFE. Data owners generate a global MAC key K through the SFE, which is shared among the parties and no one can forge the MACs. Data owners also have the shares of the random value s_i and the shares of $t_i^{(2)}$, respectively, and then compute $[q_i] = [s_i - t_i^{(2)}] + p_i * [K]$ through the underlying SFE. Data owners broadcast the sets \mathbf{p} and \mathbf{q} to P_1 . We give the offline phase in Table 5. The set $\mathbf{l}^{(1)}$ is used for one-time pad (OTP) encryption of the data in the online phase. The set $\mathbf{t}^{(1)}$ is used for MAC verification of the data after OTP encryption in the online phase. These MACs will be used to check if P_1 does local computations correctly during the online phase, as described below.

4.3 Online Phase

Next, we describe the online phase of our actively secure PFE. Each data owner has private input data (say x_j of P_j , $1 \leq j \leq u$), and needs to share his data secretly with other data owners (denoted as $[x_j]$). For example, P_j shares his private data x_j with other data owners through Shamir's secret sharing. We donot emphasize here exact threshold that should be considered according to particular application scenario.

There are $|\mathbf{ow}|$ outgoing wires and $|\mathbf{iw}|$ incoming wires ($|\mathbf{ow}| \leq |\mathbf{iw}|$). The outgoing wires would be extended and permuted to obtain the incoming wires. In other words, some outgoing wire might have several copies in the incoming wires and thus we get an extended permutation from the outgoing wires to the incoming wires. To facilitate the representation, we consider its "reverse" relationship from the incoming wires to the outgoing wires (as shown in Fig. 2), which is really a surjection map (denoted as π_3).

Now each data owner owns a share in $[x_j]$. All data owners need to do preparatory computations on the data in the input circuit. They do OTP encryption for x_j with $\mathbf{l}^{(1)}$ generated in the offline phase, denoted as $[u_j] = [x_j] + [l_j^{(1)}]$, and additionally compute $[v_j] = [t_j^{(1)}] + ([x_j] + [l_j^{(1)}]) * [K]$, where j is the outgoing wire's index corresponding to that input wire of the circuit, and both \mathbf{u} and \mathbf{v} are recovered via the underlying SFE. At the beginning of the online phase, data owners send \mathbf{u} and \mathbf{v} to P_1 . For ease of explanation, we set $y_k = x_{\pi_3(k)}$ for $k \in \{1, \dots, |\mathbf{iw}|\}$, i.e., if there is an outgoing wire assigned to $[x_j]$, then there is an incoming wire assigned to $[y_k]$ s.t. $j = \pi_3(k)$. P_1 has all the knowledge of

Table 5. Linear implementation of the Offline Phase by \mathcal{ZK}_{DS} .

Linear Implementation of The Offline Phase from \mathcal{ZK}_{DS}

This protocol invokes the underlying SFE, so we only describe the operations related to the private function.

Input Function: **P_1 shares his circuit/function.**

- P_1 secretly shares G_i with other players, $i \in \{1, \dots, g\}$.
- Players evaluate and open $[G_i] \cdot (1 - [G_i])$ for $i \in \{1, \dots, g\}$. If any of them is not 0, it means that P_1 has not entered a valid function and the players abort the protocol.

Players generate randomness for inputs and outputs of two shuffles.

- Data owners jointly generate the secretly shared random values $[l] = \{[l_1], \dots, [l_{|ow|}]\}$ and $[r] = \{[r_1], \dots, [r_{|iw|}]\}$ through the underlying SFE. l is for the input of the RG&OWD component, and l' produced from l in keeping with C is for the input of the RR&IWD component. r is prepared for the output of the RR&IWD component.
- Data owners jointly generate the secretly shared MAC key K , shared random values $[t] = \{[t_1], \dots, [t_{|ow|}]\}$ and $[s] = \{[s_1], \dots, [s_{|iw|}]\}$ by the underlying SFE. t is for the input of the RG&OWD component, and t' produced from t in keeping with C is for the input of the RR&IWD component. s is prepared for the output of the RR&IWD component.

 P_1 applies (l, t) and (l', t') to the two shuffles.

- Data owners jointly call key generation algorithm. This generates public key for ElGamal encryption and shared secret key, i.e., $[sk] = \sum_{i=1}^u [sk_i] * 2^i$, where u is the total number of data owners.
- Data owners jointly call encryption algorithm. From the plaintext shares $\{[l_1], \dots, [l_{|ow|}]\}$ and $\{[t_1], \dots, [t_{|ow|}]\}$, they get ciphertexts $\{ct_1^{(1)}, \dots, ct_{|ow|}^{(1)}\}$ and $\{ct_1'^{(1)}, \dots, ct_{|ow|}'^{(1)}\}$ which are then sent to P_1 .
- P_1 applies π_1 and re-randomization to $\{ct_1^{(1)}, \dots, ct_{|ow|}^{(1)}\}$ to obtain $\{ctp_1^{(1)}, \dots, ctp_{|ow|}^{(1)}\}$. Transforms $\{ct_1^{(1)}, \dots, ct_{|ow|}^{(1)}\}$ into $\{ct_1^{(2)}, \dots, ct_{|iw|}^{(2)}\}$ according to C . Applies π_2 and re-randomization to $\{ct_1^{(2)}, \dots, ct_{|iw|}^{(2)}\}$ and obtains $\{ctp_1^{(2)}, \dots, ctp_{|iw|}^{(2)}\}$. Similarly, from $\{ct_1'^{(1)}, \dots, ct_{|ow|}'^{(1)}\}$, gets $\{ctp_1'^{(1)}, \dots, ctp_{|ow|}'^{(1)}\}$ and $\{ctp_1'^{(2)}, \dots, ctp_{|iw|}'^{(2)}\}$.
- P_1 applies \mathcal{ZK}_{DS} to prove that he already uses two valid EP.
- Data owners jointly call decryption algorithm. From $\{ctp_1^{(1)}, \dots, ctp_{|ow|}^{(1)}\}$ and $\{ctp_1^{(2)}, \dots, ctp_{|iw|}^{(2)}\}$, they obtain $\{[l_1^{(1)}], \dots, [l_{|ow|}^{(1)}]\}$ and $\{[l_1^{(2)}], \dots, [l_{|iw|}^{(2)}]\}$. Similarly, from $\{ctp_1'^{(1)}, \dots, ctp_{|ow|}'^{(1)}\}$ and $\{ctp_1'^{(2)}, \dots, ctp_{|iw|}'^{(2)}\}$, they get the shares $\{[t_1^{(1)}], \dots, [t_{|ow|}^{(1)}]\}$ and $\{[t_1^{(2)}], \dots, [t_{|iw|}^{(2)}]\}$.

Data owners calculate and broadcast p and q .

- Data owners calculate $[p_k] = [r_k] - [l_k^{(2)}]$ and $[q_k] = [s_k] - [t_k^{(2)}] + p_k * [K]$ by the underlying SFE, $k \in \{1, \dots, |iw|\}$.
-

circuits and re-randomization, and he can compute \mathbf{u} and \mathbf{v} as

$$u'_k = u_{\pi_3(k)} = [y_k] + [l_k^{(2)}] \quad \text{and} \quad v'_k = v_{\pi_3(k)} = [t_k^{(2)}] + (y_k + l_k^{(2)}) * [K], \quad (4)$$

respectively. P_1 knows \mathbf{p} and \mathbf{q} generated in the offline phase and computes

$$d_k = u'_k + p_k = y_k + r_k \quad \text{and} \quad m_k = v'_k + q_k = s_k + (y_k + r_k) * K, \quad (5)$$

respectively, and broadcasts \mathbf{d} and \mathbf{m} to all data owners. Data owners calculate $[n_k] = [s_k] + (y_k + r_k) * [K]$ via the underlying SFE and check whether n_k is equal to m_k . If $n_k \neq m_k$, it means that P_1 has cheated during his local computation. At this point, the other parties decide whether to abort the protocol based on the properties of the underlying SFE, or return P_1 's input to continue the protocol without P_1 's involvement. If $n_k = m_k$, then P_1 is uncorrupted and data owners compute $[y_k] = d_k - [r_k]$ by the underlying SFE. For g gates that are topologically ordered, data owners can then compute

$$[z_i] = (1 - [G_i]) * ([y_{2i-1}] + [y_{2i}]) + [G_i] * [y_{2i-1}] * [y_{2i}], i \in \{1, \dots, g\} \quad (6)$$

Each data owner now has a share of $[z_i]$. The corresponding outgoing wire's index of $[z_i]$ is $w = u + i$. Data owners then continue to compute $[u_w]$ and $[v_w]$ and repeat the above operations until all g gates are computed. The outputs of the last o gates constitute the computation result of the function on the inputs.

Informally speaking, function owner P_1 uses the difference set generated in the offline phase to convert the encrypted output of one gate to the encrypted input of the upcoming gate, while maintaining a one-time MAC of all values. Once all g gates have been computed, it completes the entire computation of the function. From online/offline phases, P_1 neither learns the private inputs of data owners nor reveals the knowledge of the circuit topology. Data owners can go through the MACs to check whether the local operation of P_1 is correct without sacrificing the topology of the circuit. If some MAC authentication fails, then there exist incorrectness in P_1 's local computations. Above operations among multiple parties are performed securely via the underlying SFE protocol.

4.4 Proofs

In this section, we give simulation-based security proofs of the offline and online phases of our PFE protocol, respectively [45].

4.4.1 Proof of Offline Protocol

We construct a simulator $\mathcal{S}_{offline}$ such that a poly-time environment \mathcal{Z} cannot distinguish between a real protocol system and an ideal protocol system. We assume here a static, active corrupted adversary. A simulator $\mathcal{S}_{offline}$ simulates the ideal functionality of offline phase, which relays messages between the parties and \mathcal{Z} so that \mathcal{Z} will see the same interface as in its interaction with the real

protocol. We denote the set of corrupted parties by $\mathcal{A} \subset \{P_1, \dots, P_u\}$. Table 6 presents the simulator $\mathcal{S}_{offline}$ we construct.

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 statistically indistinguishable from the view in the real process. This view includes the corrupt player's view of the protocol execution as well as the honest player's inputs and outputs.

The view of adversaries $\mathcal{A} \setminus \{P_1\}$ includes the share of G_i , the share of random values for inputs and outputs of EP, $\{l_i\}_{i=1}^{|\mathit{ow}|}$, $\{r_i\}_{i=1}^{|\mathit{iw}|}$, $\{t_i\}_{i=1}^{|\mathit{ow}|}$, $\{s_i\}_{i=1}^{|\mathit{iw}|}$,

Table 6. simulator $\mathcal{S}_{offline}$

simulator $\mathcal{S}_{offline}$
<p>This protocol invokes the underlying SFE, so we only need to specify the simulator for the Input Function.</p> <p>Input Function:</p> <p>P_1 shares his circuit/function.</p> <p>If $P_1 \in \mathcal{A}$:</p> <ul style="list-style-type: none"> – The simulator $\mathcal{S}_{offline}$ evaluates $[G_i] * (1 - [G_i])$ for $i \in \{1, \dots, g\}$. If any value of them is not 0, then P_1 does not provide a valid function. The simulator aborts. <p>If $P_1 \notin \mathcal{A}$:</p> <ul style="list-style-type: none"> – The simulator $\mathcal{S}_{offline}$ generates a random circuit with g gates G'_i for all $i \in \{1, \dots, g\}$ and finds the mapping π_3 corresponding to this circuit. – The simulator $\mathcal{S}_{offline}$ generates shares of $[G'_i]$ for all $i \in \{1, \dots, g\}$, and sends the corresponding shares to other parties involved, respectively. – The simulator $\mathcal{S}_{offline}$ evaluates $[G'_i] * (1 - [G'_i])$ for $i \in \{1, \dots, g\}$. <p>Players generate randomness for inputs and outputs of two shuffles.</p> <ul style="list-style-type: none"> – The simulator $\mathcal{S}_{offline}$ honestly performs the protocol. <p>P_1 applies (l, t) and (l', t') to the two shuffles.</p> <p>If $P_1 \in \mathcal{A}$:</p> <ul style="list-style-type: none"> – The simulator $\mathcal{S}_{offline}$ randomly generates π_1 and π_2 then performs the ideal \mathcal{ZK}_{DS} protocol with other players. If any of the players aborts the protocol, $\mathcal{S}_{offline}$ aborts the protocol. <p>If $P_1 \notin \mathcal{A}$:</p> <ul style="list-style-type: none"> – The simulator $\mathcal{S}_{offline}$ honestly evaluates $ct^{(1)}$ and $ct'^{(1)}$. – Once P_1 inputs π_1 and π_2, the simulator $\mathcal{S}_{offline}$ performs the ideal \mathcal{ZK}_{DS} protocol twice with other players, respectively. – The simulator $\mathcal{S}_{offline}$ honestly performs next steps of the protocol. <p>Data owners calculate and broadcast p and q.</p> <ul style="list-style-type: none"> – The simulator $\mathcal{S}_{offline}$ honestly performs the protocol.

$\{l_i^{(1)}\}_{i=1}^{|\mathbf{ow}|}$, $\{l_i^{(2)}\}_{i=1}^{|\mathbf{iw}|}$, $\{t_i^{(1)}\}_{i=1}^{|\mathbf{ow}|}$, $\{t_i^{(2)}\}_{i=1}^{|\mathbf{iw}|}$, $\{ct_i^{(1)}\}_{i=1}^{|\mathbf{ow}|}$, $\{ctp_i^{(1)}\}_{i=1}^{|\mathbf{ow}|}$, $\{ct_i^{(2)}\}_{i=1}^{|\mathbf{iw}|}$, $\{ctp_i^{(2)}\}_{i=1}^{|\mathbf{iw}|}$, $\{ct_i^{\prime(1)}\}_{i=1}^{|\mathbf{ow}|}$, $\{ctp_i^{\prime(1)}\}_{i=1}^{|\mathbf{ow}|}$, $\{ct_i^{\prime(2)}\}_{i=1}^{|\mathbf{iw}|}$, $\{ctp_i^{\prime(2)}\}_{i=1}^{|\mathbf{iw}|}$, and finally, $\{p_i\}_{i=1}^{|\mathbf{iw}|}$, $\{q_i\}_{i=1}^{|\mathbf{iw}|}$. The shared values all look random and therefore are indistinguishable between ideal and real execution. $\{ct_i^{(1)}\}_{i=1}^{|\mathbf{ow}|}$, $\{ct_i^{(2)}\}_{i=1}^{|\mathbf{iw}|}$, $\{ct_i^{\prime(1)}\}_{i=1}^{|\mathbf{ow}|}$ and $\{ct_i^{\prime(2)}\}_{i=1}^{|\mathbf{iw}|}$ are ElGamal encryptions under shared secret key, and therefore are indistinguishable between ideal and real execution. $\{ctp_i^{(1)}\}_{i=1}^{|\mathbf{ow}|}$, $\{ctp_i^{(2)}\}_{i=1}^{|\mathbf{iw}|}$, $\{ctp_i^{\prime(1)}\}_{i=1}^{|\mathbf{ow}|}$ and $\{ctp_i^{\prime(2)}\}_{i=1}^{|\mathbf{iw}|}$ are valid re-randomization of ElGamal ciphertexts if the protocol does not abort due to \mathcal{ZK}_{DS} verification. $\{l_i^{(1)}\}_{i=1}^{|\mathbf{ow}|}$, $\{l_i^{(2)}\}_{i=1}^{|\mathbf{iw}|}$, $\{t_i^{(1)}\}_{i=1}^{|\mathbf{ow}|}$ and $\{t_i^{(2)}\}_{i=1}^{|\mathbf{iw}|}$ are obtained by permutation and re-randomization. The final results $\{p_i\}_{i=1}^{|\mathbf{iw}|}$ and $\{q_i\}_{i=1}^{|\mathbf{iw}|}$ are computed as a result of two shared random values, and therefore follow uniform distribution in both ideal and real executions.

The adversary P_1 has the same view as the other malicious players, except for the uniform random value set $\{a_i\}_{i=1}^{|\mathbf{ow}|}$ and $\{b_i\}_{i=1}^{|\mathbf{iw}|}$ that he alone has. The shared values all follow uniform random distribution. In the ideal functionality we also have a uniform distribution, and as a result ideal and real executions are indistinguishable to the environment \mathcal{Z} .

4.4.2 Proof of Online Protocol

We construct a simulator \mathcal{S}_{online} such that a poly-time environment \mathcal{Z} cannot distinguish between a real protocol system and an ideal protocol system. We assume here a static, active corrupted adversary. This simulator \mathcal{S}_{online} simulates the ideal functionality of online phase, which relays messages between the parties and \mathcal{Z} so that \mathcal{Z} will see the same interface as in its interaction with the real protocol. We denote the set of corrupted parties by $\mathcal{A} \subset \{P_1, \dots, P_u\}$. Table 7 presents the simulator \mathcal{S}_{online} we construct.

To see that the simulated and real processes cannot be distinguished, we will show that the view of the environment in the ideal process is statistically indistinguishable from the view in the real process. This view consists of the corrupt players' view of the protocol execution as well as the inputs and outputs of honest players. The view of adversary includes $\{u_i\}_{i=1}^{|\mathbf{ow}|}$, $\{v_i\}_{i=1}^{|\mathbf{iw}|}$, $\{d_i\}_{i=1}^{|\mathbf{iw}|}$, $\{m_i\}_{i=1}^{|\mathbf{iw}|}$, $\{n_i\}_{i=1}^{|\mathbf{iw}|}$ and $\{z_i\}_{i=1}^g$. The shared values all look random and therefore are indistinguishable between ideal and real execution.

We next show that $\{d_i\}_{i=1}^{|\mathbf{iw}|}$ and $\{m_i\}_{i=1}^{|\mathbf{iw}|}$ have uniform distribution. Observe that u_i is hidden using the random values of input wires which are shared and therefore act as one-time pad, and as P_1 prepares the two inputs, it maintains uniform distribution. Furthermore, p_i also follows uniform distribution from the secure offline protocol. The value s_i acts as a one-time pad which is shared between the players and therefore, m_i follows uniform distribution. In the ideal functionality we also have uniform distribution, and as a result ideal and real are indistinguishable to the environment \mathcal{Z} . For malicious P_1 , the distributions are the same, but we have to make sure that he has performed the input preparation correctly. In the next phase players check P_1 's computation. Player P_1 cheating

Table 7. simulator \mathcal{S}_{online}

simulator \mathcal{S}_{online}
<p>This protocol calls the underlying ideal SFE protocol and the offline protocol. The ideal offline phase protocol yields the correct \mathbf{p} and \mathbf{q}. We construct \mathcal{S}_{online} starting from the point that all parties input $[x_j]$ phase for all $j \in \{1, \dots, u\}$.</p> <p>Input Data:</p> <ul style="list-style-type: none"> – The simulator \mathcal{S}_{online} honestly performs the protocol. <p>Prepare the Input of the Circuit:</p> <ul style="list-style-type: none"> – If $P_j \in \mathcal{A}$, the simulator \mathcal{S}_{online} randomly generates $[u_j]$ and $[v_j]$. If any player aborts the protocol, \mathcal{S}_{online} aborts. – If $P_j \notin \mathcal{A}$, the simulator \mathcal{S}_{online} honestly performs the protocol. <p>Players Check P_1's Input Preparation:</p> <ul style="list-style-type: none"> – The simulator \mathcal{S}_{online} honestly performs the protocol and aborts if the checks fail, i.e., $[n_k] \neq [m_k]$ for all $k \in \{1, \dots, \mathbf{i}\mathbf{w} \}$. <p>Evaluate the Circuit:</p> <p>The simulator \mathcal{S}_{online} honestly computes each gate step by step according to the protocol.</p> <ul style="list-style-type: none"> – If $P_j \in \mathcal{A}$, the \mathcal{S}_{online} randomly generates two random values to represent the share of P_j's input about $[u_w]$ and $[v_w]$, $w = u+i, i \in \{1, \dots, g\}$. If any player aborts the protocol, \mathcal{S}_{online} aborts. – If $P_j \notin \mathcal{A}$, \mathcal{S}_{online} honestly performs the protocol. – The last o gates are the outputs of the circuit. By invoking the ideal SFE protocol, each party gets the output of the circuit.

means he does not calculate d_i and m_i correctly. To be successful, he has to somehow adjust n_i and m_i to be equal. He only has an option to adjust d_i and his share of s_i to make the equality hold. Since he does not know K , the value $d_i \cdot K$ has a uniform distribution, and therefore the probability of P_1 's modifying s_i s.t. $n_i = m_i$ is equivalent to that of guessing correct K and hence exponentially small in length of K . It follows that with overwhelming probability that P_1 's computation is correct after the checks. The simulator aborts if any check fails.

The final result z_i is a secret shared value and has a uniform distribution. For the output wires, players open their shares, and z_i is learnt by all parties. In order to make the distribution of outputs indistinguishable, the simulator has to modify his share of z_i in the ideal execution. He is able to do so and produce the exact same output for the ideal execution. This completes the proof.

5 Conclusions and Future Work

We propose a novel framework \mathcal{ZK}_{DS} for proving correct extended permutation by optimizing the structure of MSS framework. Compared with existing heavy solution \mathcal{ZK}_{EP} , \mathcal{ZK}_{DS} is more succinct and efficient in the sense that it removes the replication component \mathcal{ZK}_{rep} and that its double shuffles are

“smaller”. Significant gains on computation and communication complexities are thereby obtained. \mathcal{ZK}_{DS} seems to be optimal from the perspective of double-shuffle structure. We are also interested in other efficient methods (rather than zero-knowledge proofs) in verifying extended permutations. From \mathcal{ZK}_{DS} , we construct more compact PFE with linear active security for general purpose settings, which contributes an even less proof overhead and makes the protocol more practical.

References

1. Alhassan, M.Y., Günther, D., Kiss, Á., Schneider, T.: Efficient and scalable universal circuits. *J. Cryptol.* **33**(3), 1216–1271 (2020). <https://doi.org/10.1007/s00145-020-09346-z>, <https://doi.org/10.1007/s00145-020-09346-z>
2. Araki, T., Barak, A., Furukawa, J., Lichter, T., Lindell, Y., Nof, A., Ohara, K., Watzman, A., Weinstein, O.: Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. pp. 843–862. IEEE Computer Society (2017). <https://doi.org/10.1109/SP.2017.15>, <https://doi.org/10.1109/SP.2017.15>
3. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: Backes, M., Ning, P. (eds.) *Computer Security - ESORICS 2009*, 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5789, pp. 424–439. Springer (2009). https://doi.org/10.1007/978-3-642-04444-1_26, https://doi.org/10.1007/978-3-642-04444-1_26
4. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Pointcheval, D., Johansson, T. (eds.) *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Cambridge, UK, April 15-19, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7237, pp. 263–280. Springer (2012). https://doi.org/10.1007/978-3-642-29011-4_17, https://doi.org/10.1007/978-3-642-29011-4_17
5. 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>, <https://doi.org/10.1145/62212.62213>
6. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Tallinn, Estonia, May 15-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6632, pp. 169–188. Springer (2011). https://doi.org/10.1007/978-3-642-20465-4_11, https://doi.org/10.1007/978-3-642-20465-4_11
7. Bicer, O., Bingol, M.A., Kiraz, M.S., Levi, A.: Highly efficient and re-executable private function evaluation with linear complexity. *IEEE Transactions on Dependable and Secure Computing* pp. 1–1 (2020). <https://doi.org/10.1109/TDSC.2020.3009496>

8. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007. pp. 498–507. ACM (2007). <https://doi.org/10.1145/1315245.1315307>, <https://doi.org/10.1145/1315245.1315307>
9. 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, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. pp. 315–334. IEEE Computer Society (2018). <https://doi.org/10.1109/SP.2018.00020>, <https://doi.org/10.1109/SP.2018.00020>
10. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings. Lecture Notes in Computer Science, vol. 740, pp. 89–105. Springer (1992). https://doi.org/10.1007/3-540-48071-4_7, https://doi.org/10.1007/3-540-48071-4_7
11. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings. Lecture Notes in Computer Science, vol. 839, pp. 174–187. Springer (1994). https://doi.org/10.1007/3-540-48658-5_19, https://doi.org/10.1007/3-540-48658-5_19
12. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: Jarecki, S., Tsudik, G. (eds.) Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5443, pp. 160–179. Springer (2009). https://doi.org/10.1007/978-3-642-00468-1_10, https://doi.org/10.1007/978-3-642-00468-1_10
13. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer (2012). https://doi.org/10.1007/978-3-642-32009-5_38, https://doi.org/10.1007/978-3-642-32009-5_38
14. Demmler, D., Schneider, T., Zohner, M.: ABY - A framework for efficient mixed-protocol secure two-party computation. In: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015. The Internet Society (2015), <https://www.ndss-symposium.org/ndss2015/aby---framework-efficient-mixed-protocol-secure-two-party-computation>
15. Evans, D., Kolesnikov, V., Rosulek, M.: A pragmatic introduction to secure multi-party computation. Foundations and Trends® in Privacy and Security 2(2-3), 70–246 (2018). <https://doi.org/10.1561/33000000019>, <http://dx.doi.org/10.1561/33000000019>
16. 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.) Advances in Cryptology - EURO-

- CRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 537–556. Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_32, https://doi.org/10.1007/978-3-642-38348-9_32
17. Frikken, K.B., Atallah, M.J., Li, J.: Attribute-based access control with hidden policies and hidden credentials. *IEEE Trans. Computers* **55**(10), 1259–1270 (2006). <https://doi.org/10.1109/TC.2006.158>, <https://doi.org/10.1109/TC.2006.158>
 18. Frikken, K.B., Atallah, M.J., Zhang, C.: Privacy-preserving credit checking. In: Riedl, J., Kearns, M.J., Reiter, M.K. (eds.) *Proceedings 6th ACM Conference on Electronic Commerce (EC-2005)*, Vancouver, BC, Canada, June 5-8, 2005. pp. 147–154. ACM (2005). <https://doi.org/10.1145/1064009.1064025>, <https://doi.org/10.1145/1064009.1064025>
 19. Furukawa, J., Sako, K.: An efficient scheme for proving a shuffle. In: Kilian, J. (ed.) *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference*, Santa Barbara, California, USA, August 19-23, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2139, pp. 368–387. Springer (2001). https://doi.org/10.1007/3-540-44647-8_22, https://doi.org/10.1007/3-540-44647-8_22
 20. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) *Advances in Cryptology, Proceedings of CRYPTO '84*, Santa Barbara, California, USA, August 19-22, 1984, Proceedings. Lecture Notes in Computer Science, vol. 196, pp. 10–18. Springer (1984). https://doi.org/10.1007/3-540-39568-7_2, https://doi.org/10.1007/3-540-39568-7_2
 21. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*. p. 218–229. STOC '87, Association for Computing Machinery, New York, NY, USA (1987). <https://doi.org/10.1145/28395.28420>, <https://doi.org/10.1145/28395.28420>
 22. Groth, J.: Linear algebra with sub-linear zero-knowledge arguments. In: Halevi, S. (ed.) *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference*, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5677, pp. 192–208. Springer (2009). https://doi.org/10.1007/978-3-642-03356-8_12, https://doi.org/10.1007/978-3-642-03356-8_12
 23. Groth, J.: A verifiable secret shuffle of homomorphic encryptions. *J. Cryptol.* **23**(4), 546–579 (2010). <https://doi.org/10.1007/s00145-010-9067-9>, <https://doi.org/10.1007/s00145-010-9067-9>
 24. Groth, J., Ishai, Y.: Sub-linear zero-knowledge argument for correctness of a shuffle. In: Smart, N.P. (ed.) *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Istanbul, Turkey, April 13-17, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4965, pp. 379–396. Springer (2008). https://doi.org/10.1007/978-3-540-78967-3_22, https://doi.org/10.1007/978-3-540-78967-3_22
 25. Groth, J., Lu, S.: Verifiable shuffle of large size ciphertexts. In: Okamoto, T., Wang, X. (eds.) *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography*, Beijing, China, April 16-20, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4450, pp. 377–

392. Springer (2007). https://doi.org/10.1007/978-3-540-71677-8_25, https://doi.org/10.1007/978-3-540-71677-8_25
26. Günther, D., Kiss, Á., Scheidel, L., Schneider, T.: Poster: Framework for semi-private function evaluation with application to secure insurance rate calculation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019. pp. 2541–2543. ACM (2019). <https://doi.org/10.1145/3319535.3363251>, <https://doi.org/10.1145/3319535.3363251>
 27. Günther, D., Kiss, Á., Schneider, T.: More efficient universal circuit constructions. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10625, pp. 443–470. Springer (2017). https://doi.org/10.1007/978-3-319-70697-9_16, https://doi.org/10.1007/978-3-319-70697-9_16
 28. Hazay, C., Nissim, K.: Efficient set operations in the presence of malicious adversaries. In: Nguyen, P.Q., Pointcheval, D. (eds.) Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6056, pp. 312–331. Springer (2010). https://doi.org/10.1007/978-3-642-13013-7_19, https://doi.org/10.1007/978-3-642-13013-7_19
 29. Holz, M., Kiss, Á., Rathee, D., Schneider, T.: Linear-complexity private function evaluation is practical. In: Chen, L., Li, N., Liang, K., Schneider, S.A. (eds.) Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12309, pp. 401–420. Springer (2020). https://doi.org/10.1007/978-3-030-59013-0_20, https://doi.org/10.1007/978-3-030-59013-0_20
 30. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Sadeghi, A., Gligor, V.D., Yung, M. (eds.) 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013. pp. 955–966. ACM (2013). <https://doi.org/10.1145/2508859.2516662>, <https://doi.org/10.1145/2508859.2516662>
 31. Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: Lee, D.H., Wang, X. (eds.) Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings. Lecture Notes in Computer Science, vol. 7073, pp. 556–571. Springer (2011). https://doi.org/10.1007/978-3-642-25385-0_30, https://doi.org/10.1007/978-3-642-25385-0_30
 32. Kiss, Á., Schneider, T.: Valiant’s universal circuit is practical. In: Fischlin, M., Coron, J. (eds.) Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9665, pp. 699–728. Springer (2016). https://doi.org/10.1007/978-3-662-49890-3_27, https://doi.org/10.1007/978-3-662-49890-3_27
 33. Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) Financial Cryptography and

- Data Security, 12th International Conference, FC 2008, Cozumel, Mexico, January 28-31, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5143, pp. 83–97. Springer (2008). https://doi.org/10.1007/978-3-540-85230-8_7, https://doi.org/10.1007/978-3-540-85230-8_7
34. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. Lecture Notes in Computer Science, vol. 8043, pp. 1–17. Springer (2013). https://doi.org/10.1007/978-3-642-40084-1_1, https://doi.org/10.1007/978-3-642-40084-1_1
 35. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, R., Garay, J.A. (eds.) Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. Lecture Notes in Computer Science, vol. 8043, pp. 1–17. Springer (2013). https://doi.org/10.1007/978-3-642-40084-1_1, https://doi.org/10.1007/978-3-642-40084-1_1
 36. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4515, pp. 52–78. Springer (2007). https://doi.org/10.1007/978-3-540-72540-4_4, 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 (2009). <https://doi.org/10.1007/s00145-008-9036-8>, <https://doi.org/10.1007/s00145-008-9036-8>
 38. Lindell, Y., Riva, B.: Blazing fast 2pc in the offline/online setting with security for malicious adversaries. In: Ray, I., Li, N., Kruegel, C. (eds.) Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015. pp. 579–590. ACM (2015). <https://doi.org/10.1145/2810103.2813666>, <https://doi.org/10.1145/2810103.2813666>
 39. Lipmaa, H., Mohassel, P., Sadeghian, S.: Valiant’s universal circuit: Improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017 (2016), <https://eprint.iacr.org/2016/017>
 40. Liu, H., Yu, Y., 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.) Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12826, pp. 365–394. Springer (2021). https://doi.org/10.1007/978-3-030-84245-1_13, https://doi.org/10.1007/978-3-030-84245-1_13
 41. Liu, Y., Wang, Q., Yiu, S.: Making private function evaluation safer, faster, and simpler. *IACR Cryptol. ePrint Arch.* p. 1682 (2021), <https://eprint.iacr.org/2021/1682>
 42. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: Blaze, M. (ed.) Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA. pp. 287–302. USENIX (2004), <http://www.usenix.org/publications/library/proceedings/sec04/tech/malkhi.html>

43. Mohassel, P., Sadeghian, S.S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Athens, Greece, May 26-30, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7881, pp. 557–574. Springer (2013). https://doi.org/10.1007/978-3-642-38348-9_33, https://doi.org/10.1007/978-3-642-38348-9_33
44. Mohassel, P., Sadeghian, S.S.: How to hide circuits in MPC: an efficient framework for private function evaluation. *IACR Cryptol. ePrint Arch.* p. 137 (2013), <http://eprint.iacr.org/2013/137>
45. Mohassel, P., Sadeghian, S.S., Smart, N.P.: Actively secure private function evaluation. *IACR Cryptol. ePrint Arch.* **2014**, 102 (2014), <http://eprint.iacr.org/2014/102>
46. Mohassel, P., Sadeghian, S.S., Smart, N.P.: Actively secure private function evaluation. In: Sarkar, P., Iwata, T. (eds.) *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security*, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II. Lecture Notes in Computer Science, vol. 8874, pp. 486–505. Springer (2014). https://doi.org/10.1007/978-3-662-45608-8_26, https://doi.org/10.1007/978-3-662-45608-8_26
47. Neff, C.A.: A verifiable secret shuffle and its application to e-voting. In: Reiter, M.K., Samarati, P. (eds.) *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security*, Philadelphia, Pennsylvania, USA, November 6-8, 2001. pp. 116–125. ACM (2001). <https://doi.org/10.1145/501983.502000>, <https://doi.org/10.1145/501983.502000>
48. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference*, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7417, pp. 681–700. Springer (2012). https://doi.org/10.1007/978-3-642-32009-5_40, https://doi.org/10.1007/978-3-642-32009-5_40
49. 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). <https://doi.org/10.1093/comjnl/bxt019>, <https://doi.org/10.1093/comjnl/bxt019>
50. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*, Prague, Czech Republic, May 2-6, 1999, Proceeding. Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer (1999). https://doi.org/10.1007/3-540-48910-X_16, https://doi.org/10.1007/3-540-48910-X_16
51. Shelat, A., Shen, C.: Two-output secure computation with malicious adversaries. In: Paterson, K.G. (ed.) *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Tallinn, Estonia, May 15-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6632, pp. 386–405. Springer (2011). https://doi.org/10.1007/978-3-642-20465-4_22, https://doi.org/10.1007/978-3-642-20465-4_22

52. Valiant, L.G.: Universal circuits (preliminary report). In: Chandra, A.K., Wotschke, D., Friedman, E.P., Harrison, M.A. (eds.) Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA. pp. 196–203. ACM (1976). <https://doi.org/10.1145/800113.803649>, <https://doi.org/10.1145/800113.803649>
53. Waksman, A.: A permutation network. *J. ACM* **15**(1), 159–163 (1968). <https://doi.org/10.1145/321439.321449>, <https://doi.org/10.1145/321439.321449>
54. Yao, A.C.: Protocols for secure computations. In: 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982). pp. 160–164 (1982). <https://doi.org/10.1109/SFCS.1982.38>
55. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science (sfcs 1986). pp. 162–167 (1986). <https://doi.org/10.1109/SFCS.1986.25>
56. Zhao, S., Yu, Y., Zhang, J., Liu, H.: Valiant’s universal circuits revisited: An overall improvement and a lower bound. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11921, pp. 401–425. Springer (2019). https://doi.org/10.1007/978-3-030-34578-5_15, https://doi.org/10.1007/978-3-030-34578-5_15
57. Zhu, R., Cassel, D., Sabry, A., Huang, Y.: NANOPI: extreme-scale actively-secure multi-party computation. 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. 862–879. ACM (2018). <https://doi.org/10.1145/3243734.3243850>, <https://doi.org/10.1145/3243734.3243850>
58. Zhu, R., Huang, Y., Katz, J., Shelat, A.: The cut-and-choose game and its application to cryptographic protocols. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 1085–1100. USENIX Association, Austin, TX (Aug 2016), <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/zhu>