FAMC: Fair and Publicly Auditable Multi-Party Computation with Cheater Detection

Yong Li*[1][0000−0002−1419−6257], Yueyang Feng[1], Xi Chen[2][0000−0001−8553−827X], Jian Zhang[2], Ruxian Li[2], Kewei Lv[3,4][0000−0002−8733−2870], and Dawei Zhang[5][0000−0001−5942−8245]

[1] School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China
[2] Linklogis, Shenzhen 518063, China
[3] State Key Lab of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100193, China
[4] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 101400, China
[5] School of Computer Science & Technology, Beijing Jiaotong University, Beijing 100044, China

Abstract. Secure multi-party computation (MPC) protocols do not completely prevent malicious parties from cheating. Though numerous researches on cheating detection are proposed, most cheater detection works do not guarantee *fairness* of the protocol. That is, if the corrupted parties try to violate fairness, the output of the computation can only be obtained by themselves, but not by the honest parties. In this paper, based on the SPDZ framework, we propose a fair secure multi-party computation protocol with honest-majority by using identifiable secret sharing. This protocol can identify corrupted parties during the execution process and prevent them from undermining the fairness of the protocol. In addition, the protocol satisfies *publicly auditability*, which allows any third party to verify the correctness of the protocol's output results. The security of the protocol was proved in the universal composability framework. The protocol maintains high online efficiency when no cheating happens. And if corrupted parties try to violate the fairness, honest parties need to expend additional computation to regain the output results of the protocol, but the cost remains within a reasonable range.

Keywords: secure multi-party computation · cheater detection · fairness.

1 Introduction

The goal of secure multi-party computation (MPC) is to allow a set of mutually independent and distrust parties to jointly complete the computation of a function with their respective input while guaranteeing the privacy of their inputs.

* Corresponding author. Email:liyong@bjtu.edu.cn

MPC has found wide applications in electronic auctions, privacy-preserving data mining etc, since it was proposed by Yao in 1982 [1].

In traditional MPC, a party corrupted by a malicious adversary can learn the outputs of the honest parties and cause a false termination (called *abort*). It is desirable to be able to identify the corrupted party - cheater, i.e., to achieve *cheater detection* in the MPC protocol. Furthermore, many MPC schemes do not consider the property of *fairness* (i.e. either all parties receive the protocol output or no party does), since sacrificing fairness can be a trade-off for higher efficiency. However, it is important that all MPC parties receive the computation results, for example, in the case of electronic contract signing, it would be very problematic if malicious parties receive the signed contract while honest parties do not [2]. Fairness has always been an important topic in MPC, and ensuring that all parties have access to the output results is one of the fundamental properties required for secure multi-party computation. Meanwhile, it is crucial for all parties involved in the MPC protocol to verify that the output results are computed correctly based on their respective private inputs. This property is referred to as *publicly auditability*. Furthermore, in applications like public elections or auctions, it is also essential for the results of the election or auction to satisfy publicly auditability.

Our contributions. In this paper, we propose a Fair and publicly Auditable Multi-party Computation protocol with cheater detection (FAMC), based on SPDZ protocol [4, 5]. The security properties include:

Cheater Detection. If corrupted parties try to deviate from the protocol, then honest parties are able to identify them specifically as cheaters.

Fairness. After the completion of the protocol, all participating parties are able to obtain the output results. In other words, the protocol do not allow the case which only corrupted parties can access the output results. And corrupted parties can obtain output results only when the honest parties can obtain them.

Publicly Auditability. FAMC protocol enables any external third-party auditor to verify the correctness of the protocol computation results, including the results of the protocol's normal output results, and the results of the reconstruction by honest parties in case of cheaters undermining fairness of the protocol.

2 Related Work

Cheater Detection. Early MPC protocols such as [3], achieved cheater detection by letting each party prove in zero knowledge that they honestly follow the protocol. However, the high communication complexity of these protocols makes them unsuitable for actual deployment. The SPDZ series protocols [4, 5] have high efficiency, but the protocol itself does not provide the nature of cheater detection. The early protocol of Bendlin et al. [6] provides a weak form of cheater detection: that is, at least one honest player will identify dishonest players, but other honest players may have no clue about the identity of the cheater; MPC with *identifiable abort* [7, 8] ensures that all honest parties agree with the sub-

set of cheaters. Cunningham et al. introduced MPC with *completely identifiable abort* to ensure that all honest parties can identify all cheaters [9]. Recently, Jin et al. proposed an MPC scheme based on blockchain, which combines Pedersen commitment and non-interactive zero-knowledge proof to detect cheaters, and both parties inside and outside the protocol will be convinced of the cheater's identity [27]. Baum et al. proposed a new, lightweight approach to discernible aborts in dishonest majority MPC, and a key feature of this protocol is that the online phase is based on simple and pairwise information-theoretic MACs [29].

Publicly Auditability. Early protocols with property of publicly auditability originate from special applications, such as publicly verifiable electronic voting, online auctions, and verifiable secret sharing, etc. Cohen and Fischer proposed a public and anonymous voting scheme [10]. This voting scheme not only output the voting results but also provided a proof of correctness that can be verified by everyone. Chaum et al. proposed a publicly auditable voting scheme [11], aiming to enable voters to verify whether their ballots are correctly counted in the tallying process. Sako proposed a publicly verifiable auction protocol [12] which does not reveal the bids of non-winning parties but all auction parties can verify the validity of the winning bid, ensuring that the winning price is indeed the highest bid. The aforementioned research works focus on achieving publicly auditable protocols in specific application scenarios such as voting and auction. In the work of Hoogh [13], the idea of building a universal verifiable protocol was proposed, that is, by attaching correctness of non-interactive zero-knowledge proof to all sent messages, universal verifiability of secure multi-party computation was realized on the basis of threshold homomorphic cryptosystem [14]. In 2014, Baum et al. formally defined the concept of *publicly auditable* secure multiparty computation [15]. They supplemented and modified the SPDZ protocol to satisfy publicly auditability, that is, any third party that has access to the public records of the protocol can verify whether the output result of the protocol is correct.

Fairness. Achieving fairness in MPC, that is, all parties get or no one gets the output is a long-term research issue. Suppose the total number of all parties is n, and the number of corrupted parties is t. When $t < n/3$, assuming that there exists a synchronous point-to-point network with authentication channel, the fair MPC protocol for computational security can be constructed for any computing function [2]. For instance, Goldreich et al. successfully constructed a fair MPC protocol under the assumption of the existence of trapdoor one-way permutation [16, 3]. Furthermore, it is possible to achieve the construction of information-theoretic secure protocols, if the authenticated channel in an aforementioned synchronous point-to-point network is private. For example, Chaum et al. proposed a perfectly fair MPC protocol that guarantees information-theoretic security [17]. When $t < n/2$, in such setting, assuming that each party has access to a broadcast channel, regardless of computational security or information-theoretic security, any computational function can be used to achieve the construction of fair MPC protocol [18]. Inspired by the economic incentive and punishment mechanisms in blockchain, *fairness with penalties* [28] was achieved in Jin et

al.'s MPC scheme [27], where cheaters will be financially punished, while honest parties will be compensated. Nie et al. construct a SPDZ-like protocol that supports an identifiable abort in dishonest majority, achieving fairness when there is malicious behavior by up to $n/2$ parties acting as cheaters [30].

Both of the above situations belong to honest-majority scenarios, and when $t \geq n/2$, it is dishonest-majority. Unfortunately, as early as 1986, Cleve proved that it is impossible to achieve a completely fair MPC under dishonest-majority scenarios [19]. However, fairness is equally important in voting and electronic contract signing when $t \geq n/2$, so scholars relaxed some conditions of fairness (complete fairness) under the standard definition, and called "*partial fairness*". Studies on partial fairness mainly include optimistic model and stepwise opening model and so on.

Cunningham et al. proposed a commitment-enhanced secret sharing scheme (CESS) to achieve cheater detection [9]. Their scheme is based on SPDZ protocol, which is divided into offline and online phases. Cunningham et al.'s scheme does not provide property related to fairness, that is, although honest parties can identify each cheater, they *cannot* obtain the final correct output results.

3 Definition and Assumptions

3.1 Model Assumptions

Notations. Throughout this work, we implicitly consider a sequence of protocols parameterized by a security parameter κ. All of our MPC protocols consider arithmetic circuits over p-order fields, where p is a large prime. $\mathbb{Z}_p$ refers to the field $\{0, ..., p-1\}$; $\mathbb{Z}_p^*$ refers to $\mathbb{Z}_p/\{0\} = \{1, ..., p-1\}$, and g is a generator of a group G.

In our scheme, it is assumed that there is a direct secure communication channel between each two parties, and two available functions are required: a secure broadcast channel and a bulletin board. Each party will use one secure broadcast to distribute their private input. The purpose of using bulletin boards is to achieve publicly auditability: some public values during the protocol process are required to be recorded on the bulletin board, and after the protocol ends, any external third-party auditor can verify the correctness of the protocol output through the public records.

3.2 Scheme Definition

We proposes a fair secure multi-party computation protocol that satisfies public auditability and cheater detection. The scheme is secure in malicious adversary model with honest-majority, under the Decisional Diffie-Hellman assumption.

Fig.1 depicts the ideal functionality $\mathcal{F}_{FAMC}$ of the FAMC protocol, where all parties send their private inputs to the trusted party, and the trusted party computes the results and returns them to each party. At the same time, the trusted party also achieves auditability, which means that external third-party

auditors can verify the correctness of the computation results. In the audit step (which is new and different from the ideal functionality in [25]), auditor uses the expected output y_{recon} to audit the output of *Eval* step, accepting it if it is equal to y, otherwise identifying the cheater and executing second reconstruction to recover the correct output.

Init: Receive inputs $(Init, \mathcal{C}, p)$ from all participating parties (where $\mathcal{C}$ is the computing circuit corresponding to the computing function f, composed of addition and multiplication gates over the field $\mathbb{Z}_p$, and p is a large prime number), and execute:

1. Wait for adversary $\mathcal{A}$ to send a list of corrupted parties $P_c \subseteq \{1, 2, ..., n\}$.

Input: Receive inputs $(Input, x_i)$ from each P_i (where x_i is the private input value of P_i), and execute:

1. Save (P_i, x_i), calculate $x_i = [x_i]_{P_1} + ... + [x_i]_{P_n}$.
2. These calculated n initial shares are distributed to each party.
3. Save $(P_i, x_i, [x_1]_{P_i}, ..., [x_n]_{P_i})$.

Eval: Receive inputs $(Eval)$ from all parties, and execute:

1. If any participating party does not provide a private input value, output (**reject**).
2. Calculate $[y]_{P_i} = \mathcal{C}([x_1]_{P_i}, ..., [x_n]_{P_i})$, encrypt $[y]_{P_i}$and publish encryption.
3. Save $(P_i, x_i, [x_1]_{P_i}, ..., [x_n]_{P_i}, [y]_{P_i})$, if $i \notin P_C$. Otherwise, let adversary $\mathcal{A}$ random choose $[y]_{P_i'}$ and save $(P_i, x_i, [x_1]_{P_i}, ..., [x_n]_{P_i}, [y]_{P_i}, [y]_{P_i'})$.

Output: If the *Eval* step is successfully executed:

1. Calculate $y = [y]_{P_1} + ... + [y]_{P_i} + ... + [y]_{P_n}$, if $i \notin P_C$. Otherwise, let adversary $\mathcal{A}$ choose $[y]_{P_i'}$ instead of $[y]_{P_i}$.
2. Send $(Output, y)$ to all parties.

Audit: Receive input $(Audit, y_{recon})$ from the auditor, and if the *Eval* step is successfully executed:

1. If $y_{recon} = y$, output (**accept** y).
2. If $y_{recon} \neq y$, detect identity of cheater and add it to the list of cheaters L_{cheat}.
3. If $y_{recon} \neq y$ and $L_{cheat} = \{\emptyset\}$, public L_{cheat} and execute second reconstruction to output (**accept** y_{recon}).

Fig. 1. Ideal Functionality $\mathcal{F}_{FAMC}$

4 Achieving Fairness in MPC

4.1 Protocol Sketch

The flowchart of the FAMC protocol is shown in Fig.2, which is divided into two stages: offline and online. The offline phase refers to the phase in which all parties participate in preparing for the online interaction, that is, before the secure multi-party computation protocol is carried out, and mainly generates parameters for improving the computing efficiency of the online phase. The parameters

include global parameters, personal private and public keys, and global public keys, as well as the random numbers needed in the “input distribution" step, and the Beaver multiplication triplets [24] needed in the “calculation” step for multiplication gate operations.

In the online phase, after completing their respective input distribution, each party utilizes the private input shares of all parties to compute, according to the predefined computation circuit $\mathcal{C}$, until the final share is calculated. Then, before announcing the final share, the global public key is used for encryption, followed by output reconstruction. If the reconstruction is successful, all parties will receive output results and the protocol will end normally; If the reconstruction fails (if a cheater tampers with their final share), the honest party will decrypt their true final share after identifying the cheaters, and perform a second reconstruction to obtain the output result.

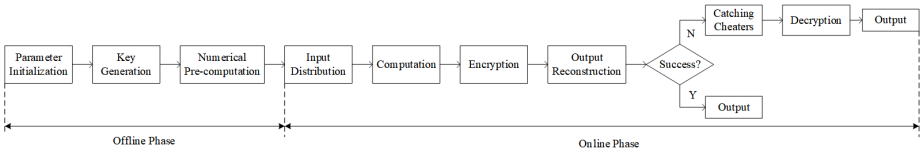

Fig. 2. Flowchart of FAMC protocol

4.2 Offline Phase

Let G be a multiplication group of a finite field $\mathbb{Z}_p$ (where p is a prime number), and let $g, h \in \mathbb{Z}_p^*$ be the generator. Take $pp = (G, p, g, h)$ as the global public parameters.

(1) **Key Generation.** Regarding the global public key pk, the idea is to allow all parties to jointly generate a part of pk, and all honest parties can collaborate to obtain the private key sk corresponding to pk, while all malicious parties cannot conspire to obtain sk. The detailed description is as follows:

Each party P_i selects a random number $sk_i \in \mathbb{Z}_p^*$ as their private key, then computes and publicly discloses the sub-public key $pk_i = h^{sk_i} \ mod \ p$, and then all parties compute the global public key $pk = \prod_{i=1}^{n} pk_i$. The private key is $sk = \sum_{i=1}^{n} sk_i$, which can be reconstructed by publishing each parties's respective sk_i.

However, in the malicious adversary model, the malicious parties are likely to refuse to cooperate, thus leading to failure of reconstructing the global private key sk. To solve this problem, our scheme uses (t,n)-*verifiable secret sharing*, allowing each party to share their own private key among n parties secretly. The specific key generation protocol $KeyGen$ is shown in Figure 3.

In the $KeyGen$ protocol, the threshold value t for the secret sharing scheme is determined as follows: when the number of parties n is even, $t = n/2+1$; when n is odd, $t = (n+1)/2$. Due to the use of verifiable secret sharing, it ensures that each party receives the correct share of the private keys of other parties. Moreover, each party P_i needs to generate an additional zero-knowledge proof to

Precondition: The global public parameters are $pp = (G, p, g, h)$, and the threshold value is set to t:

1. Party P_i randomly chooses $s \in \mathbb{Z}_p$ as their private key.
2. Party P_i chooses a random number $r \in \mathbb{Z}_p$ and calculates the commitment value $E_0 = pc(s, r) = g^s h^r$, then sends it to other parties.
3. Party P_i chooses two $t-1$ degree polynomials $F(x)$ and $G(x)$ with coefficients $(F_1, ..., F_{t-1})$ and $(G_1, ..., G_{t-1})$, respectively. They calculate commitments $\{E_j = pc(F_j, G_j) = g^{F_j} h^{G_j}\}_{j \in \{1,2,...,t-1\}}$ and publicly share them with other parties.
4. Party P_i calculates $\{[s]_{P_j} = F(j)\}_{j \in \{1,2,...,n\}}$ and $\{[r]_{P_j} = G(j)\}_{j \in \{1,2,...,n\}}$ and secretly sends $([s]_{P_j}, [r]_{P_j})$ to the party P_j.
5. After receiving the shares $([s]_{P_j}, [r]_{P_j})$ from party P_i, all other parties $\{P_j\}_{j \in \{1,2,...,n\}, j \neq i}$ confirm their shares about P_i by verifying if the equation $E([s]_{P_j}, [r]_{P_j}) = \prod_{k=0}^{t-1} E_k^{j^k}$ holds. Similarly, the protocol can only proceed when all parties receive and verify the shares distributed by others.
6. Party P_i sets their individual private key $sk_i = s$ and calculates their individual public key $pk_i = h^{sk_i}$. They also compute a NIZK proof $\pi(pk_i)$ to prove that the value sk_i in $pk_i = h^{sk_i}$ and the value of s disclosed in Step 2's commitment E_0 are the same. They send their individual public key and zero-knowledge proof to other parties.
7. All parties mutually verify their zero-knowledge proofs $\{\pi(pk_i)\}_{i \in \{1,2,...,n\}}$. Once all proofs pass the verification, calculate the global public key $pk = \prod_{i=1}^{n} pk_i$.

Fig. 3. Key generation protocol: $KeyGen$

prove that the sk_i used to compute the sub-public key pk_i and the sk_i secretly shared with other parties are the same. This zero-knowledge proof is required for step 6 of the $KeyGen$ protocol and is shown in Figure 4. After providing an interactive zero-knowledge proof based on the Camenisch-Stadler scheme [24], it is transformed into a non-interactive form by using the Fiat-Shamir paradigm [23].

(2) **Numerical Pre-computation.** In the offline phase, partial pre-computed values make each party more efficient in performing computations in the online phase. In Figure 5, the ideal function $\mathcal{F}_{SETUP}$ for pre-computation is defined, which mainly includes three parts: $Setup$, $RandomValues$, and $Triples$.

1) $Setup$: This step generates global public parameters for commitment and encryption.

2) $RandomValues$: In this step, $\mathcal{F}_{SETUP}$ needs to generate random numbers shared among n parties in the Committed Enhanced Secret Sharing (CESS) type format, with the number generated being related to the number of parties n. In the online phase, parties will use these pre-computed values to share their private inputs in the CESS format among n parties.

3) $Triples$: In this step, $\mathcal{F}_{SETUP}$ needs to generate Beaver triples [24] shared among n parties in CESS format, with the number generated being related to the number of multiplication gates m in the computation circuit. In the online

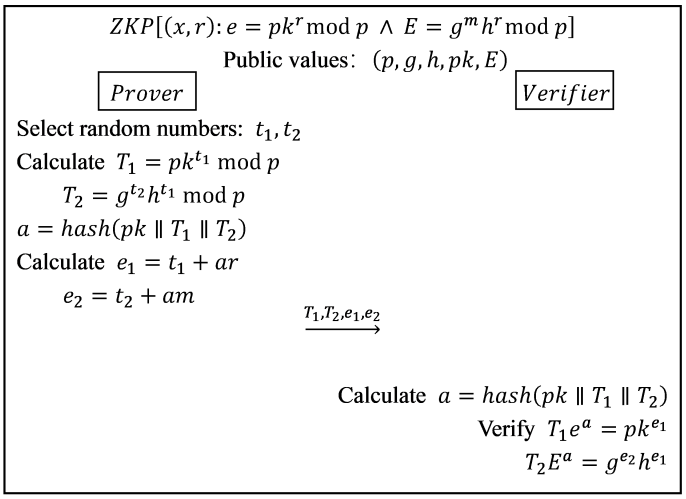

Fig. 4. Non-interactive zero knowledge proof based on Camenisch-Stadler scheme[22]

Setup: Receive input $(Setup, p)$ from all parties, $\mathcal{F}_{SETUP}$ saves prime number p. Adversary $\mathcal{A}$ sends a list of corrupted parties $P_C \subset \{1, 2, ..., n\}$.

1. Choose generators g,h, and then send g,h to adversary $\mathcal{A}$.
2. Send g,h to honest party $P_i, i \notin P_C$.

RandomValues: Receive input $(RandomValues, n)$ from all parties, execute:

1. For $i \notin P_C$, $\mathcal{F}_{SETUP}$ randomly selects $m_i \in \mathbb{Z}_p^*$ and $r_i \in \mathbb{Z}_p^*$, and then sends them to party P_i.
2. For $i \in P_C$,$\mathcal{A}$ selects $m_i \in \mathbb{Z}_p^*$ and $r_i \in \mathbb{Z}_p^*$.
3. $\mathcal{F}_{SETUP}$ calculate commitments $\{pc(m_i, r_i)\}_{i \in \{1,2,...,n\}}$, and then sends them to all parties.
4. Repeat steps 1-3 for a total of n times.

Triples: Receive input $(Triples, m)$ from all parties, execute:

1. For $i \notin P_C$, $\mathcal{F}_{SETUP}$ randomly selects a_i,r_{a_i},b_i,$r_{b_i} \in \mathbb{Z}_p^*$, and then sends them to party P_i.
2. For $i \in P_C$, $\mathcal{A}$ selects a_i,r_{a_i},b_i,r_{b_i},c_i,$r_{c_i} \in \mathbb{Z}_p^*$.
3. Define $a = \sum_{i=1}^{n} a_i$, $b = \sum_{i=1}^{n} b_i$.
4. Let $j \notin P_C$ be the smallest index of honest parties. For $i \notin P_C, i \neq j$, randomly selects c_i,$r_{c_i} \in \mathbb{Z}_p^*$. For P_j, let $c_j = ab - \sum_{i \in \{1,...,n\}, i \neq j} c_i$, and select $r_{c_j} \in \mathbb{Z}_p^*$. Send (c_i, r_{c_i}) to parties $P_i, i \notin P_C$.
5. $\mathcal{F}_{SETUP}$ calculate commitments $\{pc(a_i, r_{a_i}), pc(b_i, r_{b_i}), pc(c_i, r_{c_i})\}_{i \in \{1,2,...,n\}}$, and then sends them to all parties.
6. Repeat steps 1-5 for m times.

Fig. 5. The ideal functionality $\mathcal{F}_{SETUP}$ for the pre-calculation

phase, parties will use these pre-computed values of Beaver triples to achieve efficient multiplication operations.

4.3 Online Phase

In order to detect cheaters who may undermine the fairness of the protocol, we adopts the Committed Enhanced Secret Sharing (CESS) scheme. Each party uses all their own CESS shares (including shares of other parties' private input and one share of their own private input), calculates the final CESS share of the output result y according to a pre-defined computation circuit, and obtains the output by reconstructing all the final CESS shares. The specific process is as follows:

(1) **Input Distribution.** Each party P_i needs to share their private input x_i among n parties. This requires the random numbers generated in the $RandomValues$ step of the ideal function $\mathcal{F}_{SETUP}$ in the offline phase (Figure 5), which have already been shared among n parties. Taking a random number m as an example, the CESS share of P_i about m is:

$$< m >_{P_i} = ([m]_{P_i}, [r_m]_{P_i}, C_{m,1}, ..., C_{m,n}) \tag{1}$$

In order to share the private input x_i of party P_i among n parties, it is necessary for P_i to privately reconstruct m. This method of reconstruction is called *private reconstruction* in this scheme. To achieve private reconstruction, the Twisted-ElGamal encryption scheme [21] was used as building block. Parties P_i have already publicly disclosed their personal public key pk_i to other parties in the offline phase, so other parties P_j can use the $[r_m]_{P_j}$ included in their own CESS share $< m >_{P_j}$ to compute $e_{m,j} = pk_i^{[r_m]_{P_j}} \ mod \ p$, and send it to party P_i. On the other hand, P_i has the CESS share $< m >_{P_i}$ containing the commitment value $c_{m,j} = g^{[m]_{P_j}} h^{[r_m]_{P_j}}$ for party P_j's commitment to $[m]_{P_j}$. It is worth noting that by combining $pk_i^{[r_m]_{P_j}}$ and $g^{[m]_{P_j}} h^{[r_m]_{P_j}}$, the ciphertext under the Twisted-ElGamal encryption scheme can be obtained:

$$Enc([m]_{P_j}) = (pk_i^{[r_m]_{P_j}}, g^{[m]_{P_j}} h^{[r_m]_{P_j}}) \tag{2}$$

The notation $[m]_{P_j}$ refers to the plaintext message, while pk_i is the public key.

With their own private key sk_i, P_i can decrypt the ciphertext $Enc([m]_{P_j})$, thereby obtaining the plaintext $[m]_{P_j}$. Similarly, P_i can obtain all the secret additive shares $\{[m]_{P_j}\}_{j\in\{1,...,n\}}$ of the value m, and the sum of all the share values can be used to reconstruct m: $m = \sum_{j=1}^{n} [m]_{P_j}$. The specific private reconstruction protocol CESS.PrivRec can be found in Figure 6.

After reconstructing m, P_i uses their private input x_i to calculate $\varepsilon = x_i - m$, and broadcasts ε to all other parties. Once all parties obtained ε, they add it to their own CESS share $< m >_{P_i}$. According to the operation rules of additive secret sharing, only one party needs to perform the above addition operation, and the other parties do not need to perform any calculations. Typically, the

Precondition: Each participant P_i has a share $< x >_{P_i} = ([x]_{P_i}, [r_x]_{P_i}, c_{x,1}, ..., c_{x,n})$, and has a private key sk_i, and knows the personal public keys of all parties $\{pk_j\}_{j\in\{1,...,n\}}$.
$CESS.PrivOpen(i, j, < x >_{P_i} = ([x]_{P_i}, [r_x]_{P_i}, c_{x,1}, ..., c_{x,n}))$:
Party P_i performs the following steps, so that only party P_j can reconstruct x, and adds uncooperative parties to the cheater list L_{cheat}.

1. Use the personal public key of P_j to calculate $e_{x,i} = (pk_j)^{[r_x]_{P_i}}$.
2. Calculate a NIZK proof π_i to prove that the $[r_x]_{P_i}$ in $e_{x,i}$ and the $[r_x]_{P_i}$ in the commitment value $c_{x,i} = g^{[x]_{P_i}} h^{[r_x]_{P_i}}$ are the same.
3. Send $e_{x,i}$ and π_i to party P_j.

$CESS.PrivRec(i, j, (e_{x,i}, \pi_i), ..., (e_{x,n}, \pi_n), (c_{x,1}, ..., c_{x,n}))$:

1. Set the cheater list to be empty: $L_{cheat} = \{\emptyset\}$.
2. For $k = \{1, ..., n\}$: Check the NIZK proof π_i, and if the verification fails, add P_k to L_{cheat}.
3. If L_{cheat} is empty:
   1) If i=j:
      a) For $k = \{1, ..., n\}$: Decrypt $Enc([x]_{P_k}) = (e_{x,k}, c_{x,k})$ to obtain $[x]_{P_k}$.
      b) Calculate $x = \sum_{k=1}^{n} [x]_{P_k}$, obtaining x.
   2) Otherwise: Parties do not need to take any action.
4. Otherwise: Return (**reject**, L_{cheat}).

Fig. 6. Private reconstruction protocol: CESS.PrivRec

role requiring the addition operation is played by party P_1, who computes the CESS share regarding x_i as follows:

$$< x_i >_{P_1} = ([m]_{P_1} + \varepsilon, [r_m]_{P_1}, c_{m,1} \cdot pc(\varepsilon, 0), ..., c_{m,n}) \tag{3}$$

The remaining parties P_j compute the CESS share regarding x_i as follows:

$$\{< x_i >_{P_j} = ([m]_{P_j}, [r_m]_{P_j}, c_{m,1} \cdot pc(\varepsilon, 0), ..., c_{m,n})\}_{j\in\{2,...,n\}} \tag{4}$$

Thus, P_i's private input x_i has been shared among the n parties in the form of CESS. Similarly, all parties will use this method to distribute their respective private inputs.

(2) **Computation and Encryption.** After all parties have distributed their inputs, P_i has obtained the CESS shares $\{< x_i >_{P_j}\}_{j\in\{1,...,n\}}$ of all parties' inputs, and will then use these shares to compute the predefined computation circuit $\mathcal{C}$. The computation function consists of basic operations such as addition, constant addition, constant multiplication, and multiplication. Addition, constant addition, and constant multiplication are linear operations, which can be computed by each party locally, as the additive secret sharing shares and Pedersen commitment [20] in CESS shares are linearly homomorphic. However, multiplication is a non-linear operation, and all parties need to interact in order to complete a multiplication operation. The Beaver multiplication triplet generated in the offline stage can effectively improve the efficiency of multiplication operations.

After all parties have completed the computation of the circuit $\mathcal{C}$ according to the rules, they obtain the final CESS-shares regarding the output result y. The final CESS share of party P_i regarding y is $< y >_{P_i}$, which is given by:

$$< y >_{P_i} = ([y]_{P_i}, [r_y]_{P_i}, c_{y,1}, ..., c_{y,n}) \tag{5}$$

where $[y]_{P_i}$ represents the additive secret sharing share of y belonging to P_i; $[r_y]_{P_i}$ represents the opening value of commitment $c_{y,i}$ belonging to P_i; and $c_{y,i}$ is the commitment value of $[y]_{P_i}$ with $[r_x]_{P_i}$ as the random number, i.e. $c_{y,i} = g^{[y]_{P_i}} h^{[r_y]_{P_i}}$.

In traditional secret sharing based MPC schemes, the next step for all parties is to reconstruct the output result, i.e. each party P_i publicly reveals their own $[y]_{P_i}$ and $[r_y]_{P_i}$, calculates the output result $y = \sum_{i=0}^{n} [y]_{P_i}$, and identifies cheaters by verifying the consistency of the commitment values. In order to achieve fairness, in this scheme, each party is required to encrypt their own final share before performing the result reconstruction. The process is described as follows:

Each party P_i uses the global public key pk and the opening value $[r_y]_{P_i}$ contained in their own CESS share to compute the ciphertext $e_{P_i} = pk^{[r_y]_{P_i}} \ mod \ p$, and generates an NIZK proof $\pi(y_i)$ (using the same method as in Figure 4) to prove that the value $[r_y]_{P_i}$ in e_{P_i} and the value $[r_y]_{P_i}$ in $c_{y,i} = g^{[y]_{P_i}} h^{[r_y]_{P_i}}$ are the same (as the commitment value $c_{y,i}$ is publicly known among all parties, if a cheater tries to construct a false ciphertext and NIZK proof, the proof cannot pass the verification of honest parties). By combining $pk_i^{[r_m]_{P_j}}$ and $g^{[m]_{P_j}} h^{[r_m]_{P_j}}$ together, the ciphertext under the Twisted-ElGamal encryption scheme can also be obtained:

$$Enc([y]_{P_i}) = (e_{P_i}, c_{y,i}) = (pk^{[r_y]_{P_i}}, g^{[y]_{P_i}} h^{[r_y]_{P_i}}) \tag{6}$$

where $[y]_{P_i}$ refers to the plaintext message, and pk is the public key.

If a cheater P_k publicly reveals a tampered final share in the subsequent output reconstruction step, honest parties can decrypt $Enc([y]_{P_k})$ to obtain their true final share. Each party P_i sends $(e_{P_i}, \pi(y_i))$ to all other parties, and verifies the NIZK proof from other parties. If all proofs pass the verification, then continue to perform the output reconstruction steps.

(3) **Output reconstruction and decryption.** When all parties have computed the final CESS share, they publicly reveal $([y]_{P_i}, [r_y]_{P_i})$, and compute the output result $y = \sum_{i=1}^{n} [y]_{P_i}$. In the ideal case, all parties successfully reconstruct the output and obtain the output result y, and the protocol ends.

However, if the CESS.Rec protocol returns a list of cheaters L_{cheat}, it means that honest parties have not obtained the final output result (as the cheater publicly revealed a tampered final share). Supposing L_{cheat} includes the cheater P_c (actually, L_{cheat} may contain multiple cheaters, but we will take one cheater as an example), then honest parties need to perform additional decryption to obtain the true final share $([y]_{P_c})_{real}$ of P_c, and use $([y]_{P_c})_{real}$ to perform a second output reconstruction to obtain the output result y. The process is described as follows:

Honest party P_i must first combine the ciphertext e_{P_c} published by cheater P_c in the "Computation and Encryption" step with the commitment value $c_{y,c}$ contained in their own CESS share $< y >_{P_i}$, to form a Twisted-ElGamal type ciphertext of $([y]_{P_c})_{real}$:$Enc(([y]_c)_{real}) = (e_{P_c}, c_{y,c})$, and then decrypt it. The corresponding public key of the ciphertext $Enc(([y]_c)_{real})$ is the global public key pk, and the corresponding private key is $sk = \sum_{i=1}^{n} sk_i$. However, cheater P_c will not cooperate with honest parties by disclosing his private key sk_c, because P_c is trying to break the fairness of the protocol, and he has no reason to provide assistance to honest parties. However, in the offline stage, each party had shared their private key by (t,n)-verifiable secret sharing among n parties. Therefore, all honest parties (whose number is greater than the threshold t) can reconstruct sk_c by using Lagrange polynomial interpolation. After reconstructing sk_c, all honest parties calculate $sk = \sum_{i=1, i\neq c}^{n} sk_i + sk_c$ and decrypt the ciphertext $Enc(([y]_c)_{real})$ with sk to obtain $([y]_{P_c})_{real}$, and then perform a second reconstruction: $y = \sum_{i=1, i\neq c}^{n} y_{P_i} + ([y]_{P_c})_{real}$ to obtain the output result y, and the protocol ends.

5 Publicly Auditable MPC

The key to achieving publicly auditability in this scheme lies in the hiding, binding, and additive homomorphic properties of Pedersen commitment. During the computation process of the protocol, each participating party P_i not only computes shares $\{[x_j]_{P_i}\}_{j\in\{1,\ldots,n\}}$ according to the computation circuit $\mathcal{C}$, but also the corresponding Pedersen commitment $\{c_{x_{j,1}}, \ldots, c_{x_{j,n}}\}_{j\in\{1,\ldots,n\}}$ for each share simultaneously. In the output reconstruction step, the consistency between the decommitment values $\{[y]_{P_i}\}_{j\in\{1,\ldots,n\}}$ and the corresponding Pedersen commitments $\{c_{y,j}\}_{j\in\{1,\ldots,n\}}$ is verified to determine whether a cheater has tampered with the final share. Therefore, in this scheme, the Pedersen commitments corresponding to each party's private input share and some public values generated in the computation process are provided to the auditor (This does not disclose the party's private input for the hiding property of Pedersen commitments). The auditor can also obtain the Pedersen commitment values $\{c_{y,j}\}_{j\in\{1,\ldots,n\}}$ of the final shares by computing circuit $\mathcal{C}$ (as each party did in the protocol), and then verify the consistency of the final shares $\{[y]_{P_i}\}_{j\in\{1,\ldots,n\}}$ and the corresponding decommitment values $\{[r_y]_{P_i}\}_{j\in\{1,\ldots,n\}}$ published by each party. If the verification passes, the auditor claims that the output result y of the protocol is correct. The data provided to the auditor is called the *public record* τ and exists in a bulletin board in this scheme. Specifically, *publicly auditability* means that given the public record τ, the computation circuit $\mathcal{C}$, and the assumed output result y of the protocol, the auditor can verify whether the result y is correct. The definition of auditable correctness is as follows:

Definition 1. Auditable Correctness[15]: Assuming $\mathcal{C}$ represents a computation circuit, $(x_1, \ldots, x_n)$ represents n inputs to be computed by $\mathcal{C}$, and τ represents public records generated during the process of computing circuit $\mathcal{C}$. The protocol satisfies auditable correctness if the following conditions hold:

(1) The auditor outputs y (except with negligible probability) if $\mathcal{C}(x_1, ..., x_n) = y$, and τ is a public record obtained by computing circuit $\mathcal{C}$ with inputs $x_1, ..., x_n$.

(2) The auditor rejects y (except with negligible probability) if $\mathcal{C}(x_1, ..., x_n) \neq y$, or τ is not a public record obtained by computing circuit $\mathcal{C}$ with inputs $x_1, ..., x_n$.

Precondition: The auditor has obtained the public record τ of the protocol and outputs the result y, the final shares $\{[y]_{P_i}\}_{i\in\{1,...,n\}}$ and decommitment values $\{[r_y]_{P_i}\}_{i\in\{1,...,n\}}$.

Normal Reconstruction: The auditor uses the commitment values $\{c_{x_i,1}, ..., c_{x_i,n}\}_{i\in\{1,...,n\}}$ and public values of the multiplication operation $\{\phi_i, \delta_i\}_{i\in\{1,...,m\}}$ in the public record τ.

1. The auditor calculates the commitment values $\{c_{y,i}\}_{i\in n}$ of each participant's final share according to the calculation circuit $\mathcal{C}$.
2. Let the cheater list be $L_{cheat} = \{\emptyset\}$.
3. For $i \in \{1, ..., n\}$, verify if $pc([y]_{P_i}, [r_y]_{P_i}) = c_{y,i}$ holds. If it does not hold, add P_i to L_{cheat}.
4. If L_{cheat} is empty, calculate $y = \sum_{i\in\{1,...,n\}}[x]_{P_i}$ and the output (**accept**, y).
5. Otherwise, continue with the steps of *SecondReconstruction*.

Second Reconstruction: At this point, the auditor has obtained the cheater list L_{cheat} and identified the cheater P_c. The auditor also obtained the public values $\{pk_i\}_{i\in\{1,...,n\}}$, $\{sk_i\}_{i\in\{1,...,n\}}$, e_{P_c} and $\pi(y_c)$ from the public record τ.

1. Verify the NIZK proof $\pi(y_c)$.
2. Verify if $\{pk_i = h^{sk_i}\}_{i\in\{1,...,n\}}$ is true.
3. If both of the above verifications pass, calculate the private key $sk = \sum_{i=1}^{n} sk_i$.
4. Decrypt the ciphertext $(e_{P_c}, c_{y,c})$ using sk under the Twisted-ElGamal encryption scheme to obtain the true final share of the cheater P_c: $([y]_{P_c})_{real}$, and calculate $y' = \sum_{i=1,i\neq c}^{n}[y]_{P_i} + ([y]_{P_c})_{real}$.
5. If $y' = y$, the auditor outputs (**accept**, y). Otherwise, the auditor outputs (**reject**).

Fig. 7. Publicly auditable protocol: *Audit*

The contents of the public records τ in our scheme are shown as follows:

$$\tau = \begin{cases} pp = (G, p, g, h), \{c_{x_{i,1}}, ..., c_{x_{i,n}}\}_{i\in\{1,...,n\}} \\ \{\varphi_i, \delta_i\}_{i\in\{1,...,m\}}, \{pk_i\}_{i\in\{1,...,n\}} \\ \{e_{p_i}\}_{i\in\{1,...,n\}}, \{\pi(y_i)\}_{i\in\{1,...,n\}} \end{cases} \tag{7}$$

In the above, $pp = (G, p, g, h)$ is globally public parameter; $\{c_{x_{i,1}}, ..., c_{x_{i,n}}\}_{i\in\{1,...,n\}}$ is the Pedersen commitment values of n shares of each party's private input; $\{\varphi_i, \delta_i\}_{i\in\{1,...,m\}}$ are public values generated during multiplication operations (m represents the number of multiplication gates in computation circuit $\mathcal{C}$); $\{pk_i\}_{i\in\{1,...,n\}}$ are public keys generated by each party in the offline phase; $\{e_{p_i}\}_{i\in\{1,...,n\}}$ and $\{\pi(y_i)\}_{i\in\{1,...,n\}}$ are respectively the ciphertexts and corresponding NIZK proofs published by each party during the "Computation and

Encryption" step in the online phase. What's unique is that if honest parties perform the "Decryption" step in the protocol, they also need to add their own private key sk_i and the cheater's private key sk_c to the bulletin board, i.e. public records τ also contain $\{sk_i\}_{i\in\{1,...,n\}}$.

Specifically, the output results that need to be audited in this protocol are divided into two cases. The first is the output result of normal execution when no cheaters attempt to break the fairness of the protocol. The second is the output result obtained through "Second Reconstruction" by honest parties when a cheater attempts to break the fairness of the protocol during output reconstruction step. The publicly auditable protocol *Audit* is shown in Figure 7.

6 FAMC Protocol with Cheater Detection

We proposed a fair and auditable secure multiparty computation protocol (FAMC) that can not only detect dishonest behavior, but also identify cheaters. The $\pi_{FAMC}^{\mathcal{F}_{SETUP}}$ protocol is as follows:

(1) **Init step**: Receive input $(Init, \mathcal{C}, n, p)$ from all participating parties (where $\mathcal{C}$ is a computation circuit with n inputs composed of addition gates and multiplication gates in the $\mathbb{Z}_p$ field, and p is a large prime number). Then execute:

1) P_i sends $(Setup, p)$ to $\mathcal{F}_{SETUP}$ to obtain the global public parameter pp (which is sent to the bulletin board).

2) P_i sends $(RandomValues, n)$ to $\mathcal{F}_{SETUP}$ to obtain random numbers in CESS form that used to distribute private inputs of each party.

3) P_i sends $(Triples, m)$ to $\mathcal{F}_{SETUP}$ to obtain Beaver triples in CESS form used to calculate multiplication gates.

4) P_i executes the $KeyGen$ protocol (Figure 3) to obtain the private key sk_i, public key pk_i, global public key pk, and the secret sharing of private keys of all parties $\{[sk_j]_{P_i}\}_{j\in\{1,...,n\}}$ (the public key pk_i is sent to the bulletin board).

(2) **Input step**: P_i prepares to provide x_i as input to the protocol:

1) In order to distribute private input x_i, P_i privately builds a random number shared in CESS form using the CESS.PrivRec protocol (Figure 6) and obtains the reconstruction result m.

2) P_i broadcasts $\varepsilon = x_i - m$ (sent to all other participating parties and the bulletin board).

3) All participating parties P_j use the operation rules to locally calculate $< x_i >_{P_j} = < m + \varepsilon >_{P_j}$ (the commitment $(c_{x_{i,1}}, ..., c_{x_{i,n}})$ of x_i shares needs to be sent to the bulletin board).

(3) **Eval step**: If all parties' private inputs x_i have been shared among n parties in the *Input* step, P_i calculates circuit $\mathcal{C}$ gate-by-gate according to the operation rules:

1) Addition: For CESS shares $< x_1 >_{P_i}$ and $< x_2 >_{P_i}$, P_i locally calculates $< t >_{P_i} = < x_1 + x_2 >_{P_i}$.

2) Constant addition: For CESS share $< x_1 >_{P_i}$ and constant ϵ, P_i locally calculates $< t >_{P_i} = < x_1 + \epsilon >_{P_i}$.

3) Constant multiplication: For CESS share $< x_1 >_{P_i}$ and constant ϵ, P_i locally calculates $< t >_{P_i} =< x_1\epsilon >_{P_i}$.

4) Multiplication: For CESS shares $< x_1 >_{P_i}$ and $< x_2 >_{P_i}$, a set of Beaver triples (<a>,<b>,<c>) shared in CESS form needs to be consumed. P_i needs to interact with other participating parties to calculate $< t >_{P_i} =< x_1 x_2 >_{P_i}$ (the public values $\{\varphi_j, \delta_j\}_{j\in\{1,...,m\}}$ generated during the interaction, need to be sent to the bulletin board).

5) After completing all computations of circuit $\mathcal{C}$, P_i obtains its final CESS share $< y >_{P_i}$.

(4) **Encrypt step**: The final CESS share of P_i is represented as follows: $< y >_{P_i} = ([y]_{P_i}, [r_y]_{P_i}, c_{y,1}, ..., c_{y,n})$. The cheater P_{c1} and P_{c2} are prepared to provide the wrong final CESS share to the protocol as input:

1) The cheaters are divided into two groups, one group is P_{c1}, P_{c1} prepares the random number $[y]_{P_c}$ as the public final share, and uses $([y]_{P_c})_{real}$ to generate the ciphertext. Another group P_{c2} use the correct final share $[y]_{P_i}$ but uses randomly generated random numbers r_1 and r_2 as the generated ciphertext.

2) The first group of cheaters P_{c1} calculates the zero-knowledge proof $\pi(P_{c1})$ and ciphertext $(pk^{[r_y]_{P_i}}, g^{([y]_{P_c})_{real}} h^{[r_y]_{P_i}})$. The second group of cheaters P_{c2} also calculates the ciphertext $(pk^{r_1}, g^{[y]_{P_i}} h^{r_2})$ and zero-knowledge proof $\pi(P_{c2})$. These ciphertexts and zero-knowledge proofs are sent to all parties as well as to the bulletin board.

3) P_i uses the global public key pk and $[r_y]_{P_i}$ to calculate the ciphertext $e_{P_i} = pk^{[r_y]_{P_i}}$ and generates a zero-knowledge proof $\pi(y_i)$ (the proof method is the same as that used in Figure 4) to prove that $[r_y]_{P_i}$ in e_{P_i} and $[r_y]_{P_i}$ in the commitment $c_{y,i} = g^{[y]_{P_i}} h^{[r_y]_{P_i}}$ are the same. Then e_{P_i} and $\pi(y_i)$ are sent to all participating parties and the bulletin board.

4) P_i sets $L_{cheat} = \{\emptyset\}$ to maintain a list of cheaters. For $j \in \{1, ..., n\}$, P_i verifies the zero-knowledge proof $\pi(y_j)$ and adds any participating party who fails verification to L_{cheat}.

(5) **Output step**: If L_{cheat} is empty in the previous step, all parties execute the CESS.Rec protocol to start reconstruct the output result. There are two possible results:

1) Successful reconstruction: All parties obtain the output result y, and the protocol ends.

2) Failed reconstruction: All honest parties obtain the list of cheaters and continue to execute *Decrypt*.

(6) **Decrypt step**: All honest parties identify the cheating party P_c.

1) Each honest party P_i publicly releases their secret share of the private key sk_c for P_c: $\{[sk_c]_{P_i}\}_{i\in\{1,...,n\}, i\neq c}$.

2) Each honest party P_i then uses Lagrange interpolation to reconstruct sk_c and computes the total private key $sk = \sum_{i=1, i\neq c}^{n} sk_i + sk_c$. They send all of their private keys $\{sk_i\}_{i\in\{1,...,n\}}$ to the bulletin board for future auditing.

3) Each honest party P_i computes the decryption key sk for the Twisted-ElGamal encryption and uses it to decrypt the ciphertext $(e_{P_c}, c_{y,c})$ for P_c, obtaining the real final share $([y]_{P_c})_{real}$ of P_c. Then they reconstruct the output y

as $y = \sum_{i=1,i\neq c}^{n}[y]_{P_i} + ([y]_{P_c})_{real}$. At this point, the honest parties have obtained the correct output y and the protocol ends.

(7) **Audit step**:

1) If an auditor wishes to verify the correctness of the output y, he/she must obtain the public record τ from the bulletin board and execute the auditing protocol shown in Figure 7.

2) For the first group cheaters P_{c1}: The Pedersen commitment corresponding to the share of each parties private input is obtained from the public record τ and some public values generated during the computation (this does not disclose the participant's private input due to the hidding property of the Pedersen commitment). The auditor can also compute the circuit $\mathcal{C}$ to obtain the Pedersen commitment value $\{c_{y,j}\}_{j\in\{1,\ldots,n\}}$ of the final share (just as the parties in the protocol did), and then conduct a consistency test with the final shares $\{[y]_{P_i}\}_{j\in\{1,\ldots,n\}}$ and the release commitment values $\{[r_y]_{P_i}\}_{j\in\{1,\ldots,n\}}$ corresponding to each participant, identify the cheaters P_{c1} who do not pass the consistency test, and add P_{c1} to L_{cheat}.

3) For the second group of cheaters P_{c2}: obtain ciphertext e_{P_i} and zero-knowledge proof $\pi(y_i)$ from public record τ, and the auditor verifies the zero-knowledge proof, detects the second group of cheaters P_{c2} using different random numbers, and adds P_{c2} to L_{cheat}.

4) If only P_{c1} exists, the private key of P_{c1} is reconstructed using the Lagrange interpolation in the *Decrypt* step, achieving fairness that all parties get the correct result. Once P_{c2} exists, all cheaters will be added to the cheater list L_{cheat}, the cheater list will be published and the protocol will be aborted, so as to realize the fairness that all participants cannot get the result.

7 Security Analysis

We have proven the computational security of the $\pi_{FAMC}^{\mathcal{F}_{SETUP}}$ protocol under assumption of honest majority. This means that for any probabilistic polynomial-time (PPT) adversary, the probability of successfully breaking the protocol within a reasonable amount of time is negligible. Specifically, for any PPT adversary $\mathcal{A}$, there exists an ideal world simulator $\mathcal{S}$ such that the $\pi_{FAMC}^{\mathcal{F}_{SETUP}}$ protocol is computationally indistinguishable from the ideal function $\mathcal{F}_{FAMC}$ under polynomial-time environment entity $\mathcal{Z}$.

Theorem 1. *[21] Twisted ElGamal is IND-CPA secure (1-plaintext/2-recipient) based on the divisible DDH assumption.*

To be note that, the divisible DDH assumption is equivalent to the standard DDH assumption[22].

Theorem 2. *Assuming that the Decisional Diffie-Hellman problem (DDH) is hard in group G, the protocol $\pi_{FAMC}^{\mathcal{F}_{SETUP}}$ with oracle access to the functionality $\mathcal{F}_{SETUP}$ is a UC-secure implementation of the functionality $\mathcal{F}_{FAMC}$.*

8 Experiment and Analysis

The implementation is using Python (3.11.0) and MPyC version v0.8. The experiment simulates the process of multiple parties using the FAMC protocol to perform secure summation computation, with the predefined computation function set as the sum function, where each party's private input is an integer x_i and the output result is the sum of all parties' private inputs y. Table 1 provides an analysis of the time complexity of a single party and the communication complexity of all parties, in which the experimental parameters include the number of parties n, the number of cheaters c, and the highest power of Lagrange polynomials t. As shown in Figure 8, the time complexity of the decryption algorithm in the secondary reconstruction is directly proportional to the number of cheaters. Figure 9 shows that the communication complexity of the protocol is exponentially related to the number of cheaters.

In addition, the experiment instantiated the 5-party summation scenario and recorded the running time of each module, as shown in comparison with the 3-party summation instance in Figure 10. The experiment further compares and analyzes the FAMC protocol with some related works in terms of the properties (Table 2) and online running efficiency (Figure 11).

Table 1. Time complexity and communication complexity analysis

Modules	Distribution	Computation	Encryption	Reconstruction	Decryption(If there are cheaters)
Time complexity	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(c)$
Communication complexity	$O(n^2)$	$--$	$O(n^2)$	$O(n^2)$	$O(ct)+O(n)$

Table 2. Characteristics comparison of related works

Scheme	Cheating Detection	Cheater Detection	Fairness	Public Auditability	Parties Assumption
[4]	√	×	×	×	Dishonest majority
[25][26]	√	√	√	×	Dishonest majority
[15]	√	×	×	√	Dishonest majority
FAMC	√	√	√	√	Honest majority

9 Conclusion

This paper extends SPDZ framework and proposes an MPC protocol FAMC that satisfies cheater detection, output fairness and public auditability. This protocol

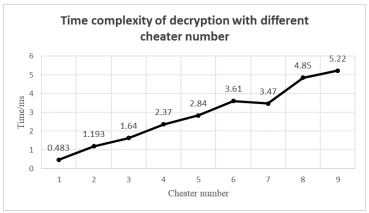

Fig. 8. Time complexity of decryption with different party number

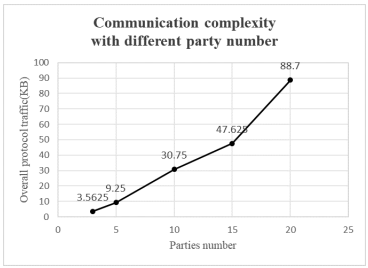

Fig. 9. Communication complexity with different party number

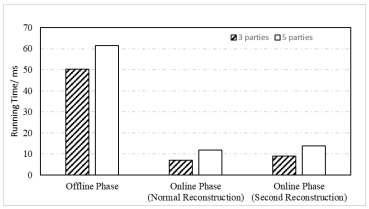

Fig. 10. The running time of each module for 3-party and 5-party instances

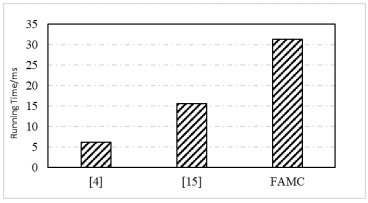

Fig. 11. The running time of different schemes in the online phase

enables honest parties to detect *all* cheaters, and the fairness of the protocol can be guaranteed. At the same time, *any* external third-party auditor can verify the computation results of the protocol. The experimental results indicate that the FAMC protocol has reasonable operational efficiency. Finding new building blocks to construct such protocol to balance security and efficiency, implementing the protocol with more complex operations and explore its performance across real-world scenarios are future research directions.

Acknowledgements Xi Chen's work is partially supported by the National Key Research and Development Program of China (2023YFB2704805) and Dawei Zhang's work is supported by the National Natural Science Foundation of China (U21A20463). The authors wish to thank Jianliang Zhou and Xiaomei Yan who help with formatting the manuscript.

References

1. Andrew Chi-Chih Y. Protocols for Secure Computations (Extended Abstract). FOCS'82. IEEE, 1982. 160-164. https://doi.org/10.1109/SFCS.1982.38
2. Lindell Y. Secure multiparty computation. Communications of the ACM, 2020, 64(1): 86-96. https://doi.org/10.1145/3387108
3. Goldreich, O., Micali, S., Wigderson, A. How to play any mental game or a completeness theorem for protocols with honest majority. STOC'87. ACM, 1987. 218-229. https://doi.org/10.1145/28395.28420

4. Damgård I, Pastro V, Smart N, et al. Multiparty computation from somewhat homomorphic encryption. Advances in Cryptology–CRYPTO 2012. LNCS 7417, Springer, 2012. 643-662. https://doi.org/10.1007/978-3-642-32009-5_38
5. Damgård I, Keller M, Larraia E, et al. Practical covertly secure MPC for dishonest majority-or: breaking the SPDZ limits. ESORICS 2013. LNCS 8134, Springer, 2013. 1-18. https://doi.org/10.1007/978-3-642-40203-6_1
6. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S. Semi-homomorphic encryption and multiparty computation. Advances in Cryptology–EUROCRYPT 2011. LNCS 6632, Springer, 2011. 169-188. https://doi.org/10.1007/978-3-642-20465-4_11
7. Baum, C., Orsini, E., Scholl, P. Efficient secure multiparty computation with identifiable abort. Theory of Cryptography. TCC 2016. LNCS 9985, Springer, 2016. 461ÍC490. https://doi.org/10.1007/978-3-662-53641-4_18
8. Ishai, Y., Ostrovsky, R., Zikas, V. Secure multi-party computation with identifiable abort. Advances in Cryptology–CRYPTO 2014, LNCS 8617, Springer, 2014. 369-386. https://doi.org/10.1007/978-3-662-44381-1_21
9. Cunningham, R., Fuller, B., Yakoubov, S. Catching MPC cheaters: Identification and openability. International Conference on Information Theoretic Security–ICITS 2017. LNCS 10681, Springer, 2017. 110-134. https://doi.org/10.1007/978-3-319-72089-0_7
10. Cohen J D, Fischer M J. A robust and verifiable cryptographically secure election scheme. SFCS'85. IEEE Computer Society, 372-382. https://doi.org/10.1109/SFCS.1985.2
11. Chaum, D., Ryan, P.Y.A., Schneider, S. A practical voter-verifiable election scheme. ESORICS 2005, LNCS 3679, Springer, 2005. 118-139. https://doi.org/10.1007/11555827_8
12. Sako K. An auction protocol which hides bids of losers. PKC 2000, LNCS 1751, Springer, 2000. 422-432. https://doi.org/10.1007/978-3-540-46588-1_28
13. Hoogh, de, S. J. A. Design of large scale applications of secure multiparty computation: secure linear programming. PhD dissertation, Technische Universiteit Eindhoven, The Netherlands, 2012. https://doi.org/10.6100/IR735328
14. Cramer R., Damgård I., Nielsen J. B. Multiparty computation from threshold homomorphic encryption. Advances in Cryptology–EUROCRYPT 2001. LNCS 2045, Springer, 2001. 280-300. https://doi.org/10.1007/3-540-44987-6_18
15. Baum C., Damgård I., Orlandi C. Publicly auditable secure multi-party computation. SCN 2014, LNCS 8642, Springer, 2014. 175-196. https://doi.org/10.1007/978-3-319-10879-7_11
16. Goldreich O. Foundations of Cryptography: Volume II - Basic Applications. Cambridge University Press, 2004.
17. Chaum D., Crépeau C., Damgard I. Multiparty unconditionally secure protocols. STOC'88. ACM Press, 1988. 11–19. https://doi.org/10.1145/62212.62214
18. Rabin, T., Ben-Or, M. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. STOC'89, ACM Press,1989. 73–85. https://doi.org/10.1145/73007.73014
19. Cleve R. Limits on the security of coin flips when half the processors are faulty. STOC'86, ACM Press, 1986. 364–369. https://doi.org/10.1145/12130.12168
20. Pedersen, Torben P. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. Advances in Cryptology–CRYPTO'91. LNCS 576, Springer, 1991. 129-140. https://doi.org/10.1007/3-540-46766-1_9
21. Chen, Y., Ma, X., Tang, C., Au, M.H. PGC: Decentralized confidential payment system with auditability. ESORICS 2020. LNCS 12308, Springer, 2020. 591-610. https://doi.org/10.1007/978-3-030-58951-6_29

22. Camenisch J, Stadler M. Efficient group signature schemes for large groups. Advances in Cryptology–CRYPTO'97. LNCS 1294, Springer, 1997. 410-424. https://doi.org/10.1007/BFb0052252
23. Fiat A., Shamir A. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. Advances in Cryptology–CRYPTO'86. LNCS 263, Springer, 1986. 186-194. https://doi.org/10.1007/3-540-47721-7_12
24. Beaver D. Efficient multiparty protocols using circuit randomization. Advances in Cryptology–CRYPTO'91. LNCS 576, Springer, 1992. 420-432. https://doi.org/10.1007/3-540-46766-1_34
25. Seo, M. Fair and Secure Multi-Party Computation with Cheater Detection. Cryptography 2021, 5, 19. https://doi.org/10.3390/cryptography5030019
26. Alper, Handan Kılınç, and Alptekin Küpçü. Optimally efficient multi-party fair exchange and fair secure multi-party computation. ACM Transactions on Privacy and Security, ACM. 2021,25(1): 1-34. https://doi.org/10.1145/3477530
27. Jin, S., Li, Y., Chen, X., Li, R. Blockchain based Publicly Auditable Multi-party Computation with Cheater Detection. ICICS 2023, LNCS 14252, Springer. 608-626. https://doi.org/10.1007/978-981-99-7356-9_36
28. Kumaresan R., Bentov I. How to Use Bitcoin to Incentivize Correct Computations. CCS'14. ACM. 2014. 30-41. https://doi.org/10.1145/2660267.2660380
29. Baum C., Melissaris N., Rachuri R., Scholl P. Cheater Identification on a Budget: MPC with Identifiable Abort from Pairwise MACs. Cryptology ePrint Archive, Report 2023/154, 2023. https://eprint.iacr.org/2023/1548.
30. L. Nie, S. Yao, J. Liu. Secure Multiparty Computation with Identifiable Abort and Fairness. 2023 7th International Conference on Cryptography, Security and Privacy (CSP 2023). IEEE, 2023. 99-106. https://doi.org/10.1109/CSP58884.2023.00023