A Scheme of Robust Privacy-preserving Multi-Party Computation via Public Verification

Keke Gai, *Senior Member, IEEE,* Dongjue Wang, Jing Yu, *Member, IEEE,* Liehuang Zhu, *Senior Member, IEEE,* and Weizhi Meng, *Senior Member, IEEE*

***Abstract*—Multi-Party Computation (MPC), as a distributed computing paradigm, is considered to be a potential solution for providing privacy-preserving for applications following the client-server model. However, traditional MPC solutions cannot satisfy the publicly verifiable requirement of the client-server model. In this paper, we propose a blockchain-based verifiable MPC solution using Pedersen's threshold secret sharing and Lifted ElGamal encryption. We first build a data distribution method using Pedersen's threshold secret sharing and symmetric encryption to protect the privacy of inputs while ensuring robustness. Then, we propose a result processing algorithm using Lifted ElGamal encryption to safeguard the privacy of the outputs. Finally, we employ non-interactive zero-knowledge proof and Pedersen commitment to publicly verify the correctness of the encrypted outputs in the smart contract, enabling the detection of malicious parties. Theoretical analysis indicates that the proposed method can publicly verify the correctness of outputs without revealing plain-text inputs and outputs, which satisfy the privacy-preserving requirements of the client-server model. Experimental evaluations have demonstrated that our proposed approach is efficient while achieving stronger privacy.**

***Index Terms*—Multi-party computation, blockchain, public verification, robustness, privacy-preserving.**

I. Introduction

PRIVACY concern is one of the major restrictions for temporary data owners to implement collaborative computations, even though merits of data sharing are well known by the public. As a distributed computation paradigm, *Multi-Party Computation* (MPC) is deemed to be an option for supporting collaborative computations among untrustworthy parties, as raw data are not directly shared between various parties [1], [2]. MPC participants can only obtain output results and intermediate values without releasing their own private inputs. Implementations of MPC are adopted in a number of domains, such as electronic auctions [3]–[6], e-voting [7]–[9], and machine learning [10], [11].

K. Gai, D. Wang and L. Zhu are with School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, 100081, China. Emails: {gaikeke,3220231818,liehuangz}@bit.edu.cn.

J. Yu is with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China, 100093. Email: yujing02@iie.ac.cn.

W. Meng is with Department of Applied Mathematics and Computer Science Cybersecurity Engineering, Technical University of Denmark, Kongens Lyngby, Denmark. Email: weme@dtu.dk.

This work is partially supported by the National Key Research and Development Program of China (Grant No. 2021YFB2701300), National Natural Science Foundation of China (Grant No. 62372044), National Defense Basic Scientific Research program of China under grant number JCKY2020602B008.

Corresponding author: Jing Yu (yujing02@iie.ac.cn)

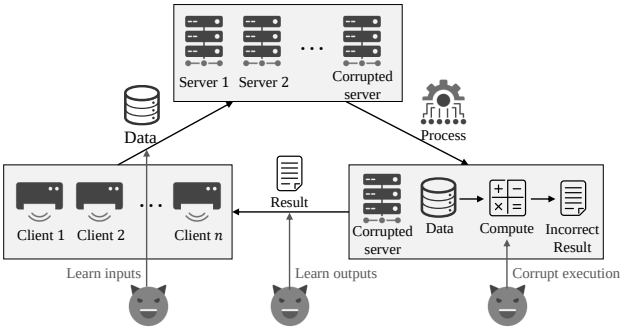


Fig. 1: Security threats of applications that follow the client-server model.

However, in the scenario of the client-server model, implementing MPC schemes encounters restrictions since a higher-level privacy-preserving scheme is required [12]. Fig. 1 displays potential security threats when applying traditional MPC schemes in the client-server model, in which servers are responsible for running MPC protocols and clients provide servers with data for obtaining outputs. The client-server mode needs to avoid servers learning plain-text inputs and outputs, while clients need to verify whether outputs are correct. Thus, additional protections are required for satisfying the security requirement of the client-server mode.

Homomorphic encryption is a promising way to implement privacy-preserving applications [13], [14] in the client-server model. To preserve the privacy of electricity consumption data, Zhan et al. [13] proposed a data aggregation scheme using the EC-ElGamal encryption algorithm with a double trapdoor decryption mechanism. To prevent the control center and fog nodes from learning the plain-text input or output, this scheme utilizes homomorphic encryption properties to perform ciphertext operations, matching the client-server model's partial security requirements. However, this scheme cannot tolerate collusion between the control center and fog nodes. Furthermore, the control center cannot verify whether the fog nodes have performed the correct computation. To address the issue of verifiable computation results, Shen et al. [14] proposed a verifiable statistical analysis scheme using threshold homomorphic encryption [15] and Shamir's threshold secret sharing [16]. The E-commerce platform service provider can verify the correctness of the computation results of cloud servers because of the verifiability of threshold homomorphic encryption. However, when the computational task fails, the E-commerce platform service provider cannot

determine which cloud server submitted the invalid result share. Moreover, this scheme lacks a public and transparent platform for result verification. The servers executing the MPC protocol perform computations independently. It is essential to verify the correctness of the computation results. It is a challenge to publicly verify the correctness of computation results in the client-server model without leaking privacy.

To overcome this challenge, we propose a verifiable MPC for client-server model, which utilizes the blockchain [17], [18] as a transparent auditing platform. Blockchain has been widely used in data auditing [19], [20] due to its transparency, immutability and scalability [21], [22]. We construct an MPC protocol using Pedersen's verifiable threshold secret sharing [23], supporting addition and multiplication operations while achieving robustness. Specifically, the threshold parameter t determines the number of servers required to reconstruct the shared secret. Therefore, even if up to $n - t$ servers are malicious, the remaining servers can still reconstruct the secret. Furthermore, to address privacy leakage concerns in the computational result verification process, we propose a privacy-preserving verification algorithm using Lifted ElGamal encryption [24], Pedersen commitment [23], and non-interactive zero-knowledge proof, which can ensure the correctness and privacy of the output results. Lifted ElGamal encryption integrates well with Pedersen commitment because of the additive homomorphic properties.

The main contributions are summarized as follows.

1) We propose a verifiable and robust MPC method, which leverages blockchain as a transparent auditing platform. We store the intermediate values of servers in the blockchain for traceability of malicious server behavior. Furthermore, we deploy the verification algorithm in the smart contract, which enables the automatic detection of malicious servers to protect the protocol execution.
2) To verify the correctness of results without revealing the plain-text outputs, we propose a privacy-preserving verification algorithm. Different from previous schemes, the proposed algorithm enables clients to identify whether each server has submitted incorrect output results without leaking privacy, which satisfies the security properties of the client-server model.
3) Theoretical analysis shows that the proposed method can achieve more comprehensive privacy protection than the previous schemes. Experiments demonstrate that the proposed method is efficient in many ways, including the computation overhead of clients and the communication overhead in the result verification phase.

The rest of this paper is organized as follows. Related work is given in Section II. Preliminaries are given in Section III. In Section IV, we provide detailed description about the proposed model. Sections V and VI present the security analysis and experimental evaluations, respectively. Finally, conclusions of this work are drawn in Section VII.

II. Related Work and Comparison

MPC was a crucial cryptography technology that was relevant for addressing data security and privacy. Many research works focused on constructing efficient MPC protocols using techniques such as garbled circuits and secret sharing. The application scenarios for secure two-party computation were limited because of the constraint on the number of participants. Goldreich et al. [28] proposed an MPC protocol utilizing the oblivious transfer protocol, which supported both boolean circuit and arithmetic circuit evaluations, extending the computation from two parties to multiple parties. Ben-Or et al. [29] constructed an MPC protocol using the Shamir's threshold secret sharing scheme [16]. This protocol utilized the homomorphic properties of Shamir's threshold secret sharing to perform secure computations on shares. However, applying this research in practical scenarios was challenging due to algorithmic complexity and efficiency limitations. The emergence of secure multiparty computation frameworks, such as MP-SPDZ [30], CrypTen [31], Cheetah [32], and Squirrel [33] had reduced the complexity of protocol development.

TABLE I: Comparison between the Proposed Scheme and Related Works

Scheme	Adversary	Ver.	Rob.	PoI	PoO	Tran.
[13]	Semi-honest	✗	✗	✓	✓	✗
[14]	Malicious	✓	✓	✓	✓	✗
[25]	Malicious	✓	✓	✓	✗	✓
[26]	Malicious	✓	✓	✓	✗	✓
[27]	Malicious	✓	✗	✓	✓	✓
Ours	Malicious	✓	✓	✓	✓	✓

"Ver." verifiable. "Rob." robust. "PoI" privacy of inputs. "PoO" privacy of outputs. "Tran." transparent result verification process.

A. Blockchain-based Multi-Party Computation

To satisfy the public audit requirements, many researchers [25], [34]–[36] focused on blockchain, which provided a transparent platform for MPC. Bentov et al. [34] proposed a fair MPC model on the Bitcoin network using a monetary penalty mechanism. The claim-or-refund mechanism contributed to the design of fair protocols and could be extended to more blockchain systems. Subsequently, Kumaresan et al. [35] improved upon Bentov's scheme [34] by enhancing the efficiency of the computation protocol in a hybrid model, which reduced the complexity of the claim-or-refund protocol script. Gao et al. [25] designed a fair MPC scheme using game theory and smart contracts, which combined a reputation system with a deposit and penalty mechanism to ensure fairness. However, this scheme [25] does not consider the privacy of the output results. In contrast to the approach of ensuring fairness through monetary penalties, Choudhuri et al. [36] transformed the fairness problem into a fair decryption problem and utilized witness encryption and the blockchain to execute fair decryption processes, which achieved a fair MPC scheme for general functions. Blockchain provided a transparent distributed platform, which satisfied the public audit requirements. When most participants were dishonest, blockchain-based MPC could ensure the fairness of transactions [37]. However, executing MPC on a blockchain platform brought the risk of privacy leaks thanks to its inherent transparency. To address privacy leakage concerns, Yang et

al. [27] utilized zero-knowledge proof techniques to construct an on-chain verifiable protocol, safeguarding privacy in the industrial Internet of Things. However, this scheme [27] is not robust enough to guarantee the normal execution of the protocol when there are malicious nodes.

B. Homomorphic Encryption-based Multi-Party Computation

Many research works [13], [14], [26] used homomorphic encryption to construct MPC protocols to satisfy the privacy-preserving requirements of the client-server model, which protected data confidentiality. Zhan et al. [13] proposed a data aggregation protocol using EC-ElGamal encryption for application in the smart grid, which ensured the confidentiality of private electricity data. However, this scheme did not consider the case of malicious fog nodes and could not verify whether fog nodes were correctly executing the computation. To achieve verifiability of the computation process, Zhou et al. [26] proposed an on-chain MPC scheme using homomorphic encryption. The computation protocol was automatically executed by smart contracts, which ensured the verifiability of the computation process. To preserve privacy during the data input phase, participants shared their private data using the additive secret sharing scheme, and the corresponding secret shares were encrypted with the public keys of various participants. Similar to the method of Zhou et al. [26], Shen et al. [14] proposed a data aggregation method by combining secret sharing and homomorphic encryption, which addressed issues related to server single points of failure and lack of verification. However, there were limitations in detecting malicious server behavior in this method. Furthermore, the verification process of the method [14] lacks transparency Both of these methods [14], [26] were unable to verify the correctness of output results while ensuring the privacy of the output results. Table I summarizes the comparison with related works discussed above. We discuss the work in terms of four metrics, including verifiability, privacy of inputs, privacy of outputs, and transparency of the verification process. Verifiability ensures that the output results are correct. Transparency ensures fairness in the verification process. The privacy of inputs is a fundamental requirement for MPC. The privacy of outputs protects the interests of the data requester.

III. Preliminaries

A. Pedersen Commitment

The Pedersen commitment [23] is a non-interactive commitment scheme with excellent hiding, binding, and additive homomorphic properties. There are three phases used in our proposed method.

- Setup: Let a cyclic group $\mathbb{G}$ of prime order p, where g and h are generators in the cyclic group $\mathbb{G}$, with their discrete logarithmic relationship being unknown.
- Commit: The committer selects a random number $r \in \mathbb{G}$ and computes the commitment for message m as $c = C(m, r) = g^m h^r$. Then, the committer sends the commitment c to the verifier.
- Open: The committer sends message m and random number r to the verifier. Then, the verifier checks whether $g^m h^r$ equals c. The verifier refuses the commitment when they are not equal.

The additive homomorphic property of this method is as follows:

$$\begin{cases} C(m_1, r_1) \times C(m_2, r_2) = C(m_1 + m_2, r_1 + r_2), \\ C(m, r)^a = C(a \times m, a \times r). \end{cases} \tag{1}$$

B. Pedersen's Verifiable Threshold Secret Sharing

The Pedersen's threshold secret sharing [23] is a verifiable secret sharing scheme, which has additive homomorphism property. Concretely, the secret sharing scheme is comprised of three algorithms: secret sharing, secret reconstruction, and verification of shares.

- Secret sharing: To share a secret s with n participants P_i for $i = 1, 2, \ldots, n$, the dealer computes a commitment $E_0 = C(s, t)$ and constructs two polynomials $F(x) = s + F_1 x + \cdots + F_{t-1} x^{t-1}$ and $G(x) = t + G_1 x + \cdots + G_{t-1} x^{t-1}$ of degree $t-1$ satisfying $F(0) = s$ and $G(0) = t$, where $t \in Z_p$ is a random number. Then, the dealer computes $s_i = F(i)$, $t_i = G(i)$, and $E_j = C(F_j, G_j)$. Finally, the dealer sends (s_i, t_i) secretly to P_i where $i = 1, 2, \ldots, n$ and broadcasts E_j where $j = 0, 1, \ldots, t-1$.
- Secret reconstruction: When receiving t shares (s_i, t_i), the receiver can reconstruct the secret s:

$$s = \sum_{i=1}^{t} \Big(\prod_{j=1, j \neq i}^{t} \frac{-j}{i-j} \Big) s_i. \tag{2}$$

- Verification of shares: During the secret sharing phase, participants P_i receive their shares from the dealer and check whether $C(s_i, t_i)$ equals $\prod_{j=0}^{t-1} E_j^{i^j}$ to verify the validity of their secret shares. Furthermore, the receiver can verify the validity of the shares in the same manner during the secret reconstruction phase.

C. Lifted ElGamal Threshold Cryptosystem

The Lifted ElGamal threshold cryptosystem is a variant of ElGamal encryption [24] with additive homomorphic properties. Specifically, the Lifted ElGamal threshold cryptosystem is comprised of three parts: key generation, encryption, and decryption.

- Key generation: Let a prime p and a generator g in the group $\mathbb{Z}_p^*$. n players $P_1, P_2, \ldots, P_n$ select the private key $s_i (i = 1, 2, \ldots, n)$ and compute $k_i = g^{s_i} \bmod p$ independently. Finally, n parties jointly compute the public key $h = \prod_{i=1}^{n} k_i = \prod_{i=1}^{n} g^{s_i} = g^{\sum_{i=1}^{n} s_i} \bmod p$.
- Encryption: Choose a random number $r \in \mathbb{Z}_p^*$ and compute the ciphertext c for message $m \in \mathbb{Z}_p^*$ as $c = Enc(m) = (c_0, c_1) = (g^r, g^m h^r)$.
- Decryption: P_i computes $t_i = c_0^{s_i} \bmod p$. Then, n parties jointly compute $t = \prod_{i=1}^{n} t_i = \prod_{i=1}^{n} c_0^{s_i} = c_0^{\sum_{i=1}^{n} s_i} \bmod p$ and $g^m = c_1 / t \bmod p$ using private key $s_i (i = 1, 2, \ldots, n)$. Then, obtain the message m by querying a discrete logarithm table or solving the discrete logarithm problem.

IV. Our Proposed Model

A. Design Goals

To realize the security properties of the client-server model, it is necessary to ensure that our MPC method satisfies the following security requirements.

Privacy-Preserving: It is significant to emphasize that useful information is not leaked during the execution of computations and result verification. The proposed method should require the servers not to access the raw inputs and outputs. During the data distribution phase and the data calculation phase, no single server or subset of servers can collude to obtain the inputs of clients. Furthermore, the proposed method needs to ensure that the blockchain nodes and servers cannot access the original outputs during the result verification phase, even in the case of collusion.

Verifiability: Ensuring the verifiability of outputs is essential to prevent clients from receiving incorrect results. The proposed method should be able to verify whether each output submitted by servers is correct to detect malicious servers. Furthermore, the proposed method needs to ensure that the computation results are publicly verifiable, allowing both clients and servers to obtain verification of the results.

Robustness: In practical applications, assume that there is a likelihood of failures in computation tasks for offline servers, it is essential to enhance the robustness to tolerate server failures. Specifically, the proposed solution needs to be able to tolerate instances where a portion of the servers are offline during the data computation or the result processing phase while ensuring the normal functioning of the protocol. Furthermore, the client should still be able to reconstruct the correct outputs using the remaining correct output shares when some malicious servers submit incorrect output shares.

B. Threat Model

We assume the maximum amount of the malicious servers is $t-1$ out of n total servers, where t is the threshold of the secret sharing scheme and $n < 2t-1$. The adversary can control these $t-1$ fixed malicious server nodes during the task execution phase. Malicious servers try to infer the original data of the benign servers or cause the failure of the computation tasks by exchanging information with each other and submitting incorrect result.

All blockchain nodes and clients are considered semi-honest nodes. We assume that clients and blockchain nodes will not collude with each other. In the proposed method, the client is the data owner and only participates in the data distribution and result decryption phases. We do not consider the case where the client submits false input data. The client follows the execution of the protocol but may attempt to infer input data from other clients. During the decryption phase, the semi-honest client decrypts the result according to the protocol process. The goal of malicious servers is to obtain raw inputs and outputs or make the client receive incorrect results. (i) $t-1$ malicious servers decrypt and exchange input data shares to reconstruct the client's original input data when receiving the input data. (ii) $t-1$ malicious servers may exchange intermediate values of the computation process to obtain information related to the raw inputs or output results during the data calculation and result processing phases. (iii) $n-t$ malicious servers may perform incorrect calculations or result processing and submit incorrect results to the blockchain. (iv) $n-t$ malicious servers may choose not to perform computations and not submit results to hinder protocol execution.

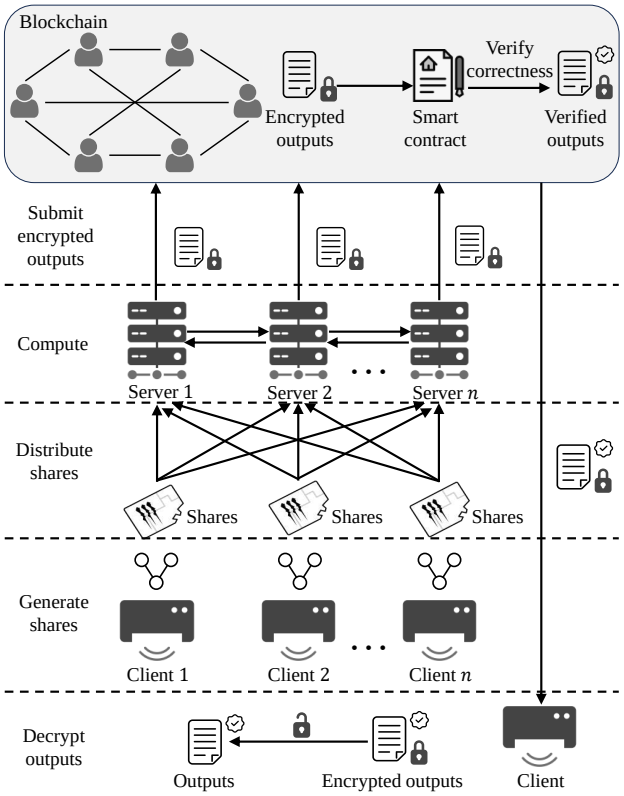


Fig. 2: The system model.

C. Model Overview

As shown in Fig. 2, our system model consists of three types of entities: clients, servers, and the blockchain. Their roles are described as follows.

Clients: The clients are data owners and task requesters in our model, which split the data using the secret sharing scheme. When the computation task is completed, the clients receive the encrypted outputs from the blockchain and decrypt them using the private key to obtain the raw outputs.

Servers: The servers are responsible for executing computations, which act as intermediate nodes between the clients and the blockchain. After completing the computation, the servers encrypt the outputs using the public key of the clients and submit them to the blockchain.

Blockchain: The blockchain serves as a transparent audit platform responsible for verifying the correctness of the outputs in our model. In particular, the feature of automatic execution by smart contracts ensures that the audit process is not susceptible to disruption.

The workflow of the proposed method is described as follows. First, clients and servers initialize system parameters and key pairs. Next, clients split and distribute data to servers

using the secret sharing scheme. The servers verify the validity of data shares and perform computational tasks. servers use the clients' public key to encrypt the computation result shares and submit them to the blockchain. The smart contract verifies the correctness of the computation result shares without revealing the original shares. Finally, the clients obtain the validated encrypted computation result shares from the blockchain. The clients use the private key to decrypt the output result shares and reconstruct the raw output result. The general process of the model is divided into the following phases, including share generation and distribution, share computation, result processing and verification, and result decryption.

1) Share generation and distribution: Clients possess the input data and need to distribute this input to servers with computational capabilities. To preserve the privacy of input data, it is crucial to ensure that servers cannot learn the raw inputs. Clients first need to split the input data into multiple shares corresponding to each respective server. Then, clients need to send the shares to the corresponding servers through a secure channel.

We employ the Pedersen's secret sharing scheme [23] to achieve the splitting of inputs. The core of secret sharing is to partition a secret into multiple random values so that the original secret remains undisclosed. Additionally, we employ commitments to ensure the validity of secret shares, preventing any tampering with the shares during the transmission process. The process of generating secret shares takes place in plaintext space. We need to avoid transmitting shares over a public channel because multiple secret shares can be used to reconstruct the raw inputs. We assume each client has negotiated a symmetric key with servers to simulate a secure channel when sharing the secret (see Section IV-D2). The secret generation and distribution phases aim to enable servers to obtain valid shares of inputs.

2) Share computation: Servers perform the corresponding computations on these shares when obtaining valid shares of inputs. We focus on addition and multiplication operations on the shares, ensuring the verifiability of the computations.

For addition operations, servers can perform the same operation on the corresponding shares due to the additive homomorphic property of the shares. Each server performs computations locally without the need for interaction. For multiplication operations, we need to utilize beaver triples [38] to facilitate the computation of the shares. We assume that servers possess multiple sets of valid triples before executing tasks. During the computation process, servers need to exchange intermediate data to reconstruct intermediate values. Since the intermediate values do not reveal any information related to the inputs, we can use a public channel to transmit intermediate data (refer to Section IV-D3).

3) Result processing and verification: To ensure clients obtain the correct results, we need to verify the correctness of the computation results. Furthermore, we utilize blockchain as a platform for result verification to enhance the transparency of the verification process. However, the public transparency of the blockchain leads to the risk of leaking output results. Servers need to encrypt the raw output results to ensure the confidentiality of the output.

We employ the Lifted ElGamal algorithm [24] to encrypt the output results because Lifted ElGamal possesses additive homomorphic properties similar to Pedersen commitments [23] (see Section IV-D4). Then, servers submit the encrypted output results to the blockchain. The smart contract verifies the correctness of the output results using the Pedersen commitment values of the output result shares (see Section IV-D5). The result processing and verification phases aim to enable the correctness and privacy of the computation results.

4) Result decryption: After completing the result verification, clients retrieve the verified encrypted output result shares from the blockchain. Then, clients collaboratively decrypt the output result shares using the private key and reconstruct the raw output results employing the reconstruction algorithm of the secret sharing scheme.

Since Pedersen's secret sharing scheme is a threshold secret sharing scheme, clients only need to collect a sufficient number of shares that meet the threshold to recover the output results. Additionally, the correctness of each share is verified, ensuring that the recovered output results are correct (refer to Section IV-D6).

D. Model Design

The workflow of the proposed method is shown in Fig. 3. The notations and their descriptions are shown in TableII. We assume there are m clients $C_i(i = 1, 2, \ldots, m)$ and n servers $S_i(i = 1, 2, \ldots, n)$ in our model, where each client $C_i(i = 1, 2, \ldots, m)$ has private data $x_i(i = 1, 2, \ldots, m)$. Multiple clients require the result of a function involving addition and multiplication, computed using their private data. Initially, clients divide their private data using the secret sharing scheme and transmit the corresponding shares to the servers. Subsequently, servers execute computations on the shares, attributable to the additive homomorphic properties inherent in the shares. For multiplication operations, servers convert these into addition operations utilizing multiplication triples. Upon completion of the computation, servers encrypt the output result shares using the clients' public key and subsequently submit these to the blockchain for verification. Next, smart contracts verify the correctness of the output result shares, ensuring no disclosure of information regarding the plain-text output results. Clients then retrieve the verified and encrypted output result shares from the blockchain. Finally, clients reconstruct the plain-text output results by decrypting the shares with their private key.

1) System Initialization: Let p and q be two large primes and $q|p-1$, where g be a generator of cyclic group $\mathbb{G}$ with order q. The p and g are public parameters stored on the blockchain. Each client C_i and server S_j has negotiated a symmetric key $K_{i,j}(i = 1, 2, \ldots, m, j = 1, 2, \ldots, n)$, used to establish a secure channel for transmitting shares. Then, clients jointly generate the key shares of Lifted ElGamal encryption according to the discrete-log based Distributed Key Generation (DKG) protocols [39]. The share $s_i \in \mathbb{Z}_p^*$ of the Lifted ElGamal private key is obtained by running the DKG protocol between clients. Subsequently, all clients computes $k_i = g^{s_i} \bmod p$ independently. Finally, all clients

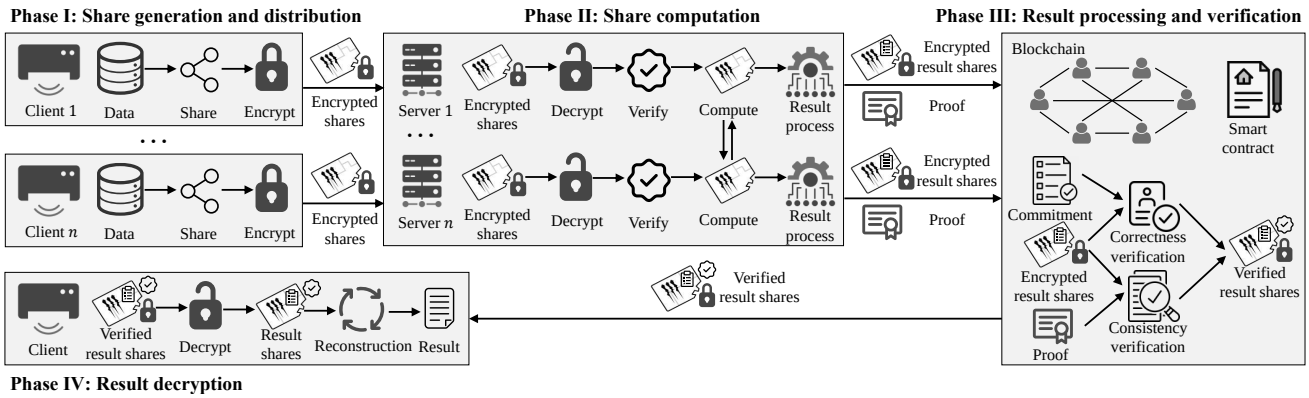


Fig. 3: The workflow of the proposed method.

TABLE II: Notations in the Proposed Method

Notation	Definition
$\mathbb{G}$	A cyclic group
p	A prime in the $\mathbb{G}$
q	The order of $\mathbb{G}$
g	A generator of $\mathbb{G}$
m	The number of clients
n	The number of servers
C_i	The identity of client i
S_i	The identity of server i
h	The public key of Lifted ElGamal cryptosystem
s_i	The private key share of Lifted ElGamal cryptosystem
$K_{i,j}$	The symmetric key negotiated by the client i and server j
t	The threshold of the Pedersen's secret sharing scheme
x_i	The private data of client i

jointly compute the public key $h = \prod_{i=1}^{n} k_i = \prod_{i=1}^{n} g^{s_i} = g^{\sum_{i=1}^{n} s_i} \bmod p$, which is a generator of the cyclic group $\mathbb{G}$ of order q and is stored on the blockchain. Furthermore, to guarantee the normal operation of multiplication, the client will generate the required multiple $t-1$ degree polynomials to submit to the blockchain. After initialization, the blockchain stores the public parameters $\{p, g, h\}$ and multiple $t-1$ degree polynomials. Each client C_i possesses Lifted ElGamal keys h, s_i and symmetric keys $K_{i,j}$ corresponding to each server. Each server S_j possesses symmetric keys $K_{i,j}$ for each corresponding client.

2) Secret Sharing-based Data Distribution: To ensure the confidentiality of the data distribution process, we combine Pedersen's verifiable threshold secret sharing [23] with symmetric encryption to achieve the distribution of the client's data. Each client $C_i (i = 1, 2, \ldots, m)$ computes the commitment for their data x_i as $E_{i,0} = C(x_i, r_i) = g^{x_i} h^{r_i} \bmod p$, where $r_i \in \mathbb{G}$ is a randomly chosen blinding factor. Then, each client C_i formulates two $t-1$ degree polynomials

$$\begin{cases} F_i(x) = x_i + F_{i,1}x + \cdots + F_{i,t-1}x^{t-1}, \\ G_i(x) = r_i + G_{i,1}x + \cdots + G_{i,t-1}x^{t-1}, \end{cases} \tag{3}$$

where $F_{i,t}$ and $G_{i,t} (t = 1, 2, \ldots, t-1)$ are the coefficients of two polynomials of degree $t-1$.

Next, the client C_i computes $s_{i,j} = F_i(j)$, $t_{i,j} = G_i(j)$, and $E_{i,\alpha} = C(F_{i,\alpha}, G_{i,\alpha}) = g^{F_{i,\alpha}} h^{G_{i,\alpha}}$, where $j = 1, 2, \ldots, n$ and $\alpha = 1, 2, \ldots, t-1$. Each client C_i broadcasts $E_{i,\alpha} (\alpha = 0, 1, \ldots, t-1)$ and encrypts $s_{i,j}$ and $t_{i,j}$ as follows:

$$\overline{s_{i,j}} = Enc(K_{i,j}, s_{i,j}), \overline{t_{i,j}} = Enc(K_{i,j}, t_{i,j}). \tag{4}$$

Finally, the client C_i sends $\{\overline{s_{i,j}}, \overline{t_{i,j}}\}$ to servers. Upon receiving the encrypted shares $\{\overline{s_{i,j}}, \overline{t_{i,j}}\}$, servers use the symmetric key $K_{i,j}$ to decrypt and obtain the original shares $\{s_{i,j}, t_{i,j}\}$. Each server S_j checks whether $g^{s_{i,j}} h^{t_{i,j}}$ equals $\prod_{\alpha=0}^{t-1} E_{i,\alpha}^{j^\alpha}$ to verify the validity of shares. Furthermore, the clients generate the multiplication triples $\langle a, b, c \rangle$ required for the computational tasks. For each share $a_i, b_i, c_i (i = 1, 2, \ldots, n)$ of the multiplication triple, the client randomly selects blinding factors $r_{a_i}, r_{b_i}, r_{c_i}$ to generate its Pedersen commitments $c_{a_i}, c_{b_i}, c_{c_i}$, which are submitted to the blockchain. Subsequently, the client encrypts the shares of the multiplication triple using the symmetric key corresponding to the server and transmits them to the respective server.

3) Data Computation: Servers perform computations according to the task function after receiving and verifying their shares. Zhou et al. [26] introduced homomorphic encryption technology to perform computations on encrypted shares within smart contracts, enabling on-chain computation but at the expense of reduced computational efficiency. Additionally, this approach poses a risk of data leakage, as it necessitates decrypting encrypted shares during multiplication operations. To address this issue, off-chain computation is employed to enhance computational efficiency and reduce the risk of data leakage. The computation protocol supports both addition and multiplication operations. For addition operations, each server can independently perform them locally without interaction. Multiplication operations necessitate the use of beaver triples [38] for computation. Our method does not address the efficiency of generating Beaver triples, as these triples can be generated in advance by a trusted third party. It is assumed that the servers already possess a sufficient number of verified multiplication triple shares $\langle a, b, c \rangle$, where $c = a \times b$. The addition operation of two private data points, $s_\alpha + s_\beta$, is used as an example to demonstrate this process. Each server performs the calculation locally and obtains a result share z_i following the addition operation.

$$z_i = s_{\alpha,i} + s_{\beta,i}, i = 1, 2, \ldots, n. \tag{5}$$

For constant multiplication operations, consider the example of $c \times s_\alpha$, with the calculation process detailed as follows:

$$z_i = c \times s_{\alpha,i}, i = 1, 2, \ldots, n. \tag{6}$$

For addition operations involving a constant and private data, the example of $c + s_\alpha$ is taken, with the calculation process outlined as follows:

$$z_i = c + s_{\alpha,i}, i = 1, 2, \ldots, n. \tag{7}$$

The above process describes an addition operation that does not require interaction, owing to the property of additive homomorphism. However, for multiplication operations, servers are unable to operate independently and require interaction. Considering the calculation of $s_\alpha \times s_\beta$ as an example, the process is as follows. Initially, each server locally calculates and broadcasts shares of $s_\alpha - a$ and $s_\beta - b$.

$$\begin{cases} d_i = s_{\alpha,i} - a_i, i = 1, 2, \ldots, n, \\ e_i = s_{\beta,i} - b_i, i = 1, 2, \ldots, n. \end{cases} \tag{8}$$

To detect malicious behavior on the server, server $S_i (i = 1, 2, \ldots, n)$ need to submit d_i and e_i to the blockchain. Subsequently, each server reconstructs d and e using the secret reconstruction algorithm, and computes the secret shares of $d \times e$ based on the client's predetermined $t-1$ degree polynomials stored in the blockchain. Specifically, $d \times e$ is split according to the sharing algorithm of Pedersen's verifiable secret sharing scheme using the $t-1$ degree polynomial stored in the blockchain.

Finally, the result share for $s_\alpha \times s_\beta$ is calculated by each server as follows:

$$z_i = c_i + e \times a_i + d \times b_i + (d \times e)_i, i = 1, 2, \ldots, n. \tag{9}$$

4) Lifted ElGamal-based Result Processing: To protect the privacy of the outputs, we should avoid the leakage of the raw results during the correctness verification process of the client-server model. For existing solutions [26], if employing smart contracts for output results verification, servers must submit the original output shares to the smart contract. The blockchain nodes and servers can access the output shares and reconstruct the output result, a consequence of the transparency of smart contracts, leading to the exposure of the computed result. To ensure confidentiality, Lifted ElGamal encryption [24] is employed to process the output results prior to performing correctness verification. Following the final computation round, each server obtains an output result share of k (e.g., $k = 256$) bits, denoted as $z_i (i = 1, 2, \ldots, n)$. To protect the privacy of the output result share, we split the output result share into multiple values. Specifically, each server S_i divides the k-bit output result shares z_i into p-bit (e.g., $p = 32$) groups, yielding k/p groups $v_j (j = 0, 1, \ldots, k/p - 1)$.

$$z_i = v_0 + v_1 \times 2^p + \cdots + v_{k/p-1} \times 2^{(k/p-1)p}. \tag{10}$$

Following the partitioning, each server encrypts these partitions using the clients' Lifted ElGamal public key. Specifically, each server S_i selects random number $l_1, l_2, ..., l_{k/p-1} \in (1, p-1)$ and computes l_0 as follows:

$$l_0 = r - (l_1 \times 2^p + l_2 \times 2^{2p} + \cdots + l_{k/p-1} \times 2^{(k/p-1)p}) \mod (p-1), \tag{11}$$

where r is the blinding factor associated with the Pedersen commitment value of the output result share z_i. The server calculates the blinding factor r for the Pedersen commitment of z_i using a similar operation to that used for the raw data shares. Server S_i encrypts the partitioned values $v_j (j = 0, 1, \ldots, k/p - 1)$ of the output result share z_i using the client's public key $pk = h$ and random numbers $l_j (j = 0, 1, \ldots, k/p - 1)$, resulting in ciphertext $\overline{z_i} = (C_0^0, C_0^1), (C_1^0, C_1^1), \ldots, (C_{k/p-1}^0, C_{k/p-1}^1)$. Specifically, the partition values $v_j (j = 0, 1, \ldots, k/p-1)$ are encrypted as follows:

$$\begin{cases} C_j^0 = g^{l_j}, j = 0, 1, \ldots, k/p - 1, \\ C_j^1 = g^{v_j} h^{l_j}, j = 0, 1, \ldots, k/p - 1, \end{cases} \tag{12}$$

where g and h are two generators of $\mathbb{Z}_p^*$, which are the same parameters utilized in the data distribution phase. Finally, the servers submit the encrypted output result share $\overline{z_i}$ to the blockchain. The encryption process is detailed as Alg. 1. To ensure clients can decrypt and obtain the correct result in subsequent steps, ciphertext consistency verification is performed. Each server S_i must prove that $C_j^0 = g^{l_j}$ and $C_j^1 = g^{v_j} h^{l_j}$ of the encrypted partition values (C_j^0, C_j^1) where $j \in \{0, 1, \ldots, k/p - 1\}$. For each set of encrypted partition values (C_j^0, C_j^1), it randomly chooses $a_j, b_j \in \mathbb{Z}_p^*$, and computes

$$\text{ZKPoC} = \begin{bmatrix} A_j^0 = g^{b_j} \\ A_j^1 = g^{a_j} h^{b_j} \\ H_j = Hash\left(C_j^0, C_j^1, A_j^0, A_j^1, g, h\right) \\ d_j = a_j + H_j v_j \\ f_j = b_j + H_j l_j \end{bmatrix}. \tag{13}$$

Then, the A_j^0, A_j^1, d_j, f_j are submitted to the blockchain. The smart contract computes $H_j = Hash\left(C_j^0, C_j^1, A_j^0, A_j^1, g, h\right)$ and verification is performed as

$$\begin{cases} g^{f_j} \stackrel{?}{=} A_j^0 {C_j^0}^{H_j}, \\ g^{d_j} h^{f_j} \stackrel{?}{=} A_j^1 {C_j^1}^{H_j}. \end{cases} \tag{14}$$

If the verification fails, the protocol is aborted. Otherwise, the verification for result correctness is further performed.

5) Blockchain-based Result Verification: To ensure the correctness of the output results, the validity of each server's submitted share is verified in the smart contract. While the transparency feature of smart contracts facilitates result verification, it also introduces the risk of privacy leakage. To address this issue, we designed an algorithm for verifying the correctness of encrypted output results using the additive homomorphic properties of Lifted ElGamal encryption and Pedersen commitment.

After receiving encrypted output result shares from the server, commitments of the raw data shares are retrieved by the smart contract from the blockchain. Specifically, the smart contract can calculate the Pedersen commitment of each raw

Algorithm 1 Outputs Encryption Algorithm using Lifted ElGamal Cryptosystem

Require: The generator g of $\mathbb{Z}_p^*$, the public key h of lifted elgamal algorithm, the blinding factor r, the partition values $z = v_0 + v_1 \times 2^p + \cdots + v_{k/p-1} \times 2^{(k/p-1)p}$

Ensure: Encrypted output share $\overline{z}$

$l_1, l_2, \ldots, l_{k/p-1} \xleftarrow{R} (1, p-1)$;
$l_0 \leftarrow r - (l_1 \times 2^p + l_2 \times 2^{2p} + \cdots + l_{k/p-1} \times 2^{(k/p-1)p}) \bmod (p-1)$;
for $i \leftarrow 0$ to $k/p-1$ **do**
  $C_i^0 \leftarrow g^{l_i}$;
  $C_i^1 \leftarrow g^{v_i} h^{l_i}$;
end for
$\overline{z} \leftarrow \left\{ \left(C_0^0, C_0^1\right), \left(C_1^0, C_1^1\right), \ldots, \left(C_{k/p-1}^0, C_{k/p-1}^1\right) \right\}$;
return $\overline{z}$;

Algorithm 2 Outputs Verification Algorithm using Lifted ElGamal Cryptosystem

Require: The commitment value c_z corresponding to the output result share, the encrypted output result share $\bar{z} = \left\{ \left(C_0^0, C_0^1\right), \left(C_1^0, C_1^1\right), \ldots, \left(C_{k/p-1}^0, C_{k/p-1}^1\right) \right\}$

Ensure: $True$ or $False$

$c_z' \leftarrow 1$;
for $i \leftarrow 0$ to $k/p-1$ **do**
  $c_z' \leftarrow c_z' \times C_i^{1^{2^{p \times i}}}$;
end for
if $c_z' = c_z$ **then**
  return $True$;
else
  return $False$;
end if

data share based on the broadcast value of the data distribution phase. Subsequently, the smart contract executes the similar operations on the initial commitments as on the data shares to obtain the commitment of output result shares. As an example of the operation on data x and y, where $x_i, y_i (i = 1, 2, \ldots, n)$ represent the data shares of x and y. For the addition operation $x + y$, the commitment is calculated as follows:

$$c_{z_i} = c_{x_i} \times c_{y_i}, i = 1, 2, \ldots, n. \tag{15}$$

For the multiplication operation $x \times y$, the commitment is calculated as follows:

$$c_{z_i} = c_{c_i} \times c_{a_i}^e \times c_{b_i}^d \times c_{(de)_i}, i = 1, 2, \ldots, n. \tag{16}$$

Furthermore, verifying the intermediate values d and e during the multiplication operation is essential. In this process, the smart contract needs the client to provide blinding factors of the Pedersen commitment. The verification process is as follows:

$$c_{x_i} \stackrel{?}{=} c_{a_i} \times c_{d_i}, c_{y_i} \stackrel{?}{=} c_{b_i} \times c_{e_i}, i = 1, 2, \ldots, n. \tag{17}$$

Finally, the smart contract obtains the Pedersen commitment c_{z_i} corresponding to the output result share.

The smart contract verifies whether the raw output result share matches the Pedersen commitment value using the encrypted output result shares $\overline{z_i}$. The verification process is as follows:

$$c_{z_i} \stackrel{?}{=} C_0^1 \times C_1^{1^{2^p}} \times \cdots \times C_{k/p-1}^{1^{2^{(k/p-1)p}}}. \tag{18}$$

Specifically, the correctness of the ciphertext verification is ensured by the additive homomorphic properties of Lifted ElGamal encryption. The completeness of Eq. (18) can be easily checked by Eq. (19).

$$\begin{aligned} & C_0^1 \times C_1^{1^{2^p}} \times \cdots \times C_{k/p-1}^{1^{2^{(k/p-1)p}}} \\ &= g^{v_0} h^{l_0} \times \left(g^{v_1} h^{l_1}\right)^{2^p} \times \cdots \times \left(g^{v_{k/p-1}} h^{l_{k/p-1}}\right)^{2^{(k/p-1)p}} \\ &= g^{v_0 + v_1 \times 2^p + \cdots + v_{k/p-1} \times 2^{(k/p-1)p}} h^{l_0 + l_1 \times 2^p + \cdots + l_{k/p-1} \times 2^{(k/p-1)p}} \\ &= g^{z_i} h^r = c_{z_i} \bmod p \end{aligned} \tag{19}$$

The verification process of the results is detailed in Alg. 2. If the verification succeeds, the calculation result of the server is correct. Otherwise, the server is malicious.

6) Result Decryption: After completing the result verification, clients retrieve encrypted output partition values from the blockchain. Subsequently, clients decrypt the partition values using the private key s_i. Specifically, the decryption process for the encrypted result partition values proceeds as follows:

$$\begin{cases} t_j = \prod_{i=1}^m C_j^{0^{s_i}} \bmod p, \\ g^{v_j} = \frac{C_j^1}{t_j} \bmod p. \end{cases} \tag{20}$$

Then, clients calculate the partition values $v_j (j = 0, 1, \ldots, k/p-1)$ of the output result share through querying a discrete logarithm table. The clients recover the raw output result share as follows:

$$z_i = v_0 + v_1 \times 2^p + \cdots + v_{k/p-1} \times 2^{(k/p-1)p}. \tag{21}$$

Finally, clients reconstruct the output result z by employing the secret sharing reconstruction algorithm.

$$z = \sum_{i=1}^{t} \left(\prod_{j=1, j \neq i}^{t} \frac{-j}{i-j} \right) z_i. \tag{22}$$

V. Security Analysis

A. Property Analysis of ZKPoC

Theorem V.1. *If the ciphertext $\{C_j^0, C_j^1\}$ is obtained by encrypting the message v_j with the randomly blinding factor l_j, the smart contract will pass the verification of the ciphertext consistency. That is, the ZKPoC is complete.*

Proof. To verify the consistency of the ciphertext $C_j^0 = g^{l_i}$ and $C_j^1 = g^{v_j} h^{l_j}$, the servers computes and sends $\{A_j^0, A_j^1, d_j, f_j\}$ to the blockchain, where $A_j^0 = g^{b_j}$, $A_j^1 = g^{a_j} h^{b_j}$, $d_j = a_j + H_j v_j$, $f_j = b_j + H_j l_j$, and $a_j, b_j \xleftarrow{R} \mathbb{Z}_p^*$. If $C_j^0 = g^{l_i}$ and $C_j^1 = g^{v_j} h^{l_j}$, the smart contract computes

$$\begin{cases} H_j = Hash(C_j^0, C_j^1, A_j^0, A_j^1, g, h), \\ g^{f_j} = g^{b_j + H_j l_j} = g^{b_j} \times \left(g^{l_j}\right)^{H_j}, \\ g^{d_j} h^{f_j} = g^{a_j + H_j v_j} h^{b_j + H_j l_j} = g^{a_j} h^{b_j} \times \left(g^{v_j} h^{l_j}\right)^{H_j}. \end{cases} \tag{23}$$

Therefore, the verification Eq. (14) holds. In other words, the servers have passed the ciphertext consistency verification and the ZKPoC is complete. □

Theorem V.2. *If the ciphertext $\{C_j^0, C_j^1\}$ is not the result of encrypting the message v_j with the randomly blinding factor l_j, the smart contract will fail the verification of the ciphertext consistency. That is, the ZKPoC is sound.*

Proof. It is assumed that there exists the ciphertext $C_j^0 = g^{l_i'}$ and $C_j^1 = g^{v_j'} h^{l_j'}$, where $v_j', l_j' \xleftarrow{R} \mathbb{Z}_p^*$. The smart contract obtains $\{A_j^0, A_j^1, d_j, f_j\}$, where $A_j^0 = g^{b_j}$, $A_j^1 = g^{a_j} h^{b_j}$, $d_j = a_j + H_j v_j$, $f_j = b_j + H_j l_j$, and $a_j, b_j \xleftarrow{R} \mathbb{Z}_p^*$. If $v_j' \neq v_j$ and $l_j' \neq l_j$, the smart contract first computes Eq. (23). Then

$$\begin{cases} A_j^0 C_j^{0\,H_j} = g^{b_j} \times \left(g^{l_j'}\right)^{H_j} \neq g^{b_j} \times \left(g^{l_j}\right)^{H_j}, \\ A_j^1 C_j^{1\,H_j} = g^{a_j} h^{b_j} \times \left(g^{v_j'} h^{l_j'}\right)^{H_j} \neq g^{a_j} h^{b_j} \times \left(g^{v_j} h^{l_j}\right)^{H_j}. \end{cases} \tag{24}$$

Therefore, the verification equation does not hold, and the server cannot pass the verification. That is, the ZKPoC is sound. □

Theorem V.3. *If the participating entities except the server cannot learn the plaintext message v_j encrypted in the ciphertext $\{C_j^0, C_j^1\}$, the ZKPoC is zero-knowledge.*

Proof. Suppose there exists a Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$, and let g, h be two generators of cyclic group $\mathbb{G}$ with order p where $h = g^s$, $s \in \mathbb{Z}_p^*$. For the plaintext message v_j, the $\mathcal{A}$ can learn that $d_j = a_j + H_j v_j$ and $C_j^1 = g^{v_j} h^{l_j}$. Since a_j is a randomly chosen number by the server, the PPT adversary $\mathcal{A}$ cannot learn v_j from d_j. For C_j^1, the $\mathcal{A}$ has

$$C_j^1 = g^{v_j} g^{s l_j} = g^{v_j + s l_j} \bmod p. \tag{25}$$

Without loss of generality, let $x_j = v_j + s l_j$. If the $\mathcal{A}$ wants to learn v_j, $x_j = ind_{g,p}(g^{x_j} \bmod p)$ needs to be calculated. However, $Pr[\mathcal{A}(g, g^{x_j}) = x_j] \leq negl(\lambda)$ due to the Discrete Logarithm Problem (DLP) is hard. Thus, it is impossible for $\mathcal{A}$ to learn x_j in PPT. Furthermore, even if $\mathcal{A}$ learns x_j, it cannot learn v_j as l_j is random. Therefore, the ZKPoC is zero-knowledge. □

B. Confidentiality and Privacy-Preserving

The proposed method ensures that servers cannot learn the raw input data of clients to protect the privacy of the input data. The clients split private inputs into secret shares using the Pedersen's verifiable secret sharing scheme. Each secret share is a random value, which does not reveal private inputs. Then, the clients encrypt different secret shares using a symmetric key negotiated with the server. Each server can only decrypt and obtain a single share. The Pedersen's verifiable secret sharing [23] is a (t, n) threshold secret sharing scheme with a threshold t, which means that only when t or more secret shares are combined can the original data be reconstructed. It is impossible to recover the private input data when $t-1$ colluding participants among the n computation nodes. Furthermore, Pedersen's verifiable secret sharing requires computing the commitment values of the polynomial coefficients during the data distribution phase. The security of the Pedersen commitment scheme [23] relies on the DLP, which has excellent hiding and binding properties. In the Pedersen commitment generation phase, introducing random blinding factors provides information-theoretic security for hiding commitment values. For the same data, using different blinding factors will generate different commitment values, which ensures that the commitment values do not reveal any meaningful information about the inputs. Consequently, the proposed method is effective in resisting collusion attacks from servers and preserving the privacy of inputs.

Moreover, the proposed method implements correctness verification of the output results using non-interactive zero-knowledge proof, Lifted ElGamal encryption [24] and Pedersen commitment [23]. The proposed correctness verification algorithm does not reveal the raw output results. Once the computation is completed, the servers split the output result shares and encrypt them using the Lifted ElGamal public key of clients. The security of Lifted ElGamal encryption ensures that the encrypted output result shares do not reveal the raw output result shares. Furthermore, the smart contract performs correctness verification of the results using the encrypted output result shares and commitment values, which do not disclose meaningful information about the raw output result shares. Then, the clients retrieve the encrypted output partition values from the blockchain and decrypt the partition values using the private key. The blockchain nodes and the servers are unable to decrypt the output result shares or reconstruct the output result. Consequently, the proposed method is effective in preserving the privacy of outputs.

C. Verifiability

The proposed method ensures the public verifiability of computation results, which detects malicious behavior by servers. The clients compute Pedersen commitment values for polynomial coefficients during the data distribution phase, which are submitted to the blockchain. To obtain commitment values corresponding to the computed results, the smart contract can perform operations on Pedersen commitments similar to those applied to secret shares thanks to additive homomorphism. Furthermore, the smart contract offers the advantages of automated execution and public transparency. Performing output result verification within the smart contract provides public verifiability for computed results. During the multiplication computation phase that relies on multiplication triples, servers are required to submit intermediate data to the blockchain. Smart contracts can verify the correctness by opening commitment values to prevent servers from tampering with intermediate data. During the output result verification phase, the proposed method utilizes ZKPoC and the additive homomorphic property of Pedersen commitment [23] to achieve correctness verification of the output results. Furthermore, the proposed outputs verification algorithm relies on the DLP. Given the hardness of the DLP, the proposed algorithm is sound.

D. Robustness

The proposed method exhibits robustness and can tolerate malicious behavior or offline status of servers. The proposed method utilizes Pedersen's verifiable (t, n) threshold secret sharing scheme [23] to construct a secure multiparty computation protocol, where the threshold t implies that t secret shares are required to recover the original secret. During the data distribution phase, the proposed method can tolerate $t-1$ malicious servers colluding to recover the original data. Any $t-1$ server cannot correctly restore the Lagrange interpolation polynomial to recover the correct data. During the data computation and result verification phases, the proposed method can tolerate the malicious behavior or offline status of up to $n-t$ servers. We assume that there are $n-t$ malicious servers that have submitted incorrect computation result shares. During the result verification phase, the smart contract will detect incorrect computation results and trace them back to the specific malicious servers. Furthermore, the client can reconstruct the original output result using the remaining correct t shares. Consequently, the proposed method ensures the continued execution of the protocol in the case of $n-t$ servers acting maliciously or going offline. The proposed method is robust and can resist the collusion of $t-1$ malicious servers in the data distribution phase and $n-t$ offline or collusion of malicious servers in the data computation and result verification phases, where t is the threshold for Pedersen's verifiable secret sharing and $n < 2t-1$.

VI. Performance Evaluation

A. Experimental Setup

The performance evaluation of our proposed method mainly focused on the following metrics. *1) Computation Overhead*: The time of performing the cryptographic operations by each entity. *2) Communication Overhead*: The communication size of each entity in the workflow. *3) Response Time*: The runtime from start to completion for a successful computational task.

We conducted experiments to show the overhead of each entity in our proposed method. All experiments were executed in a PC with Intel Core i9-13905H @2.60 GHz and 32.00 GB RAM. The virtual machine ran 64-bit Ubuntu 22.04.2 LTS, OpenJDK 11.0.19, and Hyperledger Fabric v2.4.4. We communicated with the blockchain network using Hyperledger Fabric Gateway v1.0.1 and ran chaincode using Java.

B. Computation Overhead

The computation overhead referred to the runtime of each entity in the proposed method, which involved the runtime of various cryptographic operations, such as symmetric encryption and decryption, commitment generation, and verification. Table III shows the notations of related cryptographic operations.

Table IV shows the computation overhead of each entity in the proposed method, where the numbers of clients and servers are m and n respectively. Since our proposed method didn't take into account the generation of beaver triple [38], we didn't consider the extra cost of multiplication.

TABLE III: Notations of Cryptographic Operations

Notation	Definition
T_{se}	AES encryption runtime
T_{sd}	AES decryption runtime
T_{cg}	Commitment generation runtime
T_{sp}	Polynomial generation runtime
T_{ss}	Secret share generation runtime
T_{sr}	Secret share reconstruction runtime
T_{sv}	Secret share verification runtime
T_{ee}	Lifted ElGamal encryption runtime
T_{ed}	Lifted ElGamal decryption runtime
T_{rp}	Result partition operation runtime
T_{rv}	Result verification operation runtime
T_{dc}	Share computation runtime
T_{cc}	Commitment computation runtime
T_{zp}	ZKPoC generation operation runtime
T_{zv}	ZKPoC verification operation runtime

TABLE IV: Computation Overhead of Each Entity

Entity	Computation overhead
Client	$2T_{sp} + n(T_{ss} + T_{se}) + t(T_{cg} + wT_{ed}) + T_{sr}$
Server	$m(T_{sd} + T_{sv}) + T_{dc} + T_{rp} + w(T_{ee} + T_{zp})$
Blockchain	$T_{cc} + n(wT_{zv} + T_{rv})$

Then, the servers performed computational tasks on the shares. During the result processing phase, the server split and encrypted the output results, which involved a result partition operation, w Lifted ElGamal encryption operations, and w ZKPoC generation operations, where w is the partition value number. Then, the smart contract needed to execute the verification operation on the output results, which involved a commitment computation operation, nw ZKPoC verification operations, and n result verification operations. Finally, the client needed to decrypt and reconstruct the output results, which involved tw Lifted ElGamal decryption operations and a secret share reconstruction operation.

Furthermore, we analyzed the impact of key length on computational overhead. In this experiment, the performance evaluation metric was the runtime of each server for various key lengths in each phase of the proposed method. The computation overhead did not include network request latency and the time spent on blockchain consensus. Fig. 4 illustrates the runtime of the result processing and verification phases with Lifted ElGamal key lengths of 64 bits, 128 bits, 256 bits, 512 bits, and 1024 bits. When the key length of Lifted ElGamal was increased from 64 bits to 1024 bits, the time for the encryption phase of the computation result increased from 3.88 ms to 10.06 ms, and the time for the ciphertext verification phase increased from 4.79 ms to 13.17 ms. As the key length increased, the time overhead of Lifted ElGamal encryption grew, which corresponded to an increase in the computation time for both the result processing and verification algorithms. Increasing the key length could enhance the security of the scheme but might reduce the overall performance of the scheme. In practical applications, a balance should be struck in setting the key length, ensuring both safety and efficiency.

The advantage of our method using verifiable threshold secret sharing was its resistance to the collusive attack of the server and its significant robustness. We evaluated the computation overhead associated with collusion resistance and

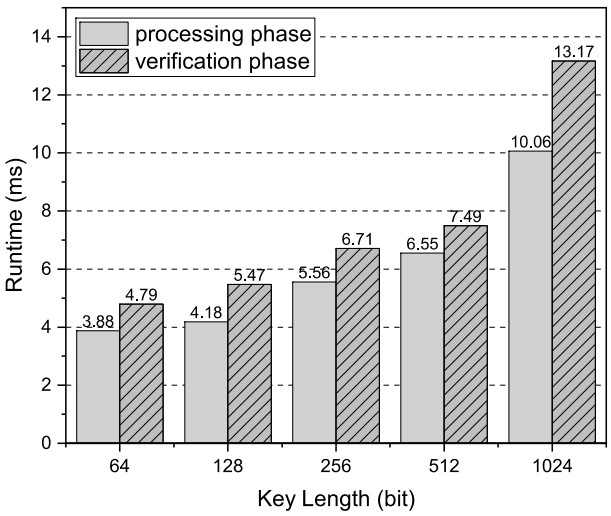


Fig. 4: The runtime of the result processing and verification phases with various Lifted ElGamal key lengths.

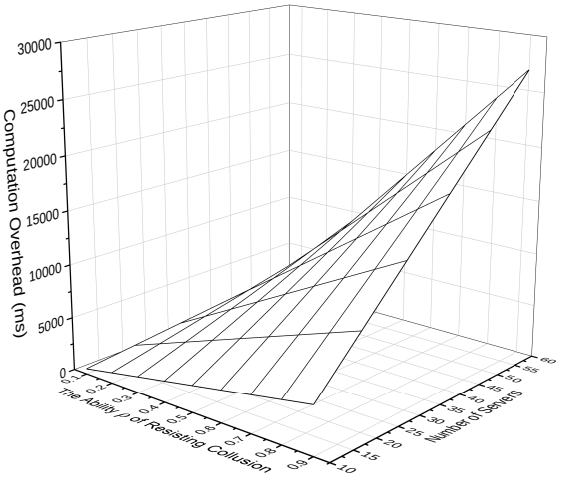


Fig. 5: The computation overhead for resisting collusion.

robustness. We considered collusion between servers involved in the computation, as described in the threat model. The collusion resistance coefficient, denoted as $\rho = \frac{t}{n}$, was used to represent the robustness of the method, where t was the threshold of the secret sharing scheme, and n was the total number of servers participating in the computation. According to Table IV, the average computation overhead T_c of clients was

$$T_c = 2T_{sp} + n(T_{ss} + T_{se}) + n\rho(T_{cg} + wT_{ed}) + T_{sr}. \quad (26)$$

Fig. 5 illustrates the effects of the collusion resistance coefficient ρ and the number n of servers on clients' computation overhead. From Fig. 5, we found that the effect of collusion resistance coefficient ρ increased on computation performance increased with the size of n. To mitigate the computational overhead on the clients, it was necessary to keep the number of deployed servers relatively small when high robust performance was needed.

TABLE V: Specific Interactions of Each Entity

Communication direction	Transmitted message
Each client $\rightarrow$ n servers	$2n$ AES ciphertexts $2n$ signatures t Pedersen commitments
Each server $\rightarrow$ Blockchain	w Lifted ElGamal ciphertexts w proof of ciphertext consistency
Blockchain $\rightarrow$ Each client	tw Lifted ElGamal ciphertexts
Each clients $\rightarrow$ $m-1$ client	tw values of the decryption process

TABLE VI: Communication Overhead of Each Entity

Entity	Communication overhead	Cost (Byte)
Each client	$2n(L(AC)+L(Sig))+tL(Comm)$ $+tw(L(EC)+L(DV))$	7240
Each server	$2m(L(AC)+L(Sig))+mtL(Comm)$ $+w(L(EC)+L(Proof))$	3956

TABLE VII: Comparison of Communication Overhead

Method	**Proof size** (Byte)
BCTV14a [40] (bn128)	312
BCTV14a [40] (mnt4)	384
BCTV14a [40] (mnt6)	424
Groth16 [41] (bn128)	137
Groth16 [41] (mnt4)	169
Groth16 [41] (mnt6)	209
GM17 [42] (bn128)	137
GM17 [42] (mnt4)	169
GM17 [42] (mnt6)	209
RVEV [43]	160
Ours	128

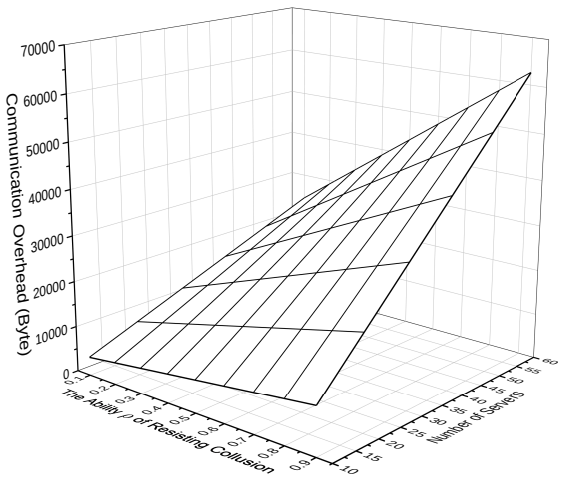
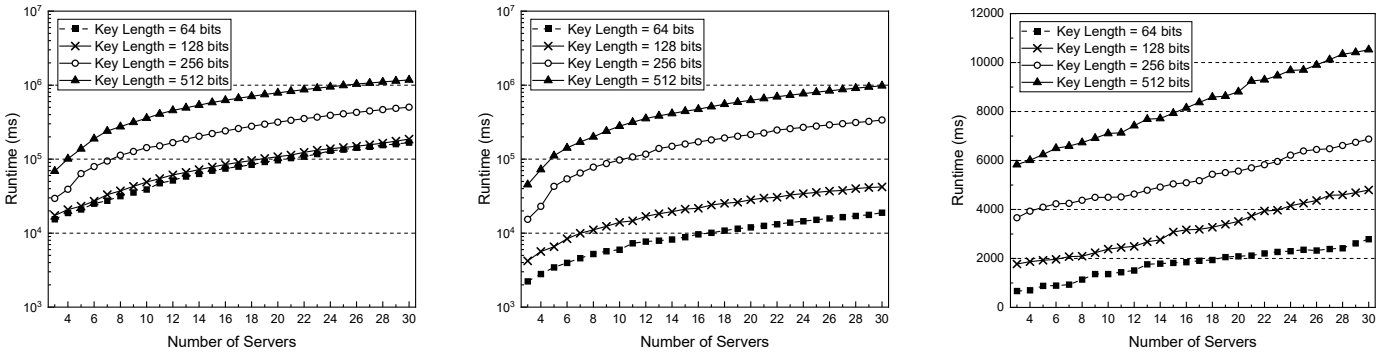


Fig. 6: The communication overhead for resisting collusion.

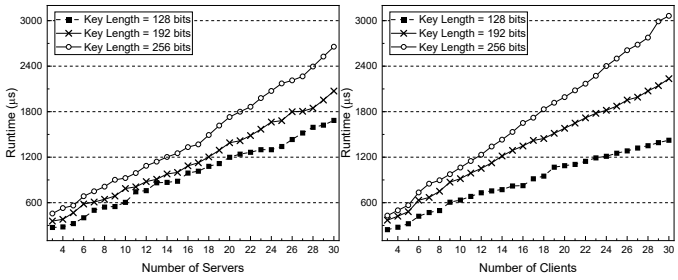
C. Communication Overhead

Table V shows the details of each entity's specific interactions. Let $L(AC)$ denoted the lengths of an AES ciphertext, $L(Proof)$ denoted the lengths of a ZKPoC, $L(Sig)$ denoted the lengths of a signature, $L(Comm)$ denoted the lengths of a Pedersen commitment, $L(EC)$ denoted the lengths of a Lifted ElGamal ciphertext, $L(DV)$ denoted the lengths of a temporary value of the decryption process, respectively. Similar to estimating computation overhead, we did not consider the extra cost of multiplication. According to Table V, the communication overhead of each client was $2n(L(AC) + L(Sig)) +$



(a) Total computational task (b) Result processing and verification (c) Result decryption

Fig. 7: The runtime with various numbers of servers and Lifted ElGamal key lengths.



(a) Secret sharing-based data distribution (b) Shares verification

Fig. 8: The runtime with various numbers of servers and clients and AES key lengths.

$tL(Comm)+tw(L(EC)+L(DV))$ and that of the server was $2m(L(AC)+L(Sig))+mtL(Comm)+w(L(EC)+L(Proof))$, where m, n, w, t were the number of clients, the number of servers, the number of partition values, the threshold of secret sharing, respectively.

During the data distribution phase, each client needed to split the input data using the Pedersen's threshold secret sharing scheme and encrypt the shares using AES symmetric encryption. Then, each client sent the following information to the servers: two AES ciphertexts, two signatures, and t Pedersen commitments. During the result processing phase, each server needed to submit the following information to the blockchain, including w Lifted ElGamal ciphertexts and w zero-knowledge proofs. Finally, during the result decryption phase, each client must retrieve the encrypted output result shares from the blockchain, including tw Lifted ElGamal ciphertexts. Additionally, for collaborative decryption, each client must reveal the temporary values generated during the decryption process to all other clients, including tw values of the decryption process. Since the value of n was usually not very large, and the growth rate of each client's communication overhead was linear with the increase in n, the communication overhead required by each client was acceptable to each client. The communication overhead of each server was not dependent on n, but was related to the number of partition values w and the number of clients m. With a suitable key length, the server's communication overhead was acceptable. Table VI shows the communication overhead of each entity with 5 clients, 10 servers, and 8 partition values.

Furthermore, we evaluated the communication overhead of the zero-knowledge proof protocol in the method. For comparison, we used the zk-SNARKs [40]–[42] scheme as our baseline, which was implemented using the libsnark[1] with the elliptic curve bn128, mnt4, and mnt6. Then, we implemented the RVEV [43] algorithm to compare communication overhead, which was used to verify whether two commitment values correspond to the same message. Table VII shows the communication overhead of different zero-knowledge proof schemes. From Table VII, we found that our method had the minimum communication overhead.

We evaluated the communication overhead for realizing resist collusion, denoted as L_c. According to Table VI, the average communication overhead L_c of clients is

$$L_c = 2n(L(AC)+L(Sig))+n\rho L(Comm) \\ +n\rho w(L(EC)+L(DV)). \quad (27)$$

Fig. 6 illustrates the effects of the collusion resistance coefficient ρ and the number n of servers on clients' communication overhead. From Fig. 6, the client communication overhead increased as the collusion resistance coefficient. The smaller the number of servers, the lower the growth rate of client communication overhead. In addition, by comparing Fig. 5 and Fig. 6, we found that the influence of the collusion resistance coefficient on computation overhead was more significant than that on communication overhead.

D. Response Time

To comprehensively understand the factors influencing performance, we conducted experiments to assess the impact of various server quantities on overall system performance. In this experiment, we evaluated the overall performance of the proposed method using the total response time of the computational task, which included the time elapsed from system initialization until the client obtained the computation results.

Total response time included the running time of each phase of the method, network request latency, and time overhead

[1] https://github.com/scipr-lab/libsnark

associated with blockchain consensus. In addition, the generation process of multiplicative triples was not simulated in the experiment. Fig. 7a illustrates the total response time of the computational task with server quantities of 3 to 30. The increase in the number of servers resulted in an increased verification overhead for smart contracts, consequently leading to an increase in the overall response time of the system. In addition, to analyze the most expensive stages in the proposed method, we evaluated the runtime of the different phases under various numbers of servers. Fig. 7b and Fig. 7c illustrate the runtime of the result processing and verification phase and the result decryption phase under various numbers of servers, respectively. In the experiment shown in Fig. 7c, each client will decrypt all result shares and reconstruct the plain-text result by selecting multiple result shares that satisfy the threshold. From the Fig. 7b and Fig. 7c, it is observed that under the same number of servers and key length, the runtime of the result decryption phase is significantly lower compared to the time consumed in the result processing and verification phase. The clients with limited computational capacity can adequately meet the performance requirements of the result decryption phase. Moreover, we evaluated the impact of AES key length on data distribution and the verification of shares. Fig. 8a and Fig. 8b illustrate the runtime of the secret sharing-based data distribution and verification of shares under various AES key lengths.

VII. Conclusions

In this paper, we proposed a blockchain-based verifiable MPC method, which provides privacy-preserving and public verification for applications following the client-server model. Specifically, we utilized blockchain as a transparent and trustworthy platform for verifying outputs. To protect input privacy and data ownership, we constructed a data distribution method that combines Pedersen's threshold secret sharing and symmetric encryption. Then, we constructed an output result processing algorithm using the Lifted ElGamal encryption to prevent the leakage of plain-text outputs. Furthermore, leveraging the additive homomorphic property of Pedersen commitment and zero-knowledge proof techniques, we proposed a privacy-preserving output verification algorithm capable of publicly verifying the correctness of outputs without disclosing the plain-text outputs. Theoretical analysis showed that the proposed method satisfied the privacy-preserving requirements of the client-server model and demonstrated feasibility. Experiments demonstrated that the proposed method was efficient in the computation overhead of clients and the communication overhead in the result verification phase.

References

[1] D. Feng and K. Yang, "Concretely efficient secure multi-party computation protocols: survey and more," *Security and Safety*, vol. 1, p. 2021001, 2022.

[2] Y. Lindell, "Secure multiparty computation," *Communications of the ACM*, vol. 64, no. 1, pp. 86–96, 2020.

[3] E.-O. Blass and F. Kerschbaum, "Borealis: Building block for sealed bid auctions on blockchains," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020, pp. 558–571.

[4] L. Liu, M. Du, and X. Ma, "Blockchain-based fair and secure electronic double auction protocol," *IEEE Intelligent Systems*, vol. 35, no. 3, pp. 31–40, 2020.

[5] T. D. Nguyen and M. T. Thai, "A blockchain-based iterative double auction protocol using multiparty state channels," *ACM Transactions on Internet Technology*, vol. 21, no. 2, pp. 1–22, 2021.

[6] B. David, L. Gentile, and M. Pourpouneh, "Fast: Fair auctions via secret transactions," in *International Conference on Applied Cryptography and Network Security*. Springer, 2022, pp. 727–747.

[7] R. Küsters, J. Liedtke, J. Müller, D. Rausch, and A. Vogt, "Ordinos: A verifiable tally-hiding e-voting system," in *2020 IEEE European Symposium on Security and Privacy*. IEEE, 2020, pp. 216–235.

[8] V. Cortier, P. Gaudry, and Q. Yang, "A toolbox for verifiable tally-hiding e-voting systems," in *European Symposium on Research in Computer Security*. Springer, 2022, pp. 631–652.

[9] E. Zaghloul, T. Li, and J. Ren, "Anonymous and coercion-resistant distributed electronic voting," in *2020 International Conference on Computing, Networking and Communications*. IEEE, 2020, pp. 389–393.

[10] Y. Miao, Z. Liu, H. Li, K.-K. R. Choo, and R. H. Deng, "Privacy-preserving byzantine-robust federated learning via blockchain systems," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2848–2861, 2022.

[11] M. Rathee, C. Shen, S. Wagh, and R. A. Popa, "Elsa: Secure aggregation for federated learning with malicious actors," in *2023 IEEE Symposium on Security and Privacy*. IEEE, 2023, pp. 1961–1979.

[12] M. Rivinius, P. Reisert, D. Rausch, and R. Küsters, "Publicly accountable robust multi-party computation," in *2022 IEEE Symposium on Security and Privacy*. IEEE, 2022, pp. 2430–2449.

[13] Y. Zhan, L. Zhou, B. Wang, P. Duan, and B. Zhang, "Efficient function queryable and privacy preserving data aggregation scheme in smart grid," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 3430–3441, 2022.

[14] H. Shen, G. Wu, Z. Xia, W. Susilo, and M. Zhang, "A privacy-preserving and verifiable statistical analysis scheme for an e-commerce platform," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2637–2652, 2023.

[15] Z. Xia, X. Yang, M. Xiao, and D. He, "Provably secure threshold paillier encryption based on hyperplane geometry," in *Information Security and Privacy: 21st Australasian Conference, ACISP 2016, Melbourne, VIC, Australia, July 4-6, 2016, Proceedings, Part II 21*. Springer, 2016, pp. 73–86.

[16] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[17] H. Zhong, Y. Sang, Y. Zhang, and Z. Xi, "Secure multi-party computation on blockchain: An overview," in *Parallel Architectures, Algorithms and Programming: 10th International Symposium, PAAP 2019, Guangzhou, China, December 12–14, 2019, Revised Selected Papers 10*. Springer, 2020, pp. 452–460.

[18] K. Gai, J. Guo, L. Zhu, and S. Yu, "Blockchain meets cloud computing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2009–2030, 2020.

[19] G. Tian, Y. Hu, J. Wei, Z. Liu, X. Huang, X. Chen, and W. Susilo, "Blockchain-based secure deduplication and shared auditing in decentralized storage," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 3941–3954, 2021.

[20] Y. Miao, K. Gai, L. Zhu, K.-K. R. Choo, and J. Vaidya, "Blockchain-based shared data integrity auditing and deduplication," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[21] X. Jia, Z. Yu, J. Shao, R. Lu, G. Wei, and Z. Liu, "Cross-chain virtual payment channels," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 3401–3413, 2023.

[22] Y. Zhang, X. Jia, B. Pan, J. Shao, L. Fang, R. Lu, and G. Wei, "Anonymous multi-hop payment for payment channel networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 01, pp. 476–485, 2024.

[23] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*. Springer, 1991, pp. 129–140.

[24] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[25] H. Gao, Z. Ma, S. Luo, and Z. Wang, "Bfr-mpc: a blockchain-based fair and robust multi-party computation scheme," *IEEE access*, vol. 7, pp. 110 439–110 450, 2019.

[26] J. Zhou, Y. Feng, Z. Wang, and D. Guo, "Using secure multi-party computation to protect privacy on a permissioned blockchain," *Sensors*, vol. 21, no. 4, p. 1540, 2021.
[27] Y. Yang, J. Wu, C. Long, W. Liang, and Y.-B. Lin, "Blockchain-enabled multiparty computation for privacy preserving and public audit in industrial iot," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 12, pp. 9259–9267, 2022.
[28] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game, or a completeness theorem for protocols with honest majority," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 307–328.
[29] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 351–371.
[30] M. Keller, "Mp-spdz: A versatile framework for multi-party computation," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 1575–1590.
[31] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4961–4973, 2021.
[32] Z. Huang, W.-j. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure Two-Party deep neural network inference," in *31st USENIX Security Symposium*, Aug. 2022, pp. 809–826.
[33] W.-j. Lu, Z. Huang, Q. Zhang, Y. Wang, and C. Hong, "Squirrel: A scalable secure Two-Party computation framework for training gradient boosting decision tree," in *32nd USENIX Security Symposium*, Aug. 2023, pp. 6435–6451.
[34] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *Annual Cryptology Conference*. Springer, 2014, pp. 421–439.
[35] R. Kumaresan, V. Vaikuntanathan, and P. N. Vasudevan, "Improvements to secure computation with penalties," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 406–417.
[36] A. R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers, "Fairness in an unfair world: Fair multiparty computation from public bulletin boards," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 719–728.
[37] S. Wu, J. Li, F. Duan, Y. Lu, X. Zhang, and J. Gan, "The survey on the development of secure multi-party computing in the blockchain," in *2021 IEEE Sixth International Conference on Data Science in Cyberspace*. IEEE, 2021, pp. 1–7.
[38] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Advances in Cryptology—CRYPTO'91: Proceedings 11*. Springer, 1992, pp. 420–432.
[39] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *Journal of Cryptology*, vol. 20, pp. 51–83, 2007.
[40] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *23rd USENIX Security Symposium*, 2014, pp. 781–796.
[41] J. Groth, "On the size of pairing-based non-interactive arguments," in *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 II 35*. Springer, 2016, pp. 305–326.
[42] J. Groth and M. Maller, "Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks," in *Annual International Cryptology Conference*. Springer, 2017, pp. 581–612.
[43] Y. Cheng, J. Ma, Z. Liu, Y. Wu, K. Wei, and C. Dong, "A lightweight privacy preservation scheme with efficient reputation management for mobile crowdsensing in vehicular networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 3, pp. 1771–1788, 2023.