Smart Contract based Multi-Party Computation with Privacy Preserving and Settlement Addressed

Xin Pei
Zhong An Information and Technology Service Co., Ltd
ShangHai, China
peixin@zhongan.io

Xuefeng Li
Zhong An Information and Technology Service Co., Ltd
ShangHai, China
linco.li@zhongan.io

Xiaochuan Wu
Zhong An Information and Technology Service Co., Ltd
ShangHai, China
wuxiaochuan@zhongan.io

Liang Sun
Zhong An Information and Technology Service Co., Ltd
ShangHai, China
sunliang@zhongan.io

Kaiyan Zheng
Zhong An Information and Technology Service Co., Ltd
ShangHai, China
zhengkaiyan@zhongan.io

***Abstract*—While the multiple party computation (*MPC*) has been suggested for over two decades, we still have not witnessed implementations that make *MPC* or its derivates deployed in a palpable reality. Difficulties stem from serial security and trust issues, typically expressed in data leakage, lazy calculation, and deny of payment etc. Many solutions rely on a third party to organize parties co-working, however, this requires each participant to totally trust the third party and delegate the source data and payment to it. In this paper, we suggest a protocol under blockchain structure to achieve efficient *MPC* without a trusted third party. The cooperation and schedule tasks are executed by a zero-knowledge coordinator, while the smart contracts is used to handle user request and safe payment. Cryptography techniques are adopted to ensure confidentiality, privacy-preserving and verifiability. All the data are transferred and calculated in the cipher state, and only the payed user can decrypt for the calculation result.**

Keywords—MPC, homomorphism, blockchain, smart contract, privacy preserving

I. INTRODUCTION

The use cases of *MPC* schemes includes electronic voting, auction, secret sharing, etc, on behalf of fairness and privacy preserving[1,9]. To illustrate a detailed requirement, let's vie a lending scenario.

"Alice possesses a house worth of 100 million dollars. One day, Alice intends to borrow 300 million from the loan companies by mortgaging her house. However, the credit limit on her guaranty is up to 100 million. Alice tries to separately request to company A, company B, company C for each loan of 100 million. For protection of guests privacy, there exists data barriers between loan companies, as well as time delay on checking the state of the guaranty. Thus, Alice might be able to collect 300 million via triple-mortgage."

This problem can be solved by involving the third party. Upon receiving Alice request, each loan company queries the third party for a decision. This requires all related companies completely trust the third party and delegate their source data. An alternative scheme called secure multi-party computation (*SMPC*) is proposed to de-centralize the third party [1,2,6]. The *SMPC* involves multiple parties to cooperate on a given calculation template *F*. The secure property requires that each party can reveal nothings else beside the overall result. In formal representation, assume *n* parties $P_1, P_2, ..., P_n$ each possesses the data secret $x_1, x_2, ..., x_n$. They cooperate to reach $(f_1, f_2, ..., f_n) \leftarrow F(x_1, x_2, ..., x_n)$. In this procedure, each party P_i only knows about its own f_i, and is innocence of any secrets $\{x_j, f_j\}_{i \neq j}$ of others. Finally, the user can retrieve $\sum f_i = F(x_1, x_2, ..., x_n)$ contributed by all parties. The *SMPC* scheme provides a distributed computing method to reach a desired target, however, it confronts with the following pain points:

- Suffer from *N*-1 attack. The secret data x_i of party P_i can be retrieved by the collude of other *N*-1 parties. The x_i is revealed by calculating $\sum f_i - \sum_{j \neq i} x_j$. Meanwhile, the *SMPC* requires each data provider to participant in the calculation by holding his own data segment $x_{i,k}$. Otherwise, the collude of others will reveal the secret $x_i = \sum_{k \in N} x_{i,k}$.
- Fail to verify computation. The use of *SMPC* must rely on each party to perform the correct calculation, and the result retrieved by the user is unverifiable.
- Inconvenient of use. To facilitate a *SMPC*, the user has to communicate with each provider for an agreement, and make rules for each data provider to segment x_i and transfer to other parties. Moreover, the results from each party needs to be serialized and aggregated by the user himself.
- Complex settlement. The *SMPC* takes no rules to ensure benefits to the honest parties, while the payment should be agreed and pledged before the computation according to the use of data sets and the work of calculation.

Our Contributions. We aim to give full play to data values and promote resource mobility while preserving data security and user privacy, via the proposed blockchain based secure multi-party computaiton(*BMPC*). a) We combine *SMPC* and blockchain[15] into *BMPC* to achieve safe settlement during the computing protocol. The smart contract[19] takes the user request and provider list as input to calculate the payment and generate a electronic contract automatically. The contract is then published on the chain and each provider is required to sign for corresponding work and the user executes the prepayment to a safe middle

address. Upon the acceptance of user confirmation or nodes consensus, the payment are separately transferred to each provider's address. b) *BMPC* specifies a coordinator to execute zero-knowledge task scheduling. The coordinator is responsible for data exchange and on chain message polling. This eliminates communication costs for all parties by bridging on-chain and off-chain querying (described in Section III). From security aspect, the coordinator cannot reveal any cipher data when scheduling, due to the secret keys are possessed by each party respectively. c) *BMPC* facilitates homomorphism encryption all through the computation by using the user's public key. This directly resolves the *N*-1 attack problem and protects the use right of the real payer, on that only the user can decrypt by the private key. d) *BMPC* adopts public verifiable computation(*VC*) technique to prove the correctness of calculation works. This requires the calculator to generate a tag for each task which can be easily verified by the on-chain nodes, so as to reach nodes consensus against deny of payment. e) Moreover, *BMPC* also uses the oblivious transfer(*OT*) structure to satisfy safe query. The anonymous inputs are params of the calculation template and concealed by the coordinator using the calculators public keys, while the decryption can only reveal the designated data segments.

II. Related Works

In general, researches on *MPC* are categorized into semi-honest and malicious adversaries from the aspect of security model. Under the assumption of semi-honest adversary model, Beaver *et al.* [9] propose a general *MPC* protocol secure against an active adversary corrupting up to *N*• 1 of the *n* participants. The protocol can be used to compute securely arithmetic circuits over any finite field. In the article[2], Nielsen *et al.* bring up a secure computing scheme for Boolean Circuits. Their online phase is similar to that of [3], while the pre-processing involves a very efficient privacy construction called *OT* protocol[4,5]. Other works in this area also include Boolean Circuits [6] and Arithmetic Circuits achievements [7].

On the other hand, more strict constraints are pushed upon the malicious adversary model of *MPC*. Jakobsen *et al.* [8] develop a practical system against such attackers, while *SPDZ*[1] and its subsequent improvements [10] greatly contribute to the efficiency. Another approach is based on the cryptosystem of Fully Homomorphic Encryption (*FHE*), proposed by Gentry [11]. In this approach all parties should firstly encrypt their input under the *FHE* scheme, and then evaluate the desired function on the ciphertexts using the homomorphic properties. The results are retrieved bya distributed decryption on the final ciphertexts. In contrast to fully homomorphic schemes, the partially homomorphic schemes[12] only allow one kind of computation (either additive or multiplicative, etc.) on encrypted data. The partially homomorphism performs much more efficient compared with *FHE*. Schemes that support partial homomorphism are for example, the Paillier scheme[13] (additive homomorphic) and the El-Gamal scheme[14] (multiplicative homomorphic).

In addition, we introduce the blockchain as a trusted system so that the contract combined with multi-parties can reach a consensus. The first blockchain system is introduced by Satoshi Nakamoto as the public ledger, which is used to record transactions of the cryptocurrency bitcoin[15]. Soon afterwards, the blockchain becomes a key technology on trust and consensus, especially in the financial and retail fields. Additionally, with the merge of smart contract technique in Ethereum [16], the scalability of blockchain is greatly improved. The smart contract provides a flexible interaction layer for customized programming [17,18], and supports digital facilitation, verification and contract negotiation, etc.

III. The Framework of Our Proposal

Figure 1. describes the main framework of the proposed *BMPC* scheme, which consists of the on-chain and the off-chain sectors. As the basic property of public chain, everything on chain including ledger, transaction, smart contract, Dapp is transparent. Thus, sensitive data should not be directly recorded on chain, as well as the private keys and data hash addresses. The chain mainly performs confirmation of contract and currency settlements. Additionally, the IPFS[21] storage system is regarded chain component due to its feasibility(e.g., FileCoin) . To accomplish secure computing in *BMPC*, many works should be implemented off chain, such as key generation, encryption/decryption, data transmission, etc. The *BMPC* requires a reliable coordinator to bridge message exchange between on chain and off chain, while keeping stable connections among participants for data processing and communications. The critical components are as follows.

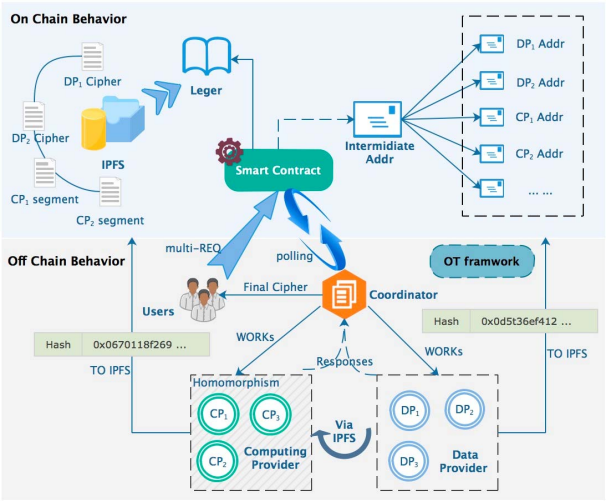

Figure 1. The framework of *BMPC* scheme

User, they designate source data and computing providers to calculate for a dedicate result. Users pay for the consumed data, computing resources and running-gas to the coordinator and the chain nodes with the coins.

Coordinator, short for *CORD*, it schedules the tasks of calculation and keeps message polling from the smart contract. The *CORD* provides calculation templates and ensures anonymous query to the data providers by applying *OT* structure. All through the scheduling, the *CORD* should be keep zero knowledge.

Smart Contract, short for *SCOT*, it supports contract generation, multi-party signing and currency settlement. The *SCOT* takes user request and providers quote to generate a contract, and requires each participant to sign as confirmation. For sake of safe settlement, the *SCOT* supplies an intermediate address for coin payment.

Data Provider, short for *DP*s, they refer to the organizations that will honestly provider what they really possess. The *DP*s are also expected to execute data pre-processing,

including segmentation, encryption and *OT* multiplication. *Computing Provider*, short for *CP*s, they are organized computing clusters or cloud hosts that provide available resources to execute the designated operations. *CP*s are able to generate a public verifiable proof on their finished computation works.

IPFS, is a blockchain feasible distributed storage system. It generates an unique hash value as accessible address to store the data.

IV. Construction of *BMPC* Scheme

In this section, we first introduce several cryptography algorithms which are necessary for *BMPC*, and then draw the procedure of scheme construction.

A. Used Algorithms

The oblivious transfer protocol endows *BMPC* with safe query property for *CORD* and *DP*s, and the homomorphism encryption enables ciphertext based operation, which makes it possible for *CORD* and *CP*s to execute zero-knowledge computation. Besides, the verifiable computation is used for *CP*s to prove the execution of assigned works.

1) *Oblivious Transfer*[*k-out of-n*]

The calculation template may involve multiple columns of data, thus the use of a derived OT_k^t protocol[20] will make sense on efficiency compared to *OT*[4,5]. In OT_k^t protocol, there are two roles, the sender (e.g., *DP*) and the receiver (e.g., *CORD*). k is number of columns provided by the sender, and t is the number of columns retrieved out of k. We assume the *CORD* sends a k-bit selection string to the *DP*. If $t=2$, $k=5$, a selection string $X=\{0,0,1,1,0\}$ can retrieve data v_3 and v_4, while the value 0 at positions 1, 2, 5 means the data values v_1, v_2 and v_5 are not selected. The *DP* is required to compute $\forall i \in k : Z_i = x_i * v_i$ respectively, and $Z=(0,0,v_3,v_4,0)$ contains the retrieved values.

The *DP*'s security needs the selection string to be well-formed, which means that the bit string X_k^t should contain exactly t 1's and k-t 0's. Meanwhile, the *CORD*'s privacy requires that the t selections are hidden from the *DP*. Different from the use cases described in [20], the source data are used in a fixed manner by the calculating template of *CORD* rather than an untrusted user. This eliminates the threat on data security as long as the *CORD* is honest in template. In aspect of privacy, all values are transferred to the *CORD* in an oblivious manner so that the real t selections are hidden.

2) *Homomorphism encryption*

A family of homomorphic encryption consists of full homomorphism (*FME*) and partial homomorphism (*PME*). The *FME* supports arbitrary computations on encrypted data[11], while the *PME* supports specified operations. An additive homomorphism can be represented in mathematical form as in (1).

$$Dec\left(Enc(A) \oplus Enc(B)\right) = A + B \tag{1}$$

where $+$ represents the operation we wish to perform on plain text (e.g. addition for additive homomorphic schemes), and $\oplus$ represents the equivalent homomorphic operation on encrypted data.

In all our protocols, we use the Paillier's cryptosystem[13] as encryption scheme with the additive homomorphic property. The public key for the Paillier encryption scheme is represented by two numbers (n, g), and the private key is (λ, μ). To encrypt a value $m \in \mathbb{Z}_n$, compute the ciphertext by $g^m r^n \bmod n^2$, where $r \in \mathbb{Z}_n$ can be randomly chosen. Then, to decrypt a value $c \in \mathbb{Z}_{n^2}$, compute the plaintext value as in (2).

$$m = \frac{c^\lambda \bmod n^2 - 1}{n} \mu \bmod n \tag{2}$$

Paillier cryptosystem as additive homomorphic is defined as in (3).

$$Dec\left(Enc(A)Enc(B) \bmod n^2\right) = A + B \bmod n \tag{3}$$

The security of Paillier encryption depends on the size of security param n. The suggested size of n is 1024 bits by *NIST*.

3) *Verifiable Computation*

A verifiable computation scheme enables a user to outsource the computation works to an untrusted computing party. The returns include the calculation result as well as a easily verifiable proof. In this way, the user can verify the correctness of the computation by simply checking the proof. Let F be a family of functions, a verifiable computation scheme *VC* for f is defined as follows:

$KeyGen(1^\lambda, f) \rightarrow (SK_f, PK_f, EK_f)$: on input a function $f \in F$, the key generation algorithm produces a secret key SK_f used for input delegation, a public verification key PK_f used to verify the correctness of the delegated computation, and a public evaluation key EK_f, hold by the server to delegate the computation of function f.

$ProbGen(SK_f, PK_f, x) \rightarrow (\sigma_x, VK_x)$: Given $x \in Dom(f)$, the delegator runs the problem generation algorithm to produce an encoding σ_x of works x and a public verification key VK_x.

$Compute(EK_f, \sigma_x) \rightarrow \sigma_y$: Given the evaluation key EK_f and the σ_x, the computing party runs the this algorithm to get an encoded form of x as $y = f(x)$.

$Verify(PK_f, VK_x, \sigma_y) \rightarrow y \text{ or } \perp$: On input the public key PK_f, the verification key VK_x, and the encoded σ_y, this algorithm returns y for pass or $\perp$ for an error.

B. Construction of BMPC Scheme

BMPC scheme consists of five steps, and most of the communication and computation tasks lie in step 4, the *Cooperation of Works*. For better illustration, we draw an overall procedure in figure 2.

1) *System Initialization.* The *BMPC* scheme requires initialization beforehand.

a) Key Generation. Participant generate the key pairs (pk_u, sk_u), (pk_{cord}, sk_{cord}), $\forall i \in M : (pk_{DP_i}, sk_{DP_i})$, $\forall j \in N : (pk_{CP_j}, sk_{CP_j})$ and each publishes *pk*, where *M* is the number of data providers, *N* is the number of computing providers.

b) Component Register. The *CORD*, *DP*s and *CP*s register themselves to the blockchain with their public keys pk_{cord}, pk_{DP}, pk_{CP}, and demonstrate available services (e.g., calculation templates, data sets, computing capacities) as well as corresponding charges.

2) *Contract Generation.* Users should make definite request to the contract by explicit definition of tasks, and the request consists of a group of selected resources and the chosen calculation template *F*.

a) The user chooses *m* out of *M* registered *DP*s to as data source $\forall i \in m : (DP_i, data_i)$, the computing resources *n* from *N* registered *CP*s, and the reliable *CORD*.

b) Finally, the request is formed in (4). Thee *SCOT* evaluates the price via formula (5) and generates the contract by combining *REQ* and the *EVL*.

$$REQ = \{\forall i \in m : (DP_i, set_i), \forall j \in n : (CP_i), CORD, f\} \quad (4)$$

The *SCOT* evaluates the price via formula (5) and generates the contract by combining *REQ* and the *EVL*. This contract requires confirmation of each participant on the agreement of works and rewards.

$$EVL = \sum_{i=1}^{m} set_i.size \times price + \sum_{j=1}^{n} f(S_j) \times price' + gas \quad (5)$$

3) *Contract Confirmation.* The *CORD* keeps periodically polling on the chain, and informs participants upon new user request. Each of the providers is supposed to use its private key for signing an agreement. To avoid flood request, we require users to mortgage the evaluated coins to the middle address generated by the *SCOT*. Upon polling the transaction, the *CORD* tells the *DP*s and *CP*s to forward signing and schedules the calculation works separately.

4) *Cooperation of Works*. The *CORD* cooperates with *DP*s and *CP*s to compute for a designated result. In order to keep providers' privacy, we use *IPFS* as inter-media for data storage.

a) The *CORD* encrypts the corresponding bit string template *X* of *f* by $\forall j \in N : (pk_{CP_j})$. Elements in *X* are then encrypted as in (6).

$$\forall j \in N : \overline{X}_j = \{E(X_{ij}, pk_{CP_j}), ..., E(X_{kj}, pk_{CP_j})\} \quad (6)$$

b) The *DP*s execute data segmentation according to the selected columns *k* and the number of computing parties *n* as in (7), reaching a $n \times k$ matrix.

$$\{V_1, V_2, ..., V_k\} \xrightarrow{segment} \begin{Bmatrix} v_{11} & \cdots & v_{1k} \\ \vdots & \ddots & \vdots \\ v_{n1} & \cdots & v_{nk} \end{Bmatrix} \quad (7)$$

Then, *DP*s generate a random number *r*, and take the user public key pk_u to encrypt the data segment $\forall i \in n : \{v_{i1}, v_{i2}, ..., v_{ik}\}$ to $\forall i \in n : \{c_{i1}, c_{i2}, ..., c_{ik}\}$, where each c_{ij} is the encryption under pk_u by additive homomorphism.

Finally, the *DP*s create $\overline{Z}_{i1}, ...\overline{Z}_{ik}$ from $\overline{X_j}$ and $\{c_{i1}, c_{i2}, ..., c_{ik}\}$ as $\overline{Z}_{ij} = \overline{X}_{ij} \times_h c_{ij}$, where $\times_h$ is the multiplication of constant against a ciphertext, allowed in the encryption scheme. Note that $\overline{Z}_{i1}, ...\overline{Z}_{ik}$ are encryptions of either 0 or c_{ij} values, and consequently, the *DP*s have no way to know which *t* values are selected. The *DP*s upload the $\forall i \in n : \{\overline{Z}_{i1}, ...\overline{Z}_{ik}\}$ to *IPFS* and return the hash address list $\{addr_1, addr_2 ...addr_n\}$ to *CORD* secretly.

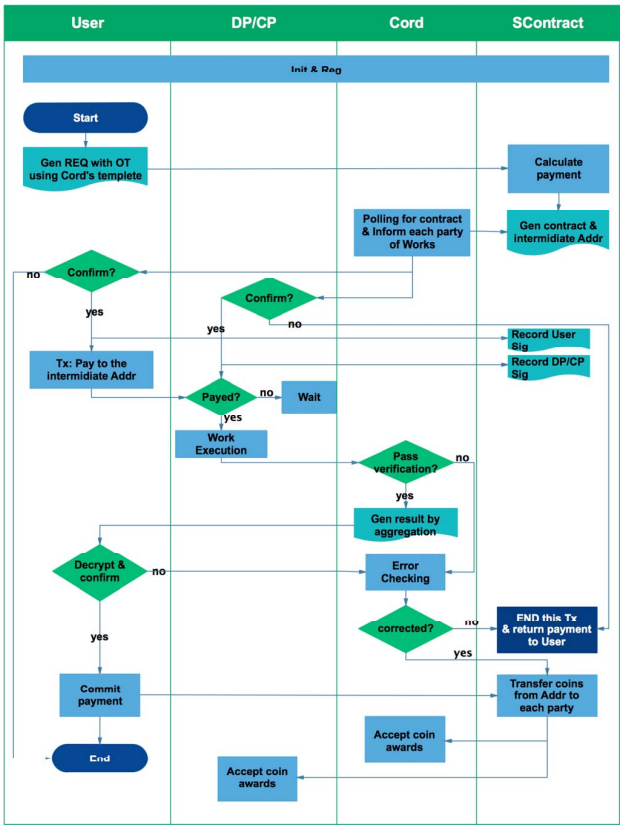

Figure 2. The procedure of BMPC scheme

c) To handle a calculation task *T*, the *CORD* divides *T* into sub-tasks $\{t_1, t_2, ..., t_n\}$ by decomposing the template *F* into $\{f_1, f_2, ..., f_{t-1}, f_{aggr}\}$ with the combination of data $\overline{Z_i}$ as *INPUT*. Note that the constructed operation f_i should also support additive homomorphism.

d) Each *CP* uses its secret key sk_{CP_j} to retrieve the OT_k^t values. When $X_{ij} = 1$ the corresponding $Z_{ij} = c_{ij}$,

and $X_{ij}=0$ will drops the corresponding $\overline{Z_{ij}}=E(0)$. Finally, the valid data is $C_i=\{c_1,...,c_t\}$. Then, the *CP* runs the sub-task $t_i(C_i)=\{f_1,f_2,...f_{t-1},C_i\}$ and returns the tuple (y_i,σ_i), where $y_i=t_i(C_i)$ and σ_i is the verifiable computing proof[22]. For example, let $t=2$, $k=5$ f_1 is $\oplus$ operation. Given inputs C_i, each calculates $y_i=t_i(C_i)=c_1\oplus c_2$ and upload y_i to *IPFS*.

e) The *CORD* verifies $\forall i\in n:\sigma_i$ by the function $Verify(\sigma_i,y_i,C_i)$ and aggregates the $\{y_i\}_{i\in n}$ by calculating $result=f_{aggr}(y_1,y_2,...,y_n)$ if correct. The *result* is then returned to the user.

5) *User decryption.* According to homomorphism theory, the user is able to decrypt the *result* by sk_u.

V. Security and Privacy Analysis

In this section, we discuss the security and privacy issues in *BMPC*. Firstly, we prove the correctness of our proposal.

A. Correctness

For a given request applying the OT_k^t protocol, the user finally intends to get the additive *t* columns from *m* data providers. The *target* is expressed as in (8).

$$target=\begin{pmatrix} V_{[0]}^1+V_{[0]}^2+...+V_{[0]}^m \\ V_{[1]}^1+V_{[1]}^2+...+V_{[1]}^m \\ ... \\ V_t^1+V_t^2+...+V_t^m \end{pmatrix} \tag{8}$$

On receiving the message of *CORD*, the *m DP*s separately execute the segmentation according to the *n* computing parties as in (9).

$$\bigcup_m\left(V_1^\tau,V_2^\tau,...,V_k^\tau\right)\rightarrow segment\ each\ V_i^\tau\ into\ n\ pieces$$

$$Segments^\tau=\bigcup_m\begin{pmatrix} v_{1,1}^\tau,v_{2,1}^\tau,...,v_{k.1}^\tau \\ ... \\ v_{1,j}^\tau,v_{2,j}^\tau,...,v_{k.j}^\tau \\ v_{1,n}^\tau,v_{2,n}^\tau,...,v_{k.n}^\tau \end{pmatrix} \tag{9}$$

The *DP*s encrypt each row (*n* rows each containing *k* values) in $Segments^\tau$ with the user's public key pk_u per elements as in (10), so that the matrix is totally encrypted.

$\forall\tau\in m$:

$$PaillierEnc(Segments^\tau)\rightarrow\bigcup_m\begin{pmatrix} c_{1,1}^\tau,c_{2,1}^\tau,...,c_{k.1}^\tau \\ ... \\ c_{1,j}^\tau,c_{2,j}^\tau,...,c_{k.j}^\tau \\ c_{1,n}^\tau,c_{2,n}^\tau,...,c_{k.n}^\tau \end{pmatrix} \tag{10}$$

Then, the *DP*s perform OT_k^t protocol by taking $\{\overline{X_j}\}_{j\in n}$ and above cipher matrix as input, where each $\overline{X_j}=\{E(pk_{CP_j},x_1),E(pk_{CP_j},x_2),...,E(pk_{CP_j},x_n)\}$, and calculate $\{\overline{Z_j^1},...,\overline{Z_j^\tau},\overline{Z_j^m}\}$ as in (11).

$$\begin{aligned}\{\overline{X_j}\}_{j\in n}\otimes\bigcup_m\begin{pmatrix} c_{1,1}^\tau,c_{2,1}^\tau,...,c_{k.1}^\tau \\ ... \\ c_{1,j}^\tau,c_{2,j}^\tau,...,c_{k.j}^\tau \\ c_{1,n}^\tau,c_{2,n}^\tau,...,c_{k.n}^\tau \end{pmatrix} \\ =\bigcup_m\begin{pmatrix} \overline{X_1}\cdot(c_{1,1}^\tau,c_{2,1}^\tau,...,c_{k.1}^\tau) \\ ... \\ \overline{X_j}\cdot(c_{1,j}^\tau,c_{2,j}^\tau,...,c_{k.j}^\tau) \\ \overline{X_n}\cdot(c_{1,n}^\tau,c_{2,n}^\tau,...,c_{k.n}^\tau) \end{pmatrix} \\ =\bigcup_m\begin{pmatrix} \overline{Z_1}=\left(E(pk_{CP_1},x_1)\times_h c_{1,1}^\tau,...,E(pk_{CP_1},x_k)\times_h c_{k,1}^\tau\right) \\ ... \\ \overline{Z_j}=\left(E(pk_{CP_j},x_1)\times_h c_{1,j}^\tau,...,E(pk_{CP_j},x_k)\times_h c_{k,j}^\tau\right) \\ \overline{Z_n}=\left(E(pk_{CP_n},x_1)\times_h c_{1,n}^\tau,...,E(pk_{CP_n},x_k)\times_h c_{k,n}^\tau\right) \end{pmatrix}\end{aligned} \tag{11}$$

After above encryption, the *DP*s upload the tuple of cipertext $\{\overline{Z_j^1},...,\overline{Z_j^\tau},\overline{Z_j^m}\}_{j\in n}$ to the IPFS and get the corresponding hash address list $\{h_1,h_2,...,h_n\}$. This list is forwarded to the *CORD* who sequentially assigns to the corresponding *CP*s. Then, each *CP* decrypts the received cipher array with its secret key sk_{CP_j} to retrieve the *t* values([0], [1],..., [a] *t* numbers out of *k*) according to OT_k^t protocol when $x_i^{real}=1$, as in (12).

$$\bigcup_m\{\overline{Z_j^\tau}\}\rightarrow\bigcup_m D(sk_{CP_j},\overline{Z_j^\tau})\rightarrow\bigcup_m\left(c_{[0],j}^\tau,c_{[1],j}^\tau,...,c_{[a],j}^\tau\right) \tag{12}$$

Using the additive homomorphism to get $\overline{C_j}$ as in (13).

$$\begin{pmatrix} c_{[0],j}^1\oplus...\oplus c_{[0],j}^\tau\oplus c_{[0],j}^m \\ c_{[1],j}^1\oplus...\oplus c_{[1],j}^\tau\oplus c_{[1],j}^m \\ ... \\ c_{[a],j}^1\oplus...\oplus c_{[a],j}^\tau\oplus c_{[a],j}^m \end{pmatrix}_{Paillier}\rightarrow\overline{C_j}=\begin{pmatrix} C_{[0],j} \\ C_{[1],j} \\ ... \\ C_{[a],j} \end{pmatrix} \tag{13}$$

Then, all *CP*s upload the calculation results $\{\overline{C_1},\overline{C_2},...,\overline{C_n}\}$ to the *IPFS* and get the corresponding hash address list $\{h_1,h_2,...,h_n\}$ to forward to the *CORD*.

Finally, The *CORD* aggregate $\{\overline{C_1},\overline{C_2},...,\overline{C_n}\}$ by additive homomorphism, as in (14).

$$\begin{aligned}\eta &= \sum\nolimits_{j\in n}^{\oplus} \overline{C_j} \\ &= \overline{C_1} \oplus \overline{C_2} \oplus, ..., \oplus \overline{C_n} \\ &= \sum\nolimits_{j\in n}^{\oplus} \begin{pmatrix} c^1_{[0],j} \oplus ... \oplus c^\tau_{[0],j} \oplus c^m_{[0],j} \\ c^1_{[1],j} \oplus ... \oplus c^\tau_{[1],j} \oplus c^m_{[1],j} \\ ... \\ c^1_{t,j} \quad \oplus ... \oplus c^\tau_{t,j} \quad \oplus c^m_{t,j} \end{pmatrix}\end{aligned} \tag{14}$$

Obviously, The η can be aggregated to the *m* dimension, mapping to the form of target as in (15).

$$\begin{cases} \eta = \sum\nolimits_{j\in n}^{\oplus} \sum\nolimits_{r\in m}^{\oplus} \begin{pmatrix} c^\tau_{[0],j} \\ c^\tau_{[1],j} \\ ... \\ c^\tau_{t,j} \end{pmatrix} = \sum\nolimits_{r\in m}^{\oplus} \begin{pmatrix} C^\tau_{[0]} \\ C^\tau_{[1]} \\ ... \\ C^\tau_{t} \end{pmatrix} \\ target = \begin{pmatrix} V^1_{[0]} + V^2_{[0]} + ... + V^m_{[0]} \\ V^1_{[1]} + V^2_{[1]} + ... + V^m_{[1]} \\ ... \\ V^1_{t} + V^2_{t} + ... + V^m_{t} \end{pmatrix} = \sum\nolimits_{r\in m}^{+} \begin{pmatrix} V^\tau_{[0]} \\ V^\tau_{[1]} \\ ... \\ V^\tau_{t} \end{pmatrix} \end{cases} \tag{15}$$

According to Paillier decryption algorithm, for each selected element c^τ_a we have (16).

$$PaillierDec(C^1_a \oplus C^2_a \oplus ... \oplus C^m_a) = V^1_a + V^2_a + ... + V^m_a \tag{16}$$

Thus, the correctness is proved as in (17).

$$PaillierDec\left(\sum\nolimits_{r\in m}^{\oplus} \begin{pmatrix} V^\tau_{[0]} \\ V^\tau_{[1]} \\ ... \\ V^\tau_{t} \end{pmatrix} \right) = \sum\nolimits_{r\in m}^{+} \begin{pmatrix} V^\tau_{[0]} \\ V^\tau_{[1]} \\ ... \\ V^\tau_{t} \end{pmatrix}, \; and \tag{17}$$

$$PaillierDec(\eta) = target$$

B. Data security

The most critical security problem is the leakage of source data. In our proposal, no one can retrieve the source data during the calculation.

1) *Zero-knowledge of CORD*. In the initialization step, the pieces of raw data are encrypted, and have never been decrypted to plaintext before receipted by the user. Thus, even if the *DP*s and *CP*s are required to tell *CORD* about the hash address of data storage, the *CORD* is still of zero-knowledge during the schedule procedure and the aggregation. Meanwhile, *CORD* does not require any of secret keys of any participant.

2) *Resistance to collusion attacks.* Most of the existing *MPC* schemes suffer from *N*-1 attacks. In our proposal, the *CORD* makes the segmentation rules in order to divide the source data into trashy pieces. Then, the raw data are encrypted twice before outsourcing: The first time encrypted to the ciphers by pk_u, and the second time encrypted by the pk_{CP_j} in the form of segmentation. Thus, any combination of the two parts among user, *CP*s and *CORD* cannot collude to reveal the raw data.

C. Safe query

The safe query property is satisfied by *OT* structure. We use *CORD* to provider a calculating template, where the requested params are fixed. The anonymity of receiver problem turns out to be the protection against *DP*s on the input params of the template. The selected bit string is encrypted by the public key of *CP*s, and transferred to corresponding *DP*s for cipher based multiplication. *DP*s do not know which values are selected, while *CP*s get the *t* out of *k* values.

D. User rights

The proposed protocol is fair to users, on that only who pays to the calculation parties can decrypt the result. Moreover, users are enabled to argue on the final result. Under the assumption that each *DP* provides the true data, the user, as well as public chain nodes can verify each step of *CP*s' calculation via the simple *VC* algorithm. During the argument, the coins on the middle address is locked by the smart contract. As long as received the commitment of the user or consensus of chain nodes, the transaction is confirmed or be canceled in case of error.

VI. Conclusion

The proposed *BMPC* scheme enables reliable and flexible multi-party computation in a blockchain environment. Additive homomorphism is used as an efficient algorithm for encryption and calculation, while the oblivious transfer and verifiable computation are also applied for protocol construction. The correctness and security/privacy issues are addressed, while the performance depends on the aggregation of each executed algorithm, which is not focused in this article. Further, we will explore an indeterminate form of user request and a biding mechanism via blockchain.

References

[1] I. Damgård, V. Pastro, N. Smart, et al. "Multiparty Computation from Somewhat Homomorphic Encryption". Cryptology Conference on Advances in Cryptology, CRYPTO. Springer-Verlag New York, Inc. pp.643-662, 2012.

[2] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. "A new approach to practical active-secure two-party computation". IACR Cryptology ePrint Archive, 2011:91.

[3] R.Bendlin, I.Damg•ard, C.Orlandi, et al. "Semi-homomorphic encryption and multi-party computation". In EUROCRYPT, pp. 169–188, 2011.

[4] Y. Ishai, M. Prabhakaran, and A. Sahai. "Founding cryptography on oblivious transfer efficiently". In CRYPTO, volume 5157 of Lecture Notes in Computer Science, pp. 572–591. Springer, 2008.

[5] S. Winkler and J. Wullschleger. "On the efficiency of classical and quantum oblivious transfer reductions". In CRYPTO, pp. 707–723, 2010.

[6] B. D. Assaf, N. Noam, and P. Benny. "Fair playMP: a system for secure multi-party computation". In ACM CCS, pp. 257–266, 2008.

[7] I. Damgård, M. Geisler, M. Krøigaard, et al. "Asynchronous Multiparty Computation: Theory and Implementation". In PKC (LNCS), Vol. 5443, pp. 160–179, 2009 .

[8] T. P. Jakobsen, M. X. Makkes, and J. D. Nielsen. "Efficient Implementation of the Orlandi Protocol". In ACNS (LNCS), Vol. 6123, pp. 255–272, 2010.

[9] D. Beaver. "Efficient multiparty protocols using circuit randomization". In CRYPTO, volume 576 of Lecture Notes in Computer Science, pp. 420–432. Springer, 1991.

[10] M. Keller, E. Orsini, and P. Scholl. "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer". In ACM CCS, pp. 830–842, 2016.

[11]C.Gentry. "Fully homomorphic encryption using ideal lattices". In STOC, ACM, pp. 169–178, 2009.

[12] L. Morris. "Analysis of partially and fully homomorphic encryption". http://gauss.ececs.uc.edu/Courses/c6056/pdf/homo-outline.pdf, 2013.

[13] P. Paillier. "Public-key cryptosystems based on composite degree residuosity classes". 17th Theory and Application of Cryptographic Techniques, EUROCRYPT'99, pp. 223–238, Berlin, Heidelberg, 1999.

[14] L. Harn, Y. Xu. "Design of generalized ElGamal type digital signature schemes based on discret logarithm". Electronics Letters, 30(24):2025-2026, 1994.

[15] S. Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System", 2008.

[16] N. Álvarez-Díaz, J. Herrera-Joancomartí, and P. Caballero-Gil. "Smart contracts based on blockchain for logistics management". The International Conference, pp.1-8, 2017.

[17] C. D. Clack, V. A. Bakshi, L. Braine. "Smart Contract Templates: foundations, design landscape and research directions". 2017.

[18] T. Chen, Z. Li, H. Zhou, et al. "Towards saving money in using smart contracts". The International Conference, pp. 81-84, 2018.

[19] G. Wood. "Ethereum: A Secure Decentralised Generalised Transaction Ledger". Ethereum Project Yellow Paper, 2014.

[20] M. Murugesan, W. Jiang, A. E. Nergiz, et al. "k-out-of-n oblivious transfer based on homomorphic encryption and solvability of linear equations". ACM Conference on Data and Application Security and Privacy. ACM, pp. 169-178, 2011.

[21] M. S. Ali, K. Dolui, F. Antonelli. "IoT data privacy via blockchains and IPFS". International Conference on the Internet of Things. ACM, 2017.

[22] S.G. Choi, J. Katz, R. Kumaresan, et al. "Multi-client non-interactive verifiable computation". Theory of Cryptography. Springer Berlin Heidelberg, pp. 499-518, 2013.