IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 15, 2019, accepted August 4, 2019, date of publication August 9, 2019, date of current version August 22, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2934147*

# BFR-MPC: A Blockchain-Based Fair and Robust Multi-Party Computation Scheme

**HONGMIN GAO, ZHAOFENG MA, SHOUSHAN LUO, AND ZHEN WANG**
[1]School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China
[2]Information Security Center, Beijing University of Posts and Telecommunications, Beijing 100876, China

Corresponding author: Zhaofeng Ma (mzf@bupt.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61272519, Grant 61170297, Grant 61572080, and Grant 61472258.

**ABSTRACT** In a general secure multi-party computation (MPC) protocol, two or more participants who do not trust each other, use their respective secret inputs to calculate a joint function in a distributed environment without a centralized organization. They can get correct outputs on the premise of ensuring privacy and independence of input. In this paper, to solve the problem of fairness and robustness in MPC, a blockchain-based multi-party computation scheme (BFR-MPC) was proposed. The blockchain maintains an open reputation system for parties as a public ledger where a more reputable party has a greater chance to be selected. The block height is used as a trusted timestamp. In each round, parties must send the correct information before the deadline. In our scheme, all parties are considered to be foresighted, and an incentive mechanism encourages parties to cooperate rather than deviate from the protocol. Because of non-cooperative parties will be immediately expelled from the protocol and will be penalized financially, the proposed scheme is robust. The penalty will be used to reward honest parties. We also proved the fairness of our scheme through Game Theory. The comparison results of the proposed scheme with other schemes show that it is a more practical scheme for MPC with high fairness and robustness.

**INDEX TERMS** Multi-party computation, blockchain, smart contract, fairness, robustness, secret sharing, game theory.

## I. INTRODUCTION

Secure Multi-party Computation (MPC) was originally formulated by Yao in 1982 [1], who has presented a solution in form of decentralization for ''The Millionaires' Problem''. The study by Goldreich et al. extended the two-party computation to multiple parties [2]. MPC is used to solve a collaborative computing problem that protects privacy between a group of untrusted participants. In this scenario, two or more parties holding secret input want to jointly calculate a function and get their own output. In this process, the participant does not get any additional information except the expected output. Specifically, a given number of participants, $P = \{P_1, ..., P_n\}$, each have a private data, respectively $x_1, ..., x_n$. Participants want to compute the value of a public function on that private data: $f(x_1, ..., x_n) = (y_1, ..., y_n)$, while keeping their own inputs secret. At the end of the computation, each participant $P_i$ dose not receive any information except knowing $(x_i, y_i)$ and the useless intermediate values derived therefrom.

At present, researchers have proposed many efficient MPC protocols [5]–[8]. Even in a lab environment, they have completed the secure multi-party computation for AES function [9]. Most of these work are focused on improving the efficiency of MPC as much as possible under the premise of a semi-honest adversary model. That is, all participants in MPC are honest, even the corrupted who are attacked are only curious, just use the observed message to analyze the extra results without deviating from the protocol. Unfortunately, this does not correspond to the actual situation. The reality is that there are malicious adversaries who will actively change the execution of the protocol to obtain the expected attack results. There are also some MPC protocols under the malicious adversary model [10]–[13]. But in reality, two things still happen:

1) When a party gets results that are not good for himself before others, he will not announce it, causing other parties to fail to get output.

The associate editor coordinating the review of this article and approving it for publication was Asad Waqar Malik.

2) Malicious adversaries constantly launch attacks to deviate from the protocol, making it unable to run smoothly

Therefore, in reality, in addition to privacy and efficiency, a practical MPC is also highly desirable to be fair (either all parties get output or none) and robust (a malicious adversary cannot mount a ''denial of service'' against the protocol).

## II. RELATED WORKS

As early as 1986, R Cleve's research [20] showed that, when more than half of the malicious parties participated, it was impossible to achieve fairness in the protocol. In 2004, Joseph Halpern and Vanessa Teague proposed a novel idea named rational multi-party computation (RMPC) [21], which uses the concept and method of Game Theory to modify the goal of traditional cryptography. The key point is to introduce the concept of rational adversary under the traditional adversary model. It is believed that the adversary is ''self-interested'' when participating in real-world protocol. It has certain motives and will always be driven by certain interests. These features redefine the strategies, behaviors, and abilities of the adversary. Anna Lysyanskaya et al. proposed a mixed-behavior model [22]. None of the parties in this model is honest. All participants are rational and selfish, and they are only interested in how to maximize their own interests. But the presence of an attacker is detected by a trusted intermediary in their model. Adam Groce et al. have proved that when all parties are rational, a reasonable fairness is possible for arbitrary functions, and Game Theory can be used to circumvent consequences of cryptography [23]. Mehrdad Nojoumian et al. combined the rational secret sharing scheme with the social secrete sharing scheme to form a new socio-rational secrete sharing scheme [24]. The main idea is to build a public trusted network, which can be regarded as a reputation system, assigning a reputation value to each participant. The reputation value is updated with the participant's behavior, and the parties are rewarded or punished according to the reputation value. Gilad Asharov et al. studied the relationship between Game Theory and secure two-party computation [25]. They transform a secure two-party computation protocol into several two-party games. The study reveals the relationship between privacy, correctness, and Nash equilibrium in secure two-party computation. The above-mentioned RMPC based on Game Theory mainly focuses on the fairness of MPC but cannot solve the problem of robustness. Moreover, such MPC protocols seem to be specifically designed to achieve fairness, and they either have a centralized trusted third-party system or lack a real background environment.

With the emergence of blockchain technology represented by Bitcoin [26], due to its characteristics of decentralization and trustable, and the incentive mechanism of Bitcoin is inseparable from Game Theory, scholars began to introduce economic incentives in Bitcoin into the MPC from the perspective of economics. It provides sufficient and real motivation for parties to participate in a MPC protocols. To the best of our knowledge, the work of Marcin Andrychowicz et al. [29], [30] was the first to implement a secure two-party lottery protocol based on the Bitcoin network. The parties in the protocol either get the output or get economic compensation (such as Bitcoin). Their work only defined a two-party protocol, but does not extend to multiple parties, and the initialization process requires multiple rounds of interaction with the Bitcoin network. Iddo Bentov and Ranjit Kumaresan define several ideal functionalities [31], e.g., $\mathcal{F}^*_{CR}$ (claim-or-refund functionality), $\mathcal{F}^*_f$ (secure computation with penalties functionality), $\mathcal{F}^*_{lot}$ (secure lottery with penalties functionality). The MPC protocol they construct only needs to call a constant round $\mathcal{F}^*_{CR}$. Ranjit Kumaresan and Iddo Bentov also studied how to use Bitcoin to motivate participants to achieve correct computations [32]. Their work includes four aspects: verifiable computation, secure computation with restricted leakage, fair secure computation and non-interactive bounty. Ranjit Kumaresan et al. also discussed how to use Bitcoin to implement a decentralized poker game [33]. In 2017, Ranjit Kumaresan et al. improved the previously proposed scheme [34] and proposed how to use the penalty mechanism to optimize the secure computation model. All of the above work are based on the Bitcoin protocol, which only studies the fairness in MPC, and has made some modifications to the Bitcoin protocol in order to meet their requirements. As we all know, the Bitcoin blockchain has great limitations in practical application scenarios. The scripting language of Bitcoin is non-Turing Complete, so it cannot support more flexible and complex functions. Moreover, the transaction confirmation time of six blocks for nearly an hour and a high transaction fee lead to the application based on Bitcoin protocol only stays at the theoretical stage. The study by Kiayias *et al.* [14] is most similar to our work. They propose a fair and robust multi-party computation protocol using a global ledger. Their work includes the following three aspects: 1) Propose a formal model of secure multi-party computation with compensation; 2) The model is UC-Secure; 3) First proposed a constant-round robust multi-party computation protocol.

In this paper, we formalized an ideal function for a fair and robust multi-party computation which we call it $\mathcal{F}^*_{FBMPC}$ (FRMPC stands for ''fair and robust multi-party computation''). Then, we proposed an outsourcing MPC scheme based on EOS blockchain to achieve our goal. At last, it proved that our scheme for MPC have a good fairness and robustness.

## III. PRELIMINARIES

### A. PUBLICLY VERIFIABLE SECRET SHARING

The earliest PVSS was proposed by Schoenmakers [38]. The most obvious feature of a non-interactive PVSS is that there is no secret channel between the secret distributor and each participant. All shares are encrypted and transmitted on the public channel with a proof of commitment. The secret share can be publicly verified.

*Init:* Generating the required system parameters using a public algorithm. When participants join the protocol, they need to first generate a pair of keys $(Pk_i, Sk_i)$ using an asymmetric encryption algorithm, publicly disclose the public key $Pk_i$ and keep the private key $Sk_i$ secret.

*Distribute:* The *Dealer* chooses a secret $s$ to share among $n$ parties $\{P_1, ..., P_n\}$. He will generate a secret share $s_i$ $(i = 1, \ldots, n)$ for each party and encrypt it with the corresponding public key $Pk_i$. Then, the ciphertext will be broadcasted on a public channel attach with a commitment $c_i = COMM(s_i)$. Anyone can verify whether the *Dealer* has sent the correct shares through a non-interactive verification algorithm $PVSS(En_{Pk_i}(s_i), c_i)$.

*Reconstruct:* The party $P_i$ uses his private key to decrypt the ciphertext and gets his share $s_i$. All parties disclose their own share. Anyone can verify the correctness of $s_i$ through an algorithm $verify(s_i, c_i)$, and exclude corrupt parties from submitting incorrect shares.

PVSS can be used to construct a secure multi-party computation protocol. Because of its feature of additive homomorphism, in an addition gate, parties only need to operate their own shares locally. When a multiplication gate, the parties are required a secondary distribution of an useless intermediate value using PVSS.

### B. GAME THEORY

A *game* $G$ consists of a set of *players* $P$, a set of *strategies* $S$ (the way of choosing *actions*) and a *pay-off function* $\boldsymbol{u}$ which used by each participant to compute his utility, i.e. $G = \{P, S, u\}$. The pay-off function $u_i : S \rightarrow R$ represents the utility of $P_i$ under different combinations of strategies.

Let $\sigma_i$ denote the strategy of player $P_i$, and $\sigma_{-i}$ be the strategies of all players other than $P_i$, $\vec{\sigma} := (\sigma_i, \sigma_{-i})$ be the strategy profile of all players. Let $(\sigma_i', \sigma_{-i})$ denote player $P_i$ replace $\sigma_i$ by $\sigma_i'$ and all the other players' strategies remain unchanged. Let $u_i(\vec{\sigma})$ mark $P_i$'s utility when the strategy profile $\vec{\sigma}$ is played.

A vector of strategies $\boldsymbol{\sigma}$ is a Nash equilibrium if, for any player $P_i$, any $\sigma_i' \neq \sigma_i$, it holds that $u_i(\sigma_i', \sigma_{-i}) < u_i(\boldsymbol{\sigma})$. This means that no one gains any advantage by deviating from the protocol as long as the other players follow the protocol. If player $P_i$ can always improve its utility by not playing $\sigma_i$, we say that the strategy $\sigma_i$ is strictly dominated.

### C. BLOCKCHAIN

The concept of blockchain originated from Nakamoto's Bitcoin white paper [26]. Its foundation is cryptography and P2P networks. The data of specific structure is organized into blocks in a certain way, and then these blocks are linked into a chain according to a certain time sequence. The data's security and unforgeability are ensured by cryptography technology and consensus mechanism. Briefly, as the basic technology of cryptocurrency such as Bitcoin, blockchain is a non-tamperable distributed and trusted ledger. It does not require a trusted third-party center but can perform trusted business processing.

#### 1) SMART CONTRACT

In the early stage of blockchain development, only cryptocurrencies like BTC, LTC were more successful applications. What really changed the blockchain was in 2013, Vitalik Buterin presented the first public blockchain platform with the built-in Turing complete programming language and introduced the concept of Smart Contract in the Ethereum white paper [27]. The concept of Smart Contract was introduced by Szabo [36], and was defined as "a computerized transaction protocol that executes the terms of a contract." In the blockchain, a Smart Contract is a piece of program code that can be automatically triggered and executed securely and reliably. On one hand, Blockchain provides a trusted execution environment for the operation of Smart Contract. On the other hand, Blockchain can achieve more complex behavior with the programmability of Smart Contract. The operation mechanism of blockchain-based Smart Contract is shown in Fig.1.

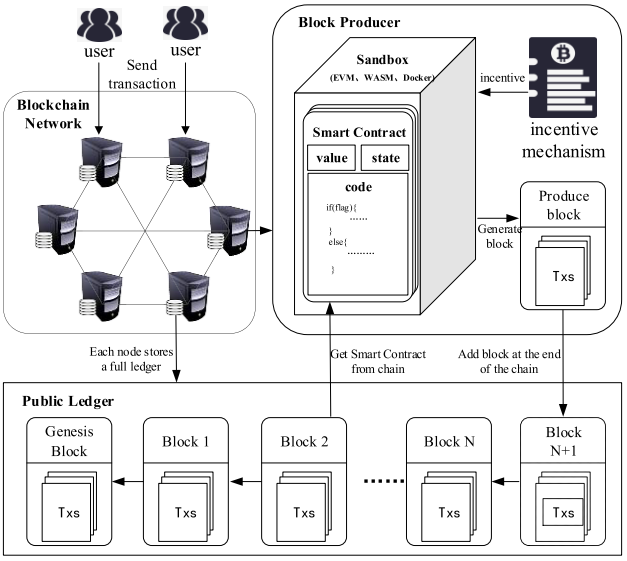



**FIGURE 1.** Operation mechanism of blockcahin-based smart contract.

The blockchain can be seen as a transaction-triggered state machine from a higher perspective and the public ledger is a world state which begins from Genesis Block. Users can connect to any node in the blockchain network to initiate transactions and the receiving node broadcasts it to the entire network through the P2P network. Due to the existence of the incentive mechanism, all Block Producer will perform the corresponding operations after receiving the transaction, and package the transaction to create a block, and then get a final result through the consensus mechanism to update the world state. The action triggered by the transaction can be invoking a smart contract from the ledger and executing it in a sandbox environment, or deploying a new smart contract. Blockchain provides the following capabilities for Smart Contract running in its environment:

*Public status:* All participants have access to the entire ledger to see the current global status of the Smart Contract.

*Timestamp server:* The blockchain can be regarded as a never-ending decentralized timestamp server with block height.

*Trusted propagation channel:* The sender can propagate a message through the blockchain, the receiver will receive the message in the next block, and the content of the message will be recorded in the ledger which cannot be tampered with and trustable.

#### 2) TRANSACTIONS OF EOS

In the EOS blockchain platform, account, address and transaction are three important components. Cryptography algorithms such as Elliptic Curve Digital Signature Algorithm (ECDSA) and Hash function ensure the security of the system. In the EOS blockchain, each user has his own account, with each account corresponding to multiple ECDSA key pairs denoted by $(Pk, Sk)$, each pair of keys corresponding to different operating permission for the account. The private key and the public key are used to sign and verify the transactions. If a user wants to call a smart contract, he needs to prepare a transaction$Tx$:

$$Tx = (Ref_{block}, t, Sig_U(Chain\_ID, Tx), \\ Action(Code, Name, Auth_U, Data))$$

$Ref_{block}$ represents the reference to the height and id of a recently generated block to prevent transactions from appearing on a fork chain and $t$ stands for the expiration time of the transaction. $Sig_U(Chain\_ID, Tx)$ is the signature on $(Chain\_ID, Tx)$ used to verify the identity of the user who initiated the transaction. *Action* denotes the operation to be performed, in which *Code* is the name of a Smart Contract, *Name* is the method in the Smart Contract, $Auth_U$ is used to authenticate whether the user who initiated the transaction has permission to call this method, *Data* are the parameters to be passed into the Smart Contract. There may be multiple actions in a transaction. A contract account can also send an action to another contract, calling each other's method, which called inline communication, and the permission of the action has the same scope as the transaction initiated by the user for the first time.

For example, we now assume Alice and Bob are two users of the EOS system. If Alice wants to pay Bob a certain amount of EOS (we assume the transfer amount is $v$), she needs to prepare a transaction $Tx$ which calls the *transfer* method of contract *eosio.token*:

$$Tx = (Ref_{block}, t, Sig_A(Chain\_ID, Tx), Action \\ (eosio.token, transfer, (Alice, active), (Alice, Bob, v)))$$

## IV. IDEAL FUNCTIONALITY

In this section, we have Designed a new ideal functionality $\mathcal{F}^*_{BRMPC}$ for MPC with fairness and robustness. We regard MPC as a repeated game while previous work consider it to be a one-time work. In the proposed ideal functionality, each party has a reputation value which is undated to his behavior each time the game is played. The greater the reputation value, the easier to be selected for the next game, so that they can earn more benefits in the future. Therefore, the participants in our model are selfish, focusing only on how they can get more benefits, which is the same as the participants in RMPC. Furthermore, the participants in $\mathcal{F}^*_{BRMPC}$ are far-sighted and care what he will get in the future which is different from RMPC.

In our model, we designed an incentive mechanism similar to the Bitcoin ecosystem for MPC protocols. Our model guarantees the following aspects:

1) No penalty will be imposed on the cooperative participant.

2) If a participant deviates from the protocol or gets output without announcing it, he will be punished financially. Honest participants will receive these penalties as compensation.

3) All participants are concerned about their reputation, so that it is easier to join the game in the future and get a long-term benefits.

Our ideal functionality for fair and robust MPC is shown in Fig. 2. We use the same notation ***coin*** as [32], which define coins as atomic entities that are fungible and cannot be duplicated. The ***coin***$(x)$ denote an item whose value is described by $x \in \mathbb{N}$.

In our ideal function $\mathcal{F}^*_{BRMPC}$, all participants are rational and far-sighted. If someone wants to be a participant, he needs to pay the deposit of at least $d$ coins in the ***deposit phase***. The ***setup phase*** will select $n$ out of $N$ players with sufficient deposit to start the current game, all of whom are considered honest at first, where $n \leq N$. In the ***input phase***, $\mathcal{F}^*_{BRMPC}$ begins to receive inputs from all participants. If the ideal functionality receives a ***corrupt-input*** from $\mathcal{S}$ for a honest party $P_s$, $0 < s \leq h$, then marks it as corrupt. After all the inputs have been collected, the ideal functionality begins to perform the computation in the***compute phase***. In the ***output phase***, $\mathcal{F}^*_{BRMPC}$ distributes the output of each honest party, and if receives a ***abort*** message or nothing from $P_s$, $0 < s \leq h$, then marks it as corrupt. If it receives a ***corrupt-output*** from $\mathcal{S}$ for a honesty party $P_s$, also marks it as corrupt. In the ***compensate phase***, all participants marked with corruption will be imposed a fine, and the penalty will be compensated to honest participants. All participants can withdraw the balance of their deposit at the ***withdraw phase***.

## V. BLOCKCHAIN-BASED FAIR AND ROBUST MPC SCHEME

In order to achieve our goal, we will combine the characteristics of PVSS and EOS blockchain to propose a fair and robust outsourcing multi-party computation scheme (BFR-MPC). The BFR-MPC proceeds in rounds. The parties execute the computation locally, which they can do until a multiplication gate, then they randomize their shares for an useless intermediate value using PVSS. This process repeats until an output gate, then participants announce their shares of output for reconstruction.

**Ideal Functionality $\mathcal{F}^{*}_{BRMPC}$**

$\mathcal{F}^{*}_{BRMPC}$ with session identifier *sid* running with parties $\{P_1,...,P_N\}, N \in \mathbb{N}$ each of whom has a reputation value $R$, a parameter $1^{\lambda}$, a honest parties $P_H$, an ideal adversary $\mathcal{S}$ that corrupts parties $\{P_c\}_{c\in C}$ proceeds as follows : Let *coin*($D$) denote the sum of deposits of all parties and $b_i$ is $P_i$'s balance in deposit. Let $d$ be a parameter representing that only if the balance of deposit is greater than $d$ can he join the game. Let $p$ denote the penalty amount and $p<d$. Inputs $x_1,...,x_n$ and outputs $y_1,...,y_n$ are initialized to $\perp$.

1) ***Deposit phase*** : Upon receiving the tuple (***deposit***, *sid*, *ssid*, $P_i$, *coin*($x$)), then *coin*($D$)= *coin*($D$+$x$), $b_i = b_i + x$, send (***deposit.receipt***, *sid*, *ssid*, $P_i$) to $P_i$.

2) ***setup phase***: Upon receiving the tuple (***setup***, *sid*, $n$), select $n$ out of $N$ players according to their reputations and balances, where $n \le N$. Let $P_H = \{P_1,...,P_n\}$.

3) ***Input phase***: Wait to receive a message (*input*, *sid*,*ssid*, $P_i$, $v_i$) from $P_i$ for all $i \in \{1,...,n\}$. Set $x_i = v_i$. Send message (***input.receipt***, *sid*, *ssid*, $P_i$) to $P_i$.

— If receives a message (***corrupt-input***, *sid*, *ssid*, $P_s$, $v_s$) from $\mathcal{S}$, $P_s \in P_H$, then:

- $C = C \cup s$ ,move $P_s$ to corrupted set.

4) ***Compute phase*:** Upon receiving all parties' inputs and sending ***input.receipt*** to all parties.

*Compute*($y_1,...,y_n$) $\leftarrow f(x_i,...,x_n)$

5) ***Output phase***: Send message (***output***,*sid*,*ssid*, $P_s$, $y_s$ ) to party $P_s$,s $\in H$

— If $P_s$ returns (***continue***, *sid*, *ssid*), then:

Send to all $P_h, h \in H$ message (*output*, *sid*,*ssid*, $P_h$, $P_s$, $y_h$).

— If $P_h$ returns (***abort***,*sid*,*ssid*), then $C = C \cup s$ ,move $P_s$ to corrupted set.

— If receives nothing from $P_s$ in this round, then $C = C \cup s$ ,move $P_s$ to corrupted set.

— If receives a message (***corrupt-output***, *sid*, *ssid*, $P_s$, $y_s$ ) from $\mathcal{S}$ , $s \in H$ ,then $C = C \cup s$ ,move $P_s$ to corrupted set.

6) ***Compensate phase:***

- For all parties in $\{P_c\}_{c\in C}$, $b_c = b_c - p$ .
- For all parties in $\{P_h\}_{h\in H}$, ***compensate***( $P_H$, $p\times|C|$, $R_H$ )

7) ***Withdraw phase***:

— If receives a message (***withdraw***, *sid*, *ssid*, $P_s$), $s \in (1,...,N)$, then *coin*($D$) = *coin*($D$ - $b_s$ ).

- Send message (***withdraw***, *sid*, *ssid*, $P_s$, coin($b_s$)) to $P_s$.
- $b_s = 0$.

**FIGURE 2.** The ideal functionality $\mathcal{F}^{*}_{BRMPC}$ for fair and robust MPC.

We use the same terminology as in [41]. The part that is executed on-chain we call it Contract, and the part executed off-chain by MPC node we call it Protocol. Let *_self* denote contract itself and *asset*($x$) denote system token in EOS blockchain. Transfer transactions are abbreviated as *transfer*($A$, $B$, *asset*($v$)) which means that account $A$ has transferred $v$ coins to account $B$.

### A. SYSTEM MODEL

Our scheme consists of three types of roles: Blockchain, User (U) and MPC Node (P). As a public ledger, the Blockchain provides a fair and trustable service for the scheme. MPC Nodes provide a MPC service for Users and reap the benefits. The MPC node needs to be registered in the MPC Node Management Contract (MPCNMContract) and a certain amount of deposit is required for registration. If a MPC node deviates from the protocol or acts unfairly (does not disclosing the secret share it holds) during the computing process, it will be deducted from his deposit as a penalty. Users can use the MPC service provided by MPC nodes to use their respective secret input to obtain the result of $f$ and pay a service charge for it. The system model of proposed scheme for fair and robust MPC is shown in Fig 3.

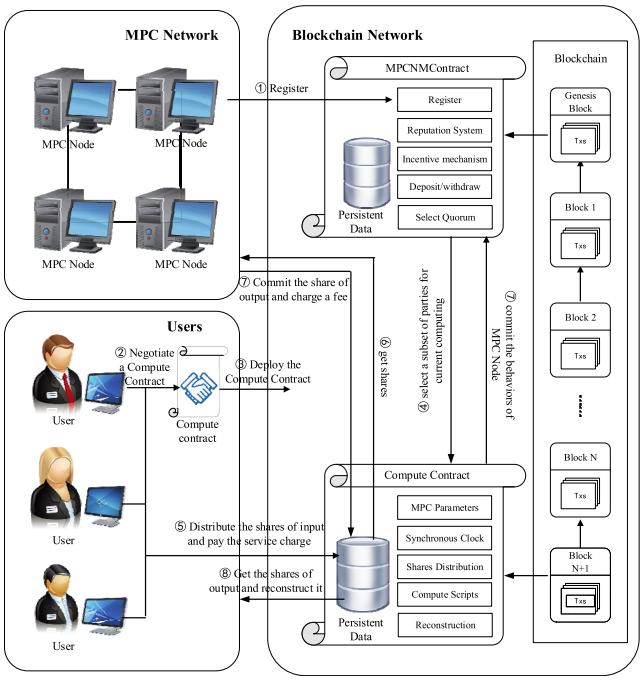



**FIGURE 3.** The system model of FBRMPC.

### B. MPC NODE MANAGEMENT CONTRACT

The contract MPCNMContract consists of the following modules, and the functions of these modules are described in detail as follows:

1) *Register:* If the participant wants to become a MPC node, it is necessary to register in this contract and to pay a certain deposit, only when the minimum deposit is met, it is possible to be selected by the Users to provide a computing service and earn income.

2) *Reputation System:* Each MPC node has a reputation value in this contract. If the MPC node is cooperative and completes the computation task successfully, the reputation value will increase, otherwise the reputation value will decrease. The reputation value determines the long-term utility of the MPC nodes.

3) *Incentive Mechanism:* The income of the MPC node comes from two parts: the fee of Users using MPC service and the penalty of corrupt MPC nodes. The Incentive Mechanism determines the actual utility of MPC nodes, which is reflected in the distribution of benefits after each service is provided.

4) *Select Quorum:* Users can select a quorum of MPC nodes according to their reputation values and deposit amounts to perform the current computation task. Poor reputation does not mean that it will not be chosen, but the probability will decrease exponentially as the reputation value declines. We should give a chance to the bad players so they can compensate for their past behavior. We also need to consider a newcomer. This is consistent with the reality.

### C. COMPUTE CONTRACT

Compute Contract is negotiated and deployed to the blockchain platform by the Users who needs to use the multi-party computation service. The MPC parameters for this current computation task, the computation scripts for each selected MPC node, etc. are specified in the Compute Contract.

The computation proceeds in rounds, and from the blockchain's perspective, each round has a time limit within which participants (Users or MPC Nodes) must correctly send the information that should be submitted to the blockchain in this round. The smart contract will verify that the information submitted is correct. At the beginning of the next round, the smart contract will also check which participant did not submit a correct message. After this, the blockchain will determine whether the computation can proceed or not (add cheaters to a corrupted parties list and if the size of honest set less than the threshold $t$ of PVSS, then abort the task). This time limit can be the height of a certain block in the future. We can let $\tau = RoundToHeight(r)$ , $r = 1, 2, 3 \ldots$ denoted the height of blockchain at the end of round $r$.

In the init phase, it will send a transaction to ***MPCNMContract*** to get a quorum MPC nodes participating in current computation task. In the first round, Users broadcast their shares encrypted by the public key of MPC nodes utilizing this contract and deposit the charge fee. The MPC nodes broadcast the useless intermediate value generated during the computation in the same way between the second round and penultimate round. In the last round, MPC nodes announce their respective output shares and charge fees. From this perspective, this contract is a trusted broadcast channel. This is somewhat equivalent to the synchronous broadcast channel in [40].

The formal definition of a Compute Contract is shown in Fig 5.

**MPCNMContract**

**Init**: Let four tuple $(A, B, R, Pk)$ denote a MPC node $P_i$, in which $A$ denote $P_i$'s account in EOS blockchain, B is $P_i$'s balance in this contract, $R$ ( $0 < R \leq 1$ ) denote $P_i$'s reputation value which is initialized to 0.5, $Pk$ is a public key of $P_i$. Let $d$ be a parameter representing the threshold of being selected, and let $p$ be the unit of penalty. Set $P$:={}.

**1)** ***Register***: receive action (***MPCNMContract***, ***Register***, *Auth*, (*from*, *s*, *Pk*, *asset(v)*))

— if ( $P_s \notin P$ ), $P_s$ is not registered,
$P = P \cup P_s$ ,
*transfer*(*from*, *_self*, *asset*(*v*)),
$P_s.A = from$ , $P_s.B = v$ , $P_s.Pk = Pk$ , $P_s.R = 0.5$

**2)** ***Deposit:*** receive action (***MPCNMContract***, ***Deposit***, *Auth*, (*from*, *s*, *asset*(*v*)))

— if ( $P_s \in P$ ), $P_s$ has already registered,
*transfer*( $P_s.A$ , *_self, asset*(*v*)),
$P_s.B = P_s.B + v$

**3)** ***SelectQuorum***: receive action (***MPCNMContract***, ***SelectQuorum***, *Auth*, (*from*, *n*))

$P_Q$=***Select***(*P*, *balance*, *reputation*, *n*),
$P_Q = \{P_1, ..., P_n\}$
Send action (***from***, ***Setup***, *Auth*,(*_self*, $P_Q$ ))

4) ***Withdraw***: receive action (***MPCNMContract***, ***Withdraw***, *Auth*, (*from*, *s*))

— if( $P_s \in P$ ), $P_s$ has already registered,
*transfer*(*_self*, $P_s.A$ , *asset*( $P_s.B$ )),
$P_s.B = 0$

5) ***Update*: receive action (*MPCNMContract***, ***Update***, *Auth*, (*from*, $P_H$ , $P_C$ ))

— For all $P_c \in P_C$
$P_c.B = P_c.B - p / P_c.R$ , $P_c.R = P_c.R - r$
$penalty = p \times |C|$
— For all $P_h \in P_H$
$P_h.B = P_h.B + (\text{penalty} \times P_h.R / sum(P_H.R))$
— *If* $|P_H| \geq t$
• $P_h.R = P_h.R + r$

**FIGURE 4.** The formal definition of a MPC node management contract.

### D. MPC PROTOCOL

In this part, we can take advantage of some studies of existing multi-party protocols based on PVSS [40]–[42]. In this paper, we mainly focus on the fairness and robustness of MPC. How to construct a secure MPC protocol using PVSS is beyond the scope of this paper. As described in [22], the parties compute secret shares of either a useless intermediate value or the outputs of the function $f$. So in the BFR-MPC, Users and MPC nodes will broadcast the shares to blockchain after the execution of each round. Rewards or penalties are automatically executed by the Smart Contract on the blockchain platform. What the Users and MPC nodes need to do at the end of each round is shown in Fig.6.

**ComputeContract**

**Init:** Set $P_H = \varnothing$ and $P_C = \varnothing$ which represent the set of honest participants and corrupt participants respectively and $U = \varnothing$ denotes the set of Users. Let $\tau$ be the current block height of blockchain , $l$ be the last round of the computation, $t$ be the threshold of PVSS and n be the quorum of MPC nodes for this computation task. Let $M_{i,\rho} = \{(Enc_{P_j.Pk}(m_j), c_j)\}$ , $0 < i \le n$ , $0 < j \le n$ , $1 < \rho \le l$ denote the message that the MPC node $P_i$ should broadcast to $P_j$ in round $\rho$ whose initial value are all $\perp$ and $computescript_{\rho,i}$ be the compute script for $P_i$ in round $\rho$ . Let $(y_1, ..., y_n)$ denote the shares of output be initialized to $\perp$ too. Set *abort* = *false*.

Send action (***MPCNMContract***, ***SelectQuorum***, *Auth*, (*_self*, *n*))

1) ***Setup***: receive action (***ComputeContract***, ***Setup***, **Auth**, (*from*, $P = \{P_1, ..., P_n\}$ ))

— if (*from* == ***MPCNMContract***)

$P_H = P$

2) ***Input***: receive action (***ComputeContract***, ***Input***, *Auth*, ($U_s$, *asset*(*v*), $\{(Enc_{P_i.Pk}(x_i), c_i), 0 \le i \le n\}$ ))

— If $\tau \le RoundToHeight(1)$

*transfer*($U_s$,*_self*, *asset*(*v*))

- If $PVSS(Enc_{P_i.Pk}(x_i), c_i)$ == ***true***
  set $U = U \cup U_s$
- If $PVSS(Enc_{P_i.Pk}(x_i), c_i)$ == ***false***
  *abort* = *true*

3) ***Withdraw***: receive action (***ComputeContract***, ***Withdraw***, *Auth*, ($U_s$ )

— if (*abort* == *true*) and $U_s \in U_H$

*transfer*(*_self*, $U_s$ , *asset*(*_self.balance* /| $U$ | )

4) ***Compute***: receive action (***ComputeContract***, ***Compute***, *Auth*, ($P_s$, $\rho$ , $\{Enc_{P_j.Pk}(d_j), c_j, 1 \le j \le n\}$ ))

— If $RoundToHeight(\rho - 1) < \tau \le RoundToHeight(\rho)$ and $P_s \in P_H$

- If $|P_H| < t$ , abort = true,
  Send action (***MPCNMContract***, ***update***, *Auth*, (*_self*, $P_H$ , $P_C$ ))
- If $\rho \ge 3$ , check the MPC node that did not submit a message in the last round. If any $M_{i,\rho-1} = \perp$ , $0 < i \le n$ , move $P_i$ to the corrupted set. $P_H = P_H / P_i$ , $P_C = P_C \cup P_i$
- If $PVSS(Enc_{P_j.Pk}(d_j), c_j)$ == ***true***
  $M_{s,\rho} = \{(Enc_{P_j.Pk}(d_j), c_j), 0 \le j \le n\}$

5) ***Output***: receive action (***ComputeContract***, ***Output***, *Auth*, ($P_s$, *v*))

— If $RoundToHeight(l-1) < \tau \le RoundToHeight(l)$ and $P_s \in P_H$

- If $|P_H| < t$ , abort = true,
  Send action (***MPCNMContract***, ***update***, *Auth*, (*_self*, $P_H$ , $P_C$ ))
- Check the MPC node that did not submit a message in the last round. If any $M_{i,l-1} = \perp$ , $0 < i \le n$ , move $P_i$ to the corrupted set. $P_H = P_H / P_i$ , $P_C = P_C \cup P_i$
- If *verify*(*v*, *COMM*(*v*)) == ***true***, $y_s = v$

6) ***Pay***: receive action (***ComputeContract***, ***Pay***, *Auth*, ($P_s$))

— If $\tau > RoundToHeight(l)$ and $P_s \in P_H$

- Check the MPC node that did not submit its share of output in the final round. If any $y_i = \perp$ , $0 < i \le n$ , move $P_i$ to the corrupted set. $P_H = P_H / P_i$ , $P_C = P_C \cup P_i$
- If $|P_H| < t$ , abort = true,
- If $|P_H| \ge t$ ,
  *transfer*(*_self*, *Ps.A*, *asset*(*_self.balance* × $P_h.R$ / *sum*( $P_H.R$ ) ))
- Send action (***MPCNMContract***, ***Update***, *Auth*, (*_self*, $P_H$ , $P_C$ ))

**FIGURE 5.** The formal definition of compute contract.

**FIGURE 5.** The formal definition of compute contract.

## VI. SCHEME ANALYSIS

In this paper, we mainly study the fairness and robustness of MPC. The security of the proposed scheme depends on a secure multi-party computation protocol based on PVSS. The verification of input, output and the intermediate values falls into another category. So in this section, we provide fairness analysis, robustness analysis and performance analysis of BFR-MPC.

### A. FARINESS ANALYSIS

*Definition 1:* In a repetitive MPC Game $G = \{P_i, \mathcal{A}_i, R_i, u_i\}$, where $1 < i \le N$, is one that is repeated an infinite number of times by different subsets of players. Each player $P_i$ has a set of action $\mathcal{A}_i$, a reputation value $R_i^\tau$, $\tau = 1, 2, 3...$ where $\tau$ denotes the number of games $P_i$ played in and a utility function $u_i$. In each game:

- A subset of $n \le N$ is chosen to participant based on their reputation value. The selected players will have an income when completing the task and the greater the reputation value, the easier it is for participants to be selected for the next game.
- The selected player $P_i$, $1 < i \le n$, will estimate his utility by the utility function $u_i(a_i, R_i)$. Players select his action $a_i$ according to $u_i$.
- The reputation $R_i$ of each player will be updated on the blockchain publicly and trustful.

We assume that each player $P_i$'s action $a_i \in \{\mathcal{C}, \mathcal{D}, \perp\}$, where $\mathcal{C}$ denotes "cooperation", $\mathcal{D}$ denotes "defection" and $\perp$ denotes that not selected to participate in this game. Players who are not selected will not affect the result of this game, nor will they affect the decision of other players, so we only consider $\mathcal{C}$ and $\mathcal{D}$ of the players in current game. Let $\mathcal{A} \stackrel{def}{=} \mathcal{A}_1 \times ... \times \mathcal{A}_n$ be an action profile and $\boldsymbol{a} = (a_1, ..., a_n) \in \mathcal{A}$ be the current game's output.

**Users:**

**1)** $\tau \leq$ ***RoundToHeight(1)***

Query ***ComputeContract*** to acquire information of MPC nodes $\{P_1,...,P_n\}$ participating in current computation task . Each User $U$ generates his shares $\{x_1,...,x_n\}$ and corresponding commitments $c_i = COMM(x_i)$ of $x_i$ . Then send transaction:

$(Ref_{block},t,Sig_U(Chain\_Id,Tx),Action(\textbf{\textit{ComputeContract}},\textbf{\textit{Input}},Auth_U,(U,asset(v),\{(Enc_{P_i.Pk}(x_i),c_i),0\leq i\leq n\})))$

**2)** ***RoundToHeight(2)*** $< \tau \leq$ ***RoundToHeight(l)***

If someone broadcasted an incorrect shares in round 1 resulting the task abort, an honest User could send the following transaction to withdraw his fee and the penalty of malicious users.

$(Ref_{block},t,Sig_U(Chain\_Id,Tx),Action(\textbf{\textit{ComputeContract}},\textbf{\textit{Withdraw}},Auth_U,(U)))$

3) $\tau >$ ***RoundToHeight(1)***

Query ***ComputeContract*** to get all shares of output and reconstruct it.

**MPC Nodes:**

**1)** ***RoundToHeight(2)*** $< \tau \leq$ ***RoundToHeight(l - 1)***

For a MPC Node $P_s$, it querys ***ComputeContract*** to acquire the compute scripts of current round, and executes the computation locally to get an intermediate value $d$. Send transaction:

$(Ref_{block},t,Sig_{P_s}(Chain\_ID,Tx),Action(\textbf{\textit{ComputeContract}},\textbf{\textit{Compute}},Auth_{P_s},(P_s,r,\{(Enc_{P_i.Pk}(d_i),c_i),0\leq i\leq n\})))$

2) ***RoundToHeight(l - 1)*** $< \tau \leq$ ***RoundToHeight(l)***

Broadcast its share of output $y_s$, send transaction:

$(Ref_{block},t,Sig_{P_s}(Chain\_Id,Tx),Action(\textbf{\textit{ComputeContract}},\textbf{\textit{Output}},Auth_{P_s},(P_s,y_s)))$

3) $\tau >$ ***RoundToHeight(l)***

Any honest MPC node can send the following transaction to get the payment of this computation task.

$(Ref_{block},t,Sig_{P_s}(Chain\_Id,Tx),Action(\textbf{\textit{ComputeContract}},\textbf{\textit{Pay}},Auth_P,(P_s)))$

**FIGURE 6.** The formal definition of MPC protocol.

*Definition 2:* Let $coin_i(\boldsymbol{a})$ denote the coin income that earned by $P_i$ in current game, $l_i(\mathbf{a}) \in \{0, 1\}$ denote whether the player $P_i$ has obtained the computation output and set $\delta(\mathbf{a}) = \sum_i l_i(\boldsymbol{a})$. For our Scheme, we make that:

$$\begin{aligned} l_i(\boldsymbol{a}) &< l_i(\boldsymbol{a'}) \text{ but } coin_i(\boldsymbol{a}) > coin_i(\boldsymbol{a'}) \\ &\Rightarrow u_i(va) > u_i(\boldsymbol{a'}) \quad (1) \\ l_i(\boldsymbol{a}) &> l_i(\boldsymbol{a'}) \text{ but } coin_i(\boldsymbol{a}) < coin_i(\boldsymbol{a'}) \\ &\Rightarrow u_i(\boldsymbol{a}) < u_i(\boldsymbol{a'}) \quad (2) \end{aligned}$$

If set a suitable incentive with appropriate rewards and penalties, the above definition are completely reasonable. Let $\Omega > 0$ be a unit of utility. The revenue of a player $P_i$ consists of three parts:

A. *Reputation:* $(R_i^{\tau}(\boldsymbol{a}) - R_i^{\tau-1}(\boldsymbol{a}))\omega_i(r) \times \Omega$, where $\omega_i(r)$ denotes a curve that represents the impact of the difference $r$ in reputation on future earnings. The $r$ and $\omega_i(r)$ are all positive.

$$R_i^{\tau}(\boldsymbol{a}) - R_i^{\tau-1}(\boldsymbol{a}) = \begin{cases} 1r, & if\ a_i = \mathcal{C} \\ -1r, & if\ a_i = \mathcal{D} \end{cases}$$

B. *Coin:* $\begin{cases} \dfrac{R_i}{\sum_{j=1}^{h} R_j}(\sum_{k=1}^{c} p_k + fee) \times \Omega, & if\ a_i = \mathcal{C} \\ -p/R_i \times \Omega, & ifa_i = \mathcal{D} \end{cases}$

C. *Output:* For a selfish adversary, the output is known to himself, and the fewer people know the result, the greater the gain. So:

$$\frac{l_i(\boldsymbol{a})}{\delta(\boldsymbol{a}) + 1} \times \Omega$$

Let $\rho_1$, $\rho_2$, $\rho_3$ denote the weights of A, B and C respectively. For a normal player, part B of each future game utility is closely associated with his reputation value, so $\rho_1 \gg \rho_2$. According to the **definition 2**, we can get $\rho_2 > \rho_3$. Let $\mathcal{C}_i(\mathcal{D}_i)$ denote player $P_i$ cooperate (defect), $\mathcal{C}_{-i}(\mathcal{D}_{-i})$ denote all the players cooperate (defect) other than $P_i$. Let $\mathcal{M}_{-i}^{h}$ denote that $h$ of players other than $P_i$ are cooperative. The following 5 situations will occur in the utility of $P_i$:

(1) If all players are cooperative and everyone receives the output. $l_i = 1, \delta = n$.

$$u_i(\boldsymbol{a})^{(\mathcal{C}_i,\mathcal{C}_{-i})} = \Omega(\rho_1 r\omega_i + \rho_2(\frac{R_i}{\sum_{j=1}^{n} R_j} \times fee) + \frac{\rho_3}{n+1})$$

(2) If player $P_i$ cooperates but other players some cooperate and some defect, $h \geq t-1, l_i = 1, \delta = n$. All players get the output, but the players who do not cooperate will be fined.

$$u_i(\boldsymbol{a})^{(\mathcal{C}_i,\mathcal{M}_{-i}^{h\geq t-1})} = \Omega(\rho_1 r\omega_i + \rho_2(\frac{R_i}{\sum_{j=1}^{h} R_j} \times (\sum_{k}^{c} p_k + fee)) + \frac{\rho_3}{n+1})$$

(3) If player $P_i$ cooperates but other players some cooperate and some defect, $h < t-1, l_i = 0, \delta = 0$. The protocol will be aborted so that no one will be able to get the output, but the players who do not cooperate will be fined.

$$u_i(\boldsymbol{a})^{(\mathcal{C}_i,\mathcal{M}_{-i}^{h<t-1})} = \Omega(\rho_2(\frac{R_i}{\sum_{j=1}^{h} R_j} \times (\sum_{k}^{c} p_k)))$$

(4) If player $P_i$ defects, but the number of players who cooperated exceeded the threshold $t$. All of the players will get the output, but player $P_i$ will be fined. $l_i = 1, \delta = n$.

$$u_i(\boldsymbol{a})^{(\mathcal{D}_i,\mathcal{M}_{-i}^{h\geq t})} = \Omega(-\rho_1 r\omega_i - \rho_2(p/R_i) + \frac{\rho_3}{n+1})$$

(5) If player $P_i$ defects, and the number of players who cooperated is less than the threshold $t$. The protocol will be

aborted so that no one will be able to get the output and the players who do not cooperate will be fined. $l_i = 0, \delta = 0$.

$$u_i(\boldsymbol{a})^{(\mathcal{D}_i, \mathcal{M}_{-i}^{h<t})} = \Omega(-\rho_1 r\omega_i - \rho_2(p/R_i))$$

We now analysis these five scenarios:

If player $P_i$ cooperates (cases (1)-(3)), regardless of other player's cooperation or defection. We have:

$$u_i^{\mathcal{C}}(\boldsymbol{a}) \geq \Omega(\rho_2(\frac{R_i}{\sum_{j=1}^{h} R_j} \times (\sum_{k}^{c} p_k)))$$

If player $P_i$ defects (cases (4)-(5)). We have:

$$u_i^{\mathcal{D}}(\boldsymbol{a}) \leq \Omega(-\rho_1 r\omega_i - \rho_2(p/R_i) + \frac{\rho_3}{n+1})$$

Because of $\rho_1 \gg \rho_2$, it is easy to prove that $u_i^{\mathcal{C}}(a) > u_i^{\mathcal{D}}(a)$. In extreme cases, if a malicious adversary registered as a newcomer, and regardless of the influence of reputation, we will have:

$$u_i^{\mathcal{D}}(\boldsymbol{a}) \leq \Omega(-\rho_2(p/R_i) + \frac{\rho_3}{n+1})$$

According to the **definition 2**, we can also get $u_i^{\mathcal{C}}(\boldsymbol{a}) > u_i^{\mathcal{D}}(\boldsymbol{a})$.

So it can be said that for a player $P_i$, $\mathcal{C}_{-i}$ strictly dominates $\mathcal{D}_{-i}$ and vector $a^{\mathcal{C}} = (a_1^{\mathcal{C}}, ..., a_n^{\mathcal{C}})$ is a strict Nash equilibrium strategy. BFR-MPC has a good fairness.

### B. ROBUSTNESS ANALYSIS

The basis of BFR-MPC are blockchain and a $(t, n)$-theshold PVSS, and it proceeds in rounds. Blockchain can provide the ability of timestamps to enable protocols to execute synchronously. Every participant who sends incorrect message out of the protocol will be detected by the Smart Contract and be kicked out at the beginning of the next round. All of this is public and transparent, running automatically in a blockchain environment. In the absence of a centralized node, this is not possible in previous work. Each participant has its own account and corresponding key pairs on the blockchain, and the identity cannot be forged. The robustness of BFR-MPC is mainly reflected in the following aspects:

1)*Setup Phase:* The ledger publicly maintains an reputation value determined by their past behavior for each participant. In order to be easily selected, participants need to accumulate their reputation values for a long time. To become an MPC node, a deposit needs to be paid in the contract, which avoids ''The Sybil Attack''. That is, the adversary cannot terminate the protocol by adding more nodes. If the protocol is aborted, the phase can be performed again, and select a new subset to start with.

2) *Input Phase:* Users are required to pay a certain amount of coins as the service fee for this current task when submitting the input. The Smart Contract will check whether the input of the Users is correct. If a User does not submit the correct input within the specified time, the service charge will be transferred to the honest Users. The existence of the service fee will enable the program to resist DoS attacks from Users.

3) *Compute Phase:* The intermediate value of the MPC node is also detected by the Smart Contract, and if a valid message is not submitted, the node will be moved to the corruption set at the beginning of the next round. As long as the number of corrupt participants does not exceed *n-t*, the protocol will be finally finalized.

In summary, any participant in the BFR-MPC that deviates from the protocol will be detected by the Smart Contract and will be prohibited from participating in the next work. The scheme uses a $(t, n)$-theshold secret share, as long as the setting of a suitable $t$, there is a certain degree of fault tolerance. Even if the protocol is aborted, Users can select a new subset to restart the agreement. So BFR-MPC has a good robustness.

### C. PERFORMANCE ANALYSIS

In this section, we provide the experimental evaluation of the proposed scheme. We have tested these on a MacBook Pro (2017) with Intel (R) Core (TM) i5 CPU that clocks at 3.1 GHz and has 8.0 GB of RAM. The code was written in Java. We downloaded a Shamir's secret shared code from GitHub.

As we all know, in all blockchain-based schemes, the efficiency of computations on the chain depends on the average block production time of the selected blockchain. For example, the average processing time for a transaction on Bitcoin is 10 minutes. BFR-MPC chooses the EOS blockchain platform. The current consensus mechanism of EOS is BFT-DPoS, which can accurately generate a block every 0.5 seconds. In BFR-MPC, all operations in **MPCNMContract** and **ComputeContract** are performed on chain.

In each round, the main time overhead of off-chain execution is a data distribution process of PVSS which includes a Shamir's secret shared and multiple asymmetric encryption executions. Their execution time is mainly related to the size of the quorum for MPC nodes that choosed to participate in the current computation task in the **Init** phase of **ComputeContract**. Fig. 7 reflects the relationship between the cost of time for each MPC per round and the size of quorum for the selected MPC nodes.

We can conclude that the cost of time for each round of BFR-MPC increases exponentially with the increase of the size of quorum for selected MPC nodes in **ComputeContract**. This is because adding MPC nodes to a specific computation task will result in more asymmetric encryption calculations per round. Even worse, a higher power polynomial is needed for secret sharing, resulting that the generated shares tasks up a larger number of bytes, making the encrypted data of the asymmetric encryption larger and consuming more time. We can see that when the quorum exceeds 40, users will make a great sacrifice in efficiency if they choose more MPC nodes to perform a computation task for a better privacy.

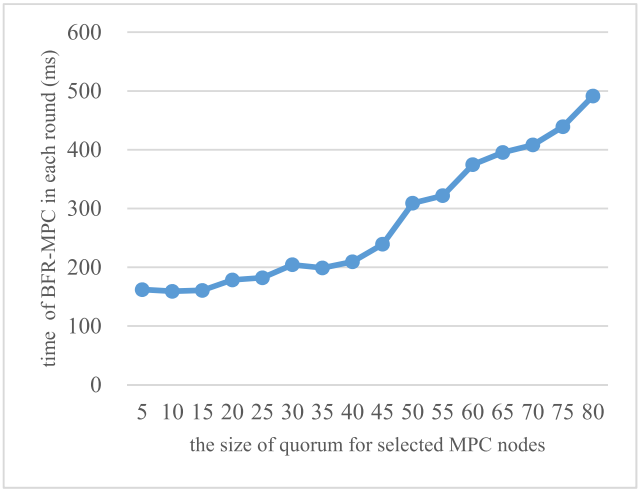



**FIGURE 7.** The time cost per round of each node varies with the size of quorum for selected MPC nodes increasing.

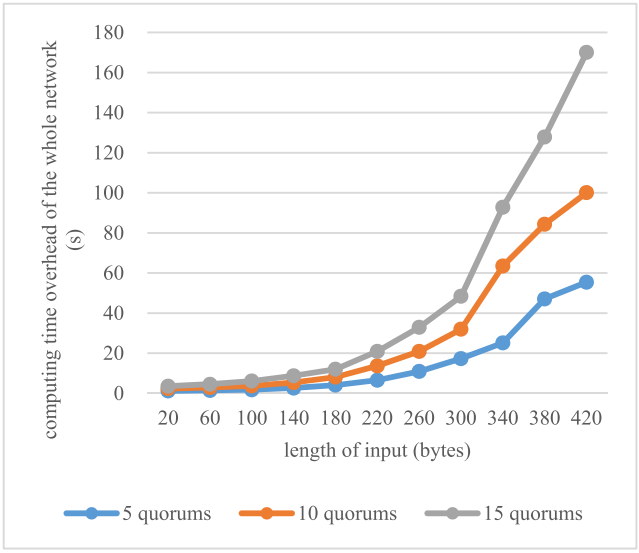



**FIGURE 8.** Computing time overhead of the whole network varies with input size.

Fig. 8 illustrates that adding quorum is costly with larger input. In our experiment, when the size of input exceeds 250 bytes, even adding a node can be very expensive.

## VII. COMPARISON WITH RELATED WORKS

In previous studies on multi-party computation, most of them only focus on the privacy of input, correctness or efficiency of computation. Different from previous researches, BFR-MPC concentrates on the fairness and robustness of MPC. Although rational multi-party computation based on cryptography and Game Theory also considers fairness, most of them only regard multi-party computation protocol as a one-time game which is not a particularly realistic situation. There are also some studies focusing on running a robust multi-party computation protocol, but they do not have an actual operating environment, and they cannot kick out malicious nodes without a centralized node.

**TABLE 1.** The comparison of BFR-MPC and some other schemes.

| | BFR-MPC | Ref. [31] | Ref. [34] | Ref. [14] |
|---|---|---|---|---|
| Operating environment | EOS | Bitcoin | Bitcoin | Ethereum |
| Fairness | √ | √ | √ | √ |
| Robustness | √ | × | × | √ |
| Commonality | √ | × | × | √ |
| Number of interactions with chain | O(n) | O(2n) | O(2n) | $O(n^2)$ |

Only papers [14], [30], [31], [33], [34] have some resemblance to our work which they are all MPC based on blockchain. We will compare BFR-MPC with these other solutions from operating environment, the time complexity of interacting with blockchain, fairness, robustness and commonality.

The solutions in [30], [33] are all proposed for multi-party computation in a special scenario, so their commonality is poor and we are not going to do much analysis of both situations. As shown in TABLE 1, all schemes are based on blockchain, and they are all have a better fairness. The schemes in [31], [34] are constructed on Bitcoin network without considering the problem of robustness, and only one malicious node can launch a DoS attack. Although the scheme in [14] considers the robustness, it can only identify the nodes that abort the protocol whose robustness is weak. In terms of time complexity of interaction with the blockchain, the scheme of [31], [34] are based on an ideal functionality called $\mathcal{F}^*_{CR}$, which needs to interact with the blockchain $2n$ times per round, the scheme of [14] needs to interact with the blockchain $n^2$ times per round, while BFR-MPC just only $n$ times. Although [31] discussed to build a universal multi-party computation protocol with PVSS, it is limited by the limitations of the Bitcoin network. Moreover, in order to achieve their goals, a slight change in the Bitcoin protocol is needed, which seems almost impossible, while our solution is based on the EOS blockchain and there are no such problems. The feasibility of implementing MPC based on Ethereum is analyzed in [14], but no specific implementation scheme is given.

In addition, any blockchain-based solution needs to discuss the performance of the blockchain platform that is selected. Bitcoin generates a block for about 10 minutes. in order to prevent forking, it is necessary to be confirmed after 6 blocks, so the time for a transaction to be officially written into the block is about 60 minutes, which is too inefficient. Moreover, the scripting language of Bitcoin is non-Turling Complete, which makes it impossible to build complex functions and is very poor in terms of scalability. Ethereum produces a block in about 15 seconds, it is generally recommended that more than 15 blocks confirm. So, the confirmation time of

a transaction has been greatly improved in Ethereum. At the same time, Ethereum has a good support from Smart Contract. However, there is also a big problem in common with Bitcoin and Ethereum, that both of them have adopted PoW (proof of work) as a consensus mechanism. For this reason, when the network is congested, transactions can often not be immediately packaged into the block unless very expensive fees are paid. This makes it that applying Bitcoin or Ethereum to our scenario [14], [30], [31], [33], [34] can be very likely: the last round of transactions have been broadcast, but several rounds later, the transactions may still not be processed. Therefore, the use of Bitcoin or Ethereum to solve the unfairness in multi-party computation can only stay in a theoretical stage and cannot be realized. These conditions are completely different in the EOS blockchain. The consensus mechanism of the EOS blockchain is BFT-DPoS, where a block is generated in 0.5 seconds, which allows transactions to reach second-level confirmation. It is completely free for a user to initiate a transaction on the EOS blockchain. Further, EOS blockchain has a more powerful account system and permission control mechanism. All these advantages make the EOS blockchain more suitable for building a fair and robust multi-party computation protocol.

In summary, compared to other related works, BFR-MPC has a better fairness, robustness, commonality and also has a good performance in efficiency. Most importantly, BFR-MPC is more practical and a realistic solution.

## VIII. CONCLUSIONS

In order to meet the requirements of fairness and robustness in the multi-party computation, a fair and robust MPC scheme based blockchain was proposed in this paper. A public reputation system is maintained in the proposed scheme where each participant has a reputation value where a more reputable party has greater chance to be selected. Then a subset was selected based on the reputation value to execute a common MPC protocol based on PVSS. The output and intermediate values in each round all need to be verified by the Smart Contract. Both sending an incorrect message and not sending any message within the specified time will be detected by Smart Contract. Further, an incentive mechanism encourage all participants to be cooperative, honest participants will gain more and more benefits, while the corrupt participants will be increasingly penalized. Blockchain maintains a ledger publicly with a non-tamperable feature, but the disclosure of the ledger means privacy issues, while MPC is precisely to solve the problem of privacy protection. In the future work, we will study what helps MPC can provide for blockchain in terms of privacy protection.

## REFERENCES


[1] A. C. Yao, "Protocols for secure computations," in *Proc. 23rd Annu. Symp. Found. Comput. Sci.*, vol. 82, Nov. 1982, pp. 160–164.

[2] O. Goldreich, S. Micali, and A. Wigderson, "How to play ANY mental game," in *Proc. 19th Annu. ACM Symp. Theory Comput. (STOC)*, 1987, pp. 218–229.

[3] Y. Aumann and Y. Lindell, "Security against covert adversaries: Efficient protocols for realistic adversaries," in *Theory of Cryptography*, S. P. Vadhan, Eds. Berlin, Germany: Springer, 2007, pp. 137–156.

[4] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP: A system for secure multi-party computation," in *Proc. 15th ACM Conf. Comput. Commun. Secur. (CCS)*, 2008, pp. 257–266.

[5] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, "Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation," in *Theory of Cryptography*. Berlin, Germany: Springer, 2006, pp. 285–304.

[6] I. Damgård, M. Geisler, M. Krøigaard, and J. B. Nielsen, "Asynchronous multiparty computation: Theory and implementation," in *Public Key Cryptography—PKC*. Berlin, Germany: Springer, 2009, pp. 160–179.

[7] M. Burkhart, M. Strasser, D. Many, X. Dimitropoulos, "SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics," in *Proc. 19th USENIX Security Symp.*, 2010, pp. 223–240.

[8] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson, "High-performance secure multi-party computation for data mining applications," *Int. J. Inf. Secur.*, vol. 11, no. 6, pp. 403–418, Nov. 2012.

[9] E. Prouff and T. Roche, "Higher-order glitches free implementation of the AES using secure multi-party computation protocols," in *Cryptographic Hardware and Embedded Systems—CHES*. Berlin, Germany: Springer, 2011, pp. 63–78.

[10] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2003, pp. 145–161.

[11] J. Nielsen, "Extending oblivious transfers efficiently—How to get robustness almost for free," IACR Cryptol. ePrint Arch., Tech. Rep. 2007/215, 2007. [Online]. Available: http://eprint.iacr.org/2007/215

[12] D. Harnik, Y. Ishai, E. Kushilevitz, and J. B. Nielsen, "OT-combiners via secure computation," *Proc. 5th Conf. Theory Cryptogr. (TCC)*, 2008, pp. 393–411.

[13] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra, "A new approach to practical active-secure two-party computation," in *Proc. 32nd Annu. Cryptol. Conf. Adv. Cryptol. (CRYPTO)*, vol. 7417, 2012, pp. 681–700.

[14] A. Kiayias, H.-S. Zhou, and V. Zikas, "Fair and robust multi-party computation using a global transaction ledger," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2016, pp. 705–734.

[15] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2012, pp. 643–662.

[16] M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making SPDZ great again," in *Advances in Cryptology—EUROCRYPT*. Cham, Switzerland: Springer, 2018, pp. 158–189.

[17] G. Spini and S. Fehr, "Cheater detection in SPDZ multiparty computation," in *Information Theoretic Security*. Cham, Switzerland: Springer, 2016, pp. 151–176.

[18] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai, "Efficient constant round multi-party computation combining BMR and SPDZ," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2015, pp. 319–338.

[19] Y. Ishai, R. Ostrovsky, and V. Zikas, "Secure multi-party computation with identifiable abort," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2014, pp. 369–386.

[20] R. Cleve, "Limits on the security of coin flips when half the processors are faulty," in *Proc. 18th Annu. ACM Symp. Theory Comput. (STOC)*, 1986, pp. 364–369.

[21] J. Halpern and V. Teague, "Rational secret sharing and multiparty computation: Extended abstract," in *Proc. 36th Annu. ACM Symp. Theory Comput. (STOC)*, 2004, pp. 623–632.

[22] A. Lysyanskaya and N. Triandopoulos, "Rationality and adversarial behavior in multi-party computation," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2006, pp. 180–197.

[23] A. Groce and J. Katz, "Fair computation with rational players," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2012, pp. 81–98.

[24] M. Nojoumian and D. R. Stinson, "Socio-rational secret sharing as a new direction in rational cryptography," in *Decision and Game Theory for Security*. Berlin, Germany: Springer 2012, pp. 18–37.

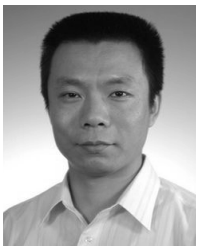
[25] G. Asharov, R. Canetti, and C. Hazay, "Towards a Game Theoretic View of Secure Computation," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer 2011, pp. 426–445.

[26] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[27] V. Buterin. (2013). *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper
[28] D. Larimer. (2017). *EOS.IO Technical White Paper*. [Online]. Available: https://steemit.com/eos/@eosio/eos-io-technical-white-paper
[29] M. Andrychowicz, S. Dziembowski, D. Malinowski, and Ł. Mazurek, "Fair two-party computations via Bitcoin deposits," in *Financial Cryptography and Data Security*. Berlin, Germany: Springer, 2014, pp. 105–121.
[30] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on Bitcoin," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 443–458.
[31] I. Bentov and R. Kumaresan, "How to Use Bitcoin to design fair protocols," in *Advances in Cryptology—CRYPTO*. Berlin, Germany: Springer, 2014, pp. 421–439, 2014.
[32] R. Kumaresan and I. Bentov, "How to Use Bitcoin to incentivize correct computations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2014, pp. 30–41.
[33] R. Kumaresan, T. Moran, and I. Bentov, "How to use Bitcoin to play decentralized poker," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2015, pp. 195–206.
[34] R. Kumaresan, V. Vaikuntanathan, and P. N. Vasudevan, "Improvements to secure computation with penalties," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2016, pp. 406–417.
[35] A. Shamir, "How to share a secret," in *Proc. Commun. ACM*, vol. 22, no. 1, pp. 612–613, 1979.
[36] N. Szabo. (1994). *Smart Contracts*. [Online]. Available: https://szabo.best.vwh.net/smart.contracts.html
[37] M. J. Osborne, *An Introduction to Game Theory*. New York, NY, USA: Oxford Univ. Press, 2004.
[38] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic voting," in *Advances in Cryptology—CRYPTO*, vol. 99. Berlin, Germany: Springer, 1999, pp. 148–164.
[39] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 839–858.
[40] M. Backes, F. Bendun, A. Choudhury, and A. Kate, "Asynchronous MPC with a strict honest majority using non-equivocation," in *Proc. ACM Symp. Princ. Distrib. Comput. (PODC)*, 2014, pp. 10–19.
[41] I. Damgård, Y. Ishai, and M. Krøigaard, "Perfectly secure multiparty computation and the computational overhead of cryptography," in *Advances in Cryptology—EUROCRYPT*. Berlin, Germany: Springer, 2010, pp. 445–465.
[42] A. Patra, A. Choudhury, and C. P. Rangan, "Efficient asynchronous verifiable secret sharing and multiparty computation," *J. Cryptol.*, vol. 28, no. 1, pp. 49–109, 2013.
[43] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority," in *Proc. 21st Annu. ACM Symp. Theory Comput.*, 1989, pp. 73–85.
[44] Z. Wang, S.-C. S. Cheung, and Y. Luo, "Information-theoretic secure multi-party computation with collusion deterrence," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 4, pp. 980–995, Apr. 2017.
[45] L. Zhou, L. Wang, Y. Sun, and T. Ai, "AntNest: Fully non-interactive secure multi-party computation," *IEEE Access*, vol. 6, pp. 75639–75649, 2018.
[46] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
[47] T. Li, H. Qin, J. Li, W. Gao, Z. Liu, and Q. Xu, "A brief survey on secure multi-party computing in the presence of rational parties," *J. Ambient Intell. Humanized Comput.*, vol. 6, no. 6, pp. 807–824, 2015.
[48] Z. Yang, K. Yang, L. Lei, K. Zheng, and V. C. M. Leung, "Blockchain-based decentralized trust management in vehicular networks," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1495–1505, Apr. 2019.

**HONGMIN GAO** was born in 1987. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Beijing University of Posts and Telecommunications. His research interests include blockchain, encode cryptography, and network and information security.

**ZHAOFENG MA** was born in 1974. He received the Ph.D. degree from Xi'an Jiaotong University, in 2004. He did his post-doctoral research work in Tsinghua University, from 2005 to 2007. Since 2007, he has been with the Beijing University of Posts and Telecommunications, Beijing, China, where he is involved in education and research work. His research interests include information security, digital rights management, and blockchain.



**SHOUSHAN LUO** was born in 1962. He is currently a Professor and a Ph.D. Supervisor with the School of Cyber Security, Beijing University of Posts and Telecommunications. His research interests include blockchain, encode cryptography, and network and information security.

**ZHEN WANG** was born in 1989. He is currently pursuing the Ph.D. degree with the School of Cyber Security, Beijing University of Posts and Telecommunications. His research interests include blockchain and digital rights management.