Constructing Fair Secure Multiparty Computation Postprints with Blockchain

Authors: Huang Jianhua, Jiang Yahui, Li Zhongcheng

Date: 2018-11-29T00:00:00+00:00

Abstract

To address the problem of achieving fairness in secure multi-party computation (MPC) when most participants are dishonest, this paper proposes a fair secure MPC protocol by constructing a penalty mechanism based on blockchain smart contracts. The protocol consists of two phases: an MPC phase based on verifiable secret sharing and a fair secret reconstruction phase, where participants can obtain the final output upon collecting t+1 correct shares. The protocol employs homomorphic commitments to verify the correctness of secret shares, utilizes a timeout mechanism to detect premature termination behavior by malicious participants, and imposes economic penalties on malicious parties. Security analysis demonstrates that honest participants can obtain the final output, or otherwise receive economic compensation; performance analysis indicates that participants are only required to deposit a one-round deposit and that the majority of complex secret share verification work is performed off-chain, thereby ensuring the execution efficiency of the protocol.

Full Text

Preamble

Constructing Fair Secure Multi-Party Computation Based on Blockchain

Huang Jianhua, Jiang Yahui, Li Zhongcheng

(School of Information Science & Engineering, East China University of Science and Technology, Shanghai 200237, China)

Abstract: This paper proposes a fair secure multi-party computation (MPC) protocol to address the problem that fairness cannot be achieved when there is no honest majority. The protocol constructs a penalty mechanism based on blockchain smart contracts. It consists of two phases: an MPC phase based

on verifiable secret sharing and a fair secret reconstruction phase. Participants can obtain the final output by collecting just t+1 correct shares. The protocol utilizes homomorphic commitments to verify the correctness of secret shares, employs timeouts to identify premature abort behaviors of malicious parties, and financially punishes aborting parties. Security analysis shows that honest participants can obtain the final output; otherwise, they receive financial compensation. Performance analysis demonstrates that the protocol requires only one coin-transfer round and that a large number of complex secret share verification operations are performed off-chain, ensuring implementation efficiency.

Keywords: secure multi-party computation; blockchain; smart contracts; fairness

0 Introduction

Secure multi-party computation (MPC), which originated from Yao' s Millionaires' Problem [1] and evolved through the work of Goldreich, Micali, and Wigderson [2], has become a prominent research topic in cryptography. MPC addresses the problem of privacy-preserving collaborative computation among mutually distrustful parties. In secure MPC scenarios, two or more parties holding secret inputs wish to jointly compute a function and obtain their respective outputs. Throughout this process, participants should learn nothing beyond their prescribed outputs. A secure MPC protocol must guarantee not only the correctness of the output and the privacy of participants' inputs but also fairness —meaning that either everyone receives the output or no one does.

Most existing MPC protocols [3–5] satisfy these security properties only against semi-honest adversaries or when the majority of participants are honest. Notable exceptions include the SPDZ line of research [6–8], which maintains input privacy and output correctness even when most participants are malicious. However, these efficient MPC protocols cannot guarantee fairness [9]. As early as 1986, Cleve [10] proved that fairness in MPC protocols cannot be achieved when more than half of the participants are dishonest. Currently, protocol designers primarily focus on security and correctness, as fairness is difficult to achieve in practice due to malicious participants aborting the protocol prematurely [11].

The most common security models are the semi-honest and malicious adversary models, which assume participants follow the protocol specification. Semi-honest participants attempt to infer others' secrets, while malicious participants actively deviate from the protocol to disrupt execution. However, in reality, participants are often rational—they act to maximize their own interests. With appropriate incentives, all (or most) participants can be deterred from malicious behavior. Consequently, a novel approach to achieving fairness involves incorporating economic penalty mechanisms into MPC protocols. Specifically, participants deposit collateral before computation begins. After computation, their behavior is verified: honest participants have their deposits returned, while malicious participants' deposits are distributed among the honest ones. A key

challenge in implementing such penalty-based fair computation is managing participant deposits and reliably adjudicating behavior without a trusted party.

Bitcoin [12] was the first digital cryptocurrency to eliminate trusted intermediaries, and Ethereum [13] extended this concept by supporting smart contracts on the blockchain. Blockchain technology, through cryptographic techniques and consensus mechanisms, solves the trust problem among mutually distrustful parties. To address the fairness problem in MPC when most participants are dishonest [14], this paper proposes a Blockchain-based Fair and Secure Multi-Party Computation protocol (BFSMPC). BFSMPC uses Ethereum smart contracts: participants submit deposits to the smart contract, which adjudicates honesty and handles deposits accordingly, while employing timeout mechanisms to detect premature termination by malicious parties. Under the cryptographic model based on blockchain, BFSMPC guarantees fairness even when most participants are dishonest. By moving extensive verification operations off-chain, BFSMPC simplifies smart contract operations and ensures protocol efficiency.

1 Related Work

MPC solves the problem of privacy-preserving collaborative computation among mutually distrustful parties. Achieving fairness in secure MPC remains a key focus and challenge in academia. Jakobsen et al. [15] designed a cloud-computing-based secret-sharing MPC protocol where each server receives all blinded outputs from users and subsequently broadcasts them to all users, enabling verification of output consistency across servers. However, this approach does not truly solve the fairness problem, as honest servers may never receive users' blinded outputs when most servers are malicious. Cleve [10] proved that complete fairness cannot be achieved in two-party computation. Although Gordon et al. [16] showed that complete fairness exists in certain specific two-party computations, these are special cases lacking generality, necessitating a relaxation of the fairness notion.

Ideal security requires computational indistinguishability between the real world and an "ideal world" where a trusted third party ensures all participants receive outputs. In reality, fairness is guaranteed only when most participants are honest. Goldreich [17] weakened this security property by modifying the ideal world to no longer guarantee fairness, while maintaining computational indistinguishability. Protocols satisfying all security properties except fairness are called "secure with abort." Katz [18] alternatively preserved the ideal world but relaxed the simulation concept, requiring that the distinguishability between real and ideal worlds be at most $1/p + \text{negl}$, where p is a specified polynomial and negl is a negligible function. Protocols meeting this condition are called "1/p-secure." Gordon and Katz [19] defined partial security for two-party computation under the standard real/ideal world paradigm, expanding the research domain of fair cryptographic protocols.

Ishai et al. [9] proposed identifiable abort protocols where all participants are

notified and can identify malicious aborting parties. Tian et al. [20] presented a UC-secure fair MPC model including fair addition and multiplication ideal functions, designing corresponding fair secure addition and multiplication protocols. Gordon et al. [11, 21] constructed fair secure computation protocols using primitive assistance, where primitives act as trusted parties without requiring prior knowledge of the computation. Fitzi et al. [21] introduced a Universal Black Box (UBB) primitive achieving complete fairness in both two-party and multi-party settings, which receives a circuit and agreement value from participants and outputs the circuit result for participants providing identical circuits. Its limitation lies in input size and runtime depending on the target function' s complexity. Gordon et al. [11] demonstrated that no "short"primitive exists for complete fairness and introduced a fair consistent secret reconstruction primitive requiring multiple invocations. These works construct fair MPC from a protocol-centric perspective, yielding largely negative results. In contrast, blockchain-based approaches treat participants as rational and leverage cryptocurrency incentives to encourage honest behavior.

Andrychowicz et al. [22] used Bitcoin to construct timed commitments, forcing participants to reveal secrets within a time limit or face financial penalties. Bentov and Kumaresan [23] formalized and abstracted Bitcoin network properties, defining ideal function $\mathcal{F}_{\mathrm{BTC}}$ and designing fair MPC protocols in the $(\mathcal{G}_{\mathrm{BTC}}, \mathcal{F}_{\mathrm{CR}})$-hybrid model. Kumaresan and Bentov [24] defined ideal function $\mathcal{F}_{\mathrm{ML}}$ and designed constant-round protocols in the $(\mathcal{G}_{\mathrm{BTC}}, \mathcal{F}_{\mathrm{ML}})$-hybrid model. Kumaresan et al. [25] improved upon [23] by reducing script complexity and on-chain costs. However, these protocols are Bitcoin-based; Bitcoin' s non-Turing-complete language limits complex functionality, requiring deposit amounts reaching $O(N^2)$ and ideal function invocations reaching $O(N)$. Zyskind [26] designed Ethereum-based smart contracts using threshold secret sharing, adding deceivers to a blacklist based on secret reconstruction to achieve secure and fair MPC with $O(N)$ deposits. Kiayias et al. [27] constructed fair and robust MPC protocols under the UC model using blockchain, but the number of currency transfer rounds equals the number of MPC protocol rounds.

2 Fair Secure Multi-Party Computation Protocol

BFSMPC builds a penalty mechanism based on Ethereum smart contracts to achieve fairness. At the protocol' s start, all participants must deposit collateral to the smart contract; otherwise, the protocol terminates. BFSMPC consists of two main phases: Phase 1 executes an unfair generic MPC protocol based on Gennaro' s scheme [28] off-chain, which employs Verifiable Secret Sharing (VSS) where secrets are shared via a t-degree random polynomial, resulting in each of the n participants receiving secret shares. Phase 2 executes a fair secret reconstruction protocol requiring multiple rounds of interaction with the smart contract. Participants first off-chain broadcast their secret shares to others, who verify and feed verification results back to the smart contract. The smart contract identifies malicious parties, whose deposits are eventually distributed

among honest parties. In this phase, honest parties can recover the secret upon collecting $t+1$ correct shares. If reconstruction fails, they receive compensation.

2.1 Off-Chain Generic MPC Protocol

Generally, three main approaches exist for constructing generic secure MPC protocols: Yao' s garbled circuits, secret sharing, and homomorphic encryption [29]. Regarding privacy protection, generic secure MPC protocols based on secret sharing often achieve better performance and can be easily extended to cloud-assisted secure computation, making MPC protocols more practical. Many generic MPC protocols operate in the semi-honest model. Although Goldreich et al. [2, 9] proposed compilers to transform semi-honest secure MPC protocols into malicious-secure ones, malicious parties can still abort prematurely after receiving outputs, preventing fairness guarantees.

To ensure fairness, BFSMPC constructs a fair secret reconstruction protocol at the output stage of the underlying generic MPC protocol. Kumaresan et al. [25] used an n-n secret sharing scheme with hash-based commitments that only prevent share tampering but cannot guarantee share correctness, leaving participants unable to verify reconstruction results. To address this, BFSMPC's underlying generic secure MPC protocol combines Gennaro' s VSS scheme with Pedersen' s homomorphic commitment scheme [30] to enable share correctness verification. This generic MPC represents the target function F as a directed graph of addition and multiplication gates, implementing arbitrary computations by executing corresponding addition and multiplication protocols. The protocol comprises three stages [31], as shown in [Figure 1: see original paper].

Stage 1: Input Stage. Each participant shares their secret input among all participants using VSS. **Stage 2: Computation Stage.** Participants execute addition and multiplication protocols on secret shares, which are verifiable. This stage outputs secret shares needed to reconstruct y, with each participant P_i receiving share y_i. **Stage 3: Output Stage.** Participants publish their shares y_i to jointly reconstruct y.

2.1.1 Input Gate In the input stage, to protect privacy, each participant P_i (acting as Dealer) uses Shamir' s secret sharing to distribute their input to other participants, employing Pedersen' s homomorphic commitment scheme to ensure share verifiability. P_i selects two random polynomials:

$$f(x) = a_0 + a_{1x} + a_{2x}^2 + \cdots + a_{tx}^t$$

$$g(x) = b_0 + b_{1x} + b_{2x}^2 + \cdots + b_{tx}^t$$

where $a_0 = \alpha$ is the secret value. P_i secretly sends share $s_j = f(j)$ to participant P_j $(j = 1, 2, \ldots, n)$ and broadcasts commitment values $A_k = g^{a_k} h^{b_k} \bmod p$ for

$k = 0, 1, \ldots, t$. Here p is a large prime agreed upon by n participants, satisfying $p = 2q+1$ where q is also prime, g is a q-order element in $\mathbb{Z}_p^*$, and h is a random element in the subgroup generated by g.

After receiving share s_j, P_j must verify its validity by checking whether their share and all others' shares lie on the same t-degree polynomial (the VSPS property). The verification proceeds as follows:

P_j computes $A'_\delta = \prod_{k=0}^{t} A_k^{\delta^k} = g^{\sum_{k=0}^{t} a_k \delta^k} h^{\sum_{k=0}^{t} b_k \delta^k} = g^{f(\delta)} h^{g(\delta)}$ for $\delta \in [1, n]$. Let λ_{ji} denote the element in row j, column i of matrix V^{-1}. Then:

$$f(j) = \sum_{i=0}^{t} \lambda_{ji} f(i) = \sum_{i=0}^{t} \lambda_{ji} a_i$$

$$g(j) = \sum_{i=0}^{t} \lambda_{ji} b_i$$

Thus:

$$A'_j = g^{\sum_{i=0}^{t} \lambda_{ji} a_i} h^{\sum_{i=0}^{t} \lambda_{ji} b_i} = \prod_{i=0}^{t} (g^{a_i} h^{b_i})^{\lambda_{ji}} = \prod_{i=0}^{t} A_i^{\lambda_{ji}}$$

P_j compares the computed A'_j with the previously broadcast A_j. If they match, P_j checks whether their share matches the commitment. If both checks pass, the received share is deemed valid; otherwise, P_j requests a correct share from the Dealer. Through this verification, each participant can detect whether the secret distributor is honest.

Security Analysis: In this scheme, the commitment broadcast by P_i contains information related to secret α only through $A_0 = g^{a_0} h^{b_0}$. For any $a_0 \in \mathbb{Z}_p$, there exists a unique b_0 such that A_0 does not leak any information about α. Thus, the process is information-theoretically secure. Moreover, the scheme resists active attacks where the secret distributor sends incorrect shares to other participants. As shown in the derivation above, if distributed shares do not lie on the same t-degree polynomial, they will fail VSPS verification, thereby detecting dishonest Dealers.

2.1.2 Addition Gate Assume participant P_j possesses verified shares α_j and β_j of secrets α and β, along with commitment-related shares ρ_j and σ_j. Public information includes commitments $A_k = g^{\alpha_k} h^{\rho_k}$ and $B_k = g^{\beta_k} h^{\sigma_k}$ for $k = 0, 1, \ldots, n$. The goal is to compute $\gamma = \alpha + \beta$. The method is as follows:

All participants can compute the commitment to γ_j as $C_j = A_j B_j = g^{\alpha_j + \beta_j} h^{\rho_j + \sigma_j}$. Since A_j and B_j have passed VSPS verification, all participants can verify C_j' s correctness by comparing it with $A_j B_j$ and determine P_j' s

honesty. Each participant computes their addition gate share as $\gamma_j = \alpha_j + \beta_j$ and publishes commitment $C_j = g^{\gamma_j} h^{\rho_j + \sigma_j}$.

Security Analysis: Due to the homomorphic commitment scheme, since A_j and B_j have passed VSPS verification, all participants can verify C_j's correctness. The published commitments leak no information about secret shares, making this addition protocol information-theoretically secure, similar to the input gate analysis.

2.1.3 Multiplication Gate Assume participant P_j possesses verified shares α_j and β_j of secrets α and β, along with commitment-related shares ρ_j and σ_j. Public information includes commitments $A_k = g^{\alpha_k} h^{\rho_k}$ and $B_k = g^{\beta_k} h^{\sigma_k}$ for $k = 0, 1, \ldots, n$. The goal is to compute $\gamma = \alpha\beta$. The method is as follows:

P_j selects two random polynomials $h(x)$ and $u(x)$ satisfying $h(0) = \gamma_j = \alpha_j \beta_j$. For $i = 1, 2, \ldots, n$, P_j (as Dealer) sends $h(i)$ to P_i and broadcasts commitments $H_k = g^{h_k} h^{u_k}$ for $k = 0, 1, \ldots, n$.

P_j must use zero-knowledge proof to demonstrate that the shared secret is indeed $\alpha_j \beta_j$. Given public information where A_j and B_j are verified commitments, P_j must prove that D_j' s opened value equals the product of A_j and B_j' s opened values. Gennaro et al. [28] provide relevant zero-knowledge proof methods.

After zero-knowledge proof verification, each P_i performs VSPS verification on P_j's broadcast commitments. If verification passes, P_i checks whether their share matches the commitment. If matched, the share is considered valid. Finally, $2t+1$ valid shares are selected to compute $\gamma_i = \sum_{j=1}^{2t+1} \lambda_{ij} h(j)$ as the multiplication gate secret share, and commitment $C_i = g^{\gamma_i} h^{\theta_i}$ is published.

Security Analysis: This multiplication scheme resists active attacks. Zero-knowledge proof ensures each P_j correctly generates $\gamma_j = \alpha_j \beta_j$. VSPS verification then checks P_j' s distribution correctness. If VSPS verification fails, P_j sent incorrect shares, and other participants request correct shares. For each participant P_i's multiplication gate output commitment C_i, others verify correctness by checking whether C_i equals $\prod_{j=1}^{2t+1} H_j^{\lambda_{ij}}$. All participants can compute the commitment to γ_i as $C_i = \prod_{j=1}^{2t+1} H_j^{\lambda_{ij}}$. Similar to the input gate analysis, this multiplication protocol is information-theoretically secure, and published content leaks no information about secret shares.

In multiplication gates, each participant must perform $n-1$ VSPS verifications. For efficiency, intermediate VSPS verification can be omitted, and instead, after all participants obtain their multiplication gate shares, a single VSPS verification can be performed on aggregated share commitments. However, this approach verifies aggregated values, cannot identify specific malicious parties, and only handles passive adversaries. If VSPS verification passes, computation continues; if not, all participants recalculate the multiplication gate with VSPS

verification performed when each participant acts as Dealer. This method cannot determine the number of malicious parties.

2.1.4 Output Gate After participants'secret inputs pass through input gates, addition and multiplication gates operate on these shares without leaking any information about secret inputs. Each participant distributing shares in addition and multiplication gates undergoes VSPS verification, detecting dishonest distributors. The output gate result comprises secret shares needed to reconstruct y, with each participant P_i receiving share y_i. Public information includes commitment A_0 to secret y and commitments A_i to shares for $i = 1, 2, \ldots, n$.

Analysis of input, addition, and multiplication gates shows that all commitments reaching the output gate are verified as correct. Therefore, in the output stage —i.e., the blockchain-enabled fair reconstruction phase—participants only need to verify whether published shares y_i match previously broadcast commitments A_i. Matching shares are considered correct, and y can be correctly recovered upon collecting $t + 1$ correct shares.

2.2 Fair Secret Reconstruction Protocol

After executing the off-chain generic MPC protocol, all participants obtain their shares and must publish them to reconstruct the secret. However, malicious participants may abort prematurely after receiving others' shares and successfully recovering the secret, preventing some participants from obtaining results and violating MPC fairness. Therefore, a fair secret reconstruction protocol is needed to detect malicious participants, penalize them, and compensate honest parties. The existence of penalty mechanisms incentivizes rational participants to avoid malicious behavior.

Implementing fair MPC via penalty mechanisms first requires solving the deposit storage problem. Kumaresan et al. [25] proposed a fair MPC protocol based on Bitcoin script language. Due to traditional Bitcoin blockchain limitations, no trusted party can hold deposits; deposits must be transferred pairwise between participants via claim-or-refund mechanisms, reaching total deposits of $O(N^2)$ and identifying only one malicious party. Kiayias et al. [27] also built on Bitcoin, where the number of transaction rounds equals the number of rounds in the malicious-secure MPC protocol. BFSMPC implements its solution using Ethereum smart contracts, which support two account types [32]: externally owned accounts controlled by user private keys and contract accounts controlled by contract code, serving as trustless trusted parties for deposit storage. In BFSMPC, all participants send deposits to a contract account for unified custody, requiring only one deposit round with total deposits of $O(N)$.

BFSMPC comprises local protocol ConLocal and smart contract ConContract. ConContract is written by a participant P_i and published to the blockchain network after all participants agree on its content; miners verify and include it in the blockchain. The following describes ConLocal and ConContract in de-

tail. ConLocal defines participants' computational operations and data sent to ConContract, while ConContract is executed by blockchain nodes, responsible for deposit custody and adjudicating participant behavior during secret reconstruction. Since smart contract execution results require consensus in blockchain networks, if participants fail to send messages to ConContract within specified time limits, they are deemed malicious.

Smart Contract ConContract for Fair Secure MPC:

- **Init:** Malicious party set $F := \emptyset$, malicious party count f initialized to 0, participant set $P := \{P_1, P_2, ..., P_n\}$, deposit flag $D := 0$, large prime p satisfying $p = 2q + 1$ (where q is prime) agreed upon by n participants, g as a q-order element in $\mathbb{Z}_p^*$, h as a random element in the subgroup generated by g, deposit amount dep, deposit array M recording deposit status (initialized to all 0s), status array J (initialized to all 0s), arrays Ch and Sh for storing commitments and shares (initialized as empty).
- **Deposit:** Upon receiving $(\text{Deposit}, dep)$ from participant P_i: Verify current time $T <$ over(0) and $\text{Ledger}[P_i] \geq dep$; confirm P_i is depositing for the first time in this computation; $\text{Ledger}[P_i] := \text{Ledger}[P_i] - dep$. If satisfied, $\text{Ledger}[\text{ConContract}] := \text{Ledger}[\text{ConContract}] + dep$; $M[i] := 1$. If $M[1]\&M[2]\&...\&M[n] = 1$, set $D := 1$.
- **ReadD:** Once $T >$ over(0): If $D \neq 1$, return deposits.
- **Compute:** Once $T \geq$ over(0) + 2: Upon receiving (ReadD, P_i) from participant P_i, verify $T <$ over(1) (ignore otherwise). If $D \neq 1$, refund deposits and terminate protocol; otherwise continue.
- **SubCom:** Upon receiving (SubCom, A_i) from P_i: Verify $T <$ over(2) (ignore otherwise). Record that P_i triggered SubCom and store $\text{Ch}[i] := A_i$.
- **SubVerify:** Once $T \geq$ over(2) + 2: Upon receiving (Verify, J_i) from P_i, verify $T <$ over(4) (ignore otherwise). Record that P_i triggered Verify. Perform bitwise AND on all verification arrays to obtain status array $J := J_1\&J_2\&...\&J_n$. Once $T >$ over(4), add participants who did not trigger Verify to F.
- **Pay:** If $J[0] = 1$, for any $P_i \notin F$: $\text{Ledger}[P_i] := \text{Ledger}[P_i] + dep + \frac{f \cdot dep}{n-f}$.
- **ReCon:** Once $T \geq$ over(5) + 2: Return shares.
- **ReturnShare:** Return Sh.
- **RePay:** Upon receiving (RePay, P_i): If $J[0] \neq 1$ and $J[i] \neq 1$, ConContract retrieves A_i from $\text{Ch}[i]$ and verifies whether the submitted share equals A_i. If equal, store the share in $\text{Sh}[i]$ and set $J[i] := 1$. When $T >$ over(5), any P_j with $J[j] \neq 1$ is added to malicious set F. This step detects two dishonest behaviors: (1) P_j submitted invalid secret shares;

(2) P_j' s share was not approved by others, and P_j failed to submit it to ConContract for verification.

Participant Local Protocol ConLocal:

- **Init:** Protocol participant set $P := \{P_1, P_2, \dots, P_n\}$, n participants agree on a large prime p where $p = 2q+1$ with q prime, g is a q-order element in $\mathbb{Z}_p^*$, h is a random element in the subgroup generated by g, deposit amount dep, and initialize verification array J_i to all zeros.
- **Check:** $U := (J, F)$; Send $(\text{Verify}, P_i, J_i)$ to ConContract; Return U.
- **ReVerify:** Upon receiving $(\text{Reverify}, \cdot)$ from P_i: Verify $T < \text{over}(5)$, ignore if not satisfied; After receiving U, if $J[0] = 1$, send (Pay, P_i) to ConContract. If $J[0] \neq 1$ and $J[i] \neq 1$, verify whether the share equals A_i. If equal, set $J[i] := 1$ and send $(\text{Reverify}, P_i, \cdot)$ to ConContract.
- **RePay:** Once $T > \text{over}(5)$: If $J[0] \neq 1$ and $J[i] = 1$ and secret recovery succeeded, once $T > \text{over}(5) + 2$, send (RePay, P_i) to ConContract. Upon ConContract receiving (RePay, P_i), it checks whether $T > \text{over}(5)$, ignores if not satisfied, otherwise if it detects $P_i \notin F$ with f malicious parties, P_i receives a refund of $dep + \frac{f \cdot dep}{n-f}$.

BFSMPC Execution Process:

Before BFSMPC execution, participants enter a pre-preparation phase to agree on deposit amounts and public parameters, generate key pairs, and broadcast public information (e.g., Ethereum addresses and public keys). Participants are assumed to have access to secret and broadcast channels, with the entire protocol running in a synchronous network. The BFSMPC execution flow is shown in [Figure 2: see original paper].

a) **Deposit Submission:** Participants submit deposits, corresponding to step 3 in [Figure 4: see original paper]. Let $\text{over}(r)$ denote the end time of round r. Participant P_i sends $(\text{Deposit}, dep)$ to ConContract before $\text{over}(0)$. If $D \neq 1$ after $\text{over}(0)$, deposits are refunded and the protocol terminates; otherwise, it continues.

b) **Commitment Submission:** Participant P_i sends $(\text{SubCom}, P_i, A_i)$ to ConContract before $\text{over}(2)$ to submit commitment A_i to their share. ConContract checks $T < \text{over}(2)$ and stores A_i in $\text{Ch}[i]$ if satisfied. Once $T > \text{over}(2)$, participants who did not send commitments are added to malicious set F. This corresponds to step 5 in [Figure 4: see original paper].

c) **Off-chain Verification:** Corresponding to steps 6 and 7 in [Figure 4: see original paper]. Before $\text{over}(3)$, participant P_i calls $(\text{SubVerify}, \cdot)$ to send their secret share (y_i, r_i) to all other participants P_j $(j = 1, 2, \dots, n, j \neq i)$. Each P_j uses verification array J_j to record verification results for other participants' shares. For multiplication gates, each participant must act as Dealer to distribute shares; zero-knowledge proof and VSPS

verification detect honesty. In addition gates, all participants can compute P_i' s addition gate output commitment. Throughout off-chain MPC, each participant's behavior is monitored to identify malicious parties and ensure correct output gate commitments.

d) **Verification Submission:** Before over(4), participant P_i sends (Verify, J_i) to ConContract. ConContract records P_i as having triggered Verify. After over(4), participants who did not trigger Verify are added to F.

e) **Secret Recovery:** If $J[0] = 1$, all participants successfully recovered the secret off-chain. For any $P_i \notin F$, ConContract sends (Pay, P_i), and participants receive refunds and terminate after confirmation. This corresponds to step 13 in [Figure 4: see original paper].

f) **Compensation Handling:** If $J[0] = 0$, some participants failed to recover the secret. If $J[i] = 1$, P_i' s share was approved by all participants. If P_i successfully recovered the secret, after $T \geq$ over(5) + 2, P_i sends (RePay, P_i) to ConContract. If $J[i] = 0$, P_i' s share was not approved, requiring submission of (Reverify, $P_i, \cdot$) to ConContract before over(5), corresponding to step 9 in [Figure 4: see original paper].

g) **Malicious Party Detection:** Upon receiving (Reverify, $P_i, \cdot$), ConContract checks $T <$ over(5). If satisfied, it verifies whether the submitted share equals A_i. If equal, it stores the share in Sh[i] and sets $J[i] := 1$. When $T >$ over(5), any P_j with $J[j] \neq 1$ is added to F. This step detects two dishonest behaviors: (1) P_j submitted invalid secret shares; (2) P_j' s share was not approved by others, and P_j failed to submit it to ConContract for verification.

h) **Failed Recovery Handling:** If participant P_i fails to recover the secret, when $T \geq$ over(5) + 2, P_i sends (ReturnShare, P_i) to ConContract (step 10 in [Figure 4: see original paper]). ConContract returns array Sh (step 11 in [Figure 4: see original paper]), which stores shares that failed off-chain verification but passed blockchain verification. P_i may have failed earlier due to insufficient correct shares; obtaining Sh may or may not yield $t + 1$ shares. However, malicious parties sending incorrect shares or aborting prematurely have been identified in previous steps, so P_i receives compensation by sending (RePay, P_i) to ConContract. If ConContract detects $P_i \notin F$ with f malicious parties, P_i receives a refund of $dep + \frac{f \cdot dep}{n-f}$.

3 Security Analysis

As derived in Section 2.1, BFSMPC uses a generic MPC protocol based on verifiable secret sharing, which can resist active attackers. In the protocol's input stage, when each participant acts as Dealer to distribute secret shares to other participants, they publish commitments to shares and secrets. This commitment scheme is information-theoretically secure and does not leak any information

about secrets. Additionally, the commitment scheme is homomorphic: for a polynomial $f(x)$, when given commitments to $f(i)$, based on the homomorphic commitment property, one can compute commitments to polynomial coefficients, enabling VSPS verification to detect whether all shares lie on the same t-degree polynomial. The specific derivation process is in Section 2.1.1. If verification passes, the Dealer can be determined to be honest, and each participant receives correct shares.

In the input stage, each participant P_i (as Dealer) uses Shamir's secret sharing to distribute inputs to other participants, employing Pedersen's homomorphic commitment scheme to ensure share verifiability. P_i selects two random polynomials $f(x)$ and $g(x)$, where $a_0 = \alpha$ is the secret value. P_i secretly sends share $s_j = f(j)$ to participant P_j and broadcasts commitment values $A_k = g^{a_k} h^{b_k} \bmod p$ for $k = 0, 1, \ldots, t$. Here p is a large prime agreed upon by n participants, satisfying $p = 2q+1$ where q is also prime, g is a q-order element in $\mathbb{Z}_p^*$, and h is a random element in the subgroup generated by g.

After receiving share s_j, P_j must verify its validity by checking whether their share and all others' shares lie on the same t-degree polynomial (the VSPS property). P_j computes $A'_\delta = \prod_{k=0}^{t} A_k^{\delta^k} = g^{f(\delta)} h^{g(\delta)}$ for $\delta \in [1, n]$. Let λ_{ji} denote the element in row j, column i of matrix V^{-1}. Then $f(j) = \sum_{i=0}^{t} \lambda_{ji} a_i$ and $g(j) = \sum_{i=0}^{t} \lambda_{ji} b_i$. Thus $A'_j = \prod_{i=0}^{t} A_i^{\lambda_{ji}}$. P_j compares the computed A'_j with the previously broadcast A_j. If they match, P_j checks whether their share matches the commitment. If both checks pass, the received share is deemed valid; otherwise, P_j requests a correct share from the Dealer. Through this verification, each participant can detect whether the secret distributor is honest.

In this scheme, the commitment broadcast by P_i contains information related to secret α only through $A_0 = g^{a_0} h^{b_0}$. For any $a_0 \in \mathbb{Z}_p$, there exists a unique b_0 such that A_0 does not leak any information about α. Thus, the process is information-theoretically secure. Moreover, the scheme resists active attacks where the secret distributor sends incorrect shares to other participants. As shown in the derivation, if distributed shares do not lie on the same t-degree polynomial, they will fail VSPS verification, thereby detecting dishonest Dealers.

4 Performance Analysis

Blockchain-based solutions must consider consensus duration. In Bitcoin, block generation takes approximately 10 minutes, with final confirmation requiring 6 blocks (about 60 minutes), which is clearly inefficient. Kumaresan et al. [25] accounted for block confirmation, making each round extremely long. Ethereum-based BFSMPC adopts the Ghost protocol [33], producing blocks every 15 seconds, offering higher efficiency than [25].

BFSMPC's first-phase generic MPC protocol outputs VSPS-verified public commitments and participants' secret shares. Participants verify share correctness off-chain. Since threshold secret sharing is used, participants need only col-

lect $t+1$ correct shares to recover secrets, without requiring all n shares to be correct. The blockchain only needs to detect when all honest participants have recovered secrets to terminate the protocol, without checking for off-chain incorrect shares. Only when participants fail to recover secrets must unverified shares be broadcast to the blockchain network for validation, identifying malicious parties to compensate honest participants. Because Ethereum-based BFSMPC includes contract accounts for storing data and value, participants need only deposit collateral once before protocol execution, significantly reducing algorithmic complexity.

For efficiency, BFSMPC' s fair reconstruction phase first performs off-chain verification. A participant P_i' s share may be disapproved for three reasons: (1) P_i sent incorrect shares; (2) P_i sent correct shares but was maliciously framed; (3) P_i aborted prematurely without publishing shares. If all participants successfully recover secrets off-chain, these malicious behaviors go undetected because fairness is already achieved. If off-chain verification fails for some participants, all unverified shares must be submitted to ConContract. As ConContract executes automatically and is controlled by no one, rational participants will send correct shares since incorrect shares are easily detected, resulting in forfeited deposits. Premature aborts are detected via timeouts: each protocol round has a time limit, and exceeding it without sending messages results in being deemed malicious. ConContract classifies P_i as malicious if: (a) deposit not submitted on time (cannot penalize, protocol terminates); (b) commitment A_i not uploaded on time; (c) verification array J_i not uploaded on time; (d) share not approved by others and not submitted to ConContract on time; (e) submitted share fails blockchain verification.

5 Conclusion

In secure MPC, fairness cannot be guaranteed when most participants are dishonest. Based on the observation that real-world attackers are rational, penalty measures can compel malicious attackers to execute protocols fairly. Blockchain' s decentralized, trustless, and immutable characteristics enable smart contracts to automatically execute code, store data, and hold value like regular accounts. This paper constructs a penalty mechanism using blockchain smart contracts and designs the fair MPC protocol BFSMPC. Participants deposit collateral to the smart contract before protocol execution, then execute an unfair secure MPC protocol based on VSS, where secret outputs are shared via a t-degree polynomial, yielding secret shares for each participant. Subsequently, a fair secret reconstruction protocol is executed where participants broadcast shares for off-chain verification, feeding results to the smart contract to identify malicious parties. The protocol is robust: not all shares need be correct; honest parties can recover secrets upon collecting $t+1$ correct shares and receive compensation if reconstruction fails. Security analysis shows participants obtain correct outputs by verifying share correctness and interaction information with the smart contract. Performance analysis demonstrates BFSMPC requires only one deposit

round with extensive verification off-chain, achieving higher efficiency than prior work [25].

References

[1] Yao C. Protocols for secure computations [C]// Proc of the 23rd IEEE Symposium on Foundations of Computer Science. Piscataway, NJ: IEEE Press, 1982: 160-164.

[2] Goldreich O, Micali S, Wigderson A. How to play any mental game [C]// Proc of the 19th Annual ACM Conference on Theory of Computing. New York: ACM Press, 1987: 218-229.

[3] Garg S, Srinivasan A. Two-round multiparty secure computation from minimal assumptions [C]// Proc of the 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2018: 468-499.

[4] Benhamouda F, Lin H. k-round MPC from k-round OT via garbled interactive circuits [EB/OL]. (2017) [2018-05-03]. https://eprint.iacr.org/2017/1125.pdf.

[5] Cuvelier E, Pereira O. Verifiable multi-party computation with perfectly private audit trail [C]// Proc of the 14th International Conference on Applied Cryptography and Network Security. Berlin: Springer, 2016: 367-385.

[6] Keller M, Pastro V, Rotaru D. Overdrive: Making SPDZ great again [C]// Proc of the 37th Annual International Conference on the Theory and Application of Cryptographic Techniques. Berlin: Springer, 2018: 158-189.

[7] Spini G, Fehr S. Cheater detection in SPDZ multiparty computation [C]// Proc of International Conference on Information Theoretic Security. Berlin: Springer, 2016: 151-176.

[8] Lindell Y, Pinkas B, Smart N P, et al. Efficient constant round multi-party computation combining BMR and SPDZ [C]// Proc of the 35th Annual Cryptology Conference. Berlin: Springer, 2015: 319-338.

[9] Ishai Y, Ostrovsky R, Zikas V. Secure multi-party computation with identifiable abort [C]// Proc of the 34th Annual Cryptology Conference on Advance in Cryptology. Berlin: Springer, 2014: 369-386.

[10] Cleve R. Limits on the security of coin flips when half the processors are faulty [C]// Proc of the 18th Annual ACM symposium on Theory of computing. New York: ACM Press, 1986: 364-369.

[11] Gordon D, Ishai Y, Moran T, et al. On complete primitives for fairness [C]// Proc of the 7th Theory of Cryptography Conference. Berlin: Springer, 2010: 91-108.

[12] Nakamoto S. Bitcoin: a peer-to-peer electronic cash system [EB/OL]. (2008) [2018-05-03]. http://bitcoin.org/bitcoin.pdf.

[13] Singhal B, Dhameja G, Panda P S. Beginning blockchain [M]. Berkeley: Apress, 2018: 219-266.

[14] Kosba A, Miller A, Shi E, et al. Hawk: the blockchain model of cryptography and privacy-preserving smart contracts [C]// Proc of IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE Press, 2016: 839-858.

[15] Jakobsen T P, Nielsen J B, Orlandi C. A framework for outsourcing of secure computation [C]// Proc of the 6th Edition of the ACM Workshop on Cloud Computing Security. New York: ACM Press, 2014: 81-92.

[16] Gordon S D, Hazay C, Katz J, et al. Complete fairness in secure two-party computation [J]. Journal of the ACM, 2011, 58 (6): 1-24.

[17] Goldreich O. Foundations of cryptography: volume 2, basic applications [M]. Cambridge: Cambridge University Press, 2004.

[18] Katz J. On achieving the best of both worlds in secure multiparty computation [C]// Proc of the 39th Annual ACM Symposium on Theory of Computing. New York: ACM Press, 2007: 11-20.

[19] Gordon S D, Katz J. Partial fairness in secure two-party computation [C]// Proc of the 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2010: 157-176.

[20] Tian Youliang, Peng Changgen, Ma Jianfeng, et al. Universally composable secure multiparty computation protocol with fairness [J]. Journal on Communications, 2014 (2): 54-62.

[21] Fitzi M, Garay J A, Maurer U M, et al. Minimal complete primitives for secure multi-party computation [J]. Journal of Cryptology, 2005, 18 (1): 37-83.

[22] Andrychowicz M, Dziembowski S, Malinowski D, et al. Secure multiparty computations on bitcoin [C]// Proc of IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE Press, 2014: 76-84.

[23] Bentov I, Kumaresan R. How to use bitcoin to design fair protocols [C]// Proc of the 34th Annual Cryptology Conference on Advance in Cryptology. Berlin: Springer, 2014: 421-439.

[24] Kumaresan R, Bentov I. How to use bitcoin to incentivize correct computations [C]// Proc of the 21st ACM Conference on Computer and Communications Security. New York: ACM Press, 2014: 30-41.

[25] Kumaresan R, Vaikuntanathan V, Vasudevan P N. Improvements to secure computation with penalties [C]// Proc of the 23rd ACM Conference on Computer and Communications Security. New York: ACM Press, 2016: 77-98.

[26] Zyskind G. Efficient secure computation enabled by blockchain technology [D]. Cambridge: Massachusetts Institute of Technology, 2016.

[27] Kiayias A, Zhou H S, Zikas V. Fair and robust multi-party computation using a global transaction ledger [C]// Proc of the 35th Annual International Con-

ference on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2016: 705-734.

[28] Gennaro R, Rabin M O, Rabin T. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography [C]// Proc of the 17th Annual ACM Symposium on Principles of Distributed Computing. New York: ACM Press, 1998: 101-111.

[29] Jiang Han, Xu Qiuliang. Secure multiparty computation in cloud computing [J]. Journal of Computer Research and Development, 2016, 53 (10): 2152-2162.

[30] Pedersen T P. Non-interactive and information-theoretic secure verifiable secret sharing [C]// Proc of Annual International Cryptology Conference on Advances in Cryptology. Berlin: Springer, 1991: 129-140.

[31] Qiu Weidong, Huang Zheng. Cryptographic protocols [M]. Beijing: Higher Education Press, 2009: 156.

[32] Jameson H. Accounts, transactions, gas and block gas limits in ethereum [EB/OL]. (2017) [2018-05-21]. https://hudsonjameson.com/2017-06-27-accounts-transactions-gas-ethereum/.

[33] Sompolinsky Y, Zohar A. Secure high-rate transaction processing in bitcoin [J]. Computer Science, 2015, 8975: 507-527.

Note: Figure translations are in progress. See original paper for figures.