

Article

Smart Reputation-Based Counter-Collusion Contracts for Cloud Verification

Xiaoli Wang ¹, Yajuan Ren ^{2,*} and Lipeng Song ¹

¹ School of Airspace Science and Engineering, Shandong University, Weihai 264209, China; wxl@sdu.edu.cn (X.W.); slp880@sdu.edu.cn (L.S.)

² The School of Computer Science and Technology, North University of China, Taiyuan 030051, China

* Correspondence: renjuan345@gmail.com

Abstract

To resist collusion between two cloud providers (CPs) in cross-verifiable outsourced computation, several smart contract-based schemes have been proposed. Their key idea is to incentivize a cloud to secretly betray collusion. In multi-round task outsourcing, collusion profits increase, and the trust built between CPs fosters collusion, making existing deposit-based anti-collusion mechanisms ineffective. To solve these problems, we propose a new solution. It differs from existing works in the following ways: (1) By introducing a reputation certificate system, it fundamentally changes the existing solutions by increasing deposits in the cat-and-mouse game. The new Outsourcing contract automatically updates a CP's reputation certificate based on its behavior, which influences further client's willingness to delegate tasks to the CP. As the increasing profits obtained by multiple collusions are much less than the corresponding delegation profit, a rational CP will not take the risk to lose future computation tasks. (2) We introduce a public Report contract. This contract is uniquely designed to report collusion attempts before a colluder's contract is signed. Since no binding colluder's contract has been signed, the reporter will not face any penalties. Consequently, our approach resolves the secrecy dilemma present in existing schemes. A further feasibility study is done by executing the contracts on an Ethereum network. Then, we conducted comparisons with state-of-the-art schemes. Both theoretical and experimental comparisons show that our scheme is the most effective in combating collusion.

Keywords: cloud computing; game theory; collusion; smart contract; verifiability

1. Introduction

Cloud computing delivers scalable computing resources via the Internet, enabling enterprises and individuals to flexibly adjust computational demands [1]. While clients frequently outsource computing tasks to CPs, these CPs may return fabricated results to reduce costs, thereby necessitating result verification [2].

Among mainstream verification approaches [3], cryptographic methods are considered impractical due to their excessive costs [4,5] (10^3 – 10^9 times higher than outsourcing). Replication-based verification requires distributing tasks to multiple providers for cross-validation [3,6]; however, employing three or more providers exceeds local computation costs [7]. Therefore, dual-provider verification becomes the only feasible solution, but it carries the critical risk of collusion—where two CPs may conspire to return identical erroneous results without being detected.



Academic Editor: Javid Taheri

Received: 6 July 2026

Revised: 31 July 2026

Accepted: 4 August 2026

Published: 7 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons](https://creativecommons.org/licenses/by/4.0/)

[Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1.1. Motivation

Current studies address the collusion problem by signing a Traitor's contract between a client and a CP to incentivize the CP to report collusion after the collusion has finished [3,6,8]. They assume that there is only one round of interaction between the two clouds, and some studies mention preventing collusion issues in multi-round interactions by increasing deposits. Figure 1 illustrates the interaction process, where two CPs independently execute outsourced tasks and submit results through the smart contract, while the direct interaction between CPs may lead to trust accumulation and potential collusion. However, deposits must be within a reasonable range to guarantee a CP's willingness to participate. Multi-round will multiply a CP's collusion profits. It will also increase the trust [9,10] between two interactive CPs, which further facilitate their collusions. When the trust exceeds a specific threshold, the existing schemes even if the maximum deposit is adopted can not prevent CP collusions. Moreover, existing research assumes that a secret act of betrayal goes undiscovered and unpunished. However, detecting such behavior is highly probable, as every act of betrayal will trigger the client's verification. Ref. [11] also highlights the inadequacies of this assumption. Last but not least, the signing of a Traitor's contract in current research will inevitably trigger Trusted Third Party (TTP) verification, leading to a verification fee.

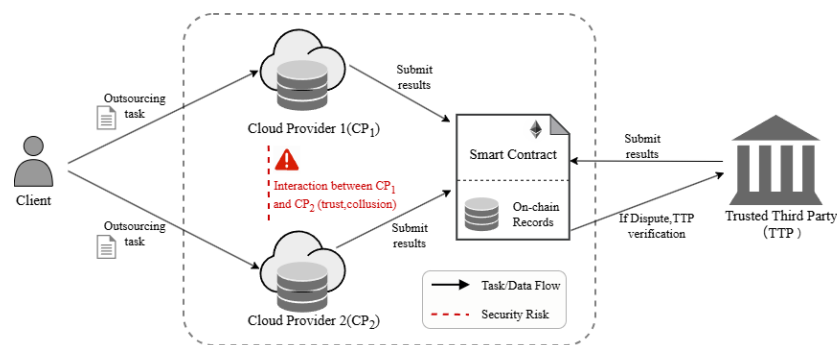


Figure 1. Interaction model.

1.2. Work and Contributions

In this paper, we propose two types of smart contracts: Outsourcing contracts and Report contracts, which differs from existing works in the following ways: (1) We propose a new Outsourcing Contract that automatically updates the reputation certificates of CPs based on their behavior. In the Report Contract, a reporter (REP) is rewarded with an increase in reputation, while the reported party is penalized with a decrease in reputation. (2) We also propose a new Report Contract, in which a cloud is required to report collusion once the other cloud initiates it. Moreover, one can report collusion publicly without worrying about the other party's punishment, as the Colluder's Contract has not yet been signed. (3) We analyze the contract under a potential trust mechanism when multi-round collusion is considered.

The contributions of our work are as follows: (i) By introducing the reputation mechanism we fundamentally change the existing counter-collusion solutions, i.e., increasing deposits and solve their ineffective problem in multi-round situations. (ii) Our contract based on advance and public reporting solves the intrinsic defect in current schemes. That is, a secrecy betrayal is never found out in multi-round situations. (iii) Our advance reporting scheme saves the cost of verifying the computation results given by the reporter.

The remainder of the paper is organized as follows. In Section 2, we provide a brief literature review of the related work. In Section 3, we present the background of smart contracts and the repeated game scenario, along with the formal definitions of all notations

employed in our scheme. In Section 4, we propose our scheme, design the Outsourcing Contract and the Report Contract, and offer a detailed description. Conduct-provable game-theoretic analyses of both our method and the Dong method [3] are given in Section 5. In Section 6, we implement the smart contracts in Solidity and execute them on Ethereum. We validate our analysis through numerical experiments and compare our model with others. Finally, Section 7 concludes the article.

2. Related Work

Verifiable computation based on cryptography has been widely studied to ensure the correctness of outsourced computations. Many verifiable computation-based-on-cryptography schemes [12–26] have been proposed. In these works, homomorphic encryption, secure multi-party computation, and interactive proof protocols are the main technologies. Several studies [12,15,16,18,20,21,26] have focused on verifiable computation through homomorphic encryption. The main idea is to improve the verification algorithm using homomorphic encryption techniques to ensure the correctness of the results. In [13,22,24,25], the problem is resolved through secure multi-party computation. The multi-party secure computing framework allows multiple parties to safely compute the results of any function or a class of functions. Several studies [14,17,19,23] also leverage interactive proof protocols to address this challenge. These solutions typically require complex cryptographic computations to generate or verify proofs, rendering them computationally expensive and impractical for real-world applications.

Verifiable computation based on replication verifies computation results by comparing multiple replicated executions. The client outsources the task to multiple CPs, cross-verifying the correctness of the results returned. Many approaches [3,8,11,27–32] based on replication have been proposed. In schemes proposed by [27–31], the client outsources the task to multiple (more than three) CPs, also known as crowd computing. Li et al. [27] outsource tasks to multiple CPs, ensuring fairness and verifiability in cloud computation through a penalty mechanism. Ma et al. [28] incentivize CPs to submit their computation results honestly through reputation. Dong et al. [29] use repeated game mechanisms to punish cheaters in cloud computing, thereby obtaining the correct solutions for tasks. The limitation of crowd computing is that it requires delegating the same task to more than three different CPs, which significantly increases the original delegation overhead. Cost analysis is reasonable when the client uses double outsourcing [30]. When employing three or more CPs, the economic cost is likely to exceed the cost of executing tasks locally on the client side. For example, in schemes by [3,8,11], the client uses double outsourcing. Recent studies [9,10] have demonstrated that trust evaluation is effective in improving participant reliability and data quality in distributed systems. For example, multidimensional trust evaluation has been adopted [9] for worker recruitment in mobile crowdsensing by jointly considering trustworthiness and task matching, while trust-based incentive mechanisms [10] have been proposed to suppress misinformation propagation by encouraging reliable participants. However, when there are only two CPs, collusion becomes the biggest issue.

Verifiable computation based on smart contracts leverages blockchain technology to enhance the trustworthiness of computation verification. Many studies [3,8,27,28,33] now leverage smart contracts to solve verifiable computing problems. Some studies [27,28] address fairness issues in cloud computing through smart contracts. The collusion between CPs has caused significant losses to the client. In order to solve the problem of collusion, several studies [3,8,33] have proposed methods utilizing smart contracts. In ref. [3], the collusion problem is analyzed and solved by constructing three smart contracts from the perspective of game theory. Dong et al. use smart contracts to stimulate tension, betrayal, and distrust between the clouds, so that clouds will not collude and cheat. In ref. [8],

Chen et al. resolved collusion through the same method as in [3] and achieved greater cost savings by outsourcing the task to multiple CPs in the form of multiple computation blocks. These studies [3,8,33] all assume that there is only one round of interaction between the two CPs, and that the betrayal goes undetected and unpunished. Smart contracts are suitable tools for addressing collusion issues in cloud computing. Therefore, by building smart contracts, we solve the confidentiality problem of betrayal in existing contracts based on contracts reported in advance.

3. Preliminaries

This section provides a brief introduction to smart contracts, presents the repeated game scenario, derives the corresponding utility functions, and analyzes the trust mechanism that arises among CPs. Finally, we summarize the notations used in our work.

3.1. Smart Contracts

Smart contracts are computer programs that automatically enforce the terms of a contract through code. These contracts are based on blockchain technology, which includes cryptocurrency platforms, most notably Ethereum. Cryptocurrencies are digital or virtual currencies that use cryptographic techniques to ensure security, control currency issuance, and verify asset transfers. Unlike traditional fiat currencies, cryptocurrencies rely on distributed ledger technology, most notably blockchain technology, to enable decentralized, transparent, and secure transactions. Smart contracts are typically based on blockchain technology, which means that the execution and state changes of the contract are recorded on an immutable distributed ledger.

Ethereum [34] is one of the first blockchain platforms to introduce the concept of smart contracts, typically written in the Solidity programming language. Each smart contract is a separate account with a dedicated address. Executing smart contracts on Ethereum requires Ether as the fuel for execution. Each computation step and storage operation requires a certain amount of ether. Smart contracts define a set of conditions and operations. When a condition is met, the contract will automatically perform the corresponding action, which may include transferring funds, updating the status, or triggering other smart contracts. Externally owned account owners can interact with smart contracts using the Ethereum Smart Contract Wallet. These wallets allow externally owned account owners to send ether and invoke contract functions.

3.2. Repeated Game Scenario

3.2.1. Utility Equation for Repeated Games

Cloud computing typically involves multiple rounds of interaction. In repeated game scenarios, CPs must consider not only the immediate benefits but also the long-term gains, as current strategies influence future payoffs. Therefore, CPs should take into account both the current and future utilities when making decisions.

The net utility of the CP_i in this round is U_i , α is the discount rate, W_i is the utility value after discounted average profit of the next round, and E_i is the probability of the CP_i being selected by the client in the next round. $\bar{W}_i$ is the average utility of the next round of participation (obtained from previous rounds of experience). On this basis, the comprehensive $\bar{U}_i$ utility is:

$$\bar{U}_i = U_i + W_i \quad (1)$$

$$W_i = \frac{1}{1+\alpha} * E_i * \bar{W}_i \quad (2)$$

3.2.2. Trust Mechanism

When CPs engage in repeated interactions and sign a Colluder's contract, trust will emerge between them, reflecting more realistic situations [35]. Successful collusion among CPs will deepen the trust between them, while acts of betrayal or reporting between CPs will undermine this trust. In the context of a Colluder's contract, credible and enforceable commitments influence the choice of strategies among CPs. The next round is whether collusion is invoked using group indirect reciprocity [36], which is the repeated encounter among a group of individuals: my actions depend on what you do to me and others.

The trust mechanism between CPs based on the principle of indirect reciprocity is as follows. Each CP_i has a degree of trust b_{ij} for another CP_j , i.e., CP_i considers that CP_j accepts the probability of its collusion attempt and complies with the collusion ($1 \geq b_{ij} \geq 0$). The CP initiates collusion when the comprehensive utility of the CP initiating collusion is greater than the comprehensive utility of the CP not initiating collusion. The trust update function looks like the following:

1. When no collusion phase is initiated between CP_i and CP_j , the trust update function is:

$$b_{ij} = b_{ij} + \frac{U_{ij} - U_{min}}{U_{max} - U_{min}} \quad (3)$$

Here, U_{max} and U_{min} represent, respectively, the maximum and minimum utility that CP_i can get from the interaction.

2. When there is a Colluder's contract phase between CP_i and CP_j , the trust update function is:

$$b_{ij} = \frac{U_{ij} - U_{distrust}}{U_{trust} - U_{distrust}} \quad (4)$$

U_{trust} and $U_{distrust}$ represent the trust and distrust of CP_i in its interactions with CP_j , respectively. This update is performed in two directions, that is, b_{ji} and b_{ji} are both updated, because CP_i and CP_j both obtain a trust update from their encounter.

3. We also define a quantity ϕ_i , which is called other CPs' satisfaction with CP_i . When CP_i and other CPs successfully collude, the value of ϕ_i is 1, and the value of all other CPs $k \neq i$ trust b_{ki} is 1. When the CP_i reports the collusion attempts by other parties to the client, the value of ϕ_i is 0, and the value of all other CPs $k \neq i$ trust b_{ki} is 0. Otherwise, we set it to empty and do not change the trust degree b .

3.3. Notations

Before presenting the newly proposed contracts, we define the notations employed throughout this work and incorporate additional terms to represent execution constraints.

- c : The cost incurred by CP for executing computing tasks.
- e : The deposit that CP submits to the client to obtain the computing task.
- r : Payment to CP after the client accepts the result.
- y, y_1, y_2 : Represents the calculation result of TTP test, the calculation result submitted by two CPs (C_1 and C_2) respectively.
- z : The cost of invoking the the TTP test to calculate the result of the task and resolve the problem.
- o : The LDR (The CP who initiates the collusion; the ringleader) incurs the costs to win over a FLR (The other CP; the follower) receive collusion.
- t : The deposit submitted by the CP signing the Colluder's contract.
- W_{ih} : In this round, no collusion or a collusion agreement, but failure to comply with the collusion contract, results in a loss, the average utility for the next round. The next round utility is not affected at this point.

- W_{ic} : After the success of this round of collusion, the next round is average utility. The next round will definitely be invited to collusion.
- W_{ir} : This refers to the next round average discounted utility of betraying the Colluder's contract when the client outsource tasks with equal probability of choosing.
- W_r : This denotes the average discounted utility obtained by a CP in the next round after signing the Report contract in the current round.

The above variables should have the following relationships:

- $r \geq 4c$: Amazon [37] and other cloud outsourcing businesses report return on investment ratios.
- $e > z + c$: The deposit should be greater than the sum of the cost of invoking the TTP and the cost of its computation task.
- $z > 2r$: Otherwise, the client does not need to outsource the task to CPs and only needs to invoke the TTP.
- $o \leq \frac{1}{2}c$: The cost of initiating the collusion is less than half the cost of its honest calculation task.
- $t > r - c + 2e - z$: This is necessary to ensure that the deposit for signing the Colluder's contract is large enough to keep the collusion commitment credible and mandatory.
- $W_{ic} > W_{ih} > W_{ir}$: When successful collusion occurs in this round, the average utility in the next round is guaranteed to be greater than in any other scenario.

4. Our Construction

To address the collusion problem between two CPs in repeated games, this section presents our proposed solution, providing a detailed description of the designed Outsourcing contract and Report contract.

4.1. The Proposed Outsourcing Contract

To prevent the undesirable outcome of collusion between two CPs (C_1, C_2) in repeated games, we further propose a new outsourcing contract. The Outsourcing contract is signed by the client side (CS) with the C_1, C_2 selected based on their reputation. Before participating in the outsourcing process, each CP is required to complete the registration procedure through the identity management mechanism. Each registered CP is associated with a unique identifier and corresponding public key, which are recorded and used throughout the auditing protocol. Therefore, a penalized CP cannot evade punishment by generating a new identity or replacing its key pair. In our reputation mechanism, the reputation certificate reflects the historical behaviors of CPs and provides an incentive mechanism for CPs to perform honestly. CPs with high-quality reputations have a better chance of being selected for tasks. Generally, the higher the reputation of CPs, the greater their probability of being chosen. However, CS only performs task publication and communication coordination, and does not have the authority to determine CP behaviors or modify reputation certificates. The verification of CP behaviors and the update of reputation certificates are automatically executed by the smart contract according to predefined verification rules.

Due to the existence of reputation certificates, the blockchain-based reputation mechanism ensures that reputation records are transparent, tamper-resistant, and enforceable. Consequently, malicious behaviors of CPs can be detected and reflected in their reputation certificates, while CPs with good reputations are more likely to obtain outsourcing tasks.

4.1.1. Reputation Scheme

The reputation scheme mainly consists of four stages. First, we define the format of the reputation certificate. Next, CS selects suitable CPs based on their reputation to assign

tasks. Then, CS evaluates the current reputation according to the strategies of the CPs. Finally, the reputation certificate is updated.

Reputation Certificate. CS selects suitable CPs for tasks based on the reputation values in the reputation certificate. The format of the reputation certificate is shown in Table 1. The different reputation values of CPs represent varying priorities for selection by CS. In each round, CS selects the two CPs with the highest reputation values to compute the task.

Table 1. Reputation certificate.

Symbol	Description
ID_i	ID of cloud provider (CP_i)
H_i	CS considers the number of times CP_i honestly computes tasks
J_i	The number of tasks received by CP_i
Sig_{CS}	Signature of the client (CS)
Sig_i	Signature of the cloud provider (CP_i)
R_i	The reputation of the cloud provider (CP_i)

Reputation Evaluation and Update. Typically, CPs choose different strategies based on various tasks, resulting in differing reputations. Generally, CPs have two reputation states: honest or malicious. The formula for reputation evaluation is as follows:

$$R_i = \mu_{i,t} \left[\frac{H_i}{J_i} * I(n) \right] \quad (5)$$

$\mu_{i,t}$ is the interaction evaluation of CS towards computing party CP_i in round t , where $\mu_{i,t} \in \{0, 1, 2\}$. This system rewards computing parties that perform well in tasks and penalizes those that fail to meet requirements. When CP_i is the reporter role in the Report contract (REP) and has signed a Report contract, $\mu_{i,t} = 2$, while the reported party (LDR) has $\mu_{j,t} = 0$. If CP_i chooses a deceptive strategy and is discovered by CS, $\mu_{i,t} = 0$; otherwise, $\mu_{i,t} = 1$. Correspondingly, H_i is the number of times CS considers CP_i honest, J_i is the number of times CS has outsourced tasks to CP_i , and $I(n) = e^{-\lambda t}$, $0 \leq I(n) \leq 1$ is the importance of the n th task, where t represents the maximum allowable time, and λ is the urgency decay coefficient. When $\lambda = 0.2$, it achieves the optimal trade-off in time sensitivity. Finally, the Outsourcing contract confirms the updates to the reputations of CPs and uploads them to the blockchain, generating the latest reputation certificates for the next round of tasks.

4.1.2. Outsourcing Contract Process

Specifically, Algorithm 1 shows the flow of the Outsourcing contract (OC).

The contract flow is as follows:

(1) CS and C_1, C_2 choose whether to sign the contract. If CP chooses to sign the contract, it is required to submit a deposit e . If either party fails to sign the contract before the first deadline of T_1 , the contract is terminated and the deposit e is refunded.

(2) CS sends task $f(x)$ to both C_1 and C_2 , requiring them to submit the results $y_i (i \in \{1, 2\})$ before T_2 .

(a) If both C_1 and C_2 fail to submit, the deposit e will be retained by CS.

(b) C_1 and C_2 submit the same result, in a non-final outsourced task, CS will refund the deposit e and pay the payment r after acceptance. In the final outsourced task, CS invokes the TTP with a probability no less than ω_1 .

(c) Otherwise, CS invokes TTP to resolve the dispute.

(3) The process for resolving disputes through TTP is as follows:

- (a) Upon receiving the invocation, TTP calculates $f(x) = y$. TTP considers any absence of submission or any submission inconsistent with y as cheating, while submissions consistent with y are considered honest.
 - (b) If one CP cheats and the other is honest, the deposit e of the honest party will be refunded and CS will pay the payment r . The amount z used to invoke TTP will be deducted from the deposit e of the cheating party, and the remaining deposit will serve as a reward for the honest party. No deposit e is refunded to the cheating party and no payment is paid by CS.
 - (c) If C_1 and C_2 both cheat, their deposits will be obtained by CS and CS does not pay the payment, CS pays the cost z of invoking TTP.
- (4) OC updates reputation certificates for C_1 and C_2 .
- (5) The above process needs to be completed before T_3 . If it is not completed before T_3 , OC needs to refund the deposit and CS needs to pay the payment to the CP that submitted the result before T_2 .

Algorithm 1: Outsourcing Contract

Input: e, r, z, T_1, T_2, T_3

```

if  $C_i(i = 1, 2)$  Sign OC then
   $C_i(i = 1, 2) : e \rightarrow \text{OC};$ 
  if  $T < T_1$  then
    CS:  $f(x) \rightarrow C_i(i = 1, 2);$ 
     $C_i(i = 1, 2)$  select whether to submit  $y_i(i = 1, 2);$ 
    if  $T < T_2$  and  $y_1 = y_2 \neq \text{NULL}$  then
      if  $f(x)$  non-final outsourced task then
        CS:  $r \rightarrow C_i(i = 1, 2);$ 
        OC:  $e \rightarrow C_i(i = 1, 2);$ 
      else
        OC invoke TTP with probability  $\geq \omega_1;$ 
      else
        if  $T > T_2$  and  $y_1 = y_2 = \text{NULL}$  then
          OC:  $2e \rightarrow \text{CS};$ 
        else
          OC invoke TTP;
          TTP compute  $f(x) = y;$  if  $y_i(i \in 1, 2) \neq y$  then
             $C_i \leftarrow \text{Cheat};$ 
          else
             $C_i \leftarrow \text{Honest};$ 
          if  $C_i(i \in 1, 2)$  Cheat and  $C_{\neg i}(\neg i \in 1, 2)$  then
            OC:  $2e - z \rightarrow C_i, z \rightarrow \text{TTP};$ 
            CS:  $r \rightarrow C_i;$ 
          else
            OC:  $2e \rightarrow \text{CS};$ 
            CS:  $z \rightarrow \text{TTP};$ 
          Require  $T < T_3;$ 
        else
          OC:  $e \rightarrow C_i(i = 1, 2);$ 
          OC: updates reputation certificates for  $C_1$  and  $C_2;$ 
          OC terminates;
        else
          OC terminates;
  
```

4.2. The Proposed Report Contract

Existing studies [3] have proposed collusion contracts to establish enforceable collusion among rational parties. To initiate such a contract, the initiator must submit a cryptographically signed request to the smart contract, which creates verifiable on-chain evidence of the initiation. In contrast, anonymous, verbal, vague, or one-off off-chain communications cannot provide the same level of credibility or enforceability between rational parties. To counter collusion, we designed a Report contract. The contract should be signed between CS and the CP that reports the collusion. We refer to this CP as the reporter (REP). The Report contract incentivizes REP to sign by providing additional benefits, such as increased reputation. The Report contract requires REP to submit irrefutable evidence to CS proving that the LDR attempted to initiate collusion and that REP refused to participate, thereby ensuring the authenticity of the report. By signing this contract, CPs are encouraged to alter their strategies, making collusive behavior less attractive. The reporting in our approach is public and does not face the dilemma of confidentiality. According to the principle of indirect reciprocity [36], if REP signed a Report contract this round, there will be no CP to initiate collusion with it in the next round.

Report Contract Process

The contract flow is as follows:

- (1) CS and REP choose whether to sign the contract. If REP chooses to sign the contract, it is required to submit *evidence* to CS. If either party fails to sign the contract before the T_2 deadline or if the *evidence* fails verification, the contract will be terminated.
- (2) REP and LDR, requiring them to submit the results $y_i (i \in \{1, 2\})$ before T_2 .
 - (a) CS assigns a reputation of $\mu_{i,t} = 2$ to REP for that round, while LDR receives a reputation of $\mu_{j,t} = 0$ for the same round.
 - (b) Otherwise, the contract does not carry out additional operations.

The irrefutable evidence submitted by REP regarding LDR's initiation of collusion and REP's refusal to sign a Colluder's contract with LDR is represented as *evidence* = {*req*, *res*}, where *req* denotes the collusion request sent by LDR to REP, and *res* denotes REP's response to that request. To gain REP's trust and prevent message tampering, each message should be digitally signed by its sender. As a result, the client can verify the authenticity and credibility of the evidence submitted by REP.

5. Game and Analysis

To demonstrate the effectiveness of our scheme, this section conducts a provable game-theoretic analysis comparing our method with Dong method.

5.1. Analysis of Our Method

The formal definition of an incomplete information game is provided in [38], with the most widely used representation being the game tree. This representation clearly illustrates the participants' decision choices, possible game outcomes, and the evolution of the game. The game starts from the root node and extends all the way to the leaf nodes. Figure 2 shows the game tree composed of the Outsourcing contract, Colluder's contract proposed in [3], and Report contract. In the game, the players are LDR and REP, i.e., $N = \{LDR, REP\}$. In the game, the action set is $A = \{Collude, \neg Collude, Report, H, C, D\}$. *Collude* means to sign the Colluder's contract, $\neg Collude$ means not to sign the Colluder's contract. *Report* means to sign the Report contract and refuse the Colluder's contract. *H* stands for honest calculation of the task and submission of the result, i.e., $y_{i,(i \in \{1,2\})} = f(x)$; *C* represent the collusive agreement where two CPs coordinate to bypass task computation and return identical invalid outputs $y_1 = y_2 \neq f(x)$; *D* represents not communicating with another

CP, not completing the calculating task, and randomly returning a result, i.e., $y_1 \neq y_2$ and $y_{i,(i \in \{1,2\})} \neq f(x)$; The game has four information sets $I = \{I_{1,1}, I_{1,2}, I_{2,1}, I_{2,2}\}$, among which the information set of LDR is $I_1 = \{I_{1,1}, I_{1,2}\}$, among which the information set of REP is $I_2 = \{I_{2,1}, I_{2,2}\}$. Assuming a total of $n \geq 2 (n \in \mathbb{N})$ CPs, the average discount utility for the next round when signing the Report contract in the current round is $W_r = \frac{n}{2} W_{ir}$. W_{ir} refers to the next round average discounted utility of signing the Report contract when CS outsource tasks with equal probability of choosing. Clearly, we need to satisfy $W_{ic} > W_{ih} > W_{ir}$. Assuming the trust degree of LDR towards REP is b_{ij} , when

$$b_{ij}(r - o + W_{ic}) + (1 - b_{ij})(r - c) > r - c + W_{ih} \quad (6)$$

is satisfied, LDR will initiate collusion with REP. Since Equation (6) clearly holds when $1 \geq b_{ij} \geq \frac{W_{ih}}{c-o+W_{ic}}$. If $n > 5$, given that $r - c + W_r > r - o + W_{ic} > r - c + W_{ih}$, REP will definitely report LDR's collusion attempt.

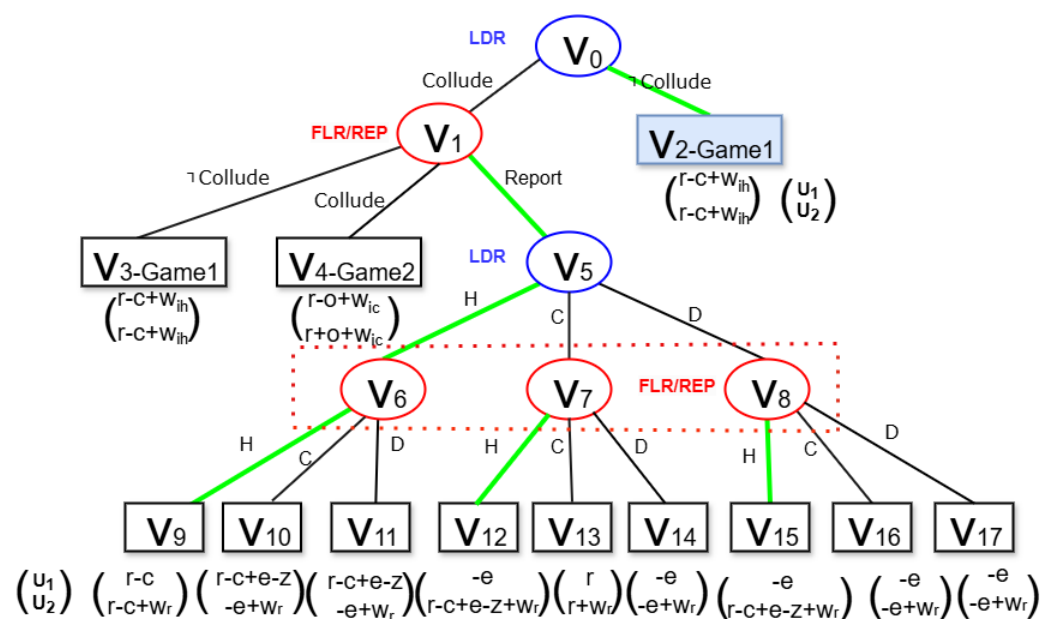


Figure 2. Game tree caused by the Outsourcing contract, Colluder's contract proposed in [3], and Report contract. Game1 is induced only by the Outsourcing contract, Game2 is induced by the Outsourcing contract and original Colluder's contract.

Theorem 1. If $n > 5$, the game in Figure 2 has a unique sequential equilibrium $(S, \beta) = ((S_1, S_2), (\beta_1, \beta_2))$ where:

$$\begin{cases} S_1 = ([0 (\text{Collude}), 1 (\neg \text{Collude})], [1 (H), 0 (C), 0 (D)]) \\ S_2 = ([0 (\text{Collude}), 0 (\neg \text{Collude}), 1 (\text{Report})], \\ \quad [1 (H), 0 (C), 0 (D)]) \\ \beta_1 = ([1 (V_0)], [1 (V_5)]) \\ \beta_2 = ([1 (V_1)], [1 (V_6), 0 (V_7), 0 (V_8)]) \end{cases}$$

The game will start from point V_0 and end at point V_2 . Two CPs both take honest actions, and no one initiates collusion.

In the final round of interaction, the client is unable to entice the CP to sign a Report contract with future benefits. Consequently, the Report contract fails to prevent collusion between CPs. The client proposes adding a detection probability in the last interaction to prevent collusion. Specifically, when two CPs submit identical computation results, there is

a probability ω_1 that triggers TTP to verify the computation results. The game tree for the final interaction is shown in the diagram below.

Here, $M = (1 - \omega_1)(r - o) - \omega_1(e + o)$, $L = (1 - \omega_1)(r + o) - \omega_1(e)$. If LDR does not raise collusion, then $M < r - c$, that is, $\omega_1 > \frac{c-o}{r+e}$. In this case, the game terminates at V_2 .

Theorem 2. If $\omega_1 > \frac{c-o}{r+e}$, the game in Figure 3 has a unique sequential equilibrium $(S, \beta) = ((S_1, S_2), (\beta_1, \beta_2))$ where:

$$\begin{cases} S_1 = ([0(\text{Collude}), 1(\neg\text{Collude})], [0(H), 1(C), 0(D)]) \\ S_2 = ([0(\text{Collude}), 1(\neg\text{Collude})], \\ \quad [0(H), 1(C), 0(D)]) \\ \beta_1 = ([1(V_0)], [1(V_3)]) \\ \beta_2 = ([1(V_1)], [0(V_5), 1(V_6), 0(V_7)]) \end{cases}$$

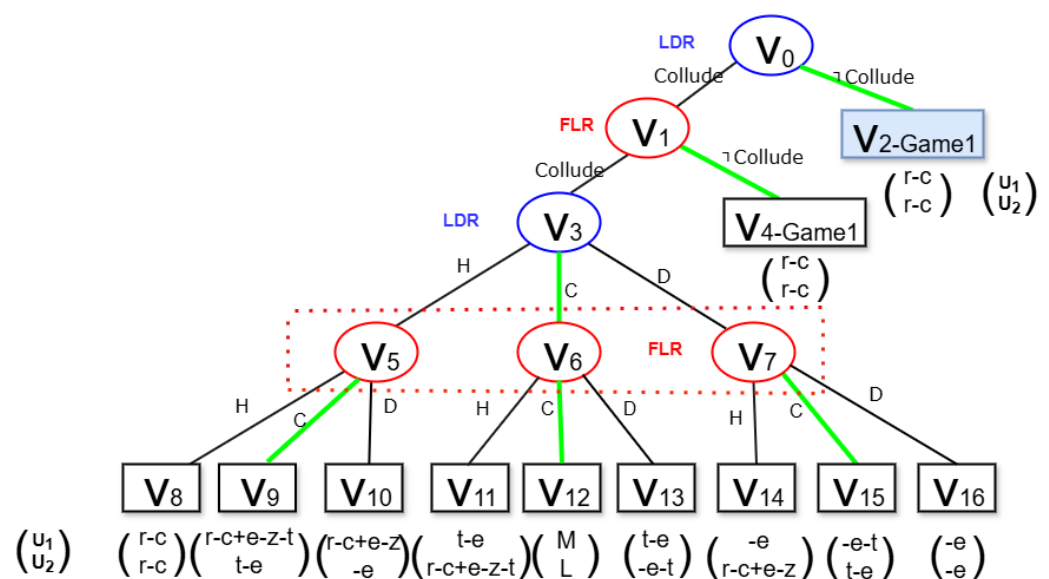


Figure 3. Game tree for the final interaction.

The game will start at point V_0 and end at point V_2 . Along the equilibrium path, both CPs choose honest behaviors and the collusion state is never reached. Therefore, the strategy profile involving mutual collusion and false-result submission is excluded from the sequential equilibrium.

5.2. Analysis of Dong Method

We analyze the efficiency of the Dong method in countering collusion in long-term games, with the impact of CPs' strategies on trust illustrated in Section 3.2. The scheme involves a client and two CPs, and is structured around a framework composed of three types of contracts. Initially, the client signs a Prisoner's contract with both CPs and outsources a computational task to them. After signing the Prisoner's contract and receiving the task, one CP (LDR) may initiate a collusion request to the other. If the second CP (FLR) accepts the request, the two CPs proceed to sign a Colluder's contract. Subsequently, one of the CPs (TRA) may choose to betray the agreement by reporting the collusion to the client and signing a Traitor's contract. Due to the strong assumption that betrayal goes undetected, which is clearly unreasonable in repeated games, betrayal undermines trust between CPs and affects future payoffs. If the condition for successful collusion agreements is to not involve a TTP, Dong's scheme becomes untenable. We will now conduct a detailed analysis of the Dong method.

Figure 4 is a game tree for the Dong method in long-term games. In the game, the players are two CPs, i.e., $N = \{LDR, FLR\}$. In the game, the action set is $A = \{Collude, \neg Collude, Report, \neg Report, H, C, D\}$. *Collude* means to sign the Colluder's contract, $\neg Collude$ means not to sign the Colluder's contract. *Report* means to sign the Traitor's contract. $\neg Report$ means not to sign the Traitor's contract. The game has five information sets $I = \{I_{1,1}, I_{1,2}, I_{2,1}, I_{2,2}, I_{2,3}\}$, among which the information set of LDR is $I_1 = \{I_{1,1}, I_{1,2}\}$, among which the information set of FLR is $I_2 = \{I_{2,1}, I_{2,2}, I_{2,3}\}$. The utility functions of LDR and FLR are represented using U_1 and U_2 , respectively. The utility of each party is listed below terminal nodes. The non-terminal selection nodes $H = \{V_0, V_1, V_3, V_6, V_7, V_8\}$; The terminal nodes $Z = \{V_2, V_4, V_5, V_9, V_{10}, V_{11}, V_{12}, V_{13}, V_{14}\}$; χ, ρ, σ , can be clearly obtained in the Figure 2. Assuming the trust degree of LDR towards FLR is b_{ij} , when

$$b_{ij}(r - o + W_{ic}) + (1 - b_{ij})(-e - o + W_{ih}) > r - c + W_{ih} \quad (7)$$

is satisfied, LDR will initiate collusion with FLR. Let $R = r - o + W_{ic}$, $S = -e - o + W_{ih}$, $P = r - c + W_{ih}$, where $R > P > S$. When $1 \geq b > \frac{P-S}{R-S}$, equation is satisfied, in which case LDR will initiate collusion against FLR. Given that $r + o + W_{ic} > r - c + W_{ih} > r - c + e - z + o + W_{ir}$, FLR will certainly cooperate with LDR in collusion.

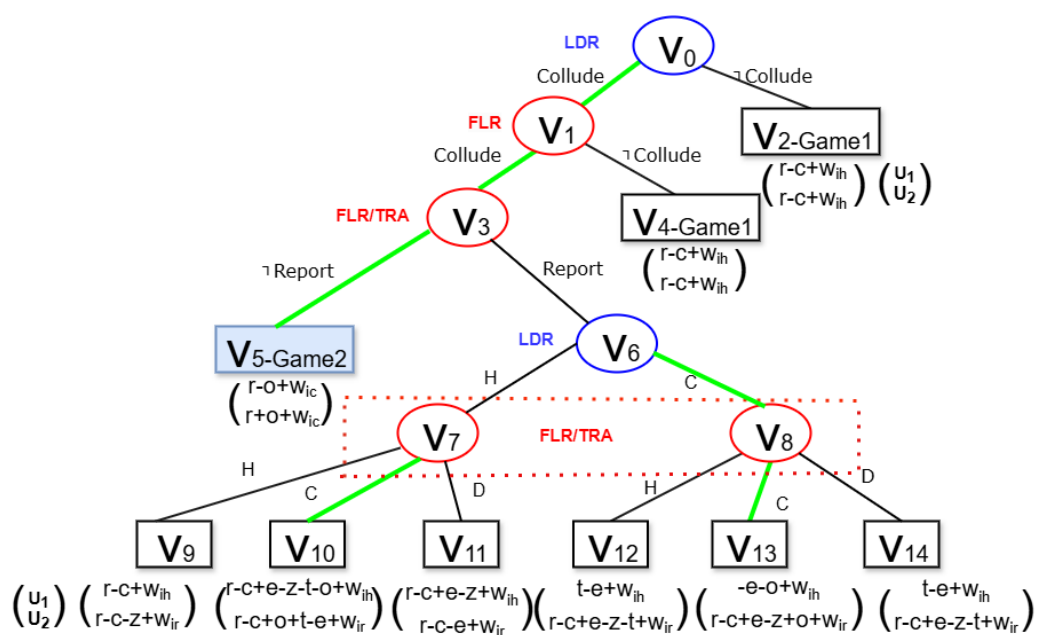


Figure 4. Game tree of Dong method in the repeated game scenarios. Game1 is induced only by the original Prisoner's contract, Game2 is induced by the original Prisoner's contract and original Colluder's contract.

Theorem 3. The game in Figure 4 has a unique sequential equilibrium [39,40] $(S, \beta) = ((S_1, S_2), (\beta_1, \beta_2))$ where:

$$\begin{cases} S_1 = ([1(Collude), 0(\neg Collude)], [0(H), 1(C), 0(D)]) \\ S_2 = ([1(Collude), 0(\neg Collude)], [1(\neg Report), 0(Report)], [0(H), 1(C), 0(D)]) \\ \beta_1 = ([1(V_0)], [1(V_6)]) \\ \beta_2 = ([1(V_1)], [1(V_3)], [0(V_7), 1(V_8)]) \end{cases}$$

The game will start at point V_0 and end at point V_5 . LDR is to always initialize the collusion coalition and send $f(\cdot)$ as the result, and FLR is to always collude with LDR and

send $f(\cdot)$ as the result. Therefore, at the equilibrium of the Dong method, the trust degree between CPs evolves to 1. We further prove that if both CPs sign the original Prisoner's contract and the Colluder's contract, neither has an incentive to break the collusive relationship by signing the original Traitor's contract. This indicates that the scheme proposed in [3] fails to prevent collusion between CPs in a repeated game environment.

5.3. Cost Analysis

We analyze the client cost for our solution. Assuming a total of num rounds of interaction, the client cost $\mathcal{C} = 2 \cdot num \cdot r + \omega_1 z$ and $\omega_1 > \frac{c-o}{r+e}$. Let $\omega_1 = \frac{c}{r+e}$. Given that $e > z$, let $e = \sigma z$, where $\sigma > 1$. Therefore, $\mathcal{C} =$

$$2 \cdot num \cdot r + \frac{cz}{r+e} = 2 \cdot num \cdot r + \frac{c}{r/z + \sigma} < 2 \cdot num \cdot r + \frac{c}{\sigma}$$

Consequently, as the CP deposit e relative to the TTP call cost z increases, the client detection cost decreases.

5.4. Proof Theorem 1

Proof. We set the strategy profile in the equilibrium is $S = (S_1, S_2)$, where:

$$\begin{cases} S_1 = [\lambda_1, (Collude), \lambda_2, (\neg Collude)], \\ \quad [\lambda_3, (H), \lambda_4, (C), \lambda_5, (D)] \\ S_2 = [\theta_1, (Collude), \theta_2, (\neg Collude), \theta_3, (Report)], \\ \quad [\theta_4, (H), \theta_5, (C), \theta_6, (D)] \end{cases}$$

□

λ_i and θ_i are unknown probabilities. They satisfy: $\lambda_i \in [0, 1], \theta_i \in [0, 1], \lambda_1 + \lambda_2 = 1, \lambda_3 + \lambda_4 + \lambda_5 = 1, \theta_1 + \theta_2 + \theta_3 = 1, \theta_4 + \theta_5 + \theta_6 = 1$. The belief system $\beta = (\beta_1, \beta_2)$ can be derived from Bayes' rule:

$$\begin{cases} \beta_1 = ([1(V_0)], [1(V_3)]) \\ \beta_2 = ([1(V_1)], [\lambda_3(V_5), \lambda_4(V_6), \lambda_5(V_7)]) \end{cases}$$

Let us use backward induction. We start with the information set $I_{2,2}$, REP makes decisions on this information set to maximize its utility, which is:

$$U_2(S; I_{2,2}, \beta) = \beta_2(V_6)U_2(S; V_6) + \beta_2(V_7)U_2(S; V_7) + \beta_2(V_8)U_2(S; V_8) = \lambda_3 U_2(S; V_6) + \lambda_4 U_2(S; V_7) + \lambda_5 U_2(S; V_8)$$

where:

$$\begin{cases} U_2(S; V_6) = \theta_4 U_2(V_9) + \theta_5 U_2(V_{10}) + \theta_6 U_2(V_{11}) \\ U_2(S; V_7) = \theta_4 U_2(V_{12}) + \theta_5 U_2(V_{13}) + \theta_6 U_2(V_{14}) \\ U_2(S; V_8) = \theta_4 U_2(V_{15}) + \theta_5 U_2(V_{16}) + \theta_6 U_2(V_{17}) \end{cases}$$

Because the following holds when $r \geq 4c, W_{ic} > W_{ih} > W_{ir}, e > z + c$, if $n > 5: U_2(V_9) > U_2(V_{10}) = U_2(V_{11}); U_2(V_{12}) > U_2(V_{13}) > U_2(V_{14}); U_2(V_{15}) > U_2(V_{16}) = U_2(V_{17})$. It can be concluded that $U_2(S; I_{2,2}, \beta)$ is maximized when $\theta_4 = 1, \theta_5 = 0, \theta_6 = 0$.

Moving backward, LDR makes the decision on information set $I_{1,2}$ to maximize its utility, which is:

$$\begin{aligned} U_1(S; I_{1,2}, \beta) &= \beta_1(V_5)U_1(S; V_5) \\ &= \lambda_3 U_1(S; V_6) + \lambda_4 U_1(S; V_7) + \lambda_5 U_1(S; V_8) \\ &= \lambda_3 U_1(V_9) + \lambda_4 U_1(V_{12}) + \lambda_5 U_1(V_{15}) \end{aligned}$$

Similarly, the following holds $U_1(V_9) > U_1(V_{12}) = U_1(V_{15})$, thus, $\lambda_3 = 1, \lambda_4 = 0, \lambda_5 = 0$ maximizes $U_1(S; I_{1,2}, \beta)$.

A level up. At $I_{2,1}$, REP tries to maximize its utility, which is:

$$\begin{aligned} U_2(S; I_{2,1}, \beta) &= \beta_2(V_1)U_2(S; V_1) \\ &= \theta_1 U_2(S; V_3) + \theta_2 U_2(V_4) + \theta_3 U_2(V_5) \\ &= \theta_1 U_2(V_{12}) + \theta_2 U_2(V_4) + \theta_3 U_2(V_9) \end{aligned}$$

Because the following holds when $r \geq 4c$, $W_{ic} > W_{ih} > W_{ir}$, $e > z + c$, $t > r - c + 2e - z$, $o \leq \frac{1}{2}c$. If $n > 5$, in the game, $U_2(V_9) > U_2(V_5) > U_2(V_3)$. Thus $\theta_1 = 0, \theta_2 = 0, \theta_3 = 1$ will maximize REP 's utility.

The top level, at $I_{1,1}$, LDR tries to maximize its utility, which is:

$$\begin{aligned} U_1(S; I_{1,1}, \beta) &= \beta_1(V_0)U_1(S; V_0) \\ &= \lambda_1 U_1(S; V_1) + \lambda_2 U_1(V_2) \\ &= \lambda_1 U_1(V_9) + \lambda_2 U_1(V_2) \end{aligned}$$

Because $U_1(V_2) > U_1(V_9)$, LDR will always choose $\lambda_1 = 0, \lambda_2 = 1$ to maximize $U_1(S; I_{1,1}, \beta)$. Similarly, E_r is the most profitable and strictly dominant strategy on each information set, so E_r is the only sequentially rational assessment.

Demonstrate that the assessment is consistent, existence strategy sequence $S^k = (S_1^k, S_2^k)$, where

$$\begin{cases} S_1^k = \left(\left[\frac{1}{k}(\text{Collude}), \frac{k-1}{k}(\neg \text{Collude}) \right], \left[\frac{k-2}{k}(H), \frac{1}{k}(C), \frac{1}{k}(D) \right] \right) \\ S_2^k = \left(\left[\frac{1}{k}(\text{Collude}), \frac{1}{k}(\neg \text{Collude}), \frac{k-2}{k}(\text{Report}) \right], \right. \\ \left. \left[\frac{k-2}{k}(H), \frac{1}{k}(C), \frac{1}{k}(D) \right] \right) \end{cases}$$

S^k is for mixed behavior strategy, distinctly, S^k converges to S . The induced belief system β^k also converges to β . The game terminates with a probability of 1 at V_2 . Thus, the unique sequential equilibrium has been proven.

6. Implementation and Results

In this section, we implement the smart contracts using Solidity and deploy them on Ethereum. Through two distinct experimental scenarios comparing our scheme with existing approaches, we demonstrate its superior efficiency across multiple metrics.

6.1. Implementation

In this paper, we implement the three smart contracts mentioned in the paper for cloud computing transactions. Remix is an IDE for Ethereum smart contracts, providing a testing environment that supports the compilation, verification, and deployment of any simulated Ethereum smart contract. Solidity has built-in data types for building all the functions of smart contracts. The contract is loosely coupled to the actual outsourced task as an external service. Actual task execution can be seen as a black box, and the contract does not need to know its internal details. We show the cost of implementing smart contracts.

The results are presented in Table 2. The cost is in the amount of gas consumed by each function, and the converted monetary value in US dollars (USD). The gas price was 1×10^{-9} ether (1 Gwei) in all transactions and the exchange rate was 1 ether = \$1926.67.

Table 2. Cost of using the smart contracts.

Contract	Operation	Cost (Gas)	Cost (USD)
Outsourcing	Init	3,077,617	5.9295
	Create	297,507	0.5732
	Bid	70,936	0.1367
	Deliver	58,423	0.1145
	Pay	95,288	0.1836
	Dispute	293,852	0.5662
Colluder	Init	1,453,200	2.7998
	Create	235,439	0.4536
	Join	21,064	0.0406
	Enforce	21,800	0.0420
Report	Init	1,385,200	2.6688
	Create	211,928	0.4083
	Join	21,320	0.0411
	Promise	21,204	0.0409

As we can see, the financial cost for using the smart contracts on the Ethereum network is low. The cost is roughly related to the computational and storage complexity of the function. For example, in the Outsourcing contract, Init (to store a contract on the blockchain) and Create cost more than other functions. For the Outsourcing contract, the total cost (for the client and the two clouds) is about 3.5 million gas (USD 6.93) if there is no dispute, or about 3.9 million gas (USD 7.50) with dispute resolution. For the Colluder's contract, the total cost is about 1.7 million gas (USD 3.33). And for the Report contract, the total cost is about 1.6 million gas (USD 3.16). The cost of smart contracts is quite low compared to the rewards paid for the cloud. Therefore, when we present a theoretical analysis, we do not consider the cost of smart contracts. If the contract is reused, costs can be further reduced.

6.2. Experiments

To evaluate the performance of our scheme in repeated games (i.e., the accuracy rate of the computation results accepted by CS), we compare our method with three representative anti-collusion schemes, namely the Dong method [3], the Chen method [8], and the Yan method [33]. The Dong method employs a Traitor's contract to incentivize a colluding party to report the collusion. The Chen method adopts a delegated computation strategy by assigning overlapping computing blocks to multiple CPs, thereby detecting collusion through inconsistent computation results. The Yan method introduces sentinel tasks into outsourced computation to identify successful collusion. The main characteristics of these methods are summarized in Table 3.

In the Dong method, a Traitor's contract is employed to address collusion issues. A CP can report collusion by submitting additional results to the CS without violating the collusion contract and obtain the corresponding reward. In the Chen method, the CS divides a task into m ($\frac{n}{2} \leq m < n$) computing blocks and distributes them among n CPs. Each block is assigned to at most two CPs, enabling collusion detection through repeated computation. We set $m = \frac{3}{4}n$, where $\frac{n}{2}$ CPs receive duplicated computing blocks. In the Yan method, the CS sends k tasks to two selected CPs in each round, including m sentinel tasks and $k - m$ ordinary tasks. Collusion can be detected through incorrect results of sentinel tasks. We set $m = k/2$ and $k = 10$.

Table 3. Comparison of anti-collision mechanisms.

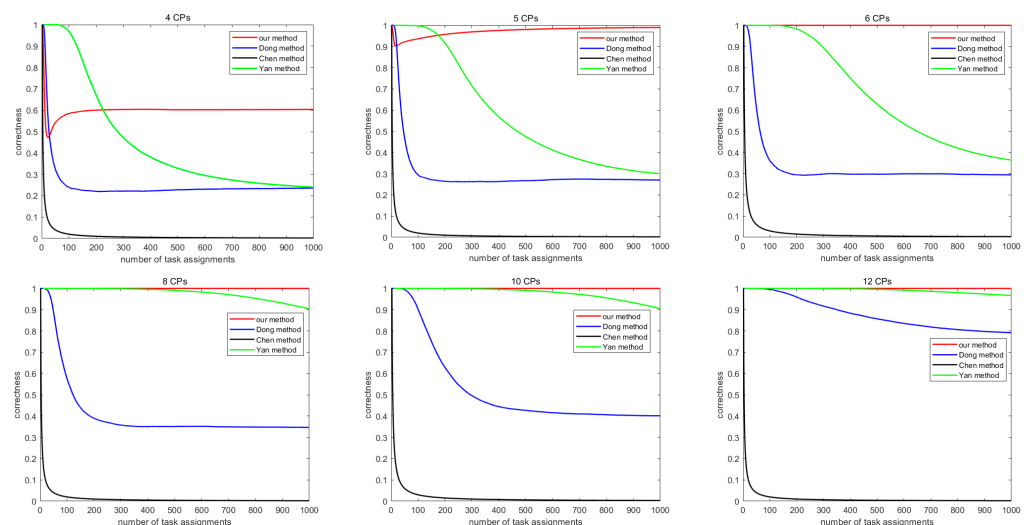
Method	Mechanism	Main Strategy	Incentive/Punishment
Dong et al. [3]	Traitor's contract	A CP reports collusion by submitting additional results and obtains the deposit of the colluding party as a reward.	Reward-based incentive for the reporter.
Chen et al. [8]	1-n delegated computation	Tasks are divided into multiple computing blocks and assigned to overlapping CPs for repeated computation.	Relies on redundant computation to ensure correctness.
Yan et al. [33]	Outsourced computation with sentry tasks	Sentinel tasks are inserted into outsourced tasks to detect successful collusion.	Uses detection probability to discourage collusion.
Our method	Report contract and reputation certificate	A CP reports collusion by providing evidence and obtains reputation incentives.	Combines incentive mechanism and reputation penalty.

6.2.1. Standardized Task Scenario

To evaluate the effectiveness of the proposed scheme under uniform task difficulty, a standardized task scenario is designed. In this setting, the client repeatedly issues tasks with similar computational complexity, simulating highly homogeneous application environments such as batch data processing. This setup helps eliminate interference from task variability and focuses on assessing CP strategies. We set the parameters as follows: CP calculates the task cost $c = 5$, the Outsourcing contract deposit $e = 65$, the LDR reward paid to the FLR $o = 2$, the payment after CS accepts the result $r = 20$, the collusion contract deposit $t = 90$, and the calling TTP fee $z = 57$. The outsourcing by client consists of 1000 rounds.

These parameters choices satisfy the analysis of our work. The simulation code was written in Matlab 9.14.0 and executed on a PC. The results presented are the average of 500 executions of each mechanism. We let the value of n vary across all experiments, using 4, 5, 6, 8, 10, and 12 sequentially, to observe the influence on the number of CPs.

The results of our simulations can be seen in Figure 5. We compare the mechanisms with respect to the ratio of the number of tasks for which the CS obtains the correct answer to the total number of task assignments.

**Figure 5.** Comparison of our method, Dong method, Chen method and Yan method: correctness vs. number of task assignments.

It can be observed from Figure 5 that our scheme is significantly better than other schemes. The existing scheme basically incentivise a CP to secretly betray a collusion after the collusion is over. In the long run, the discovery of treachery is a big possibility, especially in repeated interactions. Because betrayal collusion is found by the Colluder's contract, it will suffer huge losses, so it may tend to stable collusion. Our scheme is more effective in rewarding those who report collusion. When the number of CPs in our scheme is greater than 5, CS will definitely receive the correct answer. This is consistent with the above theoretical analysis. As the number of CPs increased, our incentive (Reputation growth of CP) became more attractive.

With the increase in the number of CPs, the correctness of the Dong method and Yan method also gradually increases, that is, the more CPs, the more difficult it is to establish trust and form collusion among CPs. In each round of the Chen method, all CPs participate in the game, so the number of CPs has little effect on the experimental results.

6.2.2. Diverse Task Scenario

To evaluate the adaptability of the proposed scheme in scenarios with significant task complexity variation, a diverse task scenario is constructed. In this setting, the client repeatedly issues tasks of three different difficulty levels, with notable differences in computational complexity and resource requirements, simulating heterogeneous application environments such as diverse data processing. This setup is designed to assess the behavioral stability of CPs under varied tasks and to verify the regulatory capability and robustness of the proposed scheme. We set the task difficulty coefficients to 0.25, 0.5, and 1, respectively. The remaining parameters vary accordingly with the difficulty coefficient. The client randomly generates three different-level tasks.

In a diverse task scenario, 100 tasks are processed. When a total of six CPs participate, their final reputations are shown in Figure 6. CP_4 once proposed collusion to CP_2 and was reported by CP_2 . As a result, CP_4 's reputation dropped to 0, and clients no longer consider CP_4 in subsequent task allocations. Meanwhile, CP_2 's reputation increased due to the reporting behavior. This indicates that the reputation mechanism in our scheme can effectively resist collusion. Figure 7 shows the task accuracy of our scheme and three comparison schemes under a diverse task scenario with six participating CPs. It can be observed that our scheme outperforms the others in terms of efficiency under the diverse task scenario. The comparison Table 4 presents the task accuracy of the four schemes in both standard and diverse task scenarios, each with five participating CPs. It can be seen that, regardless of the scenario type, our scheme effectively resists collusion, demonstrating strong adaptability across different task settings.

The computational complexity of these operations does not increase with the number of CPs. Specifically, once a CP initiates or reports a collusion behavior, the smart contract only verifies the corresponding signature and updates the reputation certificate of the involved CPs. Therefore, the computational complexity of reputation management is $O(1)$ with respect to the number of CPs. The storage complexity mainly comes from maintaining CP identities and reputation certificates. Assuming each CP stores one identity and one reputation certificate, the blockchain storage overhead is $O(N)$, where N denotes the number of CPs. This linear growth is acceptable because only metadata and reputation information are stored on-chain, while computation tasks and results remain off-chain.

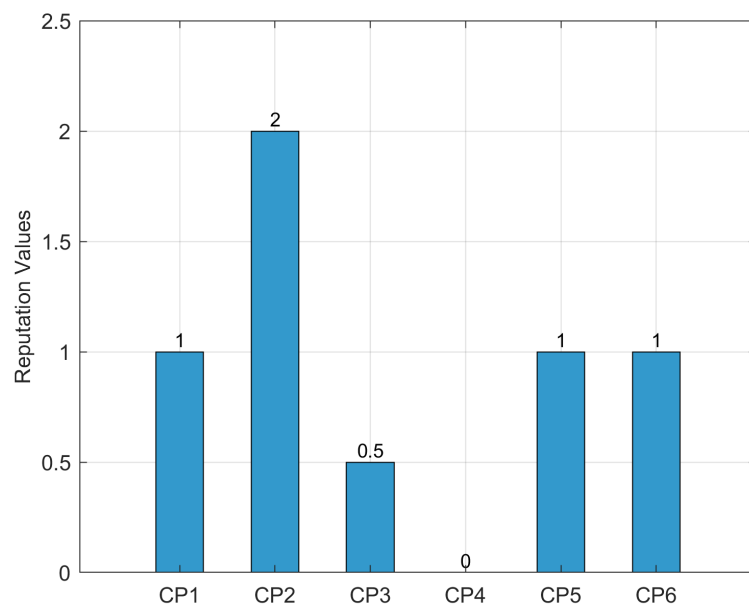


Figure 6. Reputation Evaluation of Different CPs.

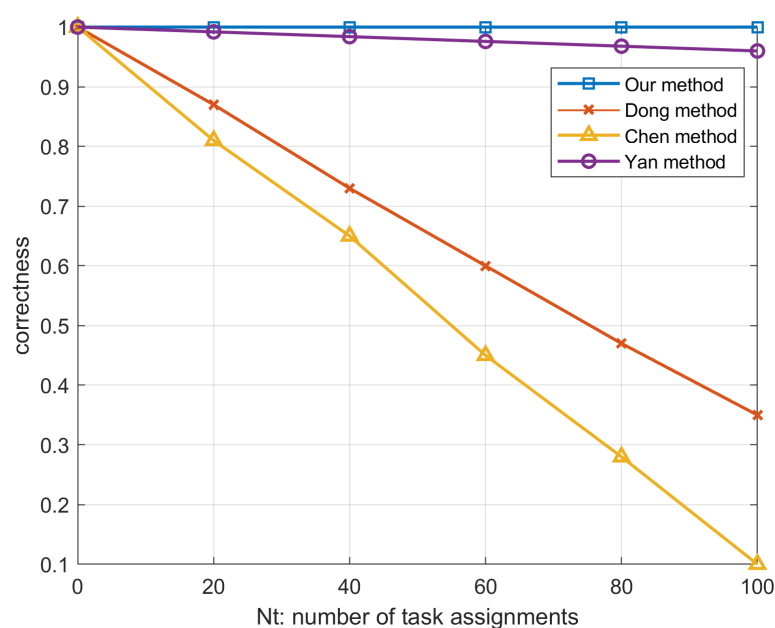


Figure 7. Correctness trends of different methods as the number of task assignments increases under the diverse task scenario.

Table 4. Task accuracy comparison of four methods under standardized and diverse task scenarios with varying task counts N_t .

Method	Standardized Task Scenario			Diverse Task Scenario		
	$N_t = 100$	$N_t = 500$	$N_t = 1000$	$N_t = 100$	$N_t = 500$	$N_t = 1000$
Our method	0.93	0.98	0.99	0.92	0.98	0.99
Dong method	0.35	0.30	0.29	0.33	0.30	0.28
Chen method	0.07	0.03	0.01	0.05	0.02	0.01
Yan method	0.98	0.51	0.35	0.96	0.48	0.33

7. Conclusions

We analyze the collusion problem in current two-cloud-provider-based verification schemes in this paper. We propose a new Outsourcing contract that automatically updates the reputation certificates of CPs based on their behavior. Our scheme fundamentally changes the existing counter-collusion solutions and solves their ineffective problem in multi-round situations. The Report contract introduced in our scheme incentivizes one cloud to report the collusion once the other cloud initiates it. As it is an advance and public reporting scheme, the strong assumption that betrayal behavior is undiscovered forever is not needed in our scheme. A rational cloud is more likely to report a collusion here, as the reward is much larger than the reward in existing contracts. Our scheme consistently outperforms the compared methods in task accuracy under both standardized and diverse task scenarios. Moreover, unlike existing schemes whose accuracy degrades as the number of tasks increases, our method maintains an accuracy above 98% across all evaluated task scales. When the number of CPs exceeds five, our scheme further guarantees that the client always receives correct computation results.

In this article, we have not considered Sybil-resistant identity management, registration cost, optimization, and appeal mechanisms. In the future, we will explore these issues to further enhance the applicability and robustness of the proposed scheme in large-scale decentralized environments.

Author Contributions: Conceptualization, X.W. and Y.R.; methodology, Y.R.; software, Y.R.; validation, X.W., Y.R. and L.S.; formal analysis, Y.R.; investigation, L.S.; resources, X.W.; data curation, X.W.; writing—original draft preparation, Y.R.; writing—review and editing, L.S.; visualization, Y.R.; supervision, L.S.; project administration, X.W.; funding acquisition, L.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Zhang, Q.; Cheng, L.; Boutaba, R. Cloud computing: State-of-the-art and research challenges. *J. Internet Serv. Appl.* **2010**, *1*, 7–18. [CrossRef]
2. Zhao, M.; Hu, C.; Song, X.; Zhao, C. Towards dependable and trustworthy outsourced computing: A comprehensive survey and tutorial. *J. Netw. Comput. Appl.* **2019**, *131*, 55–65. [CrossRef]
3. Dong, C.; Wang, Y.; Aldweesh, A.; McCorry, P.; Van Moorsel, A. Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*; ACM: New York, NY, USA, 2017; pp. 211–227.
4. Costello, C.; Fournet, C.; Howell, J.; Kohlweiss, M.; Kreuter, B.; Naehrig, M.; Parno, B.; Zahur, S. Geppetto: Versatile verifiable computation. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*; IEEE: New York, NY, USA, 2015; pp. 253–270.
5. Walfish, M.; Blumberg, A.J. Verifying computations without reexecuting them. *Commun. ACM* **2015**, *58*, 74–84. [CrossRef]
6. Zhang, Y.; Yang, D.; Xue, G.; Yu, R. Counter-collusion smart contracts for watchtowers in payment channel networks. In *Proceedings of the IEEE INFOCOM 2021—IEEE Conference on Computer Communications*; IEEE: New York, NY, USA, 2021; pp. 1–10.
7. Amazon. AWS Total Cost of Ownership (TCO) Calculator. 2023. Available online: <https://aws.amazon.com/tco-calculator/> (accessed on 3 August 2026).
8. Chen, Z.; Tian, Y.; Xiong, J.; Peng, C.; Ma, J. Towards reducing delegation overhead in replication-based verification: An incentive-compatible rational delegation computing scheme. *Inf. Sci.* **2021**, *568*, 286–316. [CrossRef]
9. Liu, Y.; He, Z.; Liang, J.; Li, Z.; Deng, Q. Multidimensional Trust Evaluation and Task Match Based Workers Recruitment Scheme for MCS. *IEEE Trans. Dependable Secur. Comput.* **2026**, *23*, 5058–5074. [CrossRef]
10. Liu, Y.; He, Z.; Liu, A.; Dai, X.; Deng, Q.; Li, Z. TDI: A Trust-Based Distributed Incentive Scheme to Promote Information Propagation. *IEEE Trans. Mob. Comput.* **2026**, *25*, 8566–8582. [CrossRef]

11. Liu, P.; Zhang, W. A new game theoretic scheme for verifiable cloud computing. In *Proceedings of the 2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*; IEEE: New York, NY, USA, 2018; pp. 1–8.
12. Li, T.; Tian, Y.; Xiong, J.; Bhuiyan, M.Z.A. FVP-EOC: Fair, verifiable, and privacy-preserving edge outsourcing computing in 5G-enabled IIoT. *IEEE Trans. Ind. Inform.* **2022**, *19*, 940–950. [[CrossRef](#)]
13. Chen, Y.; Ma, J.; Wang, X.; Zhang, X.; Zhou, H. DE-RSTC: A rational secure two-party computation protocol based on direction entropy. *Int. J. Intell. Syst.* **2022**, *37*, 8947–8967. [[CrossRef](#)]
14. Guo, J.; Du, Y.; Zhang, D.; Wu, R. PSEEMV: Provably secure and efficient emergency message verification scheme based on ECC and CRT for Space Information Network. *J. Inf. Secur. Appl.* **2023**, *73*, 103437. [[CrossRef](#)]
15. Tanwar, N. *An Efficient and Secure Blockchain Based Homomorphic Encryption for Intelligent Transport System*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 752–759.
16. Kumar, P.; Sharma, S.K.; Kadam, K.U. Empirical Evaluation of ECC Batch Verification Algorithms of Digital Signature in Wireless Sensor Network. *Electronics/Elektronika (1450-5843)* **2022**, *26*, 53. [[CrossRef](#)]
17. Devadas, L.; Goyal, R.; Kalai, Y.; Vaikuntanathan, V. Rate-1 non-interactive arguments for batch-NP and applications. In *Proceedings of the 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*; IEEE: New York, NY, USA, 2022; pp. 1057–1068.
18. Agarwal, A.; Alamati, N.; Khurana, D.; Raghuraman, S.; Rindal, P. On Black-Box Verifiable Outsourcing. In *Proceedings of the Theory of Cryptography Conference*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 158–187.
19. Mudigonda, A.S.; Williams, R.R. Time-Space Lower Bounds for Simulating Proof Systems with Quantum and Randomized Verifiers. *arXiv* **2020**, arXiv:2012.00330.
20. Chen, Y.; Xie, R.; Zhang, H.; Li, D.; Lin, W. Generation of high-order random key matrix for Hill Cipher encryption using the modular multiplicative inverse of triangular matrices. *Wirel. Netw.* **2023**, *30*, 5697–5707. [[CrossRef](#)]
21. Yu, X.; Yan, Z.; Zhang, R. Verifiable outsourced computation over encrypted data. *Inf. Sci.* **2019**, *479*, 372–385. [[CrossRef](#)]
22. Shen, J.; Liu, D.; Chen, X.; Li, J.; Kumar, N.; Vijayakumar, P. Secure Real-Time Traffic Data Aggregation with Batch Verification for Vehicular Cloud in VANETs. *IEEE Trans. Veh. Technol.* **2020**, *479*, 807–817. [[CrossRef](#)]
23. Ben-Sasson, E.; Chiesa, A.; Goldberg, L.; Gur, T.; Riabzev, M.; Spooner, N. Linear-size constant-query IOPs for delegating computation. In *Proceedings of the Theory of Cryptography Conference*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 494–521.
24. Kalai, Y.T.; Paneth, O.; Yang, L. How to delegate computations publicly. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*; ACM: New York, NY, USA, 2019; pp. 1115–1124.
25. Kalai, Y.T.; Paneth, O.; Yang, L. Delegation with updatable unambiguous proofs and PPAD-hardness. In *Proceedings of the Annual International Cryptology Conference*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 652–673.
26. Holmgren, J.; Rothblum, R. Delegating computations with (almost) minimal time and space overhead. In *Proceedings of the 2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*; IEEE: New York, NY, USA, 2018; pp. 124–135.
27. Li, L.; Zhang, T.; Sun, G.; Jin, D.; Li, N. A Fair, Verifiable and Privacy-Protecting Data Outsourcing Transaction Scheme Based on Smart Contracts. *IEEE Access* **2022**, *10*, 106873–106885. [[CrossRef](#)]
28. Ma, J.; Chen, Y.; Wang, Z.; Liu, G.; Zhu, H. A rational delegating computation protocol based on reputation and smart contract. *J. Cloud Comput.* **2021**, *10*, 51. [[CrossRef](#)]
29. Dong, L.; Mosteiro, M.A.; Singh, S. Verifiable Crowd Computing: Coping with Bounded Rationality. In *Proceedings of the International Workshop on Frontiers in Algorithmics*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 59–78.
30. Canetti, R.; Riva, B.; Rothblum, G.N. Practical delegation of computation using multiple servers. In *Proceedings of the 18th ACM Conference on Computer and Communications Security, October 2011*; ACM: New York, NY, USA, 2011. [[CrossRef](#)]
31. Distler, T.; Cachin, C.; Kapitzka, R. Resource-Efficient Byzantine Fault Tolerance. *IEEE Trans. Comput.* **2016**, *65*, 2807–2819. [[CrossRef](#)]
32. Lai, R.; Zhao, G. Blockchain for achieving accountable outsourcing computations in edge computing. *Comput. Commun.* **2023**, *200*, 17–29. [[CrossRef](#)]
33. Yan, H.; Jiang, N.; Li, K.; Wang, Y.; Yang, G. Collusion-free for cloud verification toward the view of game theory. *ACM Trans. Internet Technol. (TOIT)* **2021**, *22*, 1–21. [[CrossRef](#)]
34. Foundation, E. Ethereum’s White Paper. 2016. Available online: <https://ethereum.org/en/whitepaper/> (accessed on 3 August 2026).
35. Tomassini, M.; Pestelacci, E.; Luthi, L. Mutual trust and cooperation in the evolutionary hawks–doves game. *Biosystems* **2010**, *99*, 50–59. [[CrossRef](#)] [[PubMed](#)]
36. Taylor, C.; Nowak, M.A. Transforming the dilemma. *Evolution* **2007**, *61*, 2281–2292. [[CrossRef](#)] [[PubMed](#)]
37. Lieh, O.E.; Jarzabek, S. An adaptability-driven model and tool for analysis of service profitability. In *Proceedings of the Advanced Information Systems Engineering: 28th International Conference, CAiSE 2016 Proceedings 28, Ljubljana, Slovenia, 13–17 June 2016*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 393–408.

38. Leyton-Brown, K.; Shoham, Y. *Essentials of Game Theory: A Concise Multidisciplinary Introduction*; Morgan & Claypool Publishers: San Rafael, CA, USA, 2022.
39. Owen, G. *Game Theory*; Emerald: Leeds, UK, 2013.
40. Kreps, D.M.; Wilson, R. Sequential equilibria. *Econom. J. Econom. Soc.* **1982**, *50*, 863–894. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.