How to Construct Rational Protocols with Nash Equilibrium Consistency in the UC framework

Xiaoxia Jiang and Youliang Tian

State Key Laboratory of Public Big Data, College of Computer Science and Technology, Guizhou University, Guiyang 550025, China
15650751086@163.com, yltian@gzu.edu.cn

Abstract. The inconsistency of Nash equilibrium of rational delegated computation scheme in the UC framework will lead to the lack of strict security proof of the protocols fundamentally. The consistency proof of Nash equilibrium between the ideal world and the real world has always been a challenge in the research field. In this paper, we analyze the Nash equilibrium according to the game model of rational delegated computation, and the ideal functionality $\mathcal{F}_{RDC}$ for rational delegation of computation based on incentive-driven adversary is proposed, then we construct a rational delegated computation protocol π_{RDC} for UC-realizing the ideal functionality $\mathcal{F}_{RDC}$. In a word, the proposed rational delegated computing protocol π_{RDC} based on incentive-driven adversary has been proven to be secure in the universally composable framework, furthermore, we effectively solve the inconsistency problem of Nash equilibrium between the real world and the ideal world.

Keywords: Universally composable · Nash equilibrium · Rational delegation of computation

1 Introduction

Cloud computing is a newly-emerging business computing paradigm based on distributed technology, which takes on-demand allocation as the business model and provides users with powerful remote computing and storage capabilities. The emergence of cloud computing has brought the new flourish to delegated computation, mainly due to the fact that mobile terminal devices such as smart wearing devices, automobile internal sensors and so on used in the Internet of Things and Smart City only have limited computing power and storage capacity. Therefore, a feasible method is to upload data from these terminals to the cloud, and delegate a cloud server for calculation. We further explore that the traditional delegated computing scheme does not consider the situation that participants are rational, but once the participants are regarded as rational, which will lead to the inconsistency of Nash equilibrium between the real model and the ideal model, thus bringing security risks to the designed protocol.

Delegated computation refers to a client with limited computing resources delegates a computing task to an untrustworthy server with powerful computing power, in which not only ensures the privacy of users' data, but also guarantees the correctness (i.e. verifiability) of the results returned by the computing party. The existing delegated computing schemes can be strictly divided into traditional delegated computation and rational delegated computation. Cryptography and computational complexity theory are the basic tools used to construct traditional delegation computing schemes. **Rational delegated computation**, however, combines game theory with traditional delegated computation and is required to design appropriate incentive mechanisms to motivate participants to act honestly, in which all participants are regarded as rational and make decisions based on their own maximum utility. Although there have been some research works [9, 16, 22, 30] about the UC (Universally Composable) model [10] of delegating quantum computation, participants are not viewed as rational in these schemes. Furthermore, the previous rational delegated computation protocols do not consider a security model in the UC framework. More importantly, the existing cryptographic protocols have not solved the inconsistency problem of equilibrium between the real world and the ideal world in the UC framework. This problem makes the adversary be able to distinguish the execution of the real world from the ideal world with non-negligible probability, which will lead to the fundamental insecurity of rational cryptographic protocols.

According to the above analysis, we know that: 1. there is still a lack of rational delegated computing UC model; 2. in other rational UC models (such as [18]), there is a vital and unsolved security challenge of Nash equilibrium inconsistency. Therefore, we construct a rational delegated computing model in the UC framework, meanwhile, which effectively solves the problem of inconsistency of Nash equilibrium between the real world and the ideal world.

Related Work. In recent years, research schemes on traditional delegated computing have emerged in large numbers, which can be mainly divided into the following aspects: (1) The schemes based on cryptography technology, in which fully homomorphic encryption (FHE), homomorphic MACs and homomorphic signature are used as main tools. Gennaro et al. [19] firstly proposed the delegating computation scheme by combining FHE and Yao's garbled circuit [34]. In the same year, a FHE-based work that improved the efficiency of delegated computation was presented in [12]. Gennaro and Wichs [20] formalized the homomorphic MAC and designed a delegated computing protocol for arbitrary functions, which has been futher optimized in [4, 17]. There are also some schemes [4, 11, 24] based on homomorphic signature construction. (2) The works based on computational complexity theory include [13, 27, 28] (based on interactive proof [2, 21]), [3, 6, 7, 14, 25, 31, 33] (based on non-interactive proof) and [8, 26, 32] (i.e. the PCP theory [1] and PCP-based constructions). More recently, Zhao et al. [36] systematically reviewed a mass of researches of delegated computation.

Rational delegated computing has gradually become a research hotspot recently. Belenkiy et al. [5] proposed an incentive outsourcing computation scheme, there is a reputation or credit system so that rational contractors can calculate work correctly by setting incentives. Zhang et al. [35] also designed a reputation-based incentive protocol in crowdsourcing applications by using a rigorous repetitive game framework. Kupcu [23] deemed that there is no fair payment in [35], and could not resist irrational malicious contractors. Therefore, an outsourcing computing scheme for incentivizing malicious contractors was constructed in [23]. Game theory and smart contracts were combined in [15], which constrained the dishonest behavior of the computing party by setting up three different contracts (sub-games) to motivate participants to behave honestly. Tian et al. [29] combined the computation delegation and information theory to investigate the attack and defense limitation of players in the delegated computation.

Our Contributions. In this paper, we have completed the following work to solve the inconsistent problem of Nash equilibrium between the real world and the ideal world in the UC framework.

1. We put forward a rational delegated computing model that can solve the inconsistency of equilibrium between the ideal world and the real world in the UC framework.
2. Based on the idea of [18], we further design a UC model based on incentive-driven adversary and construct the first ideal functionality $\mathcal{F}_{RDC}$ for rational delegated computation based on the incentive-driven adversary, which provides a fair security mechanism for rational delegation of computation between the client and the computing party.
3. Besides, we design a secure protocol π_{RDC} for rational delegated computation based on the incentive-driven adversary. There are three contracts used to set reward and punishment mechanisms, so as to incentive participants to behave honestly. Homomorphic encryption is used to guarantee the privacy of data, and Pedersen Commitment is combined with PCP-based construction to ensure the correctness of the submitted results, respectively.
4. According to the composition theorem, we prove that the protocol π_{RDC} securely realize the ideal functionality $\mathcal{F}_{RDC}$ in the UC framework, that is, the proposed protocol π_{RDC} can achieve the consistency while meeting the requirements of UC security, which solves the security defects in the UC model of rational delegated computation due to the inconsistency of equilibrium.

Framework. Firstly, we propose three contracts, namely, delegation contract, corruption contract and betrayal contract, to set up the incentive mechanism for each participant in the scenario of rational delegated computation. Then we design a three-party game model between the client, the computing party and the external adversary. Based on the idea of [18], we transform the original three-party game model into a two-party attack game $\mathcal{G}_{\mathcal{M}}$ between internal participants and the external adversary. Then we obtain the Nash equilibrium in the rational delegated computing game model through the game analysis. On the basis of this Nash equilibrium, we present the first ideal functionality

$\mathcal{F}_{RDC}$ for rational delegated computation based on incentive-driven adversary. Furthermore, we design the first rational delegation computing protocol π_{RDC} based on incentive-driven adversary. Finally, we realize the security proof of the proposed protocol in the UC framework, that is, the protocol UC-securely realizes the ideal functionality $\mathcal{F}_{RDC}$; and the problem of inconsistency of Nash equilibrium between the real world and the ideal world has been solved elegantly.

Organization. The remainder of this paper is organized as follows: In section 2, we briefly describe our system model and rational cryptographic protocol design method based on incentive-driven adversary as preliminaries. In Section 3, we introduce the model assumptions and three different contracts. Then we formalize our game model and obtain Nash equilibrium through game analysis in Section 4. According to the conclusion of last section, in Section 5, we construct the ideal functionality $\mathcal{F}_{RDC}$ for rational delegated computation based on incentive-driven adversary. Then we design a rational delegated computing protocol π_{RDC} to realize ideal functionality in Section 6. In Section 7, we provide the protocol analysis, which includes security proofs and comparison with other schemes. The conclusion is given in Section 8.

2 Preliminaries

We briefly formalize our system model in Section 2.1. And we will review relevant content in rational cryptographic protocol design method based on incentive-driven adversaries mostly based on [18] in Section 2.2.

2.1 System Model

Our system model consists of four entities: the client, the computing party, the external adversary, and the trusted third party (TTP) with computing power, as shown in Fig. 1.

All participants except TTP in the model are rational that make decisions based on the own maximum expected payoff. The client and the computing party are regarded as internal participants who execute the protocol. While the external adversary is to achieve the purpose of breaking the protocol by corrupting one of the internal participants. TTP, a trusted third party with computing power, is not only responsible for the receipt and dispatch of deposits of internal participants, but also arbitrating in case of controversies between the internal participants.

In the delegation phase, the client signs a **delegation contract** with the computing party to ensure the honest implementation of the protocol between them, in which the client obtains the correct result while the computing party gains the corresponding delegation fees. The external adversary chooses whether to corrupt the client or the computing party and signs a **corruption contract** with the corrupted during the corruption phase to guarantee that the corrupted will help the external adversary disrupt the normal execution of the protocol.

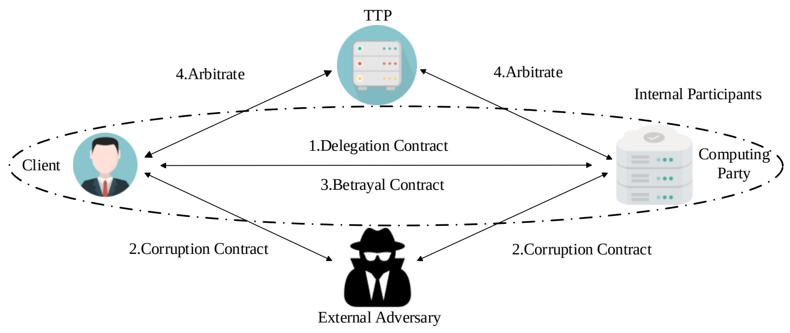

Fig. 1. System model

The corrupted internal participant may choose to sign a **betrayal contract** with another honest party in the computation phase or the verification phase, at this time, the person being corrupted is called a traitor. This contract ensures that the traitor's breach of the corruption contract is risk-free, thereby causing mistrust between the external adversary and the corrupted. The honest internal participant will send a request for arbitration to TTP if the dishonest internal participant is found in the verification stage, and the final penalty will be imposed by TTP.

2.2 Cryptography Against Incentive-driven Adversaries

Garay et al. [18] defined the rational protocol design method based on incentive-adversary in the following way.

Cryptographic Security as a Game. Firstly, the cryptographic security is formalized into an attack game, and then the utility of each player in the game model is defined.

(1) *The Attack Game* $\mathcal{G}_{\mathcal{M}}$. The attack game in [18] is a two-party zero-sum extensive game of perfect information with a horizon length 2. The protocol designer D and the external attacker A are regarded as the leader and the follower, respectively. D first chooses a probabilistic and polynomial-time computable protocol Π for computing the functionality $\mathcal{F}$, which is described as $\Pi \subseteq \{0,1\}^*$. Once receiving Π, the attacker A selects a polynomial time ITM $\mathcal{A}$ as its own strategy to destroy the protocol. Let denote by $\mathcal{G}_{\mathcal{M}}$ the above attack game, where the subscript $\mathcal{M}$ represents the attack model, which specifies all the public parameters of the game, including utilities, the functionality and description of action.

Definition 1. *Let* $\mathcal{G}_{\mathcal{M}}$ *be an attack game. A strategy profile* (Π, A) *is an* $-$ *subgame perfect equilibrium in* $\mathcal{G}_{\mathcal{M}}$ *if the following conditions hold: (a) for any*

$\Pi' \in \mathrm{ITM}^{\mathrm{n}}$, *$u_{\mathrm{D}}(\Pi', \mathrm{A}(\Pi')) \leq u_{\mathrm{D}}(\Pi, \mathrm{A}(\Pi)) + \epsilon$, and (b) for any* $\mathrm{A}' \in \mathrm{ITM}^{n}$, *$u_{\mathrm{A}}(\Pi, \mathrm{A}'(\Pi)) \leq u_{\mathrm{A}}(\Pi, \mathrm{A}(\Pi)) + \epsilon$, where u_{D} and u_{A} represent the utilities of the protocol designer and the external attacker, respectively.*

(2) *Defining the Utilities.* The method of defining adversary's utility consists of three steps in [18]:

1) The first step is to "relax" the ideal functionality $\mathcal{F}$ to a weaker ideal functionality $\langle\mathcal{F}\rangle$, which allows the attacks that the adversary wants to implement.

2) In the second step, the payoff of ideal adversary $\mathcal{S}$ is defined according to the weaker ideal functionality $\langle\mathcal{F}\rangle$ and an evaluation function v. The function v is used to map the joint view of relaxed functionality $\langle\mathcal{F}\rangle$ and environment $\mathcal{Z}$ to a real-valued payoff. $v_{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$ as a random variable to describe the payoff of by accessing functionality $\langle\mathcal{F}\rangle$ directly. The ideal expected payoff of $\mathcal{S}$ with respect to the environment $\mathcal{Z}$ is defined to be the expected value of $v_{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}$, i.e.,

$$U_I^{\langle\mathcal{F}\rangle}(\mathcal{S}, \mathcal{Z}) = E(v_{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}) \tag{1}$$

3) The third step is to assign the "best" simulator for each adversarial strategy for a certain protocol, so as to obtain the real expected payoff corresponding to the adversarial strategy based on the above-mentioned ideal expected payoff. For a functionality $\langle\mathcal{F}\rangle$ and a protocol Π, denote the class of simulators that are "good" for $\mathcal{A}$ by $\mathcal{C}_{\mathcal{A}}$, i.e, $\mathcal{C}_{\mathcal{A}} = \{\mathcal{S} \in \mathrm{ITM} \mid \forall\mathcal{Z} : \mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \mathrm{EXEC}_{\langle\mathcal{F}\rangle,\mathcal{S},\mathcal{Z}}\}$. Then the real expected payoff of the pair $(\mathcal{A}, \mathcal{Z})$ is defined as

$$U^{\Pi,\langle\mathcal{F}\rangle} = \inf_{\mathcal{S}\in\mathcal{C}_{\mathcal{A}}} U_I^{\langle\mathcal{F}\rangle}(\mathcal{S}, \mathcal{Z}) \tag{2}$$

$U^{\Pi,\langle\mathcal{F}\rangle}$ assigns a real number to each pair $(\mathcal{A}, \mathcal{Z}) \in \mathrm{ITM} \times \mathrm{ITM}$, indicating the expected payoff of $\mathcal{A}$ attacking protocol Π in environment $\mathcal{Z}$.

Further define the maximum payoff of an adversarial strategy $\mathcal{A}$ based on the real expected payoff that has been defined. At this point, the protocol needs to be executed in the "preferred" environment of the adversary, which refers to the environment that maximizes the utility of the external adversary who chooses the attack strategy $\mathcal{A}$. Thereby the maximal payoff of an adversary $\mathcal{A}$ attacking protocol Π for realizing $\langle\mathcal{F}\rangle$ with respect to a certain payoff function v is defined as

$$\hat{U}^{\Pi,\langle\mathcal{F}\rangle}(\mathcal{A}) = \sup_{\mathcal{Z}\in\mathrm{ITM}} \{\hat{U}^{\Pi,\langle\mathcal{F}\rangle}(\mathcal{A}, \mathcal{Z})\} \tag{3}$$

Finally, having defined the maximal payoff of any given adversarial strategy $\mathcal{A}$, the utility function u_{A} of the external attacker A in the attack game $\mathcal{G}_M$ can be defined. Let $(\Pi, \mathcal{A})$ terminal history in $\mathcal{G}_M$, i.e., $\mathcal{A} = \mathrm{A}(\Pi)$. Then as follows

$$u_{\mathrm{A}}(\Pi, \mathcal{A}) := \hat{\mathrm{U}}^{\Pi,\langle\mathcal{F}\rangle}(\mathcal{A}) \tag{4}$$

Since $\mathcal{G}_M$ is a zero-sum game, the utility of the protocol designer D is $u_{\mathrm{D}}(\cdot) := -u_{\mathrm{A}}(\cdot)$.

The Attack Game as a Cryptographic (Maximization) Problem.

Definition 2. *Let $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v)$ be an attack game and* Π *be a protocol that realizes $\langle\mathcal{F}\rangle$. We say that* ITM $\mathcal{A}$ *is a $\mathcal{M}$ - maximizing adversary for* Π *if*

$$\hat{U}^{\Pi,\langle\mathcal{F}\rangle}(\mathcal{A}) \overset{negl}{\approx} \sup_{\mathcal{A}' \in \mathrm{ITM}} \hat{U}^{\Pi,\langle\mathcal{F}\rangle}(\mathcal{A}) =: \hat{U}^{\Pi,\langle\mathcal{F}\rangle} \tag{5}$$

Protocol Composition.

Definition 3. *Let $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v)$ be an attack game and* Π *be a protocol that realizes $\langle\mathcal{F}\rangle$. Protocol* Π *is attack-payoff secure in $\mathcal{M}$ if $\hat{U}^{\Pi,\langle\mathcal{F}\rangle} \overset{negl}{\leq} \hat{U}^{\Phi^{\mathcal{F}},\langle\mathcal{F}\rangle}$,where $\Phi^{\mathcal{F}}$ is the dummy $\mathcal{F}$ -hybrid protocol.*

Theorem 1. *Let $\mathcal{M} = (\mathcal{F}, \langle\mathcal{F}\rangle, v)$ be an attack game and* Π *be a $\mathcal{H}$-hybrid protocol, and* Ψ *be a protocol that securely realizes $\mathcal{H}$ (in the traditional simulation-based notion of security). Then replacing in* Π *calls to $\mathcal{H}$ by invocations of protocol* Ψ *does not (noticeably) increase the utility of a $\mathcal{M}$-maximizing adversary.*

3 Assumptions and Contracts

In section 3.1, we first introduce the assumptions required by the model, and then in section 3.2, we present the three different contracts that form a restrictive relationship among them, so as to incentivize the honest behavior of players.

3.1 Model Assumptions

The client, the computing party and the external adversary under this model are all regarded as rational participants, in which the client benefits from receiving the correct calculation results; the computing party obtains payoff if completing the client's delegation tasks honestly; and the external adversary gains by attacking the protocol successfully.

Unlike other schemes, we consider not only the correctness of the calculation results, but also its privacy, namely, the calculated results should be hidden for the client during the verification process, the client will not receive the real calculation results until the verification information submitted by the computing party is verified to be right by the client. The main reason for that is order to prevent from an unallowed situation that even if the dishonest client's malicious behavior is eventually detected and required to pay a fine to the honest computing party, he still gets the corresponding benefit for owning the correct calculated result.

Due to the client and the computing party in the internal participants are not responsible for the same work, as well as they are likely to deceive each other for their own payoff, the scenario is modeled as a three-party dynamic game with perfection information between the client, the computing part and the external adversary, rather than a two-party game between the internal participants and the external adversary. It is worth noting that since TTP only acts as an impartial arbitration institution when needed and does not make different decisions for own payoff, it is not considered as a participant in the game model in this paper.

The conventions for the rational delegated computing scenario discussed in this paper are as follows:

- Assume that the client is unable to complete the computing task due to limited computing power, and can only verify the sub-tasks; while both the computing party and TTP are entities with powerful computing power.
- It is assumed that the corruption is not free, and the external adversary can choose to corrupt the client or the computing party, but not both. In order to view internal participants as more fully rational, we suppose that even if the external adversary chooses to corrupt an internal participant (the client or the computing party), the internal participant can still choose whether to accept the corruption or not. Even if the internal participant chooses to accept the corruption at this time, he can still decide to perform honestly or dishonestly in the subsequent implementation phase of the protocol.
- The client and the computing party will not conduct dishonest behavior on their own initiative. Only when the external adversary chooses the strategy of corruption may the client or the computing party deviate from the protocol. Thus, there are no malicious rumors against the external adversary in the betrayal contract.
- When the external adversary corrupts the client, which means that the client will always verify the calculation result submitted by the computing party as a failure and refuse to accept the calculated result in the final verification stage. If the computing party is corrupted, he will send an incorrect verification information to the client during the verification phase. Since the purpose of the external adversary is to break the transaction and benefit from it, this paper only considers the external adversary's attack behavior that can directly lead to the failure of the transaction rather than destroying the privacy of data (guaranteed by homomorphic encryption).
- Suppose that the client will certainly be able to submit the correct calculation results if he honestly calculates the delegated task, at this time, the honest client can also correctly verify the results submitted by the computing party regardless of the errors caused by the actual factors.
- We presume that the contracts have legal effect, either dishonest internal participants or the malicious external adversary will certainly be punished. What's more, a dishonest internal participant will suffer a loss of reputation, while this fee is not charged by TTP, but an indirect loss caused by the negative impact on future follow-up work.

3.2 The Contracts

Monetary Variables. The monetary variables used in the three contracts are specified in Table 1. The following variables are all non-negative.

Table 1. The monetary variables

variables	the descriptions of variables
w	the delegate fee paid by the client to the computing party.
c	the computing party's cost for calculating the task correctly.
s_1	the client's payoff after receiving the correct calculation results.
z	TTP's fee paid by the internal participants for arbitration.
d_{12}	the deposit that the computing party is required to place at TTP in advance.
d_{21}	the deposit that client needs to prestore at the TTP, note that $d_{12} = d_{21}$.
x_1	the client's loss of credibility because of his dishonest behavior.
x_2	the loss of reputation of the computing party for his dishonesty.
h_{31}	the fee paid by the external adversary to corrupt the client.
h_{32}	the fee paid by the external adversary to corrupt the computing party.
s_3	the payoff that the external adversary successfully sabotages the transaction.
f_{31}	the fine that the corrupted client pays to the adversary for his betrayal.
f_{32}	the fine that the corrupted computing party pays to the adversary for his betrayal.
l	the sanction imposed by law on external adversary.

Note the loss of credibility x_i represents the negative impact on the possible future transactions because of the party's dishonesty, and x_i is an increasing function of the delegation cost w, where $i = 1, 2$.

The Delegation Contract. The delegation contract shown in Fig. 2 is signed between the client and the computing party in the delegation phase, which means that the client chooses to delegate a computing task to others and the computing party also agrees to accept it. Briefly speaking, the client should pay the delegation fee w to the computing party in return for completing the calculation task correctly. In addition, the client and the computing party must pay a high amount of deposit in advance, and if one party shows dishonesty, the deposit will be acquired by the other party, so as to encourage both parties to carry out the contract honestly. The details of the contract are presented below.

(1) The contract is signed by the client and the computing party at the delegation stage. After signing the contract, the client must pre-store the delegation fee w, deposit d_{21} and arbitration fee z at the TTP; as well as the computing party should pre-store the deposit d_{12}, the arbitration fee z at the TTP.

(2) The client sends the computing task f and input x^* to the computing party, where x^* is the result of the homomorphic encryption of the real input by the client, in order to ensure the privacy of the input.

(3) The computing party is required to divide the task into several sub-tasks logically and then performs the calculations. After computing commitments to the sub-results respectively, the information of the commitments and the decommitments are sent to the TTP, and the description of sub-task division is sent to the client.

(4) The client judges whether the received description is logical or not and interacts with TTP to verify the result of a sub-task is right or not, TTP also attempts to open the commitments by the decommitment information to judge whether the computing party is honest.

(5) The honest party can send a request for arbitration to TTP if there is a dispute between the internal participants. After receiving the arbitration request, TTP will require both parties to submit relevant information and recalculate it to verify who is dishonest, and then settle the dispute as below:

- If the client is dishonest, then TTP will take the arbitration fee z paid by the client, and return the deposit d_{12} and d_{21}, arbitration fee z to the computing party, and return the delegation fee w to the client.
- If the computing party is considered dishonest, TTP will take the arbitration fee z paid by the computing party, and refund the deposit d_{12} and d_{21}, arbitration fee z and delegation fee w the client.

(6) If both of the internal participants are honest during the execution of the agreement, TTP will charge half of the arbitration fee from each of the them, return the deposit d_{21} and arbitration fee $\frac{1}{2}z$ to the client, and refund the deposit d_{12}, arbitration fee $\frac{1}{2}z$ and delegation fee w to the computing party. Meanwhile, TTP sends the encrypted result by opening the commitment to the client, thus the client can obtain the final result by performs homomorphic decryption.

Delegation Contract

- The client and the computing party are required to pre-store the fee (w, z, d_{21}) and (z, d_{12}) at TTP, respectively.
- The client sends the processed input x^* and the objective function f to the computing party.
- After completing the calculation, the computing party submits the verification information to the client, and the commitment and decommitment information about the calculated result to TTP. Then the client interacts with TTP for verification.
- If there is a dispute between the client and the computing party, TTP will perform the arbitration as follows.
  - If the client is dishonest, TTP will take the fee z from he and refund the fee w to the client, as well as refund the fee (z, d_{12}, d_{21}) to the computing party.
  - If the computing party is dishonest, TTP will take the fee z from he and refund the fee (w, z, d_{21}, d_{12}) to the client.
- If both of them are honest, TTP will refund the fee $(\frac{1}{2}z, d_{21})$ to the client and the fee $(w, \frac{1}{2}z, d_{12})$ to the computing party, then he will charge the half arbitration fee from the two internal participants respectively. Meanwhile, the client obtains the processed result from TTP.

Fig. 2. Delegation contract

The Corruption Contract. During the corruption phase, if the external adversary chooses to corrupt an internal participant, and the participant also accepts it, the parties will sign a corruption contract. The external adversary needs to pay the corruption fee to the corrupted internal participant to achieve the purpose of breaching the agreement. In order to ensure the interests of the external adversary, it is stipulated that the corrupted party is required to pay a fine to the external adversary if he betrays the corruption contract. The detailed description of the contract is as shown in Fig. 3.

Corruption Contract

- If the external adversary signs a corruption contract with the client, the external adversary must pay the corruption fee h_{31} to the client, and requires the client not to pass the verification at the verification stage to break the agreement. If the client breaches the corruption contract, a fine f_{31} must be paid to the external adversary.
- If the external adversary signs a corruption contract with the computing party, the external adversary shall pay the computing party a corruption fee h_{32} and ask the computing party to submit a wrong calculation result. If the computing party violates the contract, he is required to pay a fine f_{32} to the external adversary.

Fig. 3. Corruption contract

The Betrayal Contract. The betrayal contract shown in Fig. 4 is designed to motivate the internal participants to behave honestly. The corrupted internal participant called the traitor will sign a betrayal contract with another honest party if he wants to violate the corruption contract. The traitor can actively inform the honest party of the fact that he is in a collusion with the external adversary, meanwhile, stay honest during the execution of the agreement. Ultimately, the two parties jointly expose the evils of the external adversary, so that the external adversary will be disciplined by the law.

If the traitor is the client, the betrayal contract will be signed during the verification phase; while if the traitor is the computing party, the betrayal contract will be signed during the ccomputation phase. Once the specified time is exceeded, the betrayal contract is deemed invalid.

The existence of a betrayal contract guarantees that betrayal is risk-free. Firstly, the traitor will be exempt from the high deposit because of the honest implementation of the agreement. Secondly, the law will treat the fine in the corruption contract as invalid, but only order the refund of the corruption fee because the corruption contract is an invalid contract (see Article 52 of the Contract Law of the Peoples Republic of China) once the traitor and another honest party jointly expose the criminal of the external adversary. Moreover, since the traitor eventually chooses to be honest, avoiding the serious loss of credibility caused by the exposure of dishonest behavior. The betrayal contract sabotages a corruption contract by encouraging treachery. The fear to betrayal creates distrust between the external adversary and the corrupted, thus preventing the collusion between the external and the internal.

Betrayal Contract

- If the client is a traitor, the fine f_{31} stipulated in the corruption contract will be invalid, and the client only needs to refund the corruption fee h_{31}.
- Or If the computing party is a traitor, the fine f_{32} stipulated in the corruption contract will be invalid, and the computing party will refund the corruption fee h_{31}.
- The external adversary will be disciplined by the law which denoted as l once his malicious corruption is exposed by the traitor.

Fig. 4. Betrayal contract

4 Game Model Analysis

4.1 Game Model

The complete game shown in Fig. 5 is specifically modeled as a three-party game between the client P_1, the computing party P_2 and the external adversary P_3, and the set of participants is i.e., $P = \{P_1, P_2, P_3\}$. It's remarkable that because of our assumption, the corrupted knows the strategy of the other party is honest, meanwhile, even though the uncorrupted can not know whether the other party is honest or not, his only choice is honest, hence the game model can still be regarded as a game with perfect information.

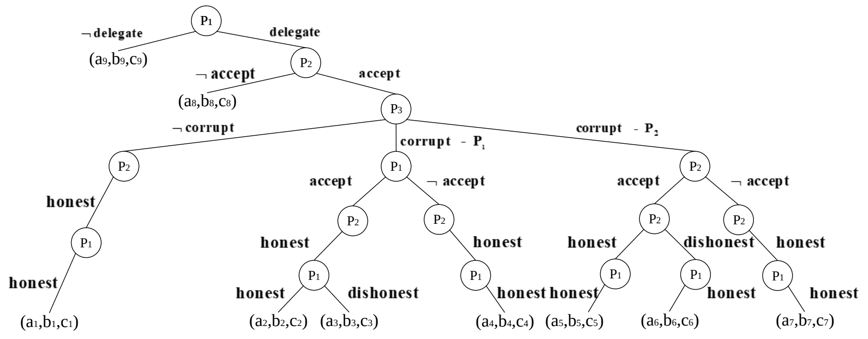

Fig. 5. Game model

The game process of rational delegated computation is described as follows:
(1) Delegation phase

Step 1: P_1 can choose to delegate a computing task to others or not, that is, the strategy set at this time is $S_{1,1} = \{delegate, \neg delegate\}$. If he chooses the strategy of "delegate", go to Step 2. Otherwise, the protocol ends.

Step 2: P_2 can choose to accept the computing task or not, and his strategy set at this time is $S_{1,2} = \{accept, \neg accept\}$. If he chooses to accept it, both of them sign the delegation contract and proceed to the next stage. Otherwise, the protocol ends.

(2) Corruption phase

Step 1: The alternative strategies for P_3 are $S_{2,3} = \{\neg corrupt, corrupt - P_1, corrupt - P_2\}$. When P_3 chooses not to corrupt any internal participants, the participants honestly execute the protocol and moves on to the next phase. When choosing corrupt P_1 or P_2, move on to the next step.

Step 2: The strategy set of P_1 or P_2 at this time can be expressed as $S_{2,1} = S_{2,2} = \{accept, \neg accept\}$, the corrupted internal participant can choose whether to accept the corruption or not. If the internal participant accepts it, then he will sign the corruption contract with the P_3 and enter the next phase.

(3) Computation phase

If P_2 is not corrupted, then the delegated task will be calculated honestly by P_2 whose strategy set is $S_{3,2} = \{honest\}$.

But the strategy set of the corrupted P_2 is $S_{3,2} = \{honest, dishonest\}$. P_2 can choose the strategy of "dishonest", which means submitting the wrong calculation results so as to perform the corruption contract but abandon the delegation contract. In addition, P_2 can also choose to be honest, i.e., to execute the delegation contract rather than the corruption contract, which indicates that the betrayal contract has been signed between P_2 and P_1.

(4) Verification phase

The the strategy set of the uncorrupted P_1 is $S_{4,1} = \{honest\}$, that is, P_1 will only perform the verification on the calculation results honestly. The verification result will be "TRUE" if P_2 executes calculation honestly, otherwise, will be "FALSE".

The the strategy set of the uncorrupted P_1 is $S_{4,1} = \{honest, dishonest\}$. When P_1 chooses to behaves honestly, namely signing a betrayal contract, which the situation is the same as before. When P_1 chooses to be dishonest, the result of the verification is always "FALSE" regardless of any submitted calculations.

The Utilities of Players. In the game model, the utilities of players can be defined as follows:

For the client, If he successfully delegates the computing task and the protocol is correctly executed, which is defined as an event E_D. When the event E_D occurs, he should pay the delegation fee w to the computing party, the arbitration fee $\frac{1}{2}z$ to the TTP, and the payoff s_1 can be obtained. Denote the client is corrupted by $E_{h_{31}}$, when the event $E_{h_{31}}$ occurs, the corruption fee h_{31} will be obtained from the adversary. The client's breach of the corruption contract is recorded as an event $E_{f_{31}}$, and the client is required to pay a fine f_{31} when the event $E_{f_{31}}$ occurs. The dishonesty of the computing party during the execution of the agreement is defined as an event $E_{d_{12}}$. When the event $E_{d_{12}}$ occurs, the deposit d_{12} can be received from the computing party. The client's dishonesty is defined as an event $E_{d_{21}}$. When the event $E_{d_{21}}$ occurs, the client should pay the deposit d_{21} to the computing party, the arbitration fee z to TTP, and suffer a loss of reputation x_1. It is defined as an event E_{b_1} that the corrupted client becomes a traitor. When the event E_{b_1} occurs, the corruption fee h_{31} shall be returned to the external adversary, and the fine f_{31} is regarded as deemed invalid.

Let $\overrightarrow{\gamma}_1 = (-w, d_{12}, -d_{21}, s_1, h_{31}, -f_{31}, -z, -h_{31}, f_{31})$ denote a payoff vector of the client, and the expected payoff of the client is computed according to our definitions as:

$$\begin{aligned} U_1 =& (s_1 - w - \frac{1}{2}z)Pr[E_D] + d_{12}Pr[E_{d_{12}}] - (d_{21} + x_1 + z)Pr[E_{d_{21}}] \\ &+ h_{31}Pr[E_{h_{31}}] - f_{31}Pr[E_{f_{31}}] + (f_{31} - h_{31})Pr[E_{b_1}] \end{aligned} \tag{6}$$

where $Pr[E_{d_{12}}E_{d_{21}}] = 0$, $Pr[E_D E_{d_{12}}] = 0$, $Pr[E_D E_{d_{21}}] = 0$, $Pr[E_{h_{31}}E_{d_{12}}] = 0$, $Pr[E_{b_1}E_{d_{21}}] = 0$, $Pr[E_{d_{21}}E_{f_{31}}] = 0$, $Pr[E_{b_1}|E_{h_{31}}E_{f_{31}}] = 1$. $E_{d_{21}}$ or $E_{f_{31}}$ may happen only if $E_{h_{31}}$ happens.

For the computing party, when the event E_D occurs, the delegated fee can be obtained from the client, while spending the cost c of honest calculation, and the arbitration fee $\frac{1}{2}z$ shall be paid to the TTP. Record the computing party is corrupted successfully as $E_{h_{32}}$, when the event $E_{h_{32}}$ occurs, the computing party will get the expense h_{32} from the adversary. The corrupted computing party deviates from the adversary is defined as an event $E_{f_{32}}$, and he will pay a fine f_{32} to the adversary when the event $E_{f_{32}}$ occurs. When the event $E_{d_{12}}$ occurs, he should pay deposit d_{12} to the client and the arbitration fee z to TTP, respectively, as well as a loss of reputation x_2 is incurred. When the event $E_{d_{21}}$ occurs, the deposit d_{21} is received from the computing party, and the cost c of calculating honestly may be lost. The corrupted computing party becomes a traitor, which is defined as an event E_{b_2}. When the event E_{b_2} occurs, the corruption fee h_{32} is refunded to the external adversary, as well as the fine f_{32} is deemed invalid.

Let $\overrightarrow{\gamma}_2 = (w, -d_{12}, d_{21}, -c, h_{32}, -f_{32}, -z, -h_{32}, f_{32})$ denote a payoff vector of the client, and the expected payoff of the client is computed according to our definitions as:

$$\begin{aligned} U_2 =& (w - c - \frac{1}{2}z)Pr[E_D] - (d_{12} + x_2 + z)Pr[E_{d_{12}}] + (d_{21} - c)Pr[E_{d_{21}}] \\ &+ h_{32}Pr[E_{h_{32}}] - f_{32}Pr[E_{f_{32}}] + (f_{32} - h_{32})Pr[E_{b_2}] \end{aligned} \tag{7}$$

where $Pr[E_{d_{12}}E_{d_{21}}] = 0$, $Pr[E_D E_{d_{12}}] = 0$, $Pr[E_D E_{d_{21}}] = 0$, $Pr[E_{h_{32}}E_{d_{21}}] = 0$, $Pr[E_{b_2}E_{d_{12}}] = 0$, $Pr[E_{d_{12}}E_{f_{32}}] = 0$, $Pr[E_{b_2}|E_{h_{32}}E_{f_{32}}] = 1$. $E_{d_{12}}$ or $E_{f_{32}}$ may happen only if $E_{h_{32}}$ happens.

For the adversary, when the event $E_{h_{31}}$ occurs, the expense of corrupting the client h_{31} shall be paid; and he will receive a fine f_{31} from the client when the event $E_{f_{31}}$ occurs. When the event $E_{h_{32}}$ occurs, the expense of corrupting the computing party h_{31} is required to pay; but when the event $E_{f_{32}}$ occurs, he will obtain a fine f_{32} from the computing party. Record the successful breach of the agreement as E_{s_3}, he will gain the payoff s_3 when the event E_{s_3} occurs. Once E_{b_1} or E_{b_2} happens, which means the adversary will be subject to legal sanctions denoted by l, as well as the corresponding corruption fee is refunded and the specified fine is viewed as invalid.

Therefore, let $\overrightarrow{\gamma}_3 = (-h_{31}, -h_{32}, -l, f_{31}, f_{32}, s_3, h_{31}, h_{32}, -f_{31}, -f_{32})$ denote a payoff vector of the external adversary. The expected payoff of the external adversary is computed according to our definitions as:

$$\begin{aligned} U_3 = &- h_{31}Pr[E_{h_{31}}] - h_{32}Pr[E_{h_{32}}] + f_{31}Pr[E_{f_{31}}] + f_{32}Pr[E_{f_{32}}] \\ &+ s_3Pr[E_{s_3}] - (l + f_{31} - h_{31})Pr[E_{b_1}] - (l + f_{32} - h_{32})Pr[E_{b_2}] \end{aligned} \tag{8}$$

where $Pr[E_{h_{31}}E_{h_{32}}] = 0$, $Pr[E_{f_{31}}E_{f_{32}}] = 0$, $Pr[E_{b_1}E_{b_2}] = 0$, $Pr[E_{h_{31}}E_{b_2}] = 0$, $Pr[E_{h_{32}}E_{b_1}] = 0$, $Pr[E_{s_3}E_{b_1}] = 0$, $Pr[E_{s_3}E_{b_2}] = 0$, $Pr[E_{b_1}|E_{h_{31}}E_{f_{31}}] = 1$, $Pr[E_{b_2}|E_{h_{32}}E_{f_{32}}] = 1$, $Pr[E_{h_{31}}|E_{f_{31}}] = 1$, $Pr[E_{h_{32}}|E_{f_{32}}] = 1$. $E_{f_{31}}$ may occur only if $E_{h_{31}}$ occurs. $E_{f_{32}}$ may occur only if $E_{h_{32}}$ occurs. E_{s_3} may occur only if $E_{h_{31}}$ or $E_{h_{32}}$ occurs.

Let a_i, b_i, c_i denote the expected utilities of the client, the computing party and the external adversary respectively when choosing the i-th strategy set. According to the above definition, the utilities of players in various strategy sets under the game model are as shown in Table 2.

Table 2. The utilities of players

	a_i	b_i	c_i
$i = 1$	$s_1 - w - \frac{1}{2}z$	$w - c - \frac{1}{2}z$	0
$i = 2$	$s_1 - w - \frac{1}{2}z$	$w - c - \frac{1}{2}z$	$-l$
$i = 3$	$h_{31} - d_{21} - x_1 - z$	$d_{21} - c$	$s_3 - h_{31}$
$i = 4$	$s_1 - w - \frac{1}{2}z$	$w - c - \frac{1}{2}z$	0
$i = 5$	$s_1 - w - \frac{1}{2}z$	$w - c - \frac{1}{2}z$	$-l$
$i = 6$	d_{21}	$h_{32} - d_{12} - x_2 - z$	$s_3 - h_{32}$
$i = 7$	$s_1 - w - \frac{1}{2}z$	$w - c - \frac{1}{2}z$	0
$i = 8$	0	0	0
$i = 9$	0	0	0

4.2 The Attack Game

With reference to [1], this game model is transformed into the attack game $\mathcal{G}_{\mathcal{M}}$. Since both the client and the computing party belong to the internal participants who execute the protocol in the game model, while the objective of the external adversary is to ultimately break the protocol by corrupting an the internal participant. The model can be regarded as a two-party attack game between the external adversary and the internal players.

The two internal participants choose a polynomial time calculable protocol in the delegation stage. Since the game is a dynamic as well as with perfect information, the internal participants send the description $\Pi_1(\mathrm{P}_1, \mathrm{P}_2) \subseteq \{0,1\}^*$ of the protocol to the external adversary P_3 in a specific way. Once receiving the message, an ITM $\mathcal{A}$ with a polynomial time is selected as the attack strategy to destroy the protocol and sent to the internal participants. The internal participants then choose the protocol $\Pi_2(\mathrm{P}_1, \mathrm{P}_2) \subseteq \{0,1\}^*$ according to the received attack strategy $\mathcal{A}$.

Let $(\Pi_1, \mathcal{A}, \Pi_2)$ denote the strategy profile of the attack game $\mathcal{G}_{\mathcal{M}}$, where $\mathcal{A}$ is a function that maps an efficient protocol to a corresponding attack strategy, and Π_2 is a function that maps an attack strategy to another valid protocol. The protocol in the internal participants can be expressed as $\Pi \subseteq \{\Pi_1, \Pi_2\}$. In addition, we use $u_1(\cdot)$, $u_2(\cdot)$ and $u_3(\cdot)$ to represent the utilities of the client, the computing party and the external adversary, respectively.

Definition 4. *Let $\mathcal{G}_{\mathcal{M}}$ be an attack game. A strategy profile $(\Pi_1, \mathrm{A}, \Pi_2)$ is an ϵ-subgame perfect equilibrium in $\mathcal{G}_{\mathcal{M}}$ if the following conditions hold:*
(a) for any $\Pi_1{}' \in \mathrm{ITM^n}$

$$u_1(\Pi_1{}', \mathrm{A}(\Pi_1{}'), \Pi_2(\mathrm{A}(\Pi_1{}'))) \leq u_1(\Pi_1, \mathrm{A}(\Pi_1), \Pi_2(\mathrm{A}(\Pi_1))) + \epsilon.$$
$$u_2(\Pi_1{}', \mathrm{A}(\Pi_1{}'), \Pi_2(\mathrm{A}(\Pi_1{}'))) \leq u_2(\Pi_1, \mathrm{A}(\Pi_1), \Pi_2(\mathrm{A}(\Pi_1))) + \epsilon.$$

(b) for any $\mathrm{A}' \in \mathrm{ITM^n}$

$$u_3(\Pi_1, \mathrm{A}'(\Pi_1), \Pi_2(\mathrm{A}'(\Pi_1))) \leq u_3(\Pi_1, \mathrm{A}(\Pi_1), \Pi_2(\mathrm{A}(\Pi_1))) + \epsilon.$$

(c) for any $\Pi_2{}' \in \mathrm{ITM^n}$

$$u_1(\Pi_1, \mathrm{A}(\Pi_1), \Pi_2{}'(\mathrm{A}(\Pi_1))) \leq u_1(\Pi_1, \mathrm{A}(\Pi_1), \Pi_2(\mathrm{A}(\Pi_1))) + \epsilon.$$
$$u_2(\Pi_1, \mathrm{A}(\Pi_1), \Pi_2{}'(\mathrm{A}(\Pi_1))) \leq u_2(\Pi_1, \mathrm{A}(\Pi_1), \Pi_2(\mathrm{A}(\Pi_1))) + \epsilon.$$

4.3 The Game Analysis

Theorem 2. *Let $\mathcal{G}_{\mathcal{M}}$ be an attack game, there is a Nash equilibrium in $\mathcal{G}_{\mathcal{M}}$ if and only if $s_1 - w - \frac{1}{2}z > h_{31} - d_{21} - x_1 - z$ and $w - c - \frac{1}{2}z > h_{32} - d_{12} - x_2 - z$.*

Proof. Firstly, the client decides whether to delegate the computing task to the computing party. Because of the limitation of the client's computing power, it is impossible to complete the computing task alone. If it is not delegated, which will result in his own payoff of 0, such as strategy set $S_9 = \{\neg \mathit{delegate}\}$. Therefore, the client will choose the strategy of "delegate".

Then it is up to the computing party to choose whether to accept the task or not, and if he chooses to not accept it, the computing party's revenue will be 0, such as the strategy set $S_8 = \{\mathit{delegate}, \neg \mathit{accept}\}$. Therefore, the computing party must choose the strategy of "accept".

It is turn for the external adversary to choose a strategy. If one of the internal players is chosen to corrupt, the internal player can decide whether to accept it or not. If the internal participant chooses not to accept, the internal participant will carry out the agreement honestly, and the external adversary gains payoff 0, such as strategy set $S_4 = \{delegate, accept, corrupt - P_1, \neg accept, honest, honest\}$ and $S_7 = \{delegate, accept, corrupt - P_2, \neg accept, honest, honest\}$. But once the internal participant chooses to accept it at this time, and at the later stage chooses to be honest, that is to say, signing a betrayal contract with another honest internal participant, which will not lead to any negative impact on the betrayer. And the utility of the traitor is greater than that of helping the external attacker and same as the situation that choosing to not accept, so the corrupted is more inclined to choose the strategy of "honest" in the subsequent implementation of the agreement, even if he signs a corruption contract with the external attacker. But this is a disastrous choice for the adversary, who will be sanctioned by the law because of the exposure of the malicious corruption, with the lowest utility $-l$ ultimately, such as strategy set $S_2 = \{delegate, accept, corrupt - P_1, accept, honest, honest\}$ and $S_5 = \{delegate, accept, corrupt - P_2, accept, honest, honest\}$. Since the existence of a betrayal contract, which will cause mistrust between the external adversary and the corrupted, the best strategy of the external adversary is to choose not corrupt any parties, so there is a Nash equilibrium $S^* = S_1 = \{delegate, accept, \neg corrupt, honest, honest\}$, i.e., for all $i = 1, 2, 3$,

$$u_i(s_i^*, s_{-i}) \geq u_i(s_i', s_{-i}) \quad \forall s_i' \neq s_i^*$$

5 Ideal Functionality $\mathcal{F}_{RDC}$ for Rational Delegation of Computation

In this section, we formalize the notion of a secure rational delegated computation by utilizing a functionality $\mathcal{F}_{RDC}$ that demonstrates the secure requirements of rational delegation of computation. First of all, we express the basic security requirements of traditional delegated computation, which includes four algorithms e.g., *Setup*, *Delegate*, *Compute* and $Verify$. Then we discuss the situation when there is an adversary who attempts to take an attack, namely corrupting an internal participant. Furthermore, we explore the case of rational delegation computing with incentive mechanisms. The functionality $\mathcal{F}_{RDC}$ that captures secure rational delegated computation is described in Fig. 6.

6 Protocol π_{RDC}

The participants of the protocol include the client P_1, the computing party P_2, the external adversary $\mathcal{A}$, and the trusted third party TTP with computing power. The protocol contains five algorithms such as *Delegate*, *Corrupt*, *Compute*, $Verify$, *Expose*, which is described as follows.

(1) Delegation phase

P_1 chooses whether to delegate the computing task to others, and if not, the protocol ends.

Functionality $\mathcal{F}_{RDC}$

$\mathcal{F}_{RDC}$ handles as follows, where P_1 is the client, P_2 is the computing party and $\mathcal{S}$ is an ideal adversary, respectively.

Setup: Execute $Setup(1^k) \rightarrow ((pk, sk), param = (g, h))$, where (pk, sk) is a pair of the homomorphic encryption keys, and $param = (g, h)$ contains public parameters of commitment.

Delegate: Upon receiving $(Delegate, sid, task\_sid(x, f), P_1, P_2)$ from P_1, perform homomorphic encryption on input x using the public key pk, i.e., $E_{pk}(x) = x^*$. Record $(Delegate, sid, task\_sid(P_1, P_2, x, f))$, and send $(Delegate, sid, task\_sid(x^*, f), P_1)$ to P_2.

Compute: Perform calculation on task and obtain $y^* = f(x^*)$. Then use Pedersen Commitment to commit to the result y^*, i.e., Com_{y^*}, and record $(Compute, sid, y^*, Com_{y^*}, Dec)$. Send $(Compute, sid, y^*, Com_{y^*})$ to P_1.

Verify: Send $(Verify, sid, Ver = TRUE)$ to P_1 and P_2. Meanwhile, perform homomorphic decryption on the result y^* of the computing task using the private key sk, i.e., $D_{sk}(y^*) = y$, then send $(Verify, sid, y)$ to P_1.

Corrupt: Upon receiving $(Corrupt, sid, P_1)$ from $\mathcal{S}$, record $(sid, Corrupted(P_1))$ and send $(Corrupt, sid, \perp)$ to $\mathcal{S}$. Upon receiving $(Corrupt, sid, P_2)$ from $\mathcal{S}$, record $(sid, Corrupted(P_2))$ and send $(Corrupt, sid, \perp)$ to $\mathcal{S}$.

Incentive mechanisms:

- If there is a record $(Delegate, sid, task\_sid(P_1, P_2, x, f))$, then the message $(Delegate, sid, Deposit(w, d_{21}, z), P_1))$ from P_1 and $(Delegate, sid, Deposit-(d_{12}, z), P_2))$ from P_2 will be received. And then send $(Verify, sid, Refund-(d_{21}, \frac{1}{2}z))$ to P_1, simultaneously, send $(Verify, sid, Refund(d_{12}, \frac{1}{2}z, w))$ to P_2.
- If there is a record $(sid, Corrupted(P_1))$, it requires $\mathcal{S}$ to send $(Corrupt, sid,$ $Deposit(h_{31}))$ and P_1 to send $(Corrupt, sid, Deposit(f_{31}), P_1)$. Afterwards, send $(Expose, sid, Refund(f_{31}))$ to P_1, send $(Expose, sid, Refund(h_{31}))$ to $\mathcal{S}$, and require $\mathcal{S}$ to send $(Expose, sid, Law(l))$.
- If there is a record $(sid, Corrupted(P_2))$, it requires $\mathcal{S}$ to send $(Corrupt, sid,$ $(h_{32}))$ and P_2 to send $(Corrupt, sid, Deposit(f_{32}), P_2)$. Then send $(Expose, sid,$ $Refund(f_{32}))$ to P_2, send $(Expose, sid, Refund(h_{32}))$ to $\mathcal{S}$, and require $\mathcal{S}$ to send $(Expose, sid, Law(l))$.

Fig. 6. Functionality $\mathcal{F}_{RDC}$.

Otherwise, P_2 will choose whether to accept the task or not, and if not, the protocol will end.

Otherwise, P_1 and P_2 sign the delegation contract, input $(Delegate, sid, P_1, P_2, TTP)$, the algorithm *Delegate* is executed as following by TTP, P_1 and P_2 jointly.

- P_1 places the delegated fee w, the deposit d_{21}, and the arbitration fee z at TTP. Also, P_2 must prestore the deposit d_{12} and the arbitration fee z at TTP.
- P_1 and P_2 jointly execute the algorithm *Setup*: input (1^k), where k is the security parameter. Generate a public-private key pair (pk_1, sk_1) of P_1, where pk_1 is used to implement homomorphic encryption of the input and verify sub-results, and sk_1 is used to implement homomorphic decryption of the calculation result, meanwhile, pk_1 is public, and sk_1 is preserved by P_1. In addition, output the public parameters $param = (g, h)$ that P_2 uses for commitment.
- P_1 uses pk_1 to conduct homomorphic encryption on the input x, and get $E_{pk_1}(x) = x^*$. P_1 sends $(Delegate, sid, x^*, f, P_1, P_2)$ to P_2 through the secret channel, where f is the computing task and enters the next stage.

(2) Corruption phase

When inputing $(Corrupt, sid, P_1, P_2, \mathcal{A})$, $\mathcal{A}$ runs the algorithm *Corrupt* with the internal participants P_1 and P_2.

- $\mathcal{A}$ chooses whether to corrupt one of the internal participants or not, and if not, goes directly to the next stage.
- If $\mathcal{A}$ chooses to corrupt P_1, the corruption information is passed to P_1 through the secret channel. If P_1 accepts this corruption, he will send $(Corrupt, sid, \perp)$ to $\mathcal{A}$ secretly. And a corruption contract is signed between P_1 and $\mathcal{A}$, in which $\mathcal{A}$ is required to pay the corruption fee h_{31} to P_1 as a reward for helping $\mathcal{A}$ break the agreement. Otherwise, proceed to the next stage.
- If $\mathcal{A}$ chooses to corrupt P_2, the corruption information is passed to P_2 secretly. If P_2 accepts it, he will send $(Corrupt, sid, \perp)$ to $\mathcal{A}$ in secret. Then $\mathcal{A}$ will sign a corruption contract with P_2, at the same time, pay the corruption fee h_{32} to P_2, which is a remuneration to breach the protocol. Otherwise, go to the next stage.

(3) Computation phase

When inputing $(Compute, sid, P_2)$, P_2 runs the algorithm *Compute*.

If P_2 chooses honesty (if P_2 has signed a corruption contract with $\mathcal{A}$ at the corruption stage, then signs a betrayal contract with P_1 at this time, which is called a traitor), then divides the computing task f into n sub-tasks according to the computational logic, that is, $f = \{f_i | i = 1, \ldots, n\}$. P_2 performs the calculation and gets the results of each sub-task

$$y_1^* = f_1(x^*), \ldots, y_n^* = f_n(x^*), \quad y^* = (y_1^*, \ldots, y_n^*) \tag{9}$$

Then P_2 uses Pedersen Commitment to commit to each sub-result respectively, i.e., $Com_{y^*} = (Com_{y_1^*}, \ldots, Com_{y_n^*})$, and sends $(Compute, sid, Com_{y^*} =$

$(Com_{y_1^*}, \ldots, Com_{y_n^*}), Dec = (Dec_1, \ldots, Dec_n), P_2, TTP)$ to TTP, where $Dec = (Dec_1, \ldots, Dec_n)$ indicates the corresponding decommitment information. Send $(Compute, sid, Information(f, Com_{y^*}), P_2, P_1)$ to P_1, which contains a detailed description of the computing task divided into n sub-tasks and the meaning of each commitment vector corresponding to each sub-task.

If P_2 chooses to be dishonest, P_2 may submit the incorrect result in the following ways.

- P_2 erroneously divides the computational task into several sub-tasks, and P_1 will receive an illogical $Informatin(f, Com_{y^*})$.
- Or P_2 divides the computing task according to logic correctly, but excludes the results of the operator task. In order to save computational resources, P_2 randomly selects the result and submits it to TTP, or even submits a wrong commitment information instead of making a commitment.

(4) Verification phase

When inputing $(Verify, sid, P_1, P_2, TTP)$, P_1, P_2 and TTP run the algorithm *Compute* together (if P_1 has signed a corruption contract with $\mathcal{A}$ during the corruption phase and chooses to be honest at this time, P_1 will sign a betrayal contract with P_2 before the start of this phase and called a traitor).

1) P_1 judges whether the received $Informatin(f, Com_{y^*})$ is reasonable according to the calculation task f_i.

- If it is considered reasonable, P_1 sends $(Verify, sid, Ver(Information) = TRUE, P_1, P_2)$ to P_2.
- If it is deemed to be unreasonable, P_1 sends $(Verify, sid, Ver(Information) = FALSE, P_1, P_2)$ to P_2 in order to inform that the verification result is incorrect and refuse to accept it. Then send $(Verify, sid, Ver(Information) = FALSE, Information(f, Com_{y^*}), f, P_1, TTP)$ to TTP for arbitration.
- If a malicious claim is made by P_1 that the validation result is incorrect, upon receipt of the validation result of P_1, P_2 sends $(Verify, sid, Ver(Inform-ation) = FALSE, Information(f, Com_{y^*}), f, P_2, TTP)$ to TTP for an arbitration request.
- If the TTP receives a request for arbitration from P_1 or P_2, the integrity of P_1 and P_2 will be determined based on whether the sub-task division of P_2 is correct or not according to receiving the calculated task f and $Informatin(f, Com_{y^*})$.

2) TTP tries to open the commitment Com_{y^*} with the decryption information Dec received from P_2, and if it cannot be opened, P_2 is deemed dishonest. If it can be opened, go to the next step.

3) P_1 performs sub-task verification as following.

- P_1 sends $(Verify, sid, Ver(f_i), P_1, TTP)$ to TTP, which indicates that P_1 wants to verify the calculation result of the sub-task f_i.
- TTP obtains the calculation result of P_2 by opening the commitment, then sends $(Verify, sid, Ver(f_i) = y_i^*, TTP, P_1)$ to P_1 immediately.

- P_1 calculates the corresponding sub-task according to f_i, obtains $y_{1i} = f_i(x)$, and then uses his own public key to homomorphically encrypt the calculation result to obtain $E_{pk_1}(y_{1i}) = y^*_{1i}$, and compares whether y^*_i and y^*_{1i} are equal. If they are equal, $(Verify, sid, Ver(f_i) = TRUE, P_1, P_2)$ will be sent to P_2 and $(Verify, sid, Ver(f_i) = TRUE, P_1, TTP)$ will be sent to TTP respectively, indicating that the verification is successful and the calculation result will be received. If they are not equal, $(Verify, sid, Ver(f_i) = FALSE, P_1, P_2)$ will be sent to P_2, and sends $(Verify, sid, Ver(f_i) = FALSE, Information(f, Com_{y^*}), x^*, f, f_i, y^*_{1i}, y^*_i, P_1, TTP)$ to TTP.
- If the correct result is submitted by P_2, while P_1 is intentionally rejected as a error result, upon receiving the incorrect validation message from P_1, P_2 sends $(Verify, sid, Ver(f_i) = FALSE, Information(f, Com_{y^*}), x^*, f, f_i, y^*_i, P_2, TTP)$ to TTP.

4) TTP executes the arbitration as following.

- If TTP receives a message of the verification success from P1 instead of receiving the arbitration request, which indicates that the internal participants are honest, skip the following steps and go straight to 5).
- If TTP receives a request for arbitration from P_1, then calculates the sub-task f_i based on the information received, obtains $y^*_{ci} = f_i(x^*)$, and compares y^*_{ci} with y^*_{1i} and y^*_i. If $y^*_{ci} \neq y^*_i$, P_2 is regarded as dishonest; if $y^*_{ci} \neq y^*_{1i}$, P_1 is judged to be dishonest.
- If TTP receives a request for arbitration from P_2, TTP first requests P_1 to submit y^*_{1i}, and if P_1 fails to submit on time, P_1 is dishonesty can be judged. Otherwise, TTP will receive y^*_{1i} from P_1, and calculate according to the above steps. TTP computes $y^*_{ci} = f_i(x^*)$, then y^*_{ci} will be compared with y^*_i and y^*_{1i}. If $y^*_{ci} \neq y^*_i$, P_2 is regarded as dishonest; if $y^*_{ci} \neq y^*_{1i}$, P_1 is deemed to be dishonest.

5) TTP performs the penalty as following.

- If P_1 is dishonest, TTP will take away the arbitration fee z paid by P_1, and return the deposit d_{12} and d_{21}, the arbitration fee z to P_2, as well as refund the delegation fee w to P_1.
- If P_2 is dishonest, TTP will take away the arbitration fee z paid by P_2, and return the deposit d_{12} and d_{21}, the arbitration fee z and the delegation fee w to P_1.
- If both P_1 and P_2 are honest in the execution of the agreement, TTP will charge half of the arbitration fee from each party, refund the deposit d_{21} and the arbitration fee $\frac{1}{2}z$ to P_1, and refund the deposit d_{12}, the arbitration fee $\frac{1}{2}z$ and the delegation fee w to P_2. Meanwhile, TTP sends the result $y^* = (y^*_1, \ldots, y^*_n)$ of task calculation to P_1, and then P_1 performs homomorphic decryption with the private key sk_1 to get the final result, i.e., $D_{sk_1}(y^*) = y$.

(5) Exposure phrase
When inputing $(Expose, sid, P_1, P_2, \mathcal{A})$, P_1, P_2 and $\mathcal{A}$ run the algorithm $Expose$ together.

- If P_2 is a traitor who signs a betrayal contract during the computation phase, at this time, P_1 and P_2 jointly will reveal the crime of $\mathcal{A}$. Because the corruption contract is invalid, the fine f_{32} in the contract is regarded as invalid, but requires refund of the corruption fee h_{32} from P_2, and $\mathcal{A}$ will be subject to legal sanctions l.
- If P_1 is a traitor who signs a betrayal contract during the verification phase, meanwhile, P_1 and P_2 jointly will expose the evil corruption of $\mathcal{A}$. Because of the invalidity of the corruption contract, the fine f_{31} in the contract is regarded as invalid, but requires refund of the corruption fee h_{31} from P_1, and $\mathcal{A}$ will be subject to legal sanctions l.

7 Protocol Analysis

7.1 Security Proof

We construct a simulator to analyze the security of π_{RDC} in the UC framework so that any environment has a negligible chance to distinguish between an ideal world execution with the simulator and $\mathcal{F}_{RDC}$ and a real world execution of an adversary with π_{RDC}. The security of π_{RDC} is formally captured by the following theorem:

Theorem 3. *The protocol π_{RDC} securely realizes the ideal functionality $\mathcal{F}_{RDC}$. I.e., for every static malicious adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ such that for all environment $\mathcal{Z}$*

$$EXEC_{\pi_{RDC},\mathcal{A},\mathcal{Z}} \approx EXEC_{\mathcal{F}_{RDC},\mathcal{S},\mathcal{Z}}$$

Proof. $\mathcal{S}$ runs an internal embedded copy of $\mathcal{A}$ denoted $\widetilde{\mathcal{A}}$, and reacts to the various events that happen during the execution in the following way.
Case 1: P_1 is corrupted.
(1) Delegation phase
(a) The ideal functionality sends $(Delegate, sid, Assign(task\_sid), P_1)$ to P_2 and then sends $(Delegate, sid, Assigned(task\_sid), P_2)$ to P_1.
(b) Then $\mathcal{F}_{RDC}$ will receive the message $(Delegate, sid, Deposit(w, d_{21}, z), P_1)$ from P_1 and $(Delegate, sid, Deposit(d_{12}, z), P_2)$ from P_2.
(c) Afterwards, the ideal functionality is executed as $\mathcal{F}_{RDC}.Setup$ and $\mathcal{F}_{RDC}.D-elegate$.
(2) Corruption phase
Receive $(Corrupt, sid, P_1)$ from $\widetilde{\mathcal{A}}$, and forward this message to $\mathcal{F}_{RDC}$.
(a) If $\mathcal{F}_{RDC}$ records $(sid, corrupted(P_1))$ and sends $(Corrupt, sid, \perp)$ to $\mathcal{S}$, then $\mathcal{S}$ sends $(Corrupt, sid, Deposit(h_{31}))$ to $\mathcal{F}_{RDC}$ and P_1 sends $(Corrupt, sid, Deposit(f_{31}, P_1))$ to $\mathcal{F}_{RDC}$, respectively.
(b) Otherwise, ignore this message and proceed the next phase.
(3) Computation phase
Because of P_2's honesty, $(Compute, sid, Information(f, Com_{y^*}), Com_{y^*} = (Com_{y_1^*}, \ldots, Com_{y_n^*}))$ will be received from $\mathcal{F}_{RDC}$ and forward this message to

$\widetilde{\mathcal{A}}$.
(4) Verification phase

(a) If $(Verify, sid, Ver(Information) = FALSE)$ is received from $\widetilde{\mathcal{A}}$, forward it to $\mathcal{F}_{RDC}$. Otherwise, receive $(Verify, sid, Ver(f_i))$ from $\widetilde{\mathcal{A}}$, and send $(Verify, sid, Ver(f_i), P_1, P_2)$ to $\mathcal{F}_{RDC}$ from P_1's interface. If $(Verify, sid, Ver(f_i)$ $= FALSE)$ is received from $\widetilde{\mathcal{A}}$, forward it to $\mathcal{F}_{RDC}$.

(b) If there is a record $(Verify, sid, Traitor(P_1))$, regardless of any message received from $\mathcal{S}$, $\mathcal{F}_{RDC}$ does the following:

- Send $(Verify, sid, Ver(Information) = TRUE)$ to P_1 and P_2.
- Send $(Verify, sid, Ver(Com_{y^*}) = TRUE)$ to P_1 and P_2.
- Send $(Verify, sid, Ver(f_i) = TRUE)$ to P_1 and P_2.
- Send $(Verify, sid, Refund(d_{21}, \frac{1}{2}z))$ to P_1 and $(Verify, sid, Refund(d_{12}, \frac{1}{2}z, w))$ to P_2. Meanwhile, send $(Verify, sid, y))$ to P_1.

(c) If there is no record $(Verify, sid, Traitor(P_1))$, $\mathcal{F}_{RDC}$ does as follows: upon receiving any message from $\mathcal{S}$, $\mathcal{F}_{RDC}$ will forward the messages to P_1 and P_2. In addition, the arbitration fee z paid by P_1 is taken away, and send $(Verify, sid, Refund(w, h_{31}, f_{31}))$ to P_1 and $(Verify, sid, Refund(d_{12}, d_{21}, z))$ to P_2.
(5) Exposure phase
If there is a record $(Compute, sid, Traitor(P_1))$.

$\mathcal{F}_{RDC}$ sends $(Expose, sid, Refund(f_{31}))$ to P_1 and $(Expose, sid, Refund(h_{31}))$ to $\mathcal{S}$, as well as requires $\mathcal{S}$ to send $(Expose, sid, Law(l))$.

Case 2: P_2 is corrupted.
(1) Delegation phase

The situation is the same as the delegation phase in case 1.
(2) Corruption phase
Receive $(Corrupt, sid, P_2)$ from $\widetilde{\mathcal{A}}$, and forward this message to $\mathcal{F}_{RDC}$.

(a) If $\mathcal{F}_{RDC}$ records $(sid, corrupted(P_2))$ and sends $(Corrupt, sid, \perp)$ to $\mathcal{S}$, then $\mathcal{S}$ sends $(Corrupt, sid, Deposit(h_{32}))$ to $\mathcal{F}_{RDC}$ and P_2 sends $(Corrupt, sid, Deposit(f_{32}), P_2)$ to $\mathcal{F}_{RDC}$, respectively.

(b) Otherwise, ignore this message and proceed the next phase.
(3) Computation phase

(a) If receiving $(Compute, sid, Information')$ from $\widetilde{\mathcal{A}}$, and then forward it to $\mathcal{F}_{RDC}$.

(b) If receiving $(Compute, sid, y^{*\prime}, Com'_{y^{*\prime}}, Dec')$ from $\widetilde{\mathcal{A}}$, then forward it to $\mathcal{F}_{RDC}$.

(c) If there is a record $(Compute, sid, Traitor(P_2))$, $\mathcal{F}_{RDC}$ sends $(Compute, sid, Information(f, Com_{y^*}), Com_{y^*})$ to P_1. Otherwise, $\mathcal{F}_{RDC}$ sends $(Compute, sid, Information', Com'_{y^{*\prime}})$ to P_1.
(4) Verification phase
If there is a record $(Compute, sid, Traitor(P_2))$, regardless of any message received from $\mathcal{S}$, $\mathcal{F}_{RDC}$ proceeds as follows:

- Send $(Verify, sid, Ver(Information) = TRUE)$ to P_1 and P_2.
- Send $(Verify, sid, Ver(Com_{y^*}) = TRUE)$ to P_1 and P_2.
- Send $(Verify, sid, Ver(f_i) = TRUE)$ to P_1 and P_2.
- Send $(Verify, sid, Refund(d_{21}, \frac{1}{2}z))$ to P_1 and $(Verify, sid, Refund(d_{12}, \frac{1}{2}z, w))$ to P_2. Meanwhile, send $(Verify, sid, y))$ to P_1.

If there is no record $(Verify, sid, Traitor(P_2))$, $\mathcal{F}_{RDC}$ does as follows:

- If $(Compute, sid, Information')$ is received from $\mathcal{S}$, send $(Verify, sid, Ver(Information') = FALSE)$ to P_1 and P_2.
- Upon $(Compute, sid, y^{*\prime}, Com'_{y^{*\prime}}, Dec')$ is received from $\mathcal{S}$. If the commitment can not be opened, send $(Verify, sid, Ver(Com'_{y^{*\prime}}) = FALSE)$ to P_1 and P_2. If the commitment information $Com'_{y^{*\prime}}$ can be opened by using Dec', send $(Verify, sid, Ver(Com'_{y^{*\prime}}) = TRUE)$ to P_1 and P_2. Then $(Verify, sid, Ver(f_i))$ will be received from P_1, and send $(Verify, sid, Ver-(f_i) = FALSE)$ to P_1 and P_2.
- Take the arbitration fee z paid by P_2, and send $(Verify, sid, Refund(d_{12}, d_{21}, z, w))$ to P_1 and $(Verify, sid, Refund(h_{32}, f_{32}))$ to P_2.

(5) Exposure phase
If there is a record $(Compute, sid, Traitor(P_2))$:
$\mathcal{F}_{RDC}$ sends $(Expose, sid, Refund(f_{32}))$ to P_2 and $(Expose, sid, Refund(h_{32}))$ to $\mathcal{S}$, as well as requires $\mathcal{S}$ to send $(Expose, sid, Law(l))$.

Case 3: both P_1 and P_2 are honest.
The ideal functionality $\mathcal{F}_{RDC}$ executes normally without algorithm *Corrupt*.

Case 4: both P_1 and P_2 are corrupted.
This situation does not occur because the scenario assumes that the external adversary cannot corrupt two internal participants simultaneously.

$\mathcal{S}$ is a good simulator. In case 1, $\mathcal{S}$ simulates the real adversary's corruption strategy by running $\widetilde{\mathcal{A}}$. Once the corruption succeeds, the output of P_1 can be controlled, that is, $(Verify, sid, Ver(Information) = FALSE)$ or $(Verify, sid, Ver(f_i) = FALSE)$, which simulates the situation that P_1 is corrupted in the real world, then P_1 and P_2 will be arbitrated. For any environment $\mathcal{Z}$, the view of the real world interacting with the protocol π_{RDC} and the real adversary $\mathcal{A}$ is indistinguishable from the view of the ideal world interacting with the ideal functionality $\mathcal{F}_{RDC}$ and the ideal adversary $\mathcal{S}$.

In case 2, $\mathcal{S}$ simulates the real adversary's corruption strategy by running $\widetilde{\mathcal{A}}$. Once the corruption succeeds, $\mathcal{S}$ controls the output of P_2, that is, $(Compute, sid, Information')$ or $(Compute, sid, y^{*\prime}, Com'_{y^{*\prime}}, Dec')$. Since $Information'$ represents the incorrect information division of the computing task, this message can be any incorrect sub-task partition information with randomness. $y^{*\prime}, Com'_{y^{*\prime}}, Dec'$ indicate respectively the erroneous calculation outcome, the commitment to the wrong result or the wrong commitment result, and the decommitment information corresponding to the wrong commitment or the wrong decommitment information. In order to save own storage and computing resources, the corrupted P_1 often chooses a random value as the calculation result, and may submit the wrong commitment information and decommitment information, which all are the results of random sampling. Because of the definition of the model, we know that the distribution of these random messages such as $y^{*\prime}, Com'_{y^{*\prime}}, Dec'$ is the same in the real world as in the ideal world. In addition, those who are not corrupted will honestly output the verification results during the verification stage, and eventually be arbitrated. For any environment $\mathcal{Z}$, the view of the real world interacting with the protocol π_{RDC} and the real adversary $\mathcal{A}$ is indistinguishable from the view of the ideal world interacting with the ideal functionality $\mathcal{F}_{RDC}$ and the ideal adversary $\mathcal{S}$.

In case 3, the strategy of the real-world adversary that not corrupting any participant is simulated, and no messages are sent in subsequent phases. Again, for any environment $\mathcal{Z}$, the view of the real world is indistinguishable from the view of the ideal world. As a result of the above analysis, it can be proved that there is an simulator $\mathcal{S}$ for every static malicious adversary, and for any environment $\mathcal{Z}$, $EXEC_{\pi_{RDC},\mathcal{A},\mathcal{Z}} \approx EXEC_{\mathcal{F}_{RDC},\mathcal{S},\mathcal{Z}}$, to prove the protocol π_{RDC} UC-securely realizes functionality $\mathcal{F}_{RDC}$.

Theorem 4. *The Nash Equilibrium between the ideal world and the real world is indistinguishable in our game model.*

Proof. According to analyzing the game model: from Theorem 2, the client is more inclined to choose to delegate the task, meanwhile the computing party is more inclined to accept this task. In the ideal world, if the adversary chooses to corrupt one of the internal participants by visiting the weaker ideal functionality $\langle \mathcal{F} \rangle$, it is stipulated in the corruption contract that if the internal participant who is corrupted breaks the corruption contract, he will pay a fine to the adversary. However, the betrayal contract indicates that if the corrupted internal participant chooses honest behavior, that is, violating the corruption contract, which will be risk-free, namely, the corruption fee is required to return to the adversary only, and because of the invalidity of the corruption contract, the penalty clause requiring the traitor to pay a fine to the adversary will also be deemed invalid. Furthermore, the corrupted internal participant will be exempted from the loss of reputation caused by dishonest implementation of the agreement by choosing to betray the corruption contract. Also, there is a situation in which the internal participant refuse the corruption and his ultimate utility is the same as that of becoming a traitor. Hence, even if the adversary chooses corruption, the internal participant may choose to accept or not accept, and the internal participant who is corrupted is more inclined to choose to become a traitor based on rational considerations. This betrayal will result in punishment for the adversary, that is, the adversary will be sanctioned by law with minimum payoff once the traitor and another internal participant expose the corruption of the adversary, which will lead to the mistrust between the adversary and the corrupted. Therefore, the best decision for the adversary is not to corrupt any internal participant, utility of which is zero. Since it is stipulated that internal participants are honest in the natural state, the strategy of both the client and the computing party is to choose honesty. The Nash Equilibrium in the ideal world is $S^*_{ideal} = S_1 = \{delegate, accept, \neg corrupt, honest, honest\}$.

The protocol π_{RDC} UC-securely realizes the functionality $\mathcal{F}_{RDC}$ because of theorem 3, and then according to Theorem 1: replacing calls to $\mathcal{F}_{RDC}$ by invocations of protocol π_{RDC} does not (obviously) increase the utility of a M-maximizing adversary. Hence the maximum utility of the adversary in the real world will not exceed zero, which means the best strategy for the adversary in the real world is also to choose not to corrupt any internal participants. Similarly, the strategy of internal participants is same as that in the ideal world to choose honesty. Therefore, the Nash equilibrium in the real world is $S^*_{real} = S_1 = \{delegate, accept, \neg corrupt, honest, honest\}$, thus proving the consistency of Nash equilibrium between the ideal world and the real world in the UC framework, i.e., $S^*_{ideal} \approx S^*_{real}$.

7.2 Comparison with Other Schemes

In this section, we compare our scheme with other schemes as shown in Table 3 and Table 4.

Table 3. Comparison with other schemes (part I)

Schemes	Type	The participants (number/type)		The adversary (number/type)
[15]	rational delegated computation	the client (1/honest)	the computing party (2/rational)	internal (1/rational)
[18]	rational secure function evaluation	the designer (1/rational)	parties (n/non-rational but honest)	external (1/rational)
our scheme	rational delegated computation	the client (1/rational)	the computing party (1/rational)	external (1/rational)

Table 4. Comparison with other schemes (part II)

Schemes	Key technologies	Correctness	Privacy	UC model	Consistency
[15]	blockchain, hash function, commitment, NIZK	Yes	Yes	No	No
[18]	signature, commitment, linear secret sharing	Yes	Yes	Yes	No
our scheme	homomorphic encryption, commitment, PCP-based construction	Yes	Yes	Yes	Yes

The scheme in [15] is still based on the hypothesis that the client is honest, does not take into account the situation that all of the participants are rational. Besides, unlike us: the adversary of the scheme is internal, that is, the cloud with delegated task. Moreover, the scheme does not provide UC model, let alone consider the consistency of Nash equilibrium between the ideal world and the real world. The scheme in [18] defines the game between the protocol designer and the rational adversary, but the parties are non-rational while honest. Although the scheme considers the UC security model, there is still no research on the consistency problem of Nash equilibrium under the UC framework. In our scheme, both the internal participants and the external adversary are regarded as fully rational. To consider some more realistic situations, we claim that the internal participant still has the right to choose whether to accept it or not when the adversary attempts to corrupt an internal participant, different from the previous schemes that the adversary can directly success in corrupting the parties without any ask. In addition, even if internal participant accepts this corruption, he can still choose to betray the adversary later, rather than helping the adversary break the agreement after the corruption is successful. It turns out that our scheme is more realistic and feasible. More importantly, we propose a UC security model for rational delegated computation and achieve the consistency of Nash equilibrium which is difficult to solve.

8 Conclusion

In this paper, we are committed to solving the major problem of inconsistency of equilibrium between the ideal world and the real world under the UC framework. Firstly, we define three contracts to set up the respective utility of each participant in the rational delegated computation scenario, so as to encourage players to stay honest to obtain their greatest payoff, in which both the internal participants and the external adversary is fully rational rather than based on the assumption of honesty. Then, according to Nash equilibrium obtained from game analysis, we formalize an ideal functionality $\mathcal{F}_{RDC}$ that represents the security requirements of rational delegation of computation, and construct a rational delegated computing protocol π_{RDC} which securely realizes the ideal functionality. Finally, we use the composition theorem to prove that the protocol UC-securely realizes the ideal functionality, that is, it satisfies the UC security as well as realizes the consistency of Nash equilibrium between the ideal world and the real world, which effectively solving this significant challenge.

9 Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments and helpful suggestions.

References

1. Arora, S., Safra, S.: Probabilistic checking of proofs:a new characterization of np. Proc.ieee Symp.on Foundations of Computer Science **45**(1), 70–122 (1998)
2. Babai, L.: Trading group theory for randomness. In: Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA. pp. 421–429 (1985)
3. Backes, M., Barbosa, M., Fiore, D., Reischuk, R.M.: ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. In: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015. pp. 271–286 (2015)
4. Backes, M., Fiore, D., Reischuk, R.M.: Verifiable delegation of computation on outsourced data. In: 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013. pp. 863–874 (2013)
5. Belenkiy, M., Chase, M., Erway, C.C., Jannotti, J., Küpçü, A., Lysyanskaya, A.: Incentivizing outsourced computation. In: Proceedings of the ACM SIGCOMM 2008 Workshop on Economics of Networked Systems, NetEcon 2008, Seattle, WA, USA, August 22, 2008. pp. 85–90 (2008)
6. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Snarks for C: verifying program executions succinctly and in zero knowledge. In: Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. pp. 90–108 (2013)
7. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012. pp. 326–349 (2012)
8. Braun, B., Feldman, A.J., Ren, Z., Setty, S.T.V., Blumberg, A.J., Walfish, M.: Verifying computations with state. In: ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013. pp. 341–357 (2013)
9. Broadbent, A., Gutoski, G., Stebila, D.: Quantum one-time programs - (extended abstract). In: Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. pp. 344–360 (2013)
10. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA. pp. 136–145 (2001)
11. Catalano, D., Fiore, D., Warinschi, B.: Homomorphic signatures with efficient verification for polynomial functions. In: Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I. pp. 371–389 (2014)
12. Chung, K., Kalai, Y.T., Vadhan, S.P.: Improved delegation of computation using fully homomorphic encryption. In: Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. pp. 483–501 (2010)
13. Cormode, G., Mitzenmacher, M., Thaler, J.: Practical verified computation with streaming interactive proofs. In: Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012. pp. 90–112 (2012)

14. Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile verifiable computation. In: 2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015. pp. 253–270 (2015)
15. 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, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 211–227 (2017)
16. Dunjko, V., Fitzsimons, J.F., Portmann, C., Renner, R.: Composable security of delegated quantum computation. In: Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II. pp. 406–425 (2014)
17. Freeman, D.M.: Improved security for linearly homomorphic signatures: A generic framework. In: Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings. pp. 697–714 (2012)
18. Garay, J.A., Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Rational protocol design: Cryptography against incentive-driven adversaries. In: 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA. pp. 648–657 (2013)
19. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings. pp. 465–482 (2010)
20. Gennaro, R., Wichs, D.: Fully homomorphic message authenticators. In: Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II. pp. 301–320 (2013)
21. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA. pp. 291–304 (1985)
22. Houshmand, M., Houshmand, M., Tan, S., Fitzsimons, J.F.: Composable secure multi-client delegated quantum computation. CoRR **abs/1811.11929** (2018)
23. Küpçü, A.: Incentivized outsourced computation resistant to malicious contractors. IEEE Trans. Dependable Sec. Comput. **14**(6), 633–649 (2017)
24. Lai, J., Deng, R.H., Pang, H., Weng, J.: Verifiable computation on outsourced encrypted data. In: Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I. pp. 273–291 (2014)
25. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013. pp. 238–252 (2013)
26. Setty, S.T.V., Braun, B., Vu, V., Blumberg, A.J., Parno, B., Walfish, M.: Resolving the conflict between generality and plausibility in verified computation. In: Eighth Eurosys Conference 2013, EuroSys '13, Prague, Czech Republic, April 14-17, 2013. pp. 71–84 (2013)

27. Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II. pp. 71–89 (2013)
28. Thaler, J., Roberts, M., Mitzenmacher, M., Pfister, H.: Verifiable computation with massively parallel interactive proofs. In: 4th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'12, Boston, MA, USA, June 12-13, 2012 (2012)
29. Tian, Y., Guo, J., Wu, Y., Lin, H.: Towards attack and defense views of rational delegation of computation. IEEE Access **7**, 44037–44049 (2019)
30. Unruh, D.: Universally composable quantum multi-party computation. In: Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings. pp. 486–505 (2010)
31. Wahby, R.S., Setty, S.T.V., Ren, Z., Blumberg, A.J., Walfish, M.: Efficient RAM and control flow in verifiable outsourced computation. In: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015 (2015)
32. Xu, G., Amariucai, G.T., Guan, Y.: Verifiable computation with reduced informational costs and computational costs. In: Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I. pp. 292–309 (2014)
33. Xu, G., Amariucai, G.T., Guan, Y.: Block programs: Improving efficiency of verifiable computation for circuits with repeated substructures. In: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore, April 14-17, 2015. pp. 405–416 (2015)
34. Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982. pp. 160–164 (1982)
35. Zhang, Y., van der Schaar, M.: Reputation-based incentive protocols in crowdsourcing applications. In: Proceedings of the IEEE INFOCOM 2012, Orlando, FL, USA, March 25-30, 2012. pp. 2140–2148 (2012)
36. Zhao, M., Hu, C., Song, X., Zhao, C.: Towards dependable and trustworthy outsourced computing: A comprehensive survey and tutorial. J. Network and Computer Applications **131**, 55–65 (2019)