Smart Contracts for Incentivized Outsourcing of Computation

Alptekin Küpçü
Koç University, Turkey

Reihaneh Safavi-Naini
University of Calgary, Canada

Abstract

Outsourcing computation allows resource limited clients to access computing on demand. Various types of clusters, grids, and clouds, such as Microsoft's Azure and Amazon's EC2, form today's outsourced computing infrastructure. A basic requirement of outsourcing is providing guarantee that the computation result is correct. We consider an automated and efficient way of achieving assurance where the computation is replicated and outsourced to two contractors by a smart contract that will decide on the correctness of the computation result, by comparing the two received results. We show that all previous incentivized outsourcing protocols with proven correctness fail when automated by a smart contract, because of copy attack where a contractor simply copies the submitted response of the other contractor. We then design an incentive mechanism that uses two lightweight response-checking protocols, and employ monetary reward, fine, and bounty to incentivize correct computation. We use game theory to model and analyze our mechanism, and prove that it has a single Nash equilibrium, corresponding to the contractors' strategy of correctly computing the result. Our work provides a foundation for incentivized computation in the smart contract setting and opens new research directions.

1 Introduction

Outsourcing computation to cloud enables a client to access computational resources of cloud providers on demand and benefit form unlimited applications that are offered by third party cloud providers. Outsourcing computation has also been considered in distributed computation settings such as SETI@home [25], where the aim is to employ unused computational resources of computing nodes around the world to obtain solution to large computational problems, and has found diverse applications such as finding COVID vaccine [19].

Verifying computation results. An essential requirement for the outsourcing scenarios above is for the client to be able to efficiently verify the correctness of the received computation results. Verifying computation dates back to the work of Babai [2] on proof systems, which was followed by a large body of influential works in theoretical computer science [12, 15]. The rise of cloud computing inspired verifiable computation systems for the cloud setting with strict efficiency requirement for the verifier (e.g., verification cost logarithmic in the input size) [11, 10, 7]. Purely cryptographic systems that use a single computing node [11, 10, 31], while attractive theoretically, have limited applications in practice because of high computation and communication cost, rigidity of parameters, and the challenge of correct implementation of complex cryptographic algorithms [14, 30].

An attractive approach to verifying the computation result is to replicate the computation on different computing nodes, and accept the result as correct if all nodes generate the same response.

This approach is natural and efficient for the client (*Problem Giver*), who only needs to compare the received responses. The challenge, however, is to design a *sound decision procedure* that correctly decides on the correct computation result when responses do not match.

Belenkiy et al. [4] initiated formalization of this problem using a *rational adversary model* when the *Problem Giver* hires two *rational Contractor(s)* (two is the minimum number for the replication approach) to perform the computation. They used game theoretic analysis to prove that by choosing the monetary values of *reward*, *fine*, and *bounty* (extra reward in certain settings), the contractors can be incentivized to perform their respective computations correctly. More specifically, they proved that the game of incentivized computation has a single Nash equilibrium that corresponds to the contractors using *Diligent* (honest) strategy. The decision protocol of the *Problem Giver* is to compare the responses, and accept if they match. The case of mismatch was handled through extra assumptions on the system such as high probability of one of the clouds being diligent, or re-outsourcing to two new clouds. It resulted in several followup works [17, 20].

Rational and cryptographic adversaries. Active adversary for analyzing security protocols are modelled as (i) *malicious* adversaries that can arbitrarily deviate from the protocol, and (ii) *rational* adversaries, that have defined utilities and deviate from the protocol to maximize their utility. Guaranteeing correctness for computation outsourced to two potentially malicious contractors is only possible if one can assume that one of the two contractors is honest. Else, no assurance about the results can be given.

This assumption, however, cannot be made when the client simply chooses two contractors among a number of bidding ones. In fact, if the client were able to obtain such a guarantee, then it would be likely that they could actually determine which of the two is the trustworthy one (e.g., through reputation or other side information). Using rational adversary model, however, does not require this assumption. The model captures deviating behaviour of clouds, whose goal is to cut their cost and receive the reward with minimum work. The model was used to analyze deployed outsourcing systems such as Truebit [28] that uses a smart contract and replicated computation to provide correctness guarantee for outsourced computation. *This is the adversary model that we use in this paper.*

Smart contracts for incentivized outsourcing. A blockchain-based smart contract (SC) is a public program that resides on the blockchain and runs on the underlying consensus based computation platform that ensures *trusted execution* of the program. Using smart contracts to manage incentivized outsourcing protocols, in addition to the attractive properties of guaranteed correctness and transparency of SC execution, has the very important property of support for native transfer of fund between user accounts, which is essential in incentivized protocols. SC computation, however, is very expensive (each instruction is run by the consensus nodes in the blockchain) and so the main computation must be performed off-chain.

Using smart contract to perform the decision process in Belenkiy et al. [4] model would appear as an attractive way of building an outsourcing service with provable correctness: (i) *Problem Giver* sends the computation description to the SC, (ii) SC chooses two contractors to perform the computation; (ii) if their submitted results match SC accepts and rewards the contractors, else, uses followup procedures, such as running the protocol with another set of two contractors, are employed to obtain the result. Using monetary compensation, according to Belenkiy et al. , will ensure that with a high probability the protocol will produce correct results. (We omit details such as registration fees and the like.)

In this work we show that this SC managed protocol based on an incentivized protocol with provable

correctness, will completely fail because of a new attack that is possible in the SC environment, called *copy attack*, in which a contractor will wait for the other contractor to send its response, and copy and submit the same response. Copy attack perfectly matches the rationality assumption, as it minimizes the computation of the copying contractor, and is possible because of (i) the inherent delay in communication with the blockchain due to the consensus algorithm, and (ii) that the SC cannot hold a secret and communication with it will be in plaintext. The attack undermines the independence of the two computations, which is the basis of correctness by replication, and thereby incentivizes computing nodes *not* to perform the correct computation.

An overview of our results. We consider outsourcing to two independent *rational contractors* using a smart contract, define possible strategies of the contractors in the new setting, and design two challenge and response protocols used by the SC to detect deviating contractors, that together with monetary incentives will provably result in correct computation result. Our proof is game theoretic and uses Nash equilibrium as the solution concept.

Defining the game. Copy attack results in a "waiting deadlock" which could leave the parties waiting indefinitely: each contractor waits to see the result of the other contractor. Rational contractors however can avoid this deadlock by using randomized submission time. The SC will use time limits to ensure timely completion of the results, and challenge-response protocols together with the payments to influence the behaviour of the players. The game is between the the two contractors, each wanting to maximize their utility. More details below.

Strategies. We start with the two basic strategies introduced in [4]: *Diligent (D)* strategy where the contractor follows the protocol, and *Lazy (L)* where the contractor uses a shortcut algorithm that produces the correct result with probability $q < 1$ (and so there is a probability that the cloud be rewarded) We assume the same algorithm is used by all Lazy contractors[1]. This is effectively the worst case in the sense that two Lazy contractors will have matching results and in Belenkiy et al. protocol, they both will receive the reward (and SC will accept the matching result). In the SC setting, however, there are four new attractive strategies: a basic *Guess (G)* strategy, where the computation result is simply guessed, and three variations of copy strategy, where the copy attacker uses one of the three basic strategies as a backup in case that its copy attempt fails. That is, a Copy attacker will choose a random time (within a well-defined interval) to copy; however, if there is no published result by the other contractor, it will use its backup strategy, which is one of the three basic types (D, L, G).

The protocol. Each contractor will submit a response that is a pair (y, z) where y is the computation result, and z is a fingerprint of the execution trace that we define as the Merkle root of a tree where leaves are ordered in time, and each represents an intermediate computation state. (That is, the first recorded state is the leftmost, followed by the second intermediate state, etc.). Matching results will happen when both contractors use Diligent, Lazy, or Copy strategies, and so to guarantee the correct final result (incentivize Diligent behavior), SC must use extra checking protocols. We introduce two challenge and response cryptographic subprotocols between the SC and the contractors, to enable the SC to detect deviating contractors. The first subprotocol is called the Match Check protocol, which is a single-round challenge and response protocol that is used when the two responses match, and is used to detect if one of the responses is a copied value. The protocol allows the SC to decide if (y, z) is obtained through a computation or is copied, or randomly generated by the G strategy. (We note that if one contractor simply guesses a response, and the second contractor can copy, and the results would match.) The second subprotocol is called the Mismatch Check protocol,

[1]The same assumption as [4, 17]

and is used when the two responses do not match. The subprotocol is a multi-round challenge and response bisection protocol that allows the SC to correctly and efficiently decide between two responses, where one is obtained through D strategy, and the second is by a cloud that uses L strategy. Proving correctness of computation is by showing that the choices of incentive values lead to a single Nash Equilibrium for the game with the strategy pair (D, D).
The challenge in designing incentives and analyzing the system is partly due to the fact that the two subprotocols cannot separate (L, L) from (D, D), and (L, G) from (D, G). That is, the Match Checkprotocol will accept the responses of two (L, L) contractors, and Mismatch Checkprotocol, when used for non-matching responses of contractors with strategy pair (L, G), will accept the contractor with strategy L. Using reasonable assumptions on the system parameters (Section 4), we prove that our solution achieves correctness of the computation result (Theorem 4.1).
Contributions: We show that copy attack has a devastating effect on all known incentivized systems [4, 17, 20]. Our game theoretic analysis requires a much wider set of strategies to capture SC setting in the game analysis, and analyse incentivization that leads to correct computation. Our system design is *optimal* in the following sense. Firstly, in Appendix B, we prove that all previous two-party incentivized protocols fail against a Copy attacker when taken directly to the SC setting. Secondly, match checking, mismatch checking, and bounty are all necessary components of the design. In particular, in Appendix C, we show that without using the match checking protocol, only using mismatch checking together with bounty is not be effective. This is shown by giving game theoretic analysis for both cases of with and without bounty (on top of mismatch checking), and proving that in both cases the Nash equilibrium will correspond to (CG, CG). These analyses lay the foundation of incentivized outsourcing to multiple rational contractors in the SC setting.
Copy attack in related works. Avizheh et al. [1] showed that copy attack will break security of an outsourcing protocol due to Canetti et al. [5] in the malicious adversary model when used in the SC setting. The protocol in [5] provided provable security against malicious adversary *assuming* at least one contractor was honest. Avizheh et al. showed security of the protocol can be restored by adding a single challenge and response step, executed when the two responses match. Restoring correctness guarantee in incentivized outsourcing in SC setting, however, needs a completely new protocol and overhaul of the mechanism design and analysis as given in this paper.

2 Preliminaries

Smart Contracts. A blockchain is a decentralized distributed ledger system that maintains a sequence of blocks that are ordered groups of transactions that are agreed upon by all system participants using a consensus algorithm. Blockchain systems allow users to have accounts and make transactions to other accounts. A smart contract is a trusted program that runs on a distributed ledger system (e.g., Ethereum), and its computation, communication, and stored values are transparent. More details can be found in [23].
Strategic Games. A strategic game is a model of interactive decision making where players choose their actions simultaneously and independently. A player's *utility* is their received payments minus their costs. We consider two-player games that can be described by a table, with rows and columns labelled by possible strategies (actions) of players 1 and 2, respectively. Each cell of the table contains a pair of real numbers corresponding to the utilities of players 1 and 2, respectively. The goal of a player is to maximize their utility. Nash equilibrium corresponds to a cell of the table where every player's strategy is the best response, given the other player's strategies. Therefore, no

single rational player would deviate from the equilibrium. In the computational setting, negligible differences in the utilities may be ignored, and players should be implementable in probabilistic polynomial time. For details, refer to [21].

Incentivizing correct computation. Belenkiy et al. [4] proposed an incentive mechanism that includes (i) reward, the money paid to a contractor that correctly performs the computation, (ii) fine, the money charged to a contractor that is detected to have produced an incorrect result, and (iii) bounty, which is the money that is paid to a contractor that correctly performs the computation while the other contractor is detected to return an incorrect result. The two contractors use two strategies D or L. Using game theoretic analysis, Belenkiy et al. proved that the game of incentivized computation has a single Nash equilibrium, which corresponds to both contractors performing the computation correctly.

Merkle Hash Tree is a binary tree that is constructed over a sequence of data elements $D = (d_1, \cdots, d_n)$ using a *collision-resistant* hash function. The leaves of the tree are the hash values of elements of D, and an internal node is the hash of the concatenation of its two child nodes. A Merkle tree construction starts from the leaves and moves to the root that is denoted by $z = MH_{root}(D)$. The *proof of consistency* for the element d_i with respect to the root z, called *Merkle proof*, is denoted by $p_i = MH_{proof}(D, d_i)$, and consists of the hash values of the siblings of nodes along the path from $H(d_i)$ to the root. Given a Merkle proof p_i for the element d_i and the root z for the data sequence D, the $VerifyMHProof(z, i, d_i, p_i)$ function verifies consistency of d_i, with respect to the Merkle tree with root z using the proof p_i. The function is efficiently (logarithmic in the sequence length) and publicly computable, and outputs *True* if the verification succeeds, and $False$ otherwise.

Representing computation and its trace. The response of a contractor consists of a calculated value, together with commitment to the computation trace. We express a computation using a Turing machine (TM) and an input tape that initially stores the input. A computation state corresponds to a *Turing machine configuration* $(state, head, tape)$ and is stored as a ***reduced configuration*** defined by [5]:

$$(state, head, tape[head]; MH_{root}(tape))$$

where $tape[head]$ denotes the tape content at the location of the $head$, and MH_{root} denotes the root of a Merkle Hash tree over the *tape*. A contractor uses the sequence of computation states to express the execution trace of their computation. Let the j^{th} reduced configuration be denoted by $rc_j = (s_j, h_j, v_j, rt_j)$, where s_j and h_j represent the *state* and *head* position, respectively, v_j represents the tape at the given head position, *tape[head]*, and $rt_j = MH_{root}(t_j)$ is the *root* of the Merkle tree on the tape t_j at that stage. Let $RC = (rc_1 \cdots rc_n)$, denote the sequence of reduced configurations of the Turing Machine, and let z denote the root of the Merkle tree that is constructed over RC. Figure 1 visualizes a sample reduced configuration with its Merkle tree, and the Merkle tree built over the sequence of reduced configurations, resulting in the z value.

3 Model

We consider a setting with three types of entities: (i) a *Problem Giver* who wants to outsource the computation of a *deterministic* function[2] $f()$ on an input x, (ii) a set of *Contractors* who are incentized to perform the computation, and (iii) a *Smart Contract (SC)* that interacts with the parties.

[2] A randomized algorithm can be outsourced after de-randomization using a pseudorandom generator.

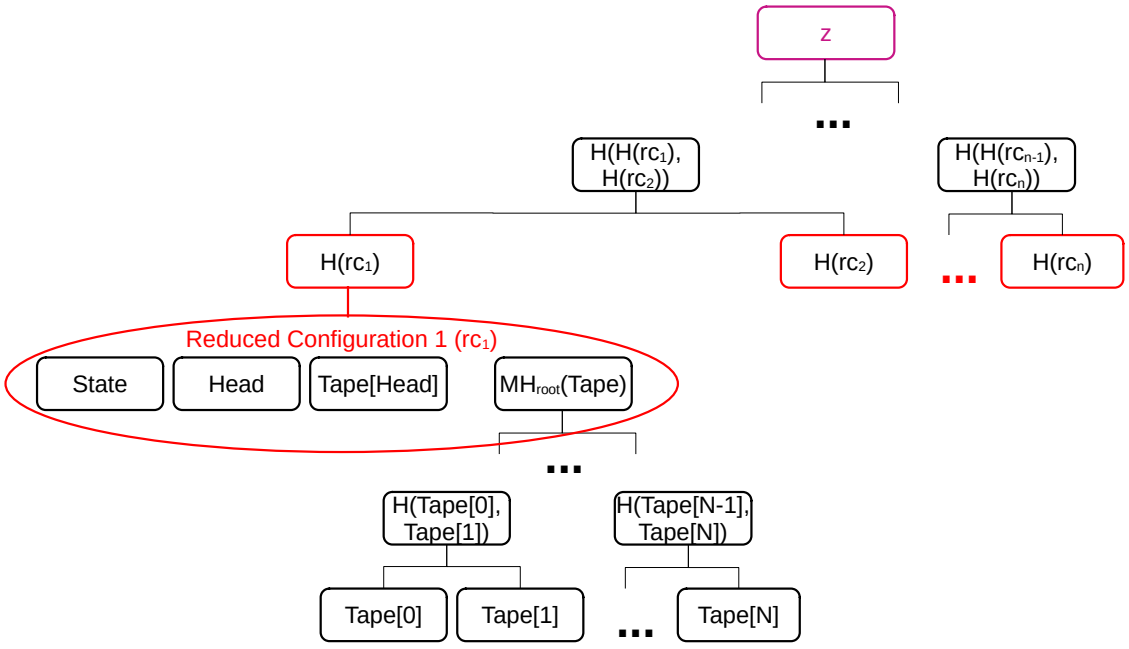

Figure 1: Merkle tree built on the sequence of reduced configurations. A reduced configuration includes the root of the Merkle tree built on the tape of that state.

The SC receives deposits from the participants, and after receiving responses from the contractors, executes a *Judge* protocol that decides on the computation result based on the received responses and possibly additional interactions with the contractors, and performs money transfers to/from contractors' accounts as specified by the protocol. The *Problem Giver* makes the required deposit to the SC in advance, and expects to obtain the correct computation result. A *Contractor* is *rational* and wants to maximize its utility that is expressed as the net reward. *The SC* is a transparent trusted program that runs on the blockchain consensus computer and executes the prescribed protocol. The SC can be created by the *Problem Giver*, or by an established service provider.

Outsourced computation. The *Problem Giver* wants the value of a function $f()$ on an input x. The function is expressed by a Turing Machine (TM) for the computation of $f()$ on the input tape that contains x. The **response** of a contractor is a pair (y, z) where y is the computation **result** (if correct, $y = f(x)$), and z is the root of a Merkle hash tree that is constructed on the sequence of reduced configurations of the TM's computation. SC randomly chooses two contractors, from a pool of available contractors. The pool is large enough that we can assume the two chosen contractors are independent.

Goals of Incentivized Outsourced Computation are the following:

1. The result received by the *Problem Giver*is correct with overwhelming probability.
2. Contractors are incentivized to participate and correctly perform the computation.
3. The computation and communication at the SC is minimal.

An implied goal of the system is that a contractor that has correctly performed the computation is always rewarded.

Strategies. There are two basic strategies, (i) **Diligent (D)** that correctly executes $f(x)$, and (ii)

Lazy (L), where the contractor deviates from correct calculation to reduce its computation cost. A lazy algorithm is referred to as a q-algorithm, and generates the correct *result* (not the correct *response*) with a non-negligible chance q (generating the correct *response* has negligible probability ϵ). A q-algorithm can be *any* maliciously constructed algorithm that performs fewer computation steps and produces an acceptable value for the computation. It can simply skip some steps of the original computation; but in all cases the contractor has a computation trace that matches its committed root of execution tree. For example, a SETI@Home [25] image processing program can simply skip processing some pixels of the image. We assume all *all Lazy contractors use the same algorithm* and so their computation results match. Thus, without additional measures, they will receive the reward. Belenkiy et al. [4] made the same assumption, inspired by the case that the same SETI@Home [25] fake clients were downloaded by multiple participants (see [4]). A maliciously constructed program can be made available to rational contractors, who will be attracted to the reduced computation and the possibility of not being caught.

New strategies. We consider a new basic **Guess (G)** strategy, where a contractor guesses the value of $f(x)$. The strategy has negligible cost and because of copy attack, can lead to matching results. The main difference between G and L strategies is that a (G, G) strategy pair will not lead to matching responses (negligible chance), while an (L, L) pair will output matching responses. By requiring the Merkle hash of the computation to be included in the contractor's response, the probability that two submitted guessed responses match will be negligible even when the computation result itself ($f(x)$) is from a small domain.

Copy strategy allows a contractor to completely skip the computation. A Copy contractor waits for the "other" contractor to submit its response to the SC and copies and submits that response as its own. This strategy is possible because of (i) SC's transparency of computation and communication, and (ii) the time interval between submitting a transaction to the blockchain network, and having it published on the blockchain. Copy strategy is very attractive because it allows a contractor to produce a matching result, and receive the reward with negligible work. However, since both contractors can use this strategy, both contractors may end up waiting indefinitely. To overcome this deadlock, a contractor will use a random time that is chosen from an appropriate range $[T_1, T_2]$, and copies the published response if exists; else, it resorts to one of the basic strategies. This leads to *Copy-Diligent (CD)*, *Copy-Lazy (CL)*, and *Copy-Guess(CG)* strategies. Thus we obtain a total of six strategies (including D, L, G), for the contractors as below.

- **Diligent** (D): Computes using the original algorithm. The response will always be accepted and rewarded. The cost is $cost(1)$.
- **Lazy** (L): Computes using a q-algorithm that is assumed common for all Lazy contractors. The *result* will be correct with probability q, and the cost will be $cost(q)$. With the use of hashing, the *response* (result together with the hash of the computation) will be correct only with negligible probability. This is a very critical observation on the Lazy contractors, first made by [4].
- **Guess** (G): Creates a random bit string that matches the format of the submission to the SC. The response will be correct with probability ϵ, and the cost is $cost(\epsilon)$.
- **Copy**: The contractor chooses a random time from a time period; if the "other" contractor has sent its response, it copies the response; else the contractor continues with one of the original strategies: D, L, and G. There are three variations: *Copy-Diligent* (CD), *Copy-Lazy* (CL), and *Copy-Guess* (CG). The cost of successful copying is $cost(\epsilon)$.

Towards a sound *Judge* protocol in the SC setting. A first approach towards constructing a

Judge protocol is to base it on the Belenkiy et al. [4] protocol with proved correctness: (i) if the two responses match, the *Judge* protocol outputs the result rewarding the contractors, (ii) else (when the responses differ) the *Judge* uses additional steps to identify the correct result and the contractor that is Diligent (if any). Belenkiy et al. and the follow-up works [20, 17] use extra assumptions or approaches such as running the protocol multiple times, implying that, in a mismatched response produced by a strategy pair (D, L) the Diligent contractor is identifiable. The following theorem proves that in the SC setting this *Judge* protocol cannot produce correct result for the *Problem Giver*using reward, bounty, and fine as incentive.

Theorem 3.1. *The incentivized computation protocol, with the possible contractor strategies D, L, CG and the* Judge *protocol above (based on [4, 20, 17]) in the smart contract setting, has a single Nash equilibrium that corresponds to the (CG, CG) strategy pair, leading to incorrect computation result for the* Problem Giver. Proof is in the Appendix.

Algorithm 1 *Problem Giver*

Set $f, r, f(), x, \tau_{SC}$.
Deposit $2r$ to the SC account.
Wait τ_{SC}.
Obtain the result y and any fines collected from SC.

Algorithm 2 Contractor

Deposit f to the SC account.
Obtain $f, r, f(), x, \tau_{SC}$ from the SC.
Run the strategy (D, L, G, CD, CL, or CG), obtaining the result y and the hash z.
Submit the response (y, z) to the SC.
If accepted by the SC, obtain r and get back the deposit of f.

Algorithm 3 *Copy* contractor

Deposit f to the SC account.
Obtain $f, r, f(), x, \tau_{SC}$ from the SC.
Pick a random time t in $[T_1, T_2]$.
At time t, check the SC.
if there is already a response (y, z) stored at the SC **then**
  Submit the same response (y, z) to the SC.
else
  Run the strategy (D, L, or G) according to the type, obtaining the result y' and the hash z'.
  Submit the response (y', z') to the SC.
end if
If accepted by the SC, obtain r and get back the deposit of f.

4 A *Judge* Protocol with Guaranteed Correctness

We first introduce notations that are used to express the working of the system, and then give (reasonable) assumptions that will be used in the game analysis. Pseudocodes for the *Problem Giver* and the contractors are in Algorithms 1, 2, and 3.

Notations:

- y, z: The *response* of a contractor, which includes the result y of the computation, together with the Merkle root z of the computation trace.
- r: The reward of a contractor in two cases, (i) when the SC receives two matching responses, and (ii) when the SC receives two conflicting responses, but the contractor succeeds in the Mismatch Check protocol.
- f: The fine charged to a contractor when their response is detected as incorrect. The fine can be enforced by requiring the contractors to make a deposit at the start of the protocol.
- $cost(1)$: The cost of the original algorithm, run by the Diligent contractors.
- $cost(q)$: The cost of a q-algorithm, run by the Lazy contractors.
- $cost(\epsilon)$: The cost of guessing and copying both. ϵ is a negligible value.
- τ_D, τ_L:, : Time to compute the function using tD and L strategies.
- τ_D: Time to compute the function using the Diligent strategy.
- τ_L: Time to compute using the Lazy strategy.
- τ_N: Network delay between a contractor and the SC.
- τ_{SC}: Smart Contract deadline for receiving computation results.
- q_S: The probability that the copying is successful for a Copy contractor, when the other contractor also uses Copy strategy. (Interestingly, our results turn out to be nicely independent of the actual value of this probability.)
- q_0: The probability that neither contractor can copy the other's response (because of closeness of random times). Note that $q_0 + 2q_S = 1$ since either one of the contractors could copy or neither could, when both contractors use Copy (they cannot both copy).
- $f()$: The function to compute, picked by the *Problem Giver*.
- x: The input to the function, picked by the *Problem Giver*.
- y: The result submitted by a contractor. Ideally, we want that $y = f(x)$.
- z: The Merkle tree root submitted by a contractor.
- A contractor's strategies are: D: Diligent, L: Lazy, G: Guess, CD: Copy-Diligent, CL: Copy-Lazy, CG: Copy-Guess.
- C refers to a Copy contractor (CD, CL, or CG) who could successfully copy.

System Parameters and Assumptions:

1. $r > cost(1)$. That is, the reward of performing the computation correctly exceeds the cost of the computation. Otherwise, a rational contractor will not join the system.
2. All Lazy contractors use the same deterministic q-algorithm. This represents for example, downloading a fake client. Therefore, the result of two Lazy contractors always match.
3. A q-algorithm produces correct computation *result* with probability q, per [4]. Note that this only holds for the correctness of the computation *result* $y = f(x)$, which is part of a contractor's *response*. The probability of producing the correct *response* (which also includes the Merkle hash) is *negligible*. When two Lazy contractors get matched, they produce the same *response*.
4. When a Lazy contractor is matched against a Diligent contractor, the probability that their responses (y, z) match, is negligible. This is because the Lazy and Diligent algorithms are different in at least one step, and so their corresponding execution trace on the same input x and their associated Merkle roots, will be different with overwhelming probability. Similarly, when a Guessing contractor gets matched against a non-copy contractor, the probability that they return the same response is negligible. The probability of guessing a response that

matches the response of another G, L, or D amounts to correct guessing of binary strings that are at least 128 or 160 bits (Merkle root), and so is negligible. [3]

5. The cost of a q-algorithm is $cost(q)$, and $cost(1) > cost(q)$. (Otherwise there is no need to employ a q-algorithm.)
6. We assume $cost(q) > cost(\epsilon)$. Thus, guessing and copying constitute the least costly actions.
7. Once a computation result is produced, it will be submitted to the SC. That is, a contractor will not add additional delay to the computation.
8. A contractor knows a good estimate of the computation time of different strategies, as well as network delay. That is, in particular, it knows upper bounds on $\tau_D > \tau_L$ and τ_N.
9. The interval $[T_1, T_2]$ that is used by the Copy contractors is $[\tau_D + \tau_N,\ \tau_{SC} - \tau_N]$. That is, a copying contractor waits for a non-copy contractor to produce and submit its response.
10. The probability that two Copy contractors pick very close random times such that neither have the opportunity to copy from the other is negligible. This can happen if the first contractor cannot copy because no result is published, and the second contractor's time is too close to the first contractor to receive its published value. Note that the random time can always be selected at coarser intervals (e.g., at multiples of τ_N). We assume the probability of selecting the exact same (coarse) time is negligible.
11. The computation deadline, τ_{SC}, is set by the smart contract and is public. This time satisfies $\tau_{SC} > \tau_D + \tau_N$ so that a Diligent strategy can succeed.
12. The interval $[T_1, T_2]$ that is used by the Copy contractors is $[\tau_D + \tau_N,\ \tau_{SC} - \tau_N - T]$ where T is τ_D if the contractor is Copy-Diligent, τ_L if the contractor is Copy-Lazy, negligible if the contractor is Copy-Guess.
13. $\tau_{SC} >> 2\tau_D + 2\tau_N$ such that Copy-Diligent is a viable strategy (a CD contractor can wait for a D contractor to finish, and if there is still no submitted response, can still execute its own Diligent computation).

Assumptions (7), (8) and (9) imply that the copy strategy C will be used after the contractor that uses D, L, or G strategy has completed and submitted its computation, and the result can be seen by the copying contractor. This implies a Copy contractor always succeeds copying the response of a non-copy contractor. Assumption (10) implies that when two Copy contractors play against each other, one of them will succeed in copying (i.e., q_0 is negligible). We show that our results turn out to be independent of this assumption and we only use it for simplicity of the presentation and analysis.

4.1 The New *Judge* protocol

Let the two contractors be denoted by $P_i, i = 1, 2$, each constructing a response using their chosen strategy and sending the pair $(y_i, z_i), i = 1, 2$, to the SC. For the Diligent strategy, we have $y_i = f(x)$. Upon the receipt of both responses, the SC runs the following *Judge* protocol:

- If $(y_1, z_1) = (y_2, z_2)$, run the *Match Check* protocol.
- Else, when $(y_1, z_1) \neq (y_2, z_2)$, run the *Mismatch Check* protocol.

[3] Recall that the difference between G and L strategies is that the response submitted by two L contractors will match, whereas the response submitted by two G contractors will *not* match, except with negligible probability. Thus, the Lazy contractor paradigm is enough to model submitting matching guesses (e.g., using the same pseudorandom seed).

Algorithm 4 Match Check

Let $n_i, i = 1, 2$ denote the lengths of $RC^i, i = 1, 2$.
Judge generates two PRNs, $rand_1$ and $rand_2$.
Judge$\rightarrow P_i : rand_i, i = 1, 2$
$P_i \rightarrow$ *Judge* : $(rc^i_{rand_i}, rc^i_{rand_i+1}, MHProof(RC^i, rc^i_{rand_i}),$
$MHProof(RC^i, rc^i_{rand_i+1}), p^i_{rand_i}), i = 1, 2.$
Judge uses $VerifyComittedReducedStep()$ on the submitted response.
The result of a contractor who passes the verification will be accepted.

Algorithm 5 Mismatch Check Protocol

$n' = \min\{n_1, n_2\}$, where n_i is the length of the sequence of reduced configurations of P_i.
$z^i = MHroot(RC^i), i = 1, 2$
Perform Committed Binary-Search (Algorithm 6)given $z^i = MHroot(RC^i), i = 1, 2$, with the two contractors to find the smallest j, where $rc^1_j = rc^2_j$ and $rc^1_{j+1} \neq rc^2_{j+1}$.
Judge $\rightarrow P_i$: j , i=1,2.
$P_i \rightarrow$*Judge* :
$(rc^i_j, MHProof(rc^i_j, RC^i), rc^i_{j+1}, MHProof(rc^i_{j+1}, RC^i), p^i_j)$
Judge verifies using $VerifyCommittedReducedStep()$.
Result of P_i is accepted if the output is True.

Checking matching submissions. Matching responses occur in all variations of Copy, and also for strategy pairs (L, L) and (D, D), and to have guaranteed correctness one must use extra checks. We prove this in Appendix C by showing that the equilibrium corresponds to (CG, CG) when the Match Check subprotocol (Algorithm 4) is not employed (instead, matching responses are simply accepted), but only Mismatch Check subprotocol (Algorithm 5) is used.
Note that a strategy pair (L, L) will result in a matching responses, and the Match Check protocol will be run. The challenged steps, however, will be responded consistently with the committed roots, and because there could be a good chance that the Match Check protocol chooses a computation step that is the same in the q-algorithm as the original computation, the Lazy approach will remain undetected. For example, consider a Lazy approach where the contractor skips the last several steps of the correct computation. This cuts down the cost of computation, and will *not* be detected by the *Judge* protocol when two L responses are received, and so the computation result will be incorrect. Our outsourcing mechanism with fines, rewards, and bounties will result in the desired (Diligent) equilibrium nevertheless. Additionally, for (L, C) strategy pair, it is possible again that the challenged step of the q-algorithm is the same as the correct algorithm, and L strategy will mistakenly be identified as D, whereas C will be penalized, since it cannot respond to the challenge as it did not perform any computation. Copying is bad with the new Match Check protocol, since it cannot respond to the *Judge* challenges.
Checking mismatching submissions. When the submitted responses do not match, the SC needs to decide which one to accept (if any). The goal is to distinguish a D strategy against L or G strategies (as well as variation of copy strategies that result in similar strategy pairs). The protocol will correctly identify D against L or G, but incorrectly reward L against G. In the case of (G, G) pair, both contractors will be fined. Yet, this is enough to achieve the desired Diligent equilibrium as we show. The Committed Binary-Search protocol is a modified version of [5] such that each

Algorithm 6 Committed Binary-Search Protocol

Input: $n_g, n_b, z^1 = MH_{root}(RC^1), z^2 = MH_{root}(RC^2)$
repeat
  $w = (n_b - n_g)/2 + n_g$
  Problem Giver $\rightarrow P_1, P_2 : w$
  $P_i \rightarrow$ *Problem Giver* : $rc_w^i, MH_{proof}(rc_w^i, RC^i)$ i=1,2
  if $VerifyMHProof(z^i, w, rc_w^i, p_w^i) = False$ **then**
    Declare i as Cheater, Exit.
  end if
  if $(rc_w^1 = rc_w^2)$, **then**
    $n_g = (n_b - n_g)/2 + n_g$,
  else
    $n_b = (n_b - n_g)/2 + n_g$
  end if
until $n_b = n_g + 1$

Strategies	Match Check Result	Strategies	Mismatch Check Result
D, D	$D+ \quad D+$	D, L	$D+ \quad L-$
D, C	$D+ \quad C-$	D, G	$D+ \quad G-$
L, L	$L+ \quad L+$	L, G	$L+ \quad G-$
L, C	$L+ \quad C-$	G, G	$G- \quad G-$
G, C	$G- \quad C-$		

Table 1: *Judge* protocol results. The worst-case for the *Problem Giver* is assumed. There is no ordering of the contractors since the result would be symmetric (e.g., D, C and C, D are the same in this representation). + indicates being rewarded, − indicates being fined.

query is verified against the submitted Merkle root, and at the end, since the responses are different, finds the first step where the two computation traces of the contractors differ. We present further details in Appendix A. The protocol is efficient and finishes in one round, significantly reducing the need for extra assumptions in Belenkiy et al. [4] or re-outsourcing as in [17].

Table 1 visualizes the *Judge* protocol results. Observe that when two Copy contractors get matched, the cases boil down to one of the cases in the table: (CD, CD) boils down to (D, C) since one (either P_1 or P_2) copies and the other executes the computation Diligently, and similarly (CL, CL) boils down to (L, C) and (CG, CG) boils down to (G, C). The *Judge* protocol identifies a Diligent contractor (if any) and always rewards them (never fines Diligent contractors). But, the *Judge* protocol may also incorrectly reward non-diligent contractors with incorrect responses.

Remark. We note that neither the Mismatch Check and Match Check protocols, nor the bounty usage alone, can lead to an equilibrium that corresponds to the correct result. However, with a well designed combination of them, we achieve the desired mechanism. This is an innovative aspect of our work.

P_1, P_2	D	L	G	CD	CL	CG
D	u_D, u_D	u_{DB}, u_-	u_{DB}, u_-	u_{DB}, u_-	u_{DB}, u_-	u_{DB}, u_-
L	-	u_L, u_L	u_{LB}, u_-	u_{LB}, u_-	u_{LB}, u_-	u_{LB}, u_-
G	-	-	u_-, u_-	u_-, u_-	u_-, u_-	u_-, u_-
CD	-	-	-	u_{CDB}, u_{CDB}	u_{CDB}, u_{CLB}	u_{CDB}, u_-
CL	-	-	-	-	u_{CLB}, u_{CLB}	u_{CLB}, u_-
CG	-	-	-	-	-	u_-, u_-

Table 2: Utility table, *with copy protection*. Note that the utilities in the table represent a symmetric game (not a symmetric matrix), thus unnecessary cells are omitted.

4.2 Game Analysis

Our *Judge* protocol design results in Table 2. Below, we detail the utilities in the table (we follow the row order):

- The utility of the strategy D against another D is $u_D = r - cost(1)$: The results would match, and the contractor will receive the reward, while paying the cost of the computation.
- The utility of the strategy D against others is $r - cost(1) + b(1-\epsilon)$: The *Judge* protocol will identify the diligent versus others, except with negligible probability, as discussed. The Diligent contractor will obtain the reward and the bounty. We approximate this utility as $r - cost(1) + b(1-\epsilon) \approx r - cost(1) + b$ and denote as u_{DB}.
- The utility of the others against the D strategy is $r\epsilon - f(1-\epsilon) - cost(q) < 0$: The *Judge* protocol will catch them against Diligent. We approximate this as $r\epsilon - f(1-\epsilon) - cost(q) \approx -f - cost(q)$ which is negative, and denote it in the table with u_-.
- The utility of strategy L, against another L is $u_L = r - cost(q)$: They both return the same response, will be able to pass the *Judge* protocol (since we assume the worst-case q-algorithm), hence they both get the reward. In any case, they pay the cost of the q-algorithm.
- The utility of the L strategy against other non-Diligent strategy (G, CD, CL, CG) is $r - cost(q) + b(1-\epsilon)$: The *Judge* protocol may (mistakenly) reward the L strategy and provide extra bounty, while fining the others. We approximate this utility as $r - cost(q) + b(1-\epsilon) \approx r - cost(q) + b$ and denote it as u_{LB}.
- The utility of the G strategy against any strategy, and the utility of copy variants (CD, CL, CG) against any non-copy strategy are all u_-. This is because they cannot respond properly to the challenges of *Judge* (guessing cannot respond to Mismatch Check and *successful copying* cannot respond to Match Check), and will be fined. This also applies to CG against CG, since in that case one of them will act like G in practice and the other will successfully copy.
- The utility of a CD contractor against any other Copy contractor is $u_{CDB} = q_S u_- + (1 - q_S) u_{DB}$: When it can successfully copy, which happens with probability q_S, it will be caught by the new *Judge* protocol, thereby getting fined and obtaining negative utility. But, when it cannot copy, which happens with probability $(1 - q_S)$, it will act Diligently, and will help catch the other contractor, obtaining u_{DB}.
- The utility of a CL contractor against any other Copy contractor is $u_{CLB} = q_S u_- + (1 - q_S) u_{LB}$: When it can successfully copy, which happens with probability q_S, it will be caught by the new *Judge* protocol, thereby getting fined and obtaining negative utility. But, when

it cannot copy, which happens with probability $(1 - q_S)$, it will act Lazily, but will not be caught by the *Judge* protocol. Instead, it will be seen as helping to catch the other contractor, thereby obtaining u_{LB}.

Intuitively, Guess or Copy-Guess strategies will fail with our Match Check protocol, since they will be caught and fined. Moreover, being completely Diligent is better than being Copy-Diligent, since the latter will be caught and fined when it successfully copies. Similarly, being Lazy is better than being Copy-Lazy. Using bounties with our *Judge* protocol with two checking protocols Mismatch Check and Match Check results in an all-Diligent equilibrium.

Theorem 4.1. *Under the reasonable assumptions stated in Section 4, and if $b > cost(1)$, then the pair of strategies (D, D) gives the only computational Nash equilibrium of the strategic game in Table 2.*

Proof. Observe that since u_- is negative, and G and CG strategies always get u_-, they are not rational anymore. Thus, we focus on D, L, CD, CL.

We start by trivial observations about the utilities:

$$u_{DB} > u_{CDB} \tag{1}$$

$$u_{LB} > u_{CLB} \tag{2}$$

since $q_S > 0$.

Next, with a series of best-response type of arguments, we show the equilibrium is (D, D).

First, observe that L is the best response against CL due to equations (12), (1) and (2).

Second, realize that the same set of equations also imply that L is the best response against CD.

Third, we show that D is the best response against L because

$$u_{DB} > u_L$$

$$r - cost(1) + b > r - cost(q) \tag{3}$$

which holds as long as

$$b > cost(1) - cost(q)$$

$$b > cost(1) \tag{4}$$

as stated in the theorem. This means, while L is the best response against all Copy strategies, if a contractor should choose L, then the other contractor is better of being D.

Lastly, D is the best response against D since $u_D > u_-$. In plain words, when the other contractor is Diligent, we should be Diligent as well, as all other options get negative utility. Therefore, no contractor has incentive to deviate from this (D, D) equilibrium. □

Corollary. *The (D, D) strategy pair results in correct computation result for the* Problem Giver*. Together with bounties, our* Judge *protocol, which is an efficient verification mechanism run with every pair of submissions, disincentivizes free riding and incentivizes Diligent behavior.*

An interesting property of the bounties in our setting is that while using them partly help change the equilibrium, they will *not* be used when all contractors are rational and hence act Diligently. Thus, bounty should not be seen as an extra expense for the *Problem Giver*.

Finally, we argue that Theorem 4.1 holds even when assumption (10) is invalid. To show this, consider a fine-grained version of u_{CDB}. When this CD contractor could successfully copy, with

probability q_S, then it will be penalized with negative utility u_-. Thus, the $q_S u_-$ part does not change. When it could not copy and therefore resorts to the Diligent strategy, it is guaranteed that it will get the reward (since it is Diligent), but about the bounty, there are two cases: Either the other contractor could copy, which happens with probability q_S for the other contractor, in which case this contractor would get the bounty, or the other Copy contractor resorted to its backup strategy, which happens with probability q_0. The latter results in the following options:

- The other contractor is CD or CL: No bounty will be obtained. The utility of this contractor would be u_D.
- The other contractor is CG: The other contractor will be caught, and this contractor will obtain bounty together with the reward, resulting in u_{DB}.

Putting all these together, what we have is that $q_S u_- + q_S u_{DB} + q_0 u_D$ or $q_S u_- + q_S u_{DB} + q_0 u_{DB}$, where the latter is exactly u_{CDB}, and the former is upper bounded by u_{CDB} since $u_{DB} > u_D$. Thus, our utility table above put an upper bound utility for CD against another Copy contractor. A very similar argument holds for CL against other Copy contractors. The following are the cases when the other Copy contractor resorted to its backup strategy:

- The other contractor is CD: This contractor will be caught and penalized, obtaining negative utility u_-.
- The other contractor is CL: No bounty will be obtained. The utility of this contractor would be u_L.
- The other contractor is CG: The other contractor will be caught, and this contractor will obtain bounty together with the reward, resulting in u_{LB}.

Putting together, we have $q_S u_- + q_S u_{LB} + q_0 u_-$ or $q_S u_- + q_S u_{LB} + q_0 u_L$ or $q_S u_- + q_S u_{LB} + q_0 u_{LB}$. The last one is exactly u_{CLB}, and the first two are upper bounded by u_{CLB} since $u_{LB} > u_-$ and $u_{LB} > u_L$.

The fact that the values u_{CDB} and u_{CLB} used in Theorem 4.1 were upper bounds mean that equations (1) and (2) still hold using their fine-grained versions. Therefore, the theorem holds even without assumption (10).

5 Related Work

There is a large body of works on outsourcing and delegation of computation with correctness and verifiability properties. Below we review the most relevant works in two groups.

Malicious Adversary Model. Interactive proof systems follow the seminal works of Goldwasser et al. [12] and Babai and Moran [3] that consider a single malicious prover, and has led to the development of *verifiable computation* systems [10] (see a survey [31] for more).

Canetti et al. [5, 6] refereed delegation protocols [9] provide correctness in presence of a single malicious contractor. Avizheh et al. [1] showed that [5] is vulnerable to copy attack in the SC setting. The protocol in [5] requires secure channels and cannot be used in SC setting.

Using replicated computation for integrity checking has been used in works such as [29]. These works do not provide formal cryptographic or game theoretic modelling and analysis of their systems.

Rational Adversary Model. Outsourcing computation to multiple independent rational entities has been popularized by projects such as SETI@Home [25] and Rosetta@Home [22] where idle CPU time of the users were employed for computing on scientific data. In these systems, the main goal is *distributing the computation load among a number of contractors*, although replication is also used to provide some level of integrity. Participation of clients is on altruistic basis using an

unfungible leader board status. Indeed, fake clients had been employed by rational contractors, thereby resulting in incorrect results [18].
The focus of this paper is on designing an *efficient mechanism that can be used by a smart contract*, with two rational contractors, both of which can deviate from the correct computation. This is similar to the settings in [13, 8, 24, 26, 27, 16], with the key addition of SC that automates (takes the role of) the referee (judge) protocol to decide on the correct computation result.
Belenkiy et al. [4] were the first to define Diligent and Lazy strategies, when outsourcing to *two* rational contractors simply comparing the returned responses. They argued the need to use *fines* in addition to *reward*, to achieve correctness. They further showed that using *bounty* for a contractor who performed Diligently against a Lazy contractor will lead to a single Nash equilibrium corresponding to correct results. They argued that *Guess* strategy need not be considered by using a "hash of the computation" that can be required for the submitted response, and will prevent the chance of a guess to match the correct response that is produced by a correct computation.
Küpçü [17] extended the framework of [4] to multiple contractors, and added altruistic and malicious contractors to the framework in addition to the rational ones. The protocol uses the results of potentially multiple rounds of outsourcing to arrive at a correct decision.
All above works in the rational setting assume that the *Problem Giver* directly interacts with the contractors, and communication channels can be secured (e.g., using TLS). Thus, in that setting, copy attack need not be considered. *All these works are vulnerable to copy attack in the SC setting* considered here.
Copy attack. When SC is used to automate outsourcing as a service, all previous incentivized protocols must be revisited to provide security against copy attack. Our results in Appendix C showed that a basic *Judge* protocol with only Match Check, even with considering bounty, cannot disincentivize dishonest behavior, and in Section 4 we showed how to guarantee correctness for the computation result. Compared to Avizheh et al. [1] in the malicious model, the challenge of our work is developing a realistic game theoretic model for the setting that captures real world restrictions of a smart contract environment, and design a set of assumptions and bounds on timing of the events without being prescriptive on the exact times. Such a refined description of the world is not necessary in malicious adversary model of [1], which relies on the assumption of the honesty of one contractor. Without this assumption, the SC does not have any reference point upon receiving two responses, and needs more complex check protocols and incentive analysis to guarantee correctness.

6 Concluding Remarks

The rise of blockchain, the attractive prospect of automating outsourcing with the possibility of using a native crypto-currency for implementing incentives, have been our motivation to study incentivized protocols in the SC setting. Surprisingly, however, because of the Copy attack, none of the previously known protocols with provable game theoretic correctness can guarantee correct results in this setting. We proposed an SC based incentivization mechanism with two checking protocols that guarantees correctness of the results. Our work is the first step to analyze incentivized SC outsourcing systems through replication. Our final *Judge* protocol with the two Mismatch Check and Match Check protocols is the first outsourcing protocol with guaranteed correct computation result, and lays the foundation for the more general case of multiple contractors. One of the challenges of our work was to provide an abstract model of the smart contract environment and behaviour of rational parties that realistically captures the effect of Copy attack. Extending our

work to multi-contractor setting will significantly complicate this model and increase the range of available strategies, including collusion strategies. We leave it as future work.

References

[1] S. Avizheh, M. Nabi, R. Safavi-Naini, and M. Venkateswarlu K. Verifiable computation using smart contracts. In *ACM CCSW*, 2019.

[2] L. Babai. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 421–429. ACM, 1985.

[3] L. Babai and S. Moran. Arthur-merlin games: a randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.

[4] M. Belenkiy, M. Chase, C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya. Incentivizing outsourced computation. In *NetEcon*, 2008.

[5] R. Canetti, B. Riva, and G. N. Rothblum. Practical delegation of computation using multiple servers. In *ACM CCS*, 2011.

[6] R. Canetti, B. Riva, and G. N. Rothblum. Two protocols for delegation of computation. In *ICITS*, 2012.

[7] K.-M. Chung, Y. Kalai, and S. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, 2010.

[8] W. Du, M. Murugesan, and J. Jia. Uncheatable grid computing. In *Algorithms and theory of computation handbook*. Chapman & Hall/CRC, 2010.

[9] U. Feige and J. Kilian. Making games short. In *ACM STOC*, 1997.

[10] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, 2010.

[11] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015.

[12] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[13] P. Golle and I. Mironov. Uncheatable distributed computations. In *CT-RSA*, 2001.

[14] S. Halevi. Advanced cryptorgaphy: Promise and challenges. https://shaih.github.io/pubs/Advanced-Cryptorgaphy.pdf, 2018.

[15] J. Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732. ACM, 1992.

[16] Y. Kong, C. Peikert, G. Schoenebeck, and B. Tao. Outsourcing computation: The minimal refereed mechanism. In *Web and Internet Economics*, 2019.

[17] A. Küpçü. Incentivized outsourced computation resistant to malicious contractors. *IEEE Transactions on Dependable and Secure Computing*, 14(6):633–649, 2017.

[18] D. Molnar. The seti@home problem. *ACM Crossroads*, Sep 2000.

[19] A. Patrizio. The coronavirus pandemic turned folding@home into an exaflop supercomputer, April 2020.

[20] V. Pham, M. Khouzani, and C. Cid. Optimal contracts for outsourced computation. In *GameSec*, 2014.

[21] A. Rapoport. Prisoner's dilemma - recollections and observations. *Game Theory as a Theory of Conflict Resolution*, 1974.

[22] Rosetta@home. http://boinc.bakerlab.org/rosetta.

[23] S. Ruoti, B. Kaiser, A. Yerukhimovich, J. Clark, and R. Cunningham. Sok: Blockchain technology and its potential use cases. *arXiv:1909.12454*, 2019.

[24] L. F. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Computer Systems*, 18(4):561–572, 2002.

[25] Seti@home. http://setiathome.berkeley.edu.

[26] D. Szajda, B. Lawson, and J. Owen. Hardening functions for large scale distributed computations. In *IEEE Security and Privacy*, 2003.

[27] D. Szajda, B. Lawson, and J. Owen. Toward an optimal redundancy strategy for distributed computations. In *IEEE Cluster Computing*, 2005.

[28] J. Teutsch and C. Reitwießner. A scalable verification solution for blockchains. *arXiv:1908.04756*, 2019.

[29] H. Ulusoy, M. Kantarcioglu, and E. Pattuk. Trustmr: Computation integrity assurance system for mapreduce. In *IEEE Big Data*, 2015.

[30] M. van Dijk and A. Juels. On the impossibility of cryptography alone for privacy-preserving cloud computing. In *USENIX HotSec*, 2010.

[31] M. Walfish and A. J. Blumberg. Verifying computations without reexecuting them. *Communications of the ACM*, 58(2):74–84, 2015.

A *Judge* Subprotocols

Verifying a Single Step of a Committed Computation. Consider two reduced configurations $rc_1 = (s_1, h_1, v_1, r_1)$ and $rc_2 = (s_2, h_2, v_2, r_2)$ that are claimed to be consecutive, and a proof of consistency p_1 for the first configuration where $p_1 = MHproof(t_1, h_1)$ where t_1 is the rc_1 configuration tape. In [5], it is shown that the referee can efficiently verify this claim by simulating a single step of the Turing machine on (s_1, h_1, v_1), and comparing the results with the values in rc_2, and outputs $True$

if the claim is valid and rc_2 is consistent with the reduced state that results from a correct single step starting from rc_1, and *False*, otherwise.
We introduce $VerifyCommittedReducedStep(rc_1, rc_2, p_1)$ that requires the published root of RC, and takes as input $rc_1, MH_{proof}(RC, rc_1), rc_2, MH_{proof}(RC, rc_2), p_1)$, where $MH_{proof}(RC, rc_i)$ is the consistency proof of rc_i against the published $MH_{root}(RC)$. Our modified $VerifyCommittedReducedStep()$ first checks the consistency of rc_i with the published root, and if True, proceeds to the single step verification as above.
Committed Binary-search Protocol. The protocol is described in Algorithm 6. It uses a binary-search subprotocol similar to [5], but with the extra checking of the proof $MH_{proof}(rc_w^i, RC^i)$ that verifies reduced configuration rc_w^i against the Merkle root of RC^i. This check is critical to prevent the copy attack during this phase, and ensures that the submitted reduced configuration belongs to the committed Merkle root. The binary search works as follows. Assume that $z^i = MH_{root}(RC^i), i = 1, 2$, for the two contractors are published. The SC asks each contractor to send the number of computation steps needed for $f(x)$, takes the smaller of the two as n_b (bad index), and sets n_g (good index) to 1. The SC then asks for the reduced configuration at $(n_b - n_g)/2 + n_g$, together with the proof of consistency of the reduced configuration with respect to the corresponding z^i. Depending on the match/mismatch of the two reduced configurations, a new query for the reduced configuration at the half interval point $[n_g, (n_b - n_g)/2 + n_g]$ or $[(n_b - n_g)/2 + n_g, n_b]$ will be formed. This process is repeated until $n_b = n_g + 1$.

B Incentivized Computation in the SC Setting with Incorrect Equilibrium

We now formally show that Belenkiy et al. and follow-up works [4, 20, 17] all lead to incorrect results for the *Problem Giver* in the equilibrium, when taken directly to the SC setting.
Analysis of the problem. First, among the copy strategies, *Copy-Guess* (CG) is the only (rational) meaningful strategy. The reason is the following: When a Copy contractor succeeds in copying, the result will be accepted regardless of the Copy contractors' type (since the *Judge* protocol is only run when $(y_1, z_1) \neq (y_2, z_2)$, but this is not the case when copying succeeds), and therefore the reward will be obtained. This happens even when two Copy contractors get matched, due to assumption (10). Since the Guess strategy has the lowest cost (see assumptions (5) and (6), the CG strategy dominates the other copy strategies. Hence, in the rest of our analysis here, we will only consider CG as the copy strategy. Furthermore, CG is always better than G, since copying may result in a better utility, while its fallback strategy (if copying is unsuccessful) is guessing anyway. Therefore, in our formal analysis below, we will consider the two originally-considered strategies (D, L) together with the new CG strategy.
Belenkiy et al. introduce *bounty* is an extra compensation that a contractor receives if they help to identify a "cheating" contractor. That is, the bounty is only paid when the two responses do not match. When *Judge* protocol identifies one response as correct and the other as incorrect, the contractor who submitted the correct response gets the bounty, in addition to the reward. Unfortunately, in the SC setting, even using bounty, the equilibrium leads to incorrect results for the *Problem Giver*.
This can be seen using Table 3. Below, we detail the utilities in the table (we follow the row order):

- The utility of the strategy D against D or C is $u_D = r - cost(1)$: The results would match, and the contractor will receive the reward, while paying the cost of the computation.

P_1, P_2	D	L	CG
D	u_D, u_D	u_{DB}, u_-	u_D, u_C
L	-	u_L, u_L	u_L, u_C
CG	-	-	u_C, u_C

Table 3: Utility table of Belenkiy et al. and follow-up works in the SC setting. Note that the utilities in the table represent a symmetric game (not a symmetric matrix), thus unnecessary cells are omitted.

- The utility of the strategy D against L is $r - cost(1) + b(1-\epsilon)$: When D is matched against L, the probability that they return the same *response* is negligible as in assumption (4).[4] The Diligent contractor will additionally obtain the bounty otherwise. We approximate this utility as $r - cost(1) + b(1-\epsilon) \approx r - cost(1) + b$ and denote as u_{DB}.
- The utility of the L strategy against the D strategy is $r\epsilon - f(1-\epsilon) - cost(q) < 0$: We use ϵ as the negligible probability that the responses match and then approximate it as zero. This means $r\epsilon - f(1-\epsilon) - cost(q) \approx -f - cost(q)$ which is negative, and we simply denote it in the table with u_-.
- The utility of the Copy strategy CG against any other strategy is $u_C = r - cost(\epsilon)$: They always end up submitting the same answer: Either they manage to copy the other's answer, or the other Copy contractor copies their answer. Hence they always obtain the reward, and only pay $cost(\epsilon)$.
- The utility of strategy L, against another non-Diligent strategy (L or CG) is $u_L = r - cost(q)$: If the other contractor is L or CG, they both return the same result, hence they both get the reward. In any case, the contractor pays the cost of the q-algorithm.

Theorem B.1. *Under the reasonable assumptions that are stated in Section 4, the pair of strategies (CG, CG) gives the only computational Nash equilibrium of the strategic game in Table 3.*

Proof. First, we show that if the other contractor is L, then the best response is D. This can be shown by proving two inequalities ($u_C > u_L$ and $u_{DB} > u_C$). First, we show:

$$\begin{aligned} u_C &> u_L \\ r - cost(\epsilon) &> r - cost(q) \end{aligned} \tag{5}$$

which is obvious since $cost(q) > cost(\epsilon)$. Second, we have:

$$\begin{aligned} u_{DB} &> u_C \\ r - cost(1) + b &> r - cost(\epsilon) \end{aligned} \tag{6}$$

We note that previous works [4, 20, 17] set $b > r > cost(1)$, which implies that $u_{DB} > u_C$.[5]
Next, consider that the other contractor is D. We show that the best response is CG. To show that, we need to show $u_C > u_D$ and $u_C > u_-$. The one involving the negative utility u_- is obvious:

$$\begin{aligned} u_C &> u_- \\ r - cost(\epsilon) &> 0 \end{aligned} \tag{7}$$

[4] Obtaining the same result has probability q, but same response has probability ϵ

[5] In the case that $b < cost(1) - cost(\epsilon)$, we would have CG as the best response against L, still making the proof valid.

per assumptions (1), (5), and (6). The other inequality is:

$$\begin{aligned} u_C &> u_D \\ r - cost(\epsilon) &> r - cost(1) \end{aligned} \tag{8}$$

which is obvious since $cost(1) > cost(\epsilon)$. Hence, (D, D) cannot be an equilibrium.
Lastly, we need to discuss what this contractor should do when the other contractor is CG. Observe that equations (8) and (5) already show that this contractor should not be Diligent or Lazy, and instead should also be CG. This makes the strategy pair (CG, CG) the only Nash equilibrium. □

Corollary. The computation result returned to the *Problem Giver* by the SC is incorrect, failing to achieve the foremost goal of outsourced computation in Section 3.

C Analysis without Match Check

We show that using the Mismatch Check protocol without the Match Check protocol does not lead to the desired equilibrium of (D, D) strategy pair. In this extended world, the six strategies defined are available to the contractors, and the SC employs the Mismatch Check protocol. The limited *Judge* protocol runs Mismatch Check when $(y_1, z_1) \neq (y_2, z_2)$, but does not run the Match Check. Essentially, when the two contractors, denoted by $P_i, i = 1, 2$, each send a pair $(y_i, z_i), i = 1, 2$, the SC runs the following *Judge* protocol.

- If $(y_1, z_1) = (y_2, z_2)$, reward both contractors and accept y_1 as the computation result.
- Else, when $(y_1, z_1) \neq (y_2, z_2)$, run an interactive *Mismatch Check* protocol with the two contractors.

This *Judge* protocol identifies a Diligent contractor (if any) and definitely rewards them. For example, in the case of the mismatch between a Guessing contractor and a Diligent contractor, the Diligent contractor always wins because the computation step that differs between the two will be run by the SC according to the specification of the correct algorithm that is known to the SC. But, the *Judge* protocol may also incorrectly reward non-diligent contractors with incorrect responses. This is because, the Committed Binary-Search will only be run when the two responses do not match, and when the two responses are from a Lazy contractor and a Guessing contractor, the Lazy contractor that has run the q-algorithm and has committed to a Merkle root with respect to the RC sequence of the q-algorithm, could be able to correctly answer the SC queries against G. Overall, it is possible that non-diligent contractors may also get the reward, instead of being fined. The *Judge* protocol, however, will never fine a Diligent contractor.
Lastly, among the copy strategies, *Copy-Guess* (CG) is the only (rational) meaningful strategy in this setting. The reason is the following: When a Copy contractor succeeds in copying, the result will be accepted regardless of the Copy contractors' type (since the Mismatch Check protocol is only run when $(y_1, z_1) \neq (y_2, z_2)$, but this is not the case when copying succeeds), and therefore the reward will be obtained. This happens even when two Copy contractors get matched, due to assumption (10). Since the Guess strategy has the lowest cost (see assumptions (5) and (6), the CG strategy dominates the other copy strategies. Hence, in the rest of our analysis in this section, we will only consider CG as the copy strategy.

P_1, P_2	D	L	G	CG
D	u_D, u_D	u_D, u_-	u_D, u_-	u_D, u_C
L	-	u_L, u_L	u_L, u_-	u_L, u_C
G	-	-	u_-, u_-	u_C, u_C
CG	-	-	-	u_C, u_C

Table 4: Utility table, *without bounty*. Note that the utilities in the table represent a symmetric game (not a symmetric matrix), thus unnecessary cells are omitted.

C.1 Without Bounty

Table 4 gives the utilities of P_1 and P_2, when the contractors use the strategies listed in Section 3. The utilities are symmetric, and so we only need to discuss the upper half of the table. We first analyse the system without bounty, and then in Section C.2, we consider bounty.

Observe that a Diligent contractor would always receive the reward, and G and L strategies may receive the reward with some probability. As noted in the discussion above, a Copy contractor always succeeds against non-Copy strategies. Moreover, since we are interested in the computational Nash equilibrium, we ignore negligible factors and for simplicity of presentation, show them as zero. To simplify the presentation, we use u_- notation to denote any utility that is negative. Below, we detail the utilities in the table (we follow the row order):

- The utility of the D strategy, independent of the other contractor is $u_D = r - cost(1)$: They obtain the reward, and pay the cost of the original algorithm.
- The utility of the L strategy against the D strategy is $r\epsilon - f(1-\epsilon) - cost(q) < 0$: Remember that when a Lazy and Diligent contractor get matched against each other, the probability that they return the same response is negligible as in assumption (4). Hence, we use ϵ as the negligible probability and then approximate it as zero. This means $r\epsilon - f(1-\epsilon) - cost(q) \approx -f - cost(q)$ which is negative, and simply denote in the table with u_-.
- The utility of the G strategy, against non-Copy strategies (D, L, or G) is $r\epsilon - f(1-\epsilon) - cost(\epsilon) < 0$: Similar to the reasoning above, $r\epsilon - f(1-\epsilon) - cost(\epsilon) \approx -f - cost(\epsilon)$ which is negative, and we simply denote it in the table with u_-.
- The utility of the Copy strategy CG against any other strategy is $u_C = r - cost(\epsilon)$: They always end up submitting the same answer: Either they manage to copy the other's answer, or the other Copy contractor copies their answer. Hence they always obtain the reward, and only pay $cost(\epsilon)$. [6]
- The utility of strategy L, against any non-Diligent strategy (L, G, or CG) is $u_L = r - cost(q)$: If the other contractor is L or CG, they both return the same result, hence they both get the reward. If the other is G, then the *Judge* protocol may still mistakenly identify this L contractor as Diligent, and would provide the reward, depending on the q-algorithm. Since we assume the worst-case q-algorithm, we assume that L is not caught against G, and is indeed rewarded. In any case, they pay the cost of the q-algorithm.

Before providing the full theorem and its proof, we provide the intuition regarding the equilibrium. Observe that a Copy-Guess strategy always obtains the reward, with minimal cost. This is because,

[6] We simplified the cost of Guess and Copy strategies both as $cost(\epsilon)$. This is why the G strategy obtains u_C against Copy strategies (since Copy contractors will simply copy the guessed response and the two results will always match, thereby not being caught by the *Judge* protocol).

when matched against a non-Copy contractor, the Copy contractor will simply copy their response, and the *Judge* protocol will not perform verification: the response will be accepted and the contractors will be rewarded. When two Copy contractors get matched, then either P_1 or P_2 will manage to copy (discarding the negligible probability that neither could copy due to very similar random timings). In this case, again both responses will match, regardless of which one guessed and which one copied, and the *Judge* protocol will accept that response without further verification. Below, we formally prove that (CG, CG) is the equilibrium.

Theorem C.1. *Under the reasonable assumptions that are stated in Section 4, the pair of strategies (CG, CG) gives the weak computational Nash equilibrium of the strategic game in Table 4.*

Proof. We will prove that CG is the best response against any other strategy, making (CG, CG) pair the (weak) equilibrium.
First, consider that the other contractor is D. We show that the best response is CG. To show that, we need to show $u_C > u_D$ and $u_C > u_-$. The one involving the negative utility u_- is obvious:

$$\begin{aligned} u_C &> u_- \\ r - cost(\epsilon) &> 0 \end{aligned} \tag{9}$$

per assumptions (1), (5), and (6). The other inequality one is:

$$\begin{aligned} u_C &> u_D \\ r - cost(\epsilon) &> r - cost(1) \end{aligned} \tag{10}$$

which is obvious since $cost(1) > cost(\epsilon)$.
Next, we show that if the other contractor is L, then the best response is again CG. This can be shown by proving three inequalities ($u_C > u_D$ and $u_C > u_-$ and $u_C > u_L$), two of which are already proven above, and the other one we prove below:

$$\begin{aligned} u_C &> u_L \\ r - cost(\epsilon) &> r - cost(q) \end{aligned} \tag{11}$$

which is obvious since $cost(q) > cost(\epsilon)$.
Next is to show that CG is the best response against G. This can be proven via three inequalities ($u_C > u_-$ and $u_C > u_D$ and $u_C > u_L$). Indeed, equations (9), (10), and (11) already show that CG is the best response against G.
Lastly, we need to discuss what this contractor should do when the other contractor is CG. Observe that equations (10) and (11) already show that this contractor should not be Diligent or Lazy. This contractor is indeed indifferent between G and CG strategies in this case. This makes the strategy pair (CG, CG) the weak Nash equilibrium. □

Corollary. The computation result returned to the *Problem Giver* by the SC is incorrect, failing to achieve the foremost goal of outsourced computation in Section 3.

C.2 With Bounty

Bounty is an extra compensation that a contractor receives if they help to identify a "cheating" contractor. That is, the bounty is only paid when the two responses do not match. When *Judge*

P_1, P_2	D	L	G	CG
D	u_D, u_D	u_{DB}, u_-	u_{DB}, u_-	u_D, u_C
L	-	u_L, u_L	u_{LB}, u_-	u_L, u_C
G	-	-	u_-, u_-	u_C, u_C
CG	-	-	-	u_C, u_C

Table 5: Utility table, *with bounty*. Note that the utilities in the table represent a symmetric game (not a symmetric matrix), thus unnecessary cells are omitted.

protocol identifies one response as correct and the other as incorrect, the contractor who submitted the correct response gets the bounty, in addition to the reward. Belenkiy et al. [4] showed that this extra payment of bounty will result in a unique Nash equilibrium that corresponds to the two contractors using Diligent strategy, resulting in the *Problem Giver* obtaining the correct computation result. In this section, we show that bounty in the SC setting is not enough to incentivize the correct behavior.

This can be seen using the right side of Table 1. In the first two rows of the right side of the table, the Diligent contractor will obtain the bounty, whereas in the last row, no contractor obtains the bounty. The interesting case is the third row, where L and G contractors' responses are compared. As described earlier, both strategies can respond correctly to a single challenge. However, the mismatch between the two responses reduces to a single computation step that will be performed by the SC, and could accept the response of L, as L has performed a computation, which, on the queried step, may match the correct computation. The G strategy, however, will fail because its rc's do not belong to a computation.

Bounty only affects the utilities below:

1. The utility of the strategy D against L or G strategies, including bounty, is $r-cost(1)+b(1-\epsilon)$: When D is matched against L or G, the contractor will additionally obtain the bounty unless both responses match, which has negligible probability due to hashing. We approximate this utility as $r - cost(1) + b(1 - \epsilon) \approx r - cost(1) + b$ and denote as u_{DB}.
   Remark: There is no change for D versus other strategies (D or CG), since they will return the same response and hence no one obtains the bounty.
2. The utility of the L strategy against G strategy, including bounty is $r-cost(q)+b(1-\epsilon)$: When L is matched against G, except with negligible probability that the responses match, the *Judge* protocol may (mistakenly) reward the L strategy and provide extra bounty, while fining the G strategy of the contractor. Similarly, we approximate this utility as $r - cost(q) + b(1 - \epsilon) \approx r - cost(q) + b$ and denote it as u_{LB}.

Thus three cells in the table will be affected (D vs L, D vs G, L vs G), resulting in Table 5 for the case with bounty.

Bounty does not help in our setting, because copying is still a meaningful strategy. Note that when the copier succeeds to copy, both responses would be the same, and hence the Match Check protocol would not be run, and no one obtains the bounty. Furthermore, as discussed above, bounty incentivizes both Diligent and Lazy strategies. The following theorem formally states that the equilibrium does not change with the bounty in our framework.

Theorem C.2. *Under the reasonable assumptions that are stated in Section 4, the pair of strategies (CG, CG) gives the weak computational Nash equilibrium of the strategic game in Table 5.*

Proof. Observe that, when the other contractor is Diligent, CG is still the best response, as shown by equations (9), (10), and (11).
Further, $u_{LB} > u_{DB}$, meaning that it is better to be Lazy than Diligent against G, since:

$$\begin{gathered} u_{LB} > u_{DB} \\ r - cost(q) + b > r - cost(1) + b \end{gathered} \tag{12}$$

which holds because $cost(q) < cost(1)$.
Moreover, when the other contractor is Lazy, it is better to be Diligent than Guessing, since:

$$\begin{gathered} u_{DB} > u_{-} \\ r - cost(1) + b > 0 \end{gathered} \tag{13}$$

which is the case due to assumption (1).
Therefore, (CG, CG) strategy pair remains a weak equilibrium. □

We further show that the desirable matchings (D, D or D, L or D, G or D, CG) that result in the acceptance of the correct result by the *Problem Giver* cannot be made an equilibrium regardless of the amount of bounty employed. Note that with the *Judge* protocol run by the SC, at least one of the contractors need to be Diligent for the accepted result to be correct.

Theorem C.3. *No non-negative value of the bounty can make (D, D) or (D, L) or (D, G) or (D, CG) strategy pair to be an equilibrium of the strategic game in Table 5.*

Proof. (outline) Firstly, if the other contractor is D, we already showed that CG is the best response. Hence, (D, D) cannot be an equilibrium.
Second, (D, CG) cannot be an equilibrium either, since if the other contractor is CG, then this contractor should be CG or G as discussed in the proof of Theorem C.1.
Third, (D, G) cannot be an equilibrium, since if the other contractor is G, then this contractor should better be L than D as shown in the proof of Theorem C.2.
Lastly, (D, L) cannot be an equilibrium, since if the other contractor is L, and then this contractor chooses the D strategy, then the other contractor would switch to CG as shown above. □

Corollary. The computation result returned to the *Problem Giver* by the SC is incorrect, failing to achieve the foremost goal of outsourced computation in Section 3.