Marvel DC: A Blockchain-Based Decentralized and Incentive-Compatible Distributed Computing Protocol

immediate

I. Abstract

We outline Marvel DC, a fully decentralized blockchain-based distributed-computing protocol which formally guarantees that computers are strictly incentivized to correctly perform requested computations. Marvel DC utilizes a reputation management protocol to ensure that, for any minority of computers not performing calculations correctly, these computers are identified and selected for computations with diminishing probability. We then outline Privacy Marvel DC, a privacy-enhanced version of Marvel DC which decouples results from the computers which computed them, making the protocol suitable for computations such as Federated Learning, where results can reveal sensitive information about that computer that computed them. We provide an implementation of Marvel DC and analyses of both protocols, demonstrating that they are not only the first protocols to provide the aforementioned formal guarantees, but are also practical, competitive with prior attempts in the field, and ready to deploy.

II. Introduction

The distributing of computation among computers has beckoned a never-before-seen level of computing power available to average users with access to as little as a smart phone and an average internet connection. Distributed computations (DCs) have typically been outsourced directly to one of the few centralized Big Tech. providers such as Amazon Web Services, or Google Cloud. Unfortunately, centralized services like these have many drawbacks. Monopoly of resources, centralized trust, restricted access for certain clients, and in the case of Federated Learning (FL), lack of diverse datasets make these centralized services unfit for many users and purposes. Full decentralization solves all of these issues. Unfortunately, decentralizing distributed-computing protocols brings many new challenges that are largely protected against in the centralized setting.

A significant issue in many existing decentralized DC implementations [1]–[8] is the accurate rewarding of computers to incentivize rational computers, computers who try to maximize their utility (e.g. in cryptocurrency tokens), to correctly perform outsourced computations. This is a consequence of the decentralized settings in which blockchain protocols are typically intended to operate in. In a decentralized protocol, players can only be expected follow an action(s) if that action(s) is(are) strong incentive compatible, resulting in strictly higher payoffs than the alternatives.

One method to combat this is, for a given computation, to use zero-knowledge (ZK) tools to prove that a computer performed the actions as prescribed by the requester [9]. Although theoretically any computation can be encoded in this way, the practicality of requiring requesters, typically with low computing power by the nature of outsourcing, to encode their computation as a ZK-circuit to allow for the proving of correct computation is an open question.

Given the 10s of billions of US dollars in revenue being generated by centralized systems [**?**] without guarantees of fair-pricing, decentralized access or computational output quality, there is clear motivation for a decentralized DC protocol that can provide such guarantees. To this end, we present Marvel DC, which formally provides all of these guarantees, beckoning the age of decentralized DC outsourcing, free from the previously unchecked stranglehold of private monopolies.

A. Our Contribution

We present Marvel DC, a blockchain-based decentralized DC protocol which addresses the many gaps that exist in outsourcing computations without the use of a trusted third-party (TTP). Namely, Marvel DC provides for the first time in literature a decentralized DC protocol which ensures rational computers are strongly incentivized to follow the protocol. Furthermore, Marvel DC utilizes reputations to isolate correctly-performing computers when selecting computers for computations. This allows Marvel DC to efficiently remove adversarial computers from the protocol. As these reputations are maintained on the blockchain itself, through careful construction (Section **??**) this reputation protocol neither affects the decentralisation or strong incentive compatibility of the protocol.

Our generic description of Marvel DC can be adapted to run on any smart-contract enabled blockchain, and as such, can make use of the vast existing communities which exist on such blockchains. This is a further improvement on protocols which require the recruitment and constant participation of an independent network of computers. In a blockchain, where computers form a subset of users, computers can be dormant until required to perform a computation. By deploying on an existing blockchain, any player in that blockchain can also participate in Marvel DC as a computer and/or requester. We summarize the main contributions of Marvel DC as follows:

Strong incentive compatible: In Section V, we describe how to program rewards such that it is strong incentive compatible for every rational player (requesters, computers and/or block-producers) in the blockchain system to follow the protocol. This is formalized in Theorem VI.1.

Handling of symmetric/asymmetric utilities: By tokenizing the protocol rewards, we are able to handle both symmetric (rational computers and requesters only want to produce good results) and asymmetric (rational computers want to be compensated financially) utilities. Thus Theorem VI.1 can be applied to both symmetric and asymmetric utilities.

Decentralization: The description of Marvel DC in Section VI-A can be translated for use on any tokenized smart-contract enabled blockchain, of which many (including Ethereum[1] and Harmony[2]) are considered truly decentralized systems due to their permissionless nature. We demonstrate this by providing a proof-of-concept implementation of Marvel DC [10] that can be deployed within such decentralized systems.

We then outline a series of privacy-enhancing amendments for Marvel DC, culminating in Privacy Marvel DC, a protocol in which results cannot be linked to the computer which provided the result, except with prohibitive additional cost to the player performing the revelation. Privacy Marvel DC retains all of the decentralization and incentive compatibility guarantees of Marvel DC, while also adding a layer of privacy which makes it appropriate for sensitive computations where knowing a certain computer computed a particular result can be used to infer private/unwanted information about the computer. This is particularly interesting in the case of FL, where computers are asked to use private data sets. To do this, we make use of existing NIZK set-membership tools. Although our privacy techniques are not novel outside of DC, the combination of decentralization, provable strong incentive compatibility and the ability to apply one smart-contract instance of Privacy Marvel DC to any computational problem with output in Euclidean space (summarized in Table I) stands as a further novel contribution.

III. Related Work

Protocol	Tokenized Rewards	Strong Incentive Compatible	Computation-Independent	Diminishing Adversarial Selection Prob.
[1]–[7]	✗	✗	✓	✓/✗
Toyoda et al. [8]	✓	✗	✓	✗
Ruckel et al. [9]	✓	✓	✗*	✗
Marvel DC	✓	✓	✓	✓

TABLE I: Comparison of incentive-aware FL protocol designs. *Computation-independence refers to the ability of a particular smart-contract encoding to be re-used for many computations. The ZK circuit-encodings of [9] must be generated anew for each type of computation, placing significant upfront costs on computation requesters.

Blockchain-based DC protocols [1]–[7] have seen significant interest. Unfortunately, all of these papers consider a blockchain or distributed system in which all parties share one utility, that is, the ability to use/benefit from a well-trained shared model, which gives correct behaviour by definition. Such an assumption makes these protocols and their resulting analyses inappropriate in the presence of untrusted peers and/or asymmetric utilities.

In [8], a protocol and general framework for incentive mechanism design, within FL protocols where players measure utility tokenomically, are proposed. Computer registration and reward distribution must all be performed by players within the system. Computer registration is performed by an "administrator", which prevents decentralization and encourages collusion. Furthermore, computation rewards are distributed based on votes of players within the system. The incentive compatibility of this choice is not considered, and is non-trivial to implement. In Marvel DC, rewards are distributed deterministically as part of the smart-contract execution on requester inputs. We prove that rational requesters always submit messages correctly to the blockchain, and thus, correct rewarding is SINCE. Moreover, [8] has no mechanism to identify Byzantine computers and diminish/remove their ability to participate in the protocol. In Marvel DC, this is achieved using reputations.

Recently, an extensive survey of existing attempts to construct privacy-preserving FL protocols [11] was published. Of the investigated works, the most promising for achieving an incentive-compatible decentralized protocol is [9]. In [9], ZK-proofs are used in the computing stage to prove that a computation was performed correctly. In a decentralized setting however, this raises many challenges, as each type of computation requires its own ZK circuit-generation/trusted set-up to generate the target function. In contrast, Marvel DC can be implemented using existing, blockchain-deployed ZK-tools and generalised rewarding functions. Moreover, as acknowledged by the authors of [9], although their methodology ensures models were trained correctly, it does not guarantee the models were trained on appropriate data. A proposed solution is using "certified sensors", equivalent to TTPs, a non-viable solution in a decentralized setting. As the rewarding functions in Marvel DC reward players based on the relative quality of the results, and not just based on the fact that a series of computations has been performed correctly, independent of the data on which the computations are being performed, we are able to avoid this issue.

IV. Preliminaries

A. Blockchains

In this paper, we are interested in a distributed set of n players $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$ interacting with one and other inside a blockchain protocol. These players send and receive stake among one another, along with the functionality to encode programs to run within the blockchain protocol in the form of *smart contracts*. A foundational assumption for the functioning of blockchain protocols is that a majority of players in the protocol act *rationally*, following the protocol if they are incentivized to do so. The non-rational players are known as

[1]https://ethereum.org/en/

[2]https://www.harmony.one/

Byzantine, and are modelled as deviating from the protocol either maliciously or randomly, and being controlled by a single adversary. This player model is known as the ByRa model [12], which we use in this paper. The ByRa model is a necessary improvement on the legacy BAR model [13] for the true consideration of incentives in distributed systems, removing any dependencies on altruistic, honest-by-default players which cannot be assumed to exist in incentive-driven protocols like blockchain/DC protocols.

Definition IV.1. The *ByRa model* consists of *Byzantine* and *Rational* players. A player is:

- *Byzantine* if they deviate arbitrarily from a recommended protocol with unknown utility function. Byzantine players are chosen and controlled by an adversary $\mathcal{A}$.
- *Rational* if they choose the strategy which maximizes their utility assuming all other players are rational.

It is the aim of this paper to construct a DC protocol that ensures rational players always follow the protocol, a property known as strong incentive compatibility in expectation. In this paper, rational players utility is measured in blockchain-based tokens. The blockchain protocol acts conceptually as a public ledger managed by a TTP. In reality, it is the following of the blockchain protocol by some majority of players using the blockchain that replicates this TTP. The blockchain protocol provides availability and correctness of the programs being run by the blockchain protocol, but does not provide privacy. That is, any player can observe the current state of all programs being run on the blockchain, and can verify that this state has been reached through the correct running of these programs. However, player inputs to these programs must be committed publicly to the blockchain before they can be passed to the smart contract, and as such, it will be an important requirement of designing a protocol involving smart contract interaction, through transactions, that the blockchain will accept these transactions in a timely fashion. In our system, this is achieved using incentivization.

B. Non-Interactive Zero-Knowledge Set Membership

The aim of this section is to outline existing *non-interactive zero-knowledge* (NIZK) tools for set membership, such as those stemming from papers like [14]–[18], which are used in our privacy-enhanced DC protocol. We define these tools generically, allowing for the adoption of any secure NIZK set-membership protocol into , as we only require a common functionality that is shared by all of them.

We define a commitment scheme *commit*, a set-membership proof scheme *SetMembership*, an NIZK proof of knowledge scheme *NIZKPoK* and a NIZK signature of knowledge scheme (*NIZKSoK*). We do not specify which instantiation of these schemes to use, as the exact choice will depend on several factors, such as efficiency, resource limitations and/or the strength of the assumptions used. Provers privately generate two bit strings, the *serial number* $\mathcal{S}$ and *randomness* $\mathcal{R}$, with $\mathcal{S},\ \mathcal{R} \in \{0,1\}^{\Theta(\kappa)}$. Provers then commit to these values by generating a commitment $com \leftarrow commit(\mathcal{S}, \mathcal{R})$. This com is then broadcast to the verifier, with the set of all commitments denoted by *Com*. It is this set of commitments *Com* to which the prover seeks to prove membership.

- $commit(m)$: A deterministic, collision-resistant function taking as input a string $m \in \{0,1\}^*$, and outputting a string $com \in \{0,1\}^{\Theta(\kappa)}$.
- $SetMembership(com, Com)$: Compresses a set of commitments *Com* and generates a membership proof π that *com* is in *Com* if $com \in Com$.
- $NIZKPoK(\mathcal{R}, \mathcal{S}, Com)$: For a set of commitments *Com*, returns a string $\mathcal{S}$ and NIZK proof of knowledge that the player running $NIZKPoK()$ knows an $\mathcal{R}$ producing a proof when running $SetMembership(commit(\mathcal{S}||\mathcal{R}), Com)$. This revelation identifies to a verifier when a proof has previously been provided for a particular, albeit unknown, commitment as the prover must reproduce $\mathcal{S}$. This is used to enforce the correct participation of players.
- $NIZKSoK(m)$: Returns a signature of knowledge that the player who chose m can also produce NIZKPoK.

In this paper, we assume the public NIZK parameters are set-up in a trusted manner[3].

C. Relayers

A fundamental requirement for transaction submission in blockchains is the payment of some transaction fee to simultaneously incentivise block producers to include the transaction, and to prevent denial-of-service/spamming attacks. However, in both the unspent transaction- and account-based models, this allows for the linking of player transactions, balances, and their associated transaction patterns.

To counteract this, we utilise the concept of *transaction relayers*, such as those used in 0x[4], Open Gas Station Network[5], and Biconomy[6]. When the user wishes to submit a transaction anonymously to the blockchain, the user publishes a proof of membership in the set of registered users to the relayer mempool, as well as the desired transaction and a signature of knowledge cryptographically binding the membership proof to the transaction, preventing tampering.

V. Constructing a Strong Incentive Compatible DC Protocol

We first describe an idealized protocol for the distribution of computation between a set of computers. We then demonstrate how to instantiate such a protocol using existing blockchain technology. In this description, we consider a requester who has a computation *calc* whose calculation the requester seeks to outsource to some subset of available computers $\mathfrak{C}$. Furthermore, the requester is aware of a threshold n_ψ such that for any random sample of n_ψ computers without replacement from $\mathfrak{C}$, a majority of those computers are rational (see Table II).

[3]Such as a Perpetual Powers of Tau ceremony, as used in Zcash https://zkproof.org/2021/06/30/setup-ceremonies/

[4]0x https://0x.org/docs/guides/v3-specification

[5]Open Gas Station Network https://docs.opengsn.org/

[6]Biconomy https://www.biconomy.io/

We consider the output of all computations in this paper as a point in l-dimensional space. To reason about the "goodness" of a computation result, for every computation we assume the existence of a deterministic function which takes the set of computation responses, and given a majority of correctly computed results, outputs a target value $\varkappa$, which is computed correctly with probability $1 - \epsilon$, for some $\epsilon < 0.5$. For deterministic calculations, the mode is such a function. In FL where results are model gradients, the Krum function [19] is such an aggregation function.

Consider the set of computation results $\{result_1, ..., result_{n_{comp}}\}$. We require for any pair ($result_+$, $result_-$), with $result_+$ a correctly computed result and $result_-$ an incorrectly computed result the following holds:

$$P(distance(result_+, \varkappa) < distance(result_-, \varkappa)) > 0.5. \quad (1)$$

Given this, for any subset of computation results taken in ascending order using the function *distance*, the expected number of these computations being correctly computed is greater than those being incorrectly computed. With respect to deterministic computations, letting $\varkappa$ be the median or mode, all correctly computed results will be distance 0 to $\varkappa$. For non-deterministic computations, it is less clear. The function used to compute $\varkappa$ must be chosen such that Equation 1 holds.

Now, consider the following DC protocol run by a TTP who enforces the correct participation of all rational players. The proceeding sections then replace this TTP, in order to achieve full decentralization, through the use of a smart contract-enabled blockchain and a strong incentive compatible protocol (Marvel DC, described in Section VI) to be run therein.

Idealized distributed computing protocol. Requesters repeatedly enter the system with independent functions for computation. A requester wishing to avail of the distributed computation of some function *calc* submits *calc* to the TTP, as well as some number of computers $n_{comp} > n_\psi$. The TTP then selects n_{comp} from the set of available computers $\mathfrak{C}$. The computers respond to the TTP with their computation of *calc*, who then sends all of the computation results to the requester.

In a distributed setting, such TTPs do not exist. Therefore, to enforce the correct participation of rational players we need to utilize a mixture of cryptography and incentivization. With this in mind, we first describe how to generically construct a reward mechanism such that rational players are strongly incentivized to follow the protocol. We then outline a reputation management protocol which maintains this incentivization, while reducing the probability that incorrectly performing computers are selected for computation.

A. Reward Mechanism

We now provide a theoretical lower-bound on the per-computer reward required to strongly incentivize the correct participation of rational computers in our DC protocol. Running computations such as sorting arrays or encryption/decryption on-chain is expensive, so we initially give the requester the opportunity to correctly submit the set of computers to reward. Computers have an opportunity to contest this by depositing a bounty on-chain, triggering the on-chain verification of the reward set. Note that verification is much cheaper than computation, but with respect to Privacy Marvel DC, this reveals some private information, which is why verification does not take place automatically. If the contest is valid, the requester loses some initial escrow, herein lower-bounded. Otherwise, the computer loses the bounty. This is described in more detail in Section VI.

For a given computation *calc*, we assume an accurate a-priori lower-bound on the cost to compute a particular computation *calc* of $cost(calc)$. This lower-bound is known by all players in the system (in reality this value can be enforced by the protocol smart-contract). Given that the payment of $Fee(tx)$ per transaction guarantees timely inclusion in the blockchain, rational computers perform the calculation of *calc* if and only if:

$$E(Reward(calc)) > cost(calc) + 2Fee(tx). \quad (2)$$

$2Fee(tx)$ is needed as computers are required to send 2 messages to the blockchain. To more accurately describe $Reward(calc)$, we introduce $0 \leq \omega, \gamma \leq 1$ with the probability of good computations being rewarded being ω, and the probability of bad computations being rewarded being γ, such that without loss of generality $\omega > \gamma$. This gives an expected payoff of $\omega Reward(calc) - (cost(calc) + 2Fee(tx)) > 0$ for following the protocol, and an expected payoff of $\gamma Reward(calc) - 2Fee(tx)$ for submitting a result but not performing the computation. Therefore, we require:

$$\omega Reward(calc) - cost(calc) - \gamma Reward(calc) > 0 \quad (3)$$

This reduces to $Reward(calc) > \frac{cost(calc)}{\omega - \gamma}$. The exact values of ω and γ depend on the computation, number of computers to be rewarded and the chosen target function. Exact values for ω and γ are difficult to predict a-priori. For deterministic computations $\omega \approx 1$, whereas for non-deterministic computations such as FL, ω will be smaller. Lower-bounding the possible value of $\omega - \gamma$ (although greater than 0) and using this value to compute $Reward(calc)$ ensures rational players follow the protocol.

To ensure a rational requester correctly submits the set of good computations to be rewarded, any positive escrow amount $escrow_{req} > Fee(tx)$ suffices to strongly incentivize the requester to do this. This can be seen by considering the payoff for submitting the correct set, which is $escrow_{req} - Fee(tx) > 0$, while the payoff for not submitting the set is 0. In the case of potential collusion of up to k computers, setting $escrow_{req} \geq k \cdot Reward(calc) + Fee(tx)$ guarantees that the requester correctly submits the set of good computations. If k is set too small by the protocol/smart contract, not submitting the reward set, and rewarding all players may be positive expectancy for the requester. Setting k equal to n_{comp} conservatively achieves this. With this lower bound on $escrow_{req}$, rational requesters always submit the correct set of good computations to the blockchain.

B. Reputation Management Protocol

In this section we describe a reputation management protocol that maintains the incentive compatibility of a DC protocol with an incentive compatible reward mechanism, while also allowing us to prioritize good computers over bad computers, meaning both short- and long-term benefits for correctly behaving computers. We consider a rating mechanism $rate()$ which, under the same assumptions of the previous section, assigns good calculations a score of 1, and bad calculations a score of 0. For a player $\mathcal{P}_i$ taking part in computations for $calc_1, calc_2, ..., calc_k$, $\mathcal{P}_i$'s *base reputation* is $bR_i = \sum_{j=1}^{k} rate(calc_j)$.

Let $iR > 0$ be the starting reputation for computers registering in the system. For a given computation, we let $\mathcal{P}_i$ be selected as computer for a computation in block at height $\mathcal{H}$ in the blockchain in direct proportion to $bR_i^{\mathcal{H}-1}(iR-1)$ as a fraction of $\sum_{j=1}^{n} bR_j^{\mathcal{H}-1} - n \cdot (iR-1)$. With this in mind, the number of computations a player $\mathcal{P}_i$ is selected for is directly proportional to:

$$probSelect_i^{\mathcal{H}} = \frac{bR_i^{\mathcal{H}-1} - (iR-1)}{\sum_{j=1}^{n} bR_j^{\mathcal{H}-1} - n \cdot (iR-1)}. \tag{4}$$

Consider a player $\mathcal{P}_i$ who includes a good computation as block proposer for a computer $\mathcal{P}_j$. This increases $\mathcal{P}_j$'s base reputation, and thus $\mathcal{P}_j$'s *probSelect*, which has long-term stake implications (more selection = more reward & more reputation = more selection ...). This negatively affects the *probSelect* of $\mathcal{P}_i$ however. Therefore, we need to reward the proposer with an increase in base reputation to counteract the increase in the computers' expected increase in base reputation. Let $E_{rep} > 0$ be the expected change in base reputation for a computer whose computation gets included on the blockchain.

For $\mathcal{P}_i$ a proposer of a block that includes k transactions containing computation results, we need the following equality to hold:

$$\begin{aligned} probSelect_i^{\mathcal{H}} &= \frac{bR_i^{\mathcal{H}} - (iR-1)}{bR_i^{\mathcal{H}} + \sum_{j \neq i} bR_j^{\mathcal{H}-1} + k \cdot E_{rep} - n \cdot (iR-1)} \\ &= \frac{bR_i^{\mathcal{H}-1} - (iR-1)}{bR_i^{\mathcal{H}-1} + \sum_{j \neq i} bR_j^{\mathcal{H}-1} - n \cdot (iR-1)}. \end{aligned} \tag{5}$$

Solving for $bR_i^{\mathcal{H}}$ in this equality gives:

$$bR_i^{\mathcal{H}} = bR_i^{\mathcal{H}-1} + k \cdot E_{rep}\Big(\frac{bR_i^{\mathcal{H}-1} - (iR-1)}{\sum_{j \neq i} bR_j^{\mathcal{H}-1} - (n-1) \cdot (iR-1)}\Big). \tag{6}$$

This means we need to add $k \cdot E_{rep}\big(\frac{bR_i^{\mathcal{H}-1} - (iR-1)}{\sum_{j \neq 1} bR_j^{\mathcal{H}-1} - (n-1) \cdot (iR-1)}\big)$ to $bR_i^{\mathcal{H}-1}$ in order to ensure the proposer is impartial, with respect to reputation and computer selection probability, to adding transactions containing computation results to the blockchain. For transactions from the requester finalising the rewards, we simply have to replace E_{rep} with the actual mean change in reputation in Equation 6, and the rest of the numbers stay the same.

VI. Marvel DC

The goal of this section is to take the ideal DC functionality of Section V and it's properties, and implement them as a set of algorithms encoded as smart contracts that can be run by a decentralized set of players without a TTP, but with access to a blockchain. We call this protocol Marvel DC. We then formally prove in Section VI-B that within Marvel DC, rational computers always follow the protocol, performing computations correctly, through strong incentive compatibility. As demonstrated in Section III, this guarantee stands alone in the field of decentralized DC. Furthermore, as Marvel DC is provided in a generic manner, it can be deployed on any decentralized blockchain, allowing the Internet of Things never-before-witnessed access to DC.

A. Algorithmic Overview

Marvel DC is a set of smart contracts provided in pseudocode in Algorithms 1, 2 and 3. The main contracts corresponding to unique phases in the protocol are labelled *Register*, *Request*, *Response* and *Finalize*. A proof-of-concept Solidity implementation of Marvel DC has been made available on Github [10]. We provide here the intuition to these encodings.

A Marvel DC instance is initialized by calling the Request contract, and lasting at least T blocks, where T is the number of blocks required for players to observe an event on-chain, send a transaction and have that transaction committed on-chain given at least $Fee(tx)$ is paid. We provide here the intuition to these encodings. Each player $\mathcal{P}_i$ has exclusive access to a token balance bal_i which is stored as a globally readable variable on the blockchain. For a token ฿, $bal_i(฿)$ is the amount of token ฿ that $\mathcal{P}_i$ owns. Players in the underlying blockchain protocol can enter Marvel DC as computers by calling the Register contract, which for a given computer deposits an escrow $escrow_{comp}$ (line 44), granting that computer a reputation of iR (line 46).

A computation request specifies the computation details $calc$, the number of computers to be selected for the computation n_{comp}, a deterministic function $f_{\varkappa}$ for selecting the target result $\varkappa$ from the set of results, the number of computers to reward n_{reward}, and the per-computer reward $Reward_i$ received by a computer if included in the set of computers to reward. The requester also must deposit an escrow of $n_{reward} \cdot Reward_i + escrow_{req}$. The *compEncKey* is the public key corresponding to the temporary public/private key pair $(compEncKey, compDecKey)$. This is a key pair generated by the requester specifically for $calc$. A randomness beacon is called (line 50), which provides a pseudo-random seed for selecting n_{comp} computers to participate in the computation in direct proportion to computer reputations (line 5), with these computers listed in the set $calc.\mathfrak{C}$.

Selected computers can then submit results for *calc* to the blockchain by calling the Response contract, with results encrypted using *calc.compEncKey*. This encryption ensures no other computer can use another computer's result, and therefore must themselves perform the computation. Given a valid response is recorded, the block producer corresponding to the response is added to *calc.proposers*. This is later used to update reputations, in line with the analysis of Section V-B.

Then, either after T blocks from when the computation *calc* was requested, or when all computers in *calc*.$\mathfrak{C}$ have responded, the requester of *calc* can complete the request by calling the Finalize contract. Calling the Finalize contract requires the requester to provide the decryption key *calc.compDecKey* corresponding to *calc.compEncKey*. If these keys correspond to a valid key pair, the requester receives back her escrow $escrow_{req}$. The contract then uses *compDecKey* to decrypt the computer responses, and identify which computers are to be rewarded (line 19). This is done by applying the pre-specified target function to the computation results, and computing a target value $\varkappa$ (line 22). The computers corresponding to the $calc.n_{reward}$ results closest to $\varkappa$ (using Euclidean distance in the provided pseudocode) are selected as the computers to reward, $\mathfrak{C}_{good}$ (line 23). The computers in $calc.\mathfrak{C}_{good}$ each receive $calc.Reward_i$. Finally, all proposers in *calc.proposers* who are also registered computers receive reputation increases of *avgRepChange* (line 30), while computers in $calc.\mathfrak{C}_{good}$ each receive an increase in reputation of 1 (line 34).

B. Protocol Properties

This section takes the Marvel DC protocol described in Section VI-A, and demonstrates its SINCE (Theorem VI.1), and the long-term benefit provided by the reputation protocol (Lemma VI.2) used therein: namely, that Byzantine computers not performing computations correctly are selected with diminishing probability.

In the following, we consider rational requesters idiosyncratically entering the system, running unique instances of the Request contract. Given this, we first show that rational computers and rational requesters are strongly incentivized to participate in the protocol. Proofs can be found in Appendix A.

Theorem VI.1. There is a strict Nash Equilibrium in which, for any computation with a per player reward $Reward_i > \frac{cost(calc)}{\omega - \gamma}$, rational computers and requesters follow the Marvel DC protocol.

This result is enough to ensure rational players follow the Marvel DC protocol. As such, a majority of results are correctly computed for every computation, agnostic to the semantics of the results. Requesters are still required to perform due dilligence when using these results as some minority may have been produced maliciously.

A consequence of using the reputation and computer-selection mechanism as described in Section V-B, Marvel DC also guarantees that Byzantine computers are selected with diminishing probability in the number of computations, converging to 0 for any minority of selected computers. This is stated formally in the following lemma.

Lemma VI.2. For a series of computations $[calc_1, calc_2, ..., calc_i]$ in Marvel DC with $Reward_i > \frac{cost(calc)}{\omega - \gamma}$ and $n_{comp} > n_{\psi}$, as the number of completed computations increases, the probability of selecting a Byzantine computer for a computation with $n_{comp} < \frac{|\mathfrak{C}|}{2}$ is strictly decreasing in expectancy and converges towards 0 as i tends to infinity.

Lemma VI.2 depends on the output of an on-chain randomness oracle being unpredictable when the Request contract is called. A requester who knows the input seed *randomSeed* to the Marvel DC computer selection protocol (line 5) can select Byzantine computers disproportionately, and use this to articially increase the reputations of Byzantine computers. With a random input seed, Marvel DC randomly draws from the set of computers in direct proportion to reputations. Existing randomness solutions, such as the Chainlink Verifiable Random Function [7], provide proofs that a provided randomness was generated correctly. Given an oracle that can produce randomness periodically, not necessarily related to Marvel DC, the guarantees of Lemma VI.2 hold. Analysis of the quality of on-chain randomness is beyond the scope of this paper.

As a direct consequence of Lemma VI.2, with reasonable choices for rewarding functions and number of computers per-computation (explored in Table II), both enforceable by the protocol, Byzantine players are eventually removed from the system. This improves the efficiency of the protocol over time, reducing the minimum requirements for computers, and as such, latency, transaction fees, and rewards.

	n_{comp} =**10**	**25**	**100**	**1000**
α =**0.45**	4.9×10^{-1}	3.1×10^{-1}	1.8×10^{-1}	8×10^{-4}
0.33	2.1×10^{-1}	4.2×10^{-2}	4.2×10^{-4}	0
0.2	3.3×10^{-2}	3.7×10^{-4}	2.1×10^{-11}	0
0.1	1.6×10^{-3}	1.6×10^{-7}	0	0
0.05	6.4×10^{-5}	3.6×10^{-11}	0	0
0.01	2.4×10^{-8}	0	0	0

TABLE II: Approximate probability of not choosing a majority of rational computers given specific starting adversarial % of computers α (left column) and selected numbers of computers n_{comp} (top row). These probabilities assume a sufficiently large population of computers such that adversarial share represents the per-selection probability of selecting an adversarial computer throughout sampling. We let 0 represent any positive number less than 10^{-14} due to precision constraints.

.

C. Privacy Marvel DC

In this section we outline a privacy enhancement to Marvel DC which we call Privacy Marvel DC. The motivation for this enhancement is to allow for an additional level of computer

[7]https://docs.chain.link/docs/chainlink-vrf/

privacy which can be seen as necessary in computations such as those in FL protocols. This additional privacy on top of the novel contributions of being SINCE, generically applicable and fully decentralized further add to the applicability and utility of our work in an even larger set of DC problems.

We present Privacy Marvel DC by describing it's key differences to Marvel DC to ensure that in an optimistic scenario, only the requester and computers involved in a computation learn the results, and that players in the system can at most infer a computer submitted a good result (or bad result), but not which of the good results (bad results). In the pessimistic scenario, all players in the blockchain observe the results, but it still holds that any player in the system can at most infer a computer submitted a good result (or bad result), but not which of the good results (bad results). In Privacy Marvel DC, there is an additional contract, *Reveal*, which is to be executed after the Response contract, and before rewards are finalized. The purpose of the Reveal contract is described later in this section.

During computer registration, computers in Privacy Marvel DC privately generate $\mathcal{S}_1,\ \mathcal{R}_1 \in \{0,1\}^{\Theta(\kappa)}$, and attach $\textit{regID}_1 \leftarrow \textit{commit}(\mathcal{S}_1, \mathcal{R}_1)$ to the registration message, as described in Section IV-B, information necessary to prove set membership at some later point. Then, when a requester requests a computation, and the indices for computation $\textit{calc}.\mathfrak{C}$ are calculated, the requester now generates a set containing the indices as specified in $\textit{calc}.\mathfrak{C}$, a set to which only computers in $\textit{calc}.\mathfrak{C}$ can prove NIZK set membership. In Privacy Marvel DC, this allows for the separation of result submission and player identity.

In addition to the deposits of Marvel DC, the requester must also deposit a pool of money necessary to incentivize relayers (Section IV-C) to publish transactions on behalf of computers involved in the computation. Given the amount of money required by one relayer to include a blockchain transaction is $\textit{fee}_r$, the additional required deposit is $n_{comp} \cdot \textit{fee}_r$ for the relaying of computer messages during the Response phase.

In the Response contract, a computer selected in $\textit{calc}.\mathfrak{C}$ can submit a NIZK proof of membership in $\textit{calc}.\mathfrak{C}$. Such a computer must also generate and submit a new $\mathcal{S}_2$, $\mathcal{R}_2 \in \{0,1\}^{\Theta(\kappa)}$ pair, and compute $\textit{regID}_2 \leftarrow \textit{commit}(\mathcal{S}_2, \mathcal{R}_2)$. Setting $m \leftarrow \langle \textit{calc}, \textit{response}, \mathcal{R}_1, \textit{regID}_2 \rangle$, the computer generates a NIZKSoK $\pi_1 \leftarrow \textit{NIZKSoK}[m]\{(\textit{regID}_1,\ \mathcal{R}_1)$: MemVerify $(\textit{calc}.\mathfrak{C},\ \textit{regID}_1) = 1\ \ \&\ \ \textit{regID}_1 = \textit{commit}(\mathcal{S}_1,\ \mathcal{R}_1)\ \}$. Finally, the computer then publishes m and π_1 to the blockchain through a relayer, who receives $\textit{fee}_r$ upon the transactions addition to the blockchain.

In the Reveal contract, the requester off-chain performs the same calculations that were done on-chain in Marvel DC to calculate the results to be rewarded. Instead of adding computer indices to $\textit{responses}_{good}$, the requester adds the corresponding $\textit{regID}_2$s. The requester publishes $\textit{responses}_{good}$ and the encryption of *calc.compDecKey* using each public key corresponding to computers in $\textit{calc}.\mathfrak{C}$ to the blockchain. However, rewards are not immediately distributed to computers in $\textit{responses}_{good}$.

In the Finalize contract, computers can choose to contest the computation of $\textit{responses}_{good}$ for up to T blocks after $\textit{responses}_{good}$ was published. If $\textit{responses}_{good}$ was computed incorrectly, any of the computers in $\textit{calc}.\mathfrak{C}$ can publish the decryption of all results, and in-so-doing prove $\textit{responses}_{good}$ was incorrectly computed of by the requester. In this case, all computers are rewarded, and the requesters escrow is destroyed. To prevent malicious computers in $\textit{calc}.\mathfrak{C}$ from attempting this, a further escrow is required, which is returned on the correct proving of miscomputation of $\textit{responses}_{good}$ by the requester.

If $\textit{responses}_{good}$ was computed correctly, any computer whose $\textit{regID}_2$ is included in $\textit{responses}_{good}$ can generate a proof of membership to $\textit{responses}_{good}$. Furthermore, as $\textit{regID}_1$ can no longer be used for future computations (using the same $\textit{regID}_1$ would reveal the same $\mathcal{R}_1$ in the next *calc*), computers must now generate a new $\mathcal{S}_3,\ \ \mathcal{R}_3 \in \{0,1\}^{\Theta(\kappa)}$ pair and corresponding $\textit{regID}_3 \leftarrow \textit{commit}(\mathcal{S}_3, \mathcal{R}_3)$.

Observation: As computers submit results through a relayer, and with an accompanying NIZKSoK π proving membership in the selected indices for computation, all players in the blockchain protocol can be sure the player submitting the message must be a selected computer. Crucially, nothing else can be learned about the submitting player's identity. Similarly, when collecting rewards, or replacing $\textit{regID}_1$, the only thing learned when a computer submits a valid message during the Finalize phase is to which set, $\textit{responses}_{good}$ or $\textit{responses} \backslash \textit{responses}_{good}$, the computer belongs.

D. Further Privacy Enhancements

There are further privacy enhancements possible for Marvel DC. One such enhancement is to detach reward collection/reputation updates from the computation. Given $\textit{responses}_{good}$ was calculated correctly, computers included in $\textit{responses}_{good}$ can instead delay their claiming of the reward and associated reputation increase arbitrarily. After a Finalize phase without arbitration, the set of $\textit{regID}_2$s corresponding to $\textit{responses}_{good}$ can be added to a pool of all recorded good responses throughout the protocol. These can then be immediately/periodically/sporadically claimed by computers, depending on the privacy requirements of the computer in question. This again uses the same NIZK set-membership techniques, except now with a larger set in which to diffuse.

VII. Implementation Analysis

In this section we analyse the costs and performance of Marvel DC and Privacy Marvel DC. We show that, on top of the unique decentralized incentive compatibility guarantees of Section VI-B, both protocols are cost-effective and practical for computers and requesters. The proof-of-concept encodings of Marvel DC used for this analysis are available here [10].

A. Gas cost of running Marvel DC and Privacy Marvel DC

There are several considerations when calculating the cost of running Marvel DC on a blockchain. In the case where a computation has a single 256-bit answer, the costs are significantly less than in the case of gradient estimation problem where answers contain thousands or millions of numbers. More concerning still is the prohibitive nature of messages in the order of MBs in many blockchain protocols. To counteract this, messages for the Response and Finalize contracts can be submitted to memory-efficient alternatives such as IPFS [20], Layer-2 chains or even through an MPC protocol between computers and requesters.

In Table III we present, for various values of n_{comp} and n_{reward}, the approximate gas and US dollar costs using the Harmony blockchain[8] for Marvel DC and Privacy Marvel DC given a 256-bit result, with all messages published on-chain. This can be extended to l-dimensional results for any $l > 1$. The key takeaway from Table III is that the running costs for players to decentralize their computational resources and requests in (Privacy) Marvel DC are practically negligible, being less than $\$0.01$ for any individual in the examples provided.

Thus, the primary consideration for players in (Privacy) Marvel DC is accurately estimating the costs for computation. This can be done through a price-discovery process between protocol participants (players vote/bid on the cost of performing particular computations), potentially involving on-chain price oracles to estimate the cost of particular types of computations without active participation from protocol participants.

The set-membership tools described in Privacy Marvel DC are pre-compiled, and currently being used in the Tornado Cash privacy protocol. We thus calculate the gas cost of the set-membership tools using the Tornado Cash library. All other operations are typical on-chain array manipulation, encryption/decryption, deposit, withdraw and writing to memory operations.

B. Performance metrics

A direct comparison of our protocol to existing distributed FL solutions under the headings of latency, throughput and communication complexity is of limited benefit. The primary reason for this is our protocol does not require protocol players to manage the blockchain, while previous standalone solutions [1], [2], [5], [6] must ensure that a majority of nodes involved in the protocols are efficiently communicating. In a decentralized blockchain setting, protocol participants are only required to listen for messages and events relevant to themselves from the blockchain. As such, decentralized protocols with similar scope to our work [8], [9] avoid such comparisons, as do we.

However there are comparisons with [8], [9] that we can perform. Using the terminology of this paper, every protocol *phase*, a period of time where an event occurs which requires a response, lasts up to T blocks. These T blocks (as used in Section VI-A) are equivalent to the time required to ensure players can submit transactions to the blockchain after a particular on-chain event, such as a computation request. Marvel DC therefore lasts up to $2T$ blocks, which covers the time for computers to respond to a request, and the time taken for the requester to reveal the decryption key. The protocol of [8] lasts at least $4T$ blocks to ensure workers are incentivized to submit at least one correct model update (using the terminology of [8], 2 model update rounds are needed for this to be the case). The additional costs are due to requesters and computers being required to respond twice each after the initial request. The protocol of [9] requires at least $3T$ blocks, as computers must commit to the data-set to be used, before submitting a computation result. All three protocols, including Marvel DC, are equipped to spawn arbitrarily many computations in parallel.

[8]https://explorer.harmony.one/, Accessed: 17/10/2023

	Marvel DC		Privacy Marvel DC		
$\{n_{comp}, n_{reward}\}$	Comp.	Req.	Comp.-No Arbitration	Comp.-Arb.	Req.
$\{5, 1\}$	60 (0.19)	763 (2)	828 (3)	541 (2)	1,429 (5)
$\{5, 3\}$	60 (0.19)	777 (2)	828 (3)	541 (2)	1,569 (5)
$\{10, 2\}$	60 (0.19)	918 (3)	828 (3)	541 (2)	2,686 (9)
$\{10, 5\}$	60 (0.19)	960 (3)	828 (3)	541 (2)	3,176 (10)
$\{25, 5\}$	60 (0.19)	1,425 (5)	828 (3)	541 (2)	7,868 (25)
$\{25, 13\}$	60 (0.19)	1,537 (5)	828 (3)	541 (2)	9,729 (31)

TABLE III: Amortized gas costs in 1,000s for computers and requesters in Marvel DC and Privacy Marvel DC for several choices of computers to select n_{comp} and computers to reward n_{reward}. The equivalent costs of using the Harmony blockchain in thousandths of a US dollar are included in round brackets.

VIII. Conclusion

We present Marvel DC, a SINCE blockchain-based decentralized DC protocol which stands as a new standard in constructing fully decentralized DC protocols. This is achieved through a novel combination of strong incentivization of rational computers in the presence of Byzantine computers and reputation-aware computer selection. Furthermore, we outline Privacy Marvel DC, which utilizes privacy-enhancing techniques that can be bootstrapped to the core Marvel DC protocol to allow for computations on sensitive data without compromising the privacy of the computers participating in the protocol. We demonstrate in Section VII that, in addition to the unique incentivization guarantees of Marvel DC and Privacy Marvel DC exhibited in Section VI, these protocols are cost-effective and ready to deploy, while providing provable performance and running-time improvements on existing state-of-the art. Much work remains to ensure DC protocols remain incentive compatible and practical where computations produce large outputs, with storage being a limiting resources in mainstream blockchain protocols. Marvel DC and Privacy Marvel DC serve as key protocols with which to continue this research.

References

[1] Y. Li, C. Chen, N. Liu, H. Huang, Z. Zheng, and Q. Yan, "A blockchain-based decentralized federated learning framework with committee consensus," *IEEE Network*, vol. 35, no. 1, pp. 234–241, 2021.

[2] X. Bao, C. Su, Y. Xiong, W. Huang, and Y. Hu, "FLChain: A blockchain for auditable federated learning with trust and incentive," in *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)*, pp. 151–159, 2019.

[3] U. Majeed and C. S. Hong, "FLchain: Federated learning via mec-enabled blockchain network," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1–4, 2019.

[4] C. Ma, J. Li, M. Ding, L. Shi, T. Wang, Z. Han, and H. V. Poor, "When federated learning meets blockchain: A new distributed learning paradigm." https://arxiv.org/abs/2009.09338, 2020. Retrieved: 30/06/2022.

[5] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained on-device federated learning," *IEEE Communications Letters*, vol. 24, no. 6, pp. 1279–1283, 2020.

[6] J. Weng, J. Weng, J. Zhang, M. Li, Y. Zhang, and W. Luo, "Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2438–2455, 2021.

[7] M. H. ur Rehman, K. Salah, E. Damiani, and D. Svetinovic, "Towards blockchain-based reputation-aware federated learning," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 183–188, 2020.

[8] K. Toyoda and A. N. Zhang, "Mechanism design for an incentive-aware blockchain-enabled federated learning platform," in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 395–403, 2019.

[9] T. Rückel, J. Sedlmeir, and P. Hofmann, "Fairness, integrity, and privacy in a scalable blockchain-based federated learning system," *Comput. Netw.*, vol. 202, jan 2022.

[10] Github. https://github.com/The-CTra1n/MarvelDC, 2022.

[11] Z. Liu, J. Guo, W. Yang, J. Fan, K.-Y. Lam, and J. Zhao, "Privacy-preserving aggregation in federated learning: A survey." https://arxiv.org/abs/2203.17005, 2022. Retrieved: 30/06/2022.

[12] C. McMenamin, V. Daza, and M. Pontecorvi, "Achieving state machine replication without honest players," in *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, (New York, NY, USA), p. 1–14, Association for Computing Machinery, 2021.

[13] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin, "Bar gossip," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, (USA), p. 191–204, USENIX Association, 2006.

[14] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from Bitcoin," in *2013 IEEE Symposium on Security and Privacy*, pp. 397–411, 2013.

[15] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from Bitcoin," in *2014 IEEE Symposium on Security and Privacy*, pp. 459–474, 2014.

[16] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology – EUROCRYPT 2016* (M. Fischlin and J.-S. Coron, eds.), (Berlin, Heidelberg), pp. 305–326, Springer Berlin Heidelberg, 2016.

[17] D. Benarroch, M. Campanelli, D. Fiore, K. Gurkan, and D. Kolonelos, "Zero-knowledge proofs for set membership: Efficient, succinct, modular," in *Financial Cryptography and Data Security* (N. Borisov and C. Diaz, eds.), (Berlin, Heidelberg), pp. 393–414, Springer Berlin Heidelberg, 2021.

[18] K. Gurkan, K. W. Jie, and B. Whitehat, "Community proposal: Semaphore: Zero-knowledge signaling on Ethereum." https://github.com/appliedzkp/semaphore, 2020. Retrieved: 19/04/2022.

[19] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), p. 118–128, Curran Associates Inc., 2017.

[20] IPFS. https://ipfs.io/.

Appendix

Theorem VI.1. There is a strict Nash Equilibrium in which, for any computation with a per player reward $\mathit{Reward}_i > \frac{\mathit{cost}(\mathit{calc})}{\omega-\gamma}$, rational computers and requesters follow the Marvel DC protocol.

Proof. Consider a Request($\mathit{requester}, *$) instance corresponding to a computation calc, and computers selected for computation $\mathit{calc}.\mathfrak{C}$. Based on $n_{comp} > n_{\psi}$, the majority of computers in $\mathit{calc}.\mathfrak{C}$ are rational.

First consider a rational requester. Correctly running Finalize ($\mathit{calc}, *$) allows the requester to receive back $\mathit{calc.escrow}_{req}$, and as such, rational requesters follow the protocol. Consider now rational computers. If the requester correctly runs Finalize ($\mathit{calc}, *$), then $\mathit{calc}.\varkappa$ and $\mathit{calc}.\mathfrak{C}_{good}$ are generated correctly. Therefore, if all rational computers follow the protocol, the assumption under which we chose Reward_i in Section V, for a given rational computer $\mathcal{C}_i$ correctly running Response(calc, $*$), $\mathcal{C}_i$ is included in $\mathit{calc}.\mathfrak{C}_{good}$ with probability ω. If $\mathcal{C}_i$ incorrectly runs Response(calc, $*$), $\mathcal{C}_i$ is included in $\mathit{calc}.\mathfrak{C}_{good}$ with probability of at most γ. By our choice of Reward_i, we have seen in Section V, given $\mathit{calc}.\mathfrak{C}_{good}$ is generated correctly and computers included in $\mathit{calc}.\mathfrak{C}_{good}$ receive this with probability 1, this is sufficient for rational computers to compute the result correctly, equivalent to calling Response($\mathit{calc}, *$). Therefore, rational computers and requesters follow the protocol if $\mathit{Reward}_i > \frac{\mathit{cost}(\mathit{calc})}{\omega-\gamma}$. □

*

Proof. As $\mathit{Reward}_i > \frac{\mathit{cost}(\mathit{calc})}{\omega-\gamma}$, from Theorem VI.1 rational computers follow the protocol. Let α be the share of computers that are Byzantine. We know a majority of computers selected are rational, as $n_{comp} > n_{\psi}$. Therefore, Byzantine computers are rewarded with probability $\gamma < \omega$. For a given computation, the expected reputation increase of a selected Byzantine computer is γ, while the expected increase for a selected rational computer is ω. Given n_{comp} are selected for the computation, the expected number of these being rational computers is $(1-\alpha)n_{comp}$, while the number of selected Byzantine computers is αn_{comp}. Furthermore, this means the expected increase in reputation for rational computers is $(1-\alpha)n_{comp}\omega$, while the expected increase in reputation for Byzantine computers is $\alpha n_{comp}\gamma$. At the beginning of the protocol, the probability of selecting a Byzantine player from the set of all computers is in direct proportion to starting reputation. Given initial reputations of iR, after the first computation, the selection probability of a Byzantine computer reduces in expectancy to:

$$\frac{\alpha(|\mathfrak{C}| \cdot \mathit{iR} + n_{comp}\gamma)}{|\mathfrak{C}| \cdot \mathit{iR} + n_{comp}\big((1-\alpha)\omega + \alpha\gamma\big)}. \tag{7}$$

First it be can see that

$$\frac{\alpha(|\mathfrak{C}| \cdot \mathit{iR} + n_{comp}\gamma)}{|\mathfrak{C}| \cdot \mathit{iR} + n_{comp}\big((1-\alpha)\omega + \alpha\gamma\big)} < \alpha \tag{8}$$

meaning Byzantine selection probability is decreasing. To prove that Byzantine selection probability tends to 0 in the number of computations as described in the Lemma statements, let α_k be the Byzantine computer selection probability after k computations. We have the expected Byzantine selection probability after $k+1$ computations, denoted , α_{k+1}, is:

$$\begin{aligned} &\frac{\alpha_k(|\mathfrak{C}| \cdot iR + n_{comp}\gamma)}{|\mathfrak{C}| \cdot iR + n_{comp}\big((1-\alpha_k)\omega + \alpha_k\gamma\big)} \\ &= \frac{\alpha_k(|\mathfrak{C}| \cdot iR + \gamma n_{comp})}{|\mathfrak{C}| \cdot iR + n_{comp}\omega - \alpha_k n_{comp}(\omega - \gamma)}. \end{aligned} \tag{9}$$

We have already seen α_{k+1} equals

$$\frac{\alpha_k(|\mathfrak{C}| \cdot iR + n_{comp}\gamma)}{|\mathfrak{C}| \cdot iR + n_{comp}\omega - \alpha_k n_{comp}(\omega - \gamma)} < \alpha_k. \tag{10}$$

which implies:

$$\frac{(|\mathfrak{C}| \cdot iR + n_{comp}\gamma)}{|\mathfrak{C}| \cdot iR + n_{comp}\omega - \alpha_k n_{comp}(\omega - \gamma)} < 1. \tag{11}$$

Letting the term on the left be r_k, we can see r_k is decreasing in k as:

- $n_{comp}(\omega - \gamma) > 0$ (because $\omega > \gamma$).
- $0 < \alpha_{k+1} < \alpha_k$.

These together mean the negative term in the denominator of r_k, $\alpha_k n_{comp}(\omega - \gamma)$, is increasing (towards 0) and as such the denominator of r_k is increasing. Therefore $\alpha_k < \alpha_0 r_0^k$, with $r_0 < 1$. The result follows. □

Algorithm 1 Computer Selection Protocol

```
function $genProbSelect()$
    $minRep \leftarrow min(Reps)$
    $denominator \leftarrow \sum(Reps) - length(Reps) \cdot (minRep - 1)$
    return $[((i - (minRep - 1))/denominator)\ for\ i \in Reps]$    ▷ Probability formula from Section V-B

function $selectComputers(randomSeed, n_{comp})$
    $ctr \leftarrow 0$
    $calc.\mathfrak{C} \leftarrow []$
    $probSelect \leftarrow genProbSelect(Reps)$
    $randomSeed \leftarrow commit(randomSeed)$
    while $ctr < n_{comp}$ do
        $i \leftarrow 0$
        $sumReps \leftarrow probSelect[i]$
        while $randomSeed > sumReps$ do
            $sumReps \leftarrow sumReps + (probSelect[i++] * (2^256))$
        if $\neg(i \in calc.\mathfrak{C})$ then
            $calc.\mathfrak{C}.append(i)$
            $ctr \leftarrow ctr + 1$
        $randomSeed \leftarrow commit(randomSeed)$
```

Algorithm 2 Reputation Management

```
function $rateComputations(calc, n_{reward}, compDecKey, f_{\varkappa})$
    $\mathfrak{C}_{good} \leftarrow []$
    $results \leftarrow decrypt(calc.responses, compDecKey)$
    $\varkappa \leftarrow f_{\varkappa}(results)$
    for $i \in [1, ..., n_{reward}]$ do    ▷ add the $n_{reward}$ closest results to $\varkappa$ to $\mathfrak{C}_{good}$
        $\mathfrak{C}_{good}.append(result.\mathcal{C}) \wedge results.remove(result)$ with $distance(result, \varkappa) = min(distance(results, \varkappa))$
    $\mathfrak{C}_{bad} \leftarrow results.\mathcal{C}$    ▷ all results not already removed in the for loop are bad results, not to be rewarded
    return $\mathfrak{C}_{good}$, $\mathfrak{C}_{bad}$

function $updateReputations(\mathfrak{C}_{good}, \mathfrak{C}_{bad}, calc)$
    $avgRepChange \leftarrow length(\mathfrak{C}_{good})/(length(\mathfrak{C}_{good}) + length(\mathfrak{C}_{bad}))$
    $denominator \leftarrow \sum(Reps) - (length(Reps) - 1) \cdot (iR - 1)$
    for $blockProposer \in calc.proposers$ do    ▷ in-line with the results from Section V-B, block proposers rep. changes should be done before updating computers
        $Reps[blockProposer] \leftarrow Reps[blockProposer] + (avgRepChange \cdot ((Reps[blockProposer] - (iR - 1))/(denominator - Reps[blockProposer])))$    ▷ Necessary for SINCE of proposers/requester
    $Reps[calc.requester] \leftarrow Reps[calc.requester] + (avgRepChange \cdot ((Reps[blockProposer] - (iR - 1))/(denominator - Reps[blockProposer])))$    ▷ Requester of successfully resolved computation must receive increase in reputation, in line with Section V-B
    $Reps[tx.blockProposer] \leftarrow Reps[tx.blockProposer] + (avgRepChange \cdot ((Reps[blockProposer] - (iR - 1))/(denominator - Reps[blockProposer])))$    ▷ Proposer including the Finalize transaction must also receive increase in reputation, in line with Section V-B
    $Reps[\mathfrak{C}_{good}] \leftarrow Reps[\mathfrak{C}_{good}] + 1$
```

Algorithm 3 Marvel DC smart contract pseudocode

```
35: $\mathfrak{C} \leftarrow []$    ▷ Set of active computers
36: $iR \leftarrow getInitialRep()$
37: $Reps \leftarrow [iR$ for $i \in \mathfrak{C}]$
38: $T \leftarrow getFinalizeDeadline()$    ▷ Globally-defined finalize deadline
39: $escrow_{comp}, escrow_{req} \leftarrow getEscrows()$ ▷ Globally-defined escrow amounts, in line with Section V
40: $n_\psi \leftarrow getMinNumComputersPerComputation()$    ▷ Set $n_\psi$ in-line with requirements from Section V
41: $tFunctions \leftarrow getTargetFunctions()$ ▷ Define allowable target functions. In reality, this can be updated during the protocol

42: Register
43: upon $\langle REGISTER \rangle$ from $\mathcal{P}$ with $\mathcal{P} \notin \mathfrak{C} \wedge \mathcal{P}.balance > escrow_{comp}$ do ▷ add computer to the system
44:     $\mathcal{P}.transfer(escrow_{comp}, contract)$    ▷ Registration cost to prevent Sybil attacks
45:     $\mathfrak{C}.append(\mathcal{P})$
46:     $Reps.append(iR)$

47: Request
48: upon
    $\langle REQUEST, calc, n_{comp}, n_{reward}, compEncKey, Reward_i, f_{\varkappa} \rangle$ from requester with $n_{comp} > n_\psi \wedge requester.balance > ((n_{reward}, \cdot Reward_i) + escrow_{req}) \wedge$
    $f_{\varkappa} \in tFunctions$ do
49:     $requester.transfer((n_{reward}, \cdot Reward_i) + escrow_{req}, contract)$    ▷ Transfer total reward plus requester escrow to contract
50:     $calc.\mathfrak{C} \leftarrow selectComputers(genRandom(), n_{comp})$    ▷ Select computers for computation
51:     $responses \leftarrow []$
52:     $proposers \leftarrow []$    ▷ Array of the players who recorded each $\langle RESPONSE, * \rangle$ transaction
53:     $start \leftarrow Blockchain.height$    ▷ Record current height of blockchain
54:     $step \leftarrow computing$

55: Response
56: upon $tx \leftarrow \langle RESPONSE, calc, result \rangle$ from $\mathcal{C} \in calc.\mathfrak{C}$
    with $calc.step = computing \wedge Blockchain.height < calc.start + T$ do
57:     $calc.responses.append(result)$    ▷ result should be the computer $\mathcal{C}$'s result of computing calc, encrypted using calc.compEncKey
58:     if $tx.blockProposer \in \mathfrak{C}$ then
59:         $calc.proposers.append(tx.blockProposer)$

60: Finalize
61: upon $tx \leftarrow \langle FINALIZE, calc, compDecKey \rangle$ from $calc.requester$
    with $valid(compDecKey, calc.compEncKey)$
    $\wedge \big((calc.step = computing \wedge Blockchain.height < calc.start + T$
    $\wedge length(calc.responses) = calc.n_{comp}) \vee (calc.step = computing$
    $\wedge Blockchain.height \geq calc.start + T)\big)$ do
62:     $calc.transfer(escrow, calc.requester)$    ▷ Returns the escrow to the requester
63:     $calc.proposers.append(calc.requester)$    ▷ Required for SINCE of requester
64:     if $tx.blockProposer \in \mathfrak{C}$ then    ▷ Required for SINCE of proposers
65:         $calc.proposers.append(tx.blockProposer)$
66:     $calc.step \leftarrow finalized$
67:
    $\mathfrak{C}_{good}, \mathfrak{C}_{bad} \leftarrow rateComputations(calc, calc.n_{reward}, compDecKey, calc.f_{\varkappa})$ ▷ Function which deterministically evaluates the goodness of returned computations, returning the indices of good and bad computers
68:     $calc.transfer(calc.Reward_i, \mathfrak{C}_{good})$
69:     $updateReputations(\mathfrak{C}_{good}, \mathfrak{C}_{bad}, calc)$
```