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

Conor McMenamin^{a,b}, Vanesa Daza^{a,c}

^aDepartment of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona, Spain ^bNOKIA Bell Labs, Nozay, France ^cCYBERCAT - Center for Cybersecurity Research of Catalonia

1. Abstract

Decentralized computation outsourcing should allow anyone to access the large amounts of computational power that exists in the Internet of Things. Unfortunately, when trusted third parties are removed to achieve this decentralization, ensuring an outsourced computation is performed correctly remains a significant challenge. In this paper, we provide a solution to this problem.

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. Furthermore, 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

^{*}This document is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 814284 *Email addresses: conor.mcmenamin@upf.edu (Conor McMenamin),
vanesa.daza@upf.edu (Vanesa Daza)

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.

Keywords: Distributed Computing, Decentralization, Blockchain, Incentives

2. Introduction

The distributing of computation among computers has beckened a neverbefore-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 data-sets 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 [16, 1, 20, 19, 13, 27, 26, 25] 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. In decentralized settings such as blockchain protocols, players only 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 [24]. 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. Furthermore, the pre-computation work required by requesters in [24] is itself intensive (3-40 minutes using 16 virtual CPU cores, and 64 GB RAM), and not appropriate for computationally-restricted requesters, defeating a significant purpose of DC protocols. In this paper, we address all of these issues.

2.1. Our Contribution

We present Marvel DC, a generic (can be adapted to any computational problem outputting results in Euclidean space) 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. This is a significant advancement in a field where existing claims of incentive compatibility do not account for computers trying to maximize their tokens, with no viable solution, to the best of our knowledge, for the distributing of tokenized rewards in a distributed and decentralized manner.

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 5.2) this reputation protocol neither affects the decentralisation or strong incentive compatibility of the protocol. When using a fixed number of computers per computation, our protocol shares all of the benefits of [8]. Our 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 compatibility in expectation. In Section 5, we describe how to program rewards such that it is strong incentive compatible in expectation for every rational player (requesters, computers and/or block-producers) in the blockchain system to follow the protocol.
- Handling of symmetric /asymmetric utilities. By tokenising 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.
- Decentralization. Encoding Marvel DC in a generic manner for any tokenized smart-contract enabled blockchain, all players in such a blockchain system are able to participate in Marvel DC. Furthermore, all of the tools used in Marvel DC are already in use in smart-contract enabled decentralized systems such as Ethereum ¹ and Harmony ² (our smart-contract encoding [10] is written in Solidity [10], a programming language interpretable by both blockchains), demonstrating our encoding can utilize the decentralisation of such blockchain ecosystems.

On top of these main contributions, Marvel DC also retains additional desirable properties that should be present in any DC protocol:

• Fair selection. Our protocol uses a decentralized pseudorandom function (such as a verifiable on-chain oracles³, or those used in [5]) to provide a random seed used to select computers based on reputation. This removes the possibility of reputation-based manipulation by clusters of colluding computers who may try to prioritize the selection of computers in the colluding cluster, as is allowed in existing works [8, 7, 24].

¹https://ethereum.org/en/

²https://www.harmony.one/

³https://docs.chain.link/docs/chainlink-vrf/

- Proper management of new peers. All new computers entering Marvel DC are given the same initial reputation, which eventually becomes significantly lower than the average reputation of active honestly-behaving computers.
- Low communication complexity & cost. By utilising a blockchain, computers participating in Marvel DC are only required to monitor the blockchain for computation requests, and respond when they themselves are selected to participate in the computation. Furthermore, by utilising cost-effective blockchains such as Harmony, the costs for participation in Marvel DC are fractions of a US dollar, as shown in Section 7.

Given these properties of Marvel DC, we then outline a series of privacy-enhancing amendments, culminating in Privacy Marvel DC, a protocol in which results cannot be linked to the computer which provided the result, except with significant additional cost to the player who causes the revelation. Privacy Marvel DC retains all of the decentralisation 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 set.

To do this, we make use of existing ZK set-membership tools (described in Section 4.2). These tools are used in protocols such as Zerocoin [22] and Tornado Cash⁴, with the latter of which currently deployed and in use on the Ethereum blockchain. Computers encrypt results, and publish them anonymously to the blockchain through a network of relayers: blockchain participants who are incentivized to submit transactions on behalf of other players (explained in detail in Section 4.3). The set of good and bad results are published as in Marvel DC, and as results are not attached to the computer that sent them, computers prove

 $^{^4}$ https://tornado.cash/

membership among the set of computers submitting good/bad results using the aforementioned ZK set-membership tools.

Although our privacy techniques are not novel outside of DC, and in fact reduce privacy compared to DC protocols like [24], the combination of decentralisation, proven 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 1) stands as an additional novel contribution.

3. Related Work

Protocol	Tokenized Rewards	Strong Incentive Compatible	Computation-Independent	Diminishing Adversarial Selection Prob.	
[16, 1, 20, 19, 13, 27, 26]	X	X	✓	✓/ X	
Toyoda et al. [25]	✓	×	✓	×	
Ruckel et al. [24]	✓	✓	X *	×	
Marvel DC	/	✓	/	✓	

Table 1: 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 [24] must be generated anew for each type of computation, placing significant upfront costs on computation requesters.

There are many intertwined areas of research regarding the decentralized outsourcing of computations to distributed sets of potentially untrusted peers. Strong advancements have been made with respect to single computer outsourcing, with [28, 17, 9] providing variations of such pairwise protocols.

These 2-player protocols involve several rounds of communication between the requester and computer. However, none discuss the problem of computer selection or rewarding in the presence of many competing computers. This limits their scope for outsourcing computations where the participation of many computers is required, such as in FL.

Blockchain-based DC protocols [16, 1, 20, 19, 13, 27, 26] have been well-studied also. 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 [25], 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 strong incentive compatible. Moreover, [25] 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 privacypreserving FL protocols[18] was published. Of the investigated works, the
most promising for achieving an incentive-compatible decentralized protocol
is [24]. In [24], 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 [24], 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.

4. Preliminaries

4.1. Blockchains

In this paper, we are interested in a distributed set of n players $\{P_1, ..., P_n\}$ 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 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 [21], which we use in this paper. The ByRa model is a necessary improvement on the legacy BAR model [15] 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 4.1. 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 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. Based on similar assumptions to [14], 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 protocol provides availability and correctness of the programs being run through the 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.

4.2. Zero-Knowledge Primitives

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 [22, 2, 11, 3, 12], 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 Marvel DC, as we only require a common functionality that is shared by all of them.

In the rest of the paper, we let * denote any value. Computers privately generate two bit strings, the serial number S and randomness r, with S, $r \in \{0,1\}^{O(\kappa)}$ for some security parameter κ . Computers then commit to these values by generating a commitment $com \leftarrow f_{com}(S,r)$ where $f_{com}(*)$ is some cryptographically-secure commitment function (such as a Pedersen Commitment [23]). This com is then published to the blockchain, with the set of all commitments denoted by Com. With this in mind, we now define the key functions that allow us to reason about the NIZK proving of the membership of some commitment com in Com:

• Verify(π): For a ZK proof π , returns 1 if and only if π is a valid proof of

knowledge.

- MemVerify(Com, com): Returns 1 if and only if $com \in Com$.
- NIZKPoK{(com, r): MemVerify(Com, com) = 1 & com = f_{com}(S, r) } → π: Returns a NIZK proof of knowledge of a commitment com and randomness r which satisfies MemVerify(Com, com)=1 and com = f_{com}(S, r). The variables (com, r) are assumed to be known only by the prover, while all other variables and functions are know by the verifier. Specifically, this function depends on the serial number S being revealed. This revelation identifies to a verifier when a proof has previously been provided for a particular, albeit unknown, commitment as the prover must reproduce S. This is used in conjunction with an escrow to enforce the correct participation of computers in our privacy-enhanced DC protocol.
- NIZKSoK(m){(com, r) : MemVerify(Com, com) = 1 & com = f_{com}(S, r)
 → π: for an arbitrary message m, this function returns an NIZK proof which proves that the person who chose m can also produce NIZKPoK{ (com, r) : MemVerify(Com, com) = 1 & com = f_{com}(S, r) }. As such, NIZKSoK(m) acts as a signature of knowledge on m, as coined in [22].

In this paper, we assume the public NIZK parameters are set-up in a trusted manner 5 .

4.3. Relayers

A fundamental requirement for transaction submission in blockchains is the payment of some transaction fee to simultaneously incentivize block producers to include the transaction, and to prevent denial-of-service/spamming attacks. However, this allows for the linking of player transactions, balances, and their associated transaction patterns. To counteract this, we utilize the concept of

⁵Such as a Perpetual Powers of Tau ceremony, as used in Zcash https://zkproof.org/ 2021/06/30/setup-ceremonies/

 $transaction \ relayers^6.$

When a computer wishes to submit a transaction anonymously to the block-chain, the computer publishes a proof of membership 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. As the relayer can verify the proof of membership, the relayer can also be sure that if the transaction is sent to certain smart-contracts (such as those used in our DC protocols), the relayer will receive a fee. As such, a model in which players submit transactions to relayers, who then in turn submit those transactions to the blockchain for a guaranteed fee is a straightforward extension of a model in which players submit transactions directly to the blockchain themselves. Furthermore, if the transaction contains a proof of membership that is NIZK and the message is broadcast anonymously (using the onion routing (Tor) protocol⁷ for example), the relayer can only infer that the player sending the transaction is a member of a particular set.

5. Constructing a Strong Incentive Compatible DC Protocol

For the sake of comprehensibility, 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 wishes to outsource to some subset of available computers C. Furthermore, the requester is aware of a threshold n_{ψ} such that for any random sample of n_{ψ} computers without replacement from C, a majority of those computers are rational.

We consider the output of all computations in this paper as a point in ldimensional space. To reason about the goodness of a computation result, for

⁶Ox https://ox.org/docs/guides/v3-specification, Open Gas Station Network https://docs.opengsn.org/, Rockside https://rockside.io/, Biconomy https://www.biconomy.io/

⁷https://www.torproject.org/

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 τ , 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 [4] is such an aggregation function. We let dist(x,y) be the Euclidean distance function. For $x,y \in \mathbb{R}^l$, $dist(x,y) = \sqrt{(x_1-y_1)^2 + ... + (x_l-y_l)^2}$.

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 for any $\sigma > 0$:

$$P(dist(result_+, \tau) < \sigma) > P(dist(result_-, \tau) < \sigma).$$
 (1)

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

Now, consider the following DC protocol run by a trusted third party (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 6) 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 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 incentivisation. 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 incentivisation, while reducing the probability that incorrectly performing computers are selected for computation.

5.1. Reward Mechanism

In this section, we provide a theoretical lower-bound on the per-computer reward required to strongly incentivize the correct participation of rational computers in our DC protocol. We describe this reward generically in terms of a cost to compute a particular computation, fees required to post a transaction to the blockchain, and the probability of being rewarded for good and bad computations. We also describe an escrow amount to be deposited by the requester when submitting the computation, which is received back after correctly computing the set of computers to reward. 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 the escrow. Otherwise, the computer loses the bounty. This is described in more detail in Section 6.

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:

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

To more accurately describe Reward(calc), we introduce $0 \le \omega, \gamma \le 1$ with the probability of good computations being rewarded being ω , and the probability of bad computations being rewarded being γ , such that WLOG $\omega > \gamma$. 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:

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

Additionally, in a smart-contract blockchain, actions must be triggered by one or more players by sending transactions to the blockchain. In Marvel DC, the distribution of rewards is initially controlled by the requester (if the requester fails to trigger correct reward distribution, computers can eventually initiate the rewarding mechanism). 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. This can be seen as the payoff for submitting the correct set is $escrow_{req} - Fee(tx) > 0$, while the payoff for not submitting the set is 0. However, in the case of potential collusion of up to k computers, setting $escrow_{req} \ge k \cdot Reward(calc) + Fee(tx)$ 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.

5.2. Reputation Management Protocol

In the previous section, we identified that all rational players in the system follow the protocol given no changes to reputation. However, the use of a reputation-based selection process prioritizes good computers over bad computers, meaning both short- and long-term benefits for correctly behaving computers. Therefore, using reputation-based computer selection is desirable. In this section we describe a reputation management protocol that maintains the incentive compatibility of a DC protocol with an incentive compatible reward mechanism.

We consider computations for which a rating mechanism rates results as either "good" or "bad" exists. Correctly performed computations are rated good with probability ω , while incorrectly computations are rated good with probability γ . We construct a function from this rating mechanism, rate(), which assigns good calculations a score of 1, and bad calculations a score of 0. For a player P_i taking part in computations for $calc_1$, $calc_2$, ..., $calc_k$, P_i 's base reputation is $baseRep_i = \sum_{j=1}^k rate(calc_j)$.

Let initRep > 0 be the starting reputation for computers registering in the system. For a given computation, P_i is selected as computer for a computation in block at height H in the blockchain in direct proportion to $baseRep_i^{H-1}(initRep-1)$ as a fraction of $\sum_{j=1}^{n} baseRep_j^{H-1} - n \cdot (initRep-1)$. We subtract (initRep-1) from $baseRep_i^{H-1}$ to normalize the base reputations for selection probability purposes, and so there is no benefit for computers rejoining, particularly in the case where base reputations are increasing over time. With this in mind, the number of computations a player P_i is selected for is directly proportional to:

$$probSelect_{i}^{H} = \frac{baseRep_{i}^{H-1} - (initRep - 1)}{\sum_{j=1}^{n} baseRep_{j}^{H-1} - n \cdot (initRep - 1)}.$$
 (4)

Consider a player P_i who includes a good computation as block proposer for a computer. This increases that computers base reputation, and thus that computers probSelect. This has long-term stake implications, as discussed at the beginning of Section 5.2. Although the computation result is encrypted, the expected change in probSelect for an included computer is positive, which negatively affects the probSelect of the block proposer. 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 $E_{repChange} > 0$ be the expected change in base reputation for a computer whose computation gets included on the blockchain.

For P_i a proposer of a block that includes k transactions containing computation results, we need the following equality to hold:

$$baseRep_{i}^{H} = baseRep_{i}^{H-1} + k \cdot E_{repChange} \left(\frac{baseRep_{i}^{H-1} - (initRep - 1)}{\sum_{j \neq i} baseRep_{j}^{H-1} - (n - 1) \cdot (initRep - 1)} \right). \quad (5)$$

This means we need to add $k \cdot E_{repChange} \left(\frac{baseRep_i^{H-1} - (initRep-1)}{\sum_{j \neq 1} baseRep_j^{H-1} - (n-1) \cdot (initRep-1)} \right)$ to $baseRep_i^{H-1}$ 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 $E_{repChange}$ with the actual mean change in reputation in Equation 5, and the rest of the numbers stay the same.

6. Marvel DC

The goal of this section is to take the ideal DC functionality of Section 5, and implement it as a set of algorithms encoded as smart-contracts that can be run by a decentralized (without a TTP) set of players with access to a blockchain. We call this protocol Marvel DC. In Section 5, we identified values for computer rewards, reputations changes and computer/requester escrows which ensure the

participation of rational players if they can be enforced. In this section, we describe how all of these values can be enforced using a blockchain, and as such, that it is possible to implement the idealized DC protocol in a fully decentralized manner.

6.1. Algorithmic Overview

We outline the Marvel DC protocol as a set of pseudo-code smart contracts encodings provided in Algorithms 1 and 2. These contracts are labelled: *Register, Request, Response* and *Finalize*. A Solidity implementation of Marvel DC has also been made publicly available on Github [10].

A Marvel DC instance can spawn an indefinite number of computation instances, each 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, including a graphic representation of the protocol flow in Figure 1. A privacy-enhancing implementation of Marvel DC is then described in Section 6.3.

Each player P_i owns (has exclusive access to) a set of token balances bal_i which are stored as a globally accessible variable on the blockchain. For a token B, $bal_i(B)$ is the amount of token B that 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 10), granting that computer a reputation of initRep (line 12).

Computation request instances are initialized by calling the Request contract, which specifies the computation details calc, the number of computers to be selected for the computation n_{comp} , a deterministic function f_{τ} for selecting the target result τ 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 deposits $n_{reward} \cdot Reward_i + escrow_{req}$. The compEncKey is the public key corresponding

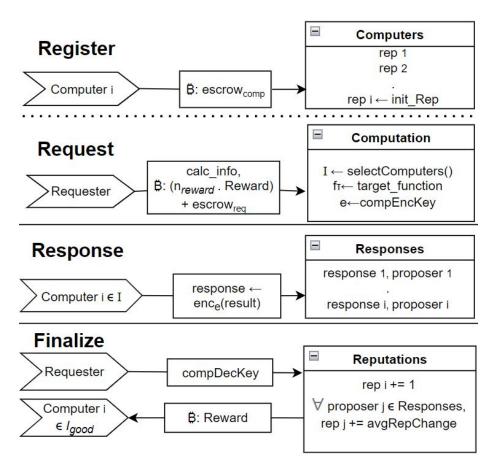


Figure 1: Marvel DC information and token flow. \Breve{B} : indicates the transfer of tokens.

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 16), 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.I.

Computers selected for a particular computation calc, identified in calc.I, can then submit results for calc to the blockchain by calling the Response contract for up to T blocks after the computation is requested. These results should be encrypted using calc.compEncKey (line 3). These encrypted results are decrypted in the Finalize phase (line 3) for rewarding purposes. 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 (line 25). This is used later to update reputations, in line with the analysis of Section 5.2.

Then, either after T blocks from when the computation calc was requested, or when all computers in calc.I 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 match up, 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 1). This is done by applying the pre-specified target function to the computation results, and computing a target value τ (line 4). The computers corresponding to the $calc.n_{reward}$ results closest to τ using Euclidean distance are selected as the computers to reward, I_{good} (line 5). The computers in $calc.I_{good}$ each receive $calc.Reward_i$. Finally, all registered computers in calc.proposers receive reputation increases of avgRepChange (line 12), while computers in $calc.I_{good}$ each receive an increase in reputation of 1 (line 16).

Algorithm 1 Marvel DC smart contract pseudocode

```
1: C \leftarrow []

    ▷ Set of active computers

 2: \ initRep \leftarrow getInitialRep()
 3: Reps \leftarrow [initRep \text{ for } i \in C]
 4: T \leftarrow getFinalizeDeadline()
                                                                           ▷ Globally-defined finalize deadline
 5: escrow_{comp}, escrow_{req} \leftarrow getEscrows() \Rightarrow Globally-defined escrow amounts, in line with Section
 6: n_{\psi} \leftarrow getMinNumComputersPerComputation()
                                                                      \triangleright Set n_{\psi} in-line with requirements from
    Section 5
 7: tFunctions \leftarrow getTargetFunctions() \triangleright Define allowable target functions. In reality, this can be
    updated during the protocol
 8: Register
 9: upon \langle REGISTER \rangle from P with P \notin C and P.balance > escrow_{comp} do \Rightarrow add computer to
    the system
10:
         P.transfer(escrow_{comp}, contract)
                                                                 \triangleright Registration cost to prevent Sybil attacks
         C.append(P)
11:
         Reps.append(initRep)
12:
13: Request
14: upon
    \langle \bar{R}EQUEST, calc, n_{comp}, n_{reward}, compEncKey, Reward_i, f_{	au} \rangle from requester with n_{comp} > 1
    n_{\psi} and requester.balance > ((n_{reward}, Reward_i) + escrow_{req}) and f_{\tau} \in tFunctions do
15:
        requester.transfer((n_{reward}, \cdot Reward_i) + escrow_{req}, contract)  \triangleright Transfer total reward plus
    requester escrow to contract
16:
         I \leftarrow selectComputers(genRandom(), n_{comp})
                                                                          \triangleright Select computers for computation
         \begin{array}{l} responses \leftarrow [] \\ proposers \leftarrow [] \end{array}
17:
                                 \triangleright Array of the players who recorded each \langle RESPONSE, * \rangle transaction
18:
19:
         start \leftarrow Blockchain.height
                                                                         > Record current height of blockchain
20:
         step \leftarrow computing
21: Response
22: upon tx \leftarrow \langle RESPONSE, calc, result \rangle from c \in I
     \textbf{with} \ \ calc.step = \ \ computing \ \textbf{and} \ \ Blockchain.height \ \ < calc.start + T \quad \textbf{do} 
         calc.responses.append(result) ▷ result should be the computer c's result of computing calc,
    encrypted using calc.compEncKey
24:
         if tx.blockProposer \in C then
25:
             calc.proposers.append(tx.blockProposer)
26: Finalize
27: upon tx \leftarrow \langle FINALIZE, calc, compDecKey \rangle from calc.requester
    with valid(compDecKey, calc.compEncKey)
    and ((calc.step = computing and Blockchain.height < calc.start + T)
    and len(calc.responses) = calc.n_{comp}) or (calc.step = computing and Blockchain.height <math>\geq calc.start + T)) do
28:
         calc.transfer(escrow, calc.requester)
                                                                        \triangleright Returns the escrow to the requester
29:
         calc.proposers.append(calc.requester)
                                                                             \triangleright Required for SINCE of requester
30:
         if tx.blockProposer \in C then
                                                                            ▶ Required for SINCE of proposers
              calc.proposers.append(tx.blockProposer)
31:
32:
         calc.step \leftarrow finalized
33:
         I_{good}, I_{bad} \leftarrow rateComputations(calc, calc.n_{reward}, compDecKey, calc.f_{\tau})
                                                                                                          ▶ Function
    which deterministically evaluates the goodness of returned computations, returning the indices
    of good and bad computers
34:
         calc.transfer(calc.Reward_i, I_{good})
35:
         updateReputations(I_{good}, I_{bad}, calc)
```

Algorithm 2 Computer Selection Protocol

```
1: \ \mathbf{function} \ \mathit{genProbSelect}()
         minRep \leftarrow min(Reps)
 3:
         denominator \leftarrow \sum (Reps) - len(Reps) \cdot (minRep - 1)
 4:
        return [((i - (minRep - 1))/denominator) for i \in Reps]
                                                                                      ▷ Probability formula from
 5: function selectComputers(randomSeed, n_{comp})
 7:
        probSelect \leftarrow genProbSelect(Reps)

randomSeed \leftarrow SHA(randomSeed)
 8:
 9:
         while ctr < n_{comp} do
10:
11:
             sumReps \leftarrow probSelect[i]
12:
13:
             while randomSeed > sumReps do
                 sumReps \leftarrow sumReps + (probSelect[i + +] * (2^256))
14:
15:
             if not(i \in I) then
16:
                 I.append(i)
                 ctr \leftarrow ctr + 1
18:
             randomSeed \leftarrow SHA(randomSeed)
```

Algorithm 3 Reputation Management

```
1: function rateComputations(calc, n_{reward}, compDecKey, f_{\tau})
                            I_{good} \leftarrow [] \\ results \leftarrow decrypt(calc.responses, compDecKey)
   3:
   4:
                             \tau \leftarrow f_{\tau}(results)
                            for i \in [1,...,n_{reward}] do
                                                                                                                                                                                                       \triangleright add the n_{reward} closest results to \tau to I_{good}
   6:
                                         I_{good}.append(result.c) and results.remove(result) with dist(result, \tau) =
                min(dist(results, \tau))
   7:
                           I_{bad} \leftarrow results.c \triangleright all results not already removed in the for loop are bad results, not to be
               rewarded
   8:
                           return I_{good}, I_{bad}
  9: function updateReputations(I_{good}, I_{bad}, calc)
                              \begin{array}{l} avgRepChange \leftarrow len(I_{good})/(len(I_{good}) + len(I_{bad})) \\ denominator \leftarrow \sum (Reps) - (len(Reps) - 1) \cdot (initRep - 1) \end{array}
10:
11:
                              for blockProposer \in calc.proposers do \triangleright in-line with the results from Section 5.2, block
12:
               proposers rep. changes should be done before updating computers
13:
                                            Reps[blockProposer] \leftarrow Reps[blockProposer] +
                (avgRepChange \cdot ((Reps[blockProposer] -
                (initRep - 1))/(denominator - Reps[blockProposer])))
                                                                                                                                                                                                                                                                                               \triangleright Necessary for SINCE of
               proposers/requester
                              Reps[calc.requester] \leftarrow Reps[calc.requester] + (avgRepChange \cdot ((Reps[blockProposer] - (Reps[blockProposer] - (Re
                (initRep - 1))/(denominator - Reps[blockProposer]))) \triangleright Requester of successfully resolved
                computation must receive increase in reputation, in line with Section 5.2
                               Reps[tx.blockProposer] \leftarrow Reps[tx.blockProposer] +
                (avgRepChange \cdot ((Reps[blockProposer] - (Reps[blockProposer] - (Re
                (initRep - 1))/(denominator - Reps[blockProposer])))
                                                                                                                                                                                                                                                               ▷ Proposer including the Finalize
                transaction must also receive increase in reputation, in line with Section 5.2
                              Reps[I_{good}] \leftarrow Reps[I_{good}] + 1
```

Algorithm 4 Marvel DC for player P_i as a Requester

```
1: function initialize(calc)
2: compEncKey, compDecKey \leftarrow generateKeyPair()
3: Reward_i \leftarrow SINCE reward to guarantee participation of computers
4: n_{comp} \leftarrow x with x > n_{\psi}
5: f_{\tau}getTargetFunction(calc) \triangleright Select target function for computation
6: broadcast\langle REQUEST, calc, n_{comp}, n_{reward}, compEncKey, Reward_i, f_{\tau} \rangle
7: upon len(calc.responses) = calc.n_{comp} or (Blockchain.height = calc.start + FD) do
8: broadcast\langle FINALIZE, calc, calc.compDecKey \rangle
```

Algorithm 5 Marvel DC for player P_i as a computer

- 1: $\mathbf{broadcast}\langle REGISTER \rangle$
- 2: upon $\langle REQUEST, calc, n_{comp}, n_{reward}, compEncKey, Reward_i, f_{\tau} \rangle$ with $i \in calc.I$ do
- $\textbf{B:} \quad \textbf{broadcast} \langle RESPONSE, calc, result \leftarrow encrypt(compute(calc), calc.compEncKey) \rangle$

6.2. Protocol Properties

In the following results, we make the assumption that rational requesters randomly enter the system, running unique instances of the Request contract. Under this assumption, we first show that rational computers and rational requesters are strongly incentivized to participate in the protocol.

Theorem 6.1. 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 protocol.

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

First consider a rational requester. Correctly running Finalize (calc, *) allows the requester to receive back $calc.escrow_{req}$, and as such, rational requesters follow the protocol.

Consider now rational computers. If the requester correctly runs Finalize (calc,*), then $calc.\tau$ and $calc.I_{good}$ are generated correctly. Therefore, if all rational computers follow the protocol, the assumption under which we chose $Reward_i$ in Section 5, for a given rational computer c_i correctly running Repsonse(calc,*), c_i is included in $calc.I_{good}$ with probability ω . If c_i incorrectly runs Response(calc,*), c_i is included in $calc.I_{good}$ with probability of at

most γ . By our choice of $Reward_i$, we have seen in Section 5, given $calc.I_{good}$ is generated correctly and computers included in $calc.I_{good}$ receive this with probability 1, this is sufficient for rational computers to compute the result correctly, equivalent to calling Response(calc, *).

Therefore, rational computers and requesters follow the protocol if $Reward_i > \frac{cost(calc)}{\omega - \gamma}$

This result is enough to ensure rational players follow the Marvel DC protocol. However, because of the use of the same reputation and computer-selection mechanism as described in Section 5.2, 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 6.2. For a series of computations $[calc_1, calc_2, ..., calc_i]$ 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 $n_{comp} < \frac{|C|}{2}$ is strictly decreasing in expectancy and converging to 0 as i tends to infinity.

Proof. As $Reward_i > \frac{cost(calc)}{\omega - \gamma}$, from Theorem 6.1 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 initRep, after the

first computation, the selection probability of a Byzantine computer reduces in expectancy to:

$$\frac{\alpha(|C| \cdot initRep + n_{comp}\gamma)}{|C| \cdot initRep + n_{comp}((1 - \alpha)\omega + \alpha\gamma)}.$$
 (6)

First it be can see that

$$\frac{\alpha(|C| \cdot initRep + n_{comp}\gamma)}{|C| \cdot initRep + n_{comp}((1 - \alpha)\omega + \alpha\gamma)} < \alpha$$
(7)

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:

$$\frac{\alpha_{k}(|C| \cdot initRep + n_{comp}\gamma)}{|C| \cdot initRep + n_{comp}((1 - \alpha_{k})\omega + \alpha_{k}\gamma)}$$

$$= \frac{\alpha_{k}(|C| \cdot initRep + \gamma n_{comp})}{|C| \cdot initRep + n_{comp}\omega - \alpha_{k}n_{comp}(\omega - \gamma)}.$$
(8)

We have already seen α_{k+1} equals

$$\frac{\alpha_k(|C| \cdot initRep + n_{comp}\gamma)}{|C| \cdot initRep + n_{comp}\omega - \alpha_k n_{comp}(\omega - \gamma)} < \alpha_k. \tag{9}$$

which implies:

$$\frac{(|C| \cdot initRep + n_{comp}\gamma)}{|C| \cdot initRep + n_{comp}\omega - \alpha_k n_{comp}(\omega - \gamma)} < 1.$$
(10)

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.

Remark 6.3. Lemma 6.2 depends on the output of the on-chain randomness oracle being unpredictable when Request is called. Existing solutions, such as the Chainlink VRF ⁸, provide proofs that provided randomness was generated correctly. Analysis of the quality of this randomness is beyond the scope of this work.

As a direct consequence of Lemma 6.2, with reasonable choices for rewarding functions and number of computers per-computation (explored in Table 3), 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.

6.3. 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 privacy which can be seen as necessary in computations such as those in FL protocols. The privacy provided is based on existing, well-known ZK techniques. However, this additional privacy on top of the novel contributions of being strong incentive compatible, 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 addi-

⁸https://docs.chain.link/docs/chainlink-vrf/

tional 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 S_1 , $r_1 \in \{0,1\}^{O(\kappa)}$, and attach $regID_1 \leftarrow f_{com}(S_1, r_1)$ to the registration message, as described in Section 4.2. Then, when a requester requests a computation, and the indices for computation I are calculated, the requester now generates a Merkle Tree containing the indices as specified in I. This Merkle Tree is the structure to which only selected computers submitting results can prove membership in ZK. Results are therefore associated to a ZK set-membership proof which reveals nothing about the player that generated the proof. In Privacy Marvel DC, this separates result submission and player identity. To maintain this separation of identity from result, ZK set-membership proofs are required in the updated Finalize contract, described later in this section.

In addition to the deposits of Marvel DC, the requester must also deposit a pool of money necessary to incentivize relayers (as described in Section 4.3) 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 Fee(relay), the additional required deposit is $n_{comp} \cdot Fee(relay)$ for the relaying of computer messages during the Respnse phase.

In the Response contract for Privacy Marvel DC, computers selected in I privately generate a ZK-proof proving their membership in I. Formally, given a computation result of result, the computer sets $response \leftarrow encrypt$ (compute(calc), calc.compEncKey). The computer also generates a new S_2 , $r_2 \in \{0,1\}^{O(\kappa)}$ pair, and computes $regID_2 \leftarrow f_{com}(S_2, r_2)$. Setting $m \leftarrow \langle calc, response, r_1, regID_2 \rangle$, the computer generates a NIZKSoK $\pi_1 \leftarrow NIZK-SoK[m]\{(regID_1, r_1) : \text{MemVerify } (I, regID_1) = 1 \& regID_1 = f_{com}(S_1, r_1) \}$. Finally, the computer then publishes m and π_1 to the blockchain through a relayer, who receives Fee(relay) 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, but instead of adding computer indices to $responses_{good}$, the requester adds the corresponding $regID_2$ s. The requester publishes $responses_{good}$ and the encryption of calc.compDecKey using each public key corresponding to computers in calc.I to the blockchain. However, rewards are not immediately distributed to computers in $responses_{good}$.

In the Finalize contract, computers now have a chance to contest the computation of $responses_{good}$ for up to T blocks after the Reveal contract is called. If $responses_{good}$ was computed incorrectly, any of the computers in calc.I can publish the decryption of all results and pass them into the rateComputations function of Marvel DC, proving the incorrect computation of $responses_{good}$ by the requester. In this case, all computers are rewarded, and the requesters escrow is destroyed. To prevent malicious computers in calc.I from attempting this proof in order to reveal computation results, a further escrow is required, which is returned on the correct proving of miscomputation of $responses_{good}$ by the requester.

If instead $responses_{good}$ was computed correctly, any computer whose $regID_2$ is included in $responses_{good}$ can generate a proof of membership to $responses_{good}$. Furthermore, as $regID_1$ can no longer be used for future computations (using the same $regID_1$ would reveal the same r_1 in the next calc), the computer generates a new S_3 , $r_3 \in \{0,1\}^{O(\kappa)}$ pair and corresponding $regID_3 \leftarrow f_{com}(S_3, r_3)$. Setting $m \leftarrow \langle calc, false, regID_3 \rangle$, the computer with $regID_2$ in $responses_{good}$ then generates a NIZKSoK $\pi_2 \leftarrow NIZKSoK[m]\{(regID_2, r_2) : \text{MemVerify } (responses_{good}, regID_2) = 1 & regID_2 = f_{com}(S_2, r_2) \}$. Finally, the computer publishes m and π_2 to the blockchain. In this variation of the protocol, computers not included in $responses_{good}$ must also respond, updating there $regID_1$. If this is not done, computers forfeit $escrow_{comp}$ and any reputation earned.

Observation 6.4. 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, but nothing else can be learned about the submitting player's identity. Similarly, when collecting rewards, or replacing $regID_1$, the only thing learned when a computer submits a valid message during the Finalize phase is to which set, $responses_{good}$ or $responses_{good}$, the computer belongs.

6.4. 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 $responses_{good}$ was calculated correctly, computers included in $responses_{good}$ can instead delay their claiming of the reward and associated reputation increase arbitrarily. After a Finalize phase without arbitration, the set of $regID_2$ s corresponding to $responses_{good}$ 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 setmembership techniques, except now with a larger set in which to diffuse.

7. Implementation Analysis

In this section we analyse Marvel DC and Privacy Marvel DC as described in Sections 6 and 6.3. We show that on top of the unique formal guarantees of Section 6.2, both protocols are cost-effective and practical for both computers and requesters. Our reputation and computer selection mechanisms perform analogously to those graphed in [8]. Crucially however, our protocols are proven to be strong incentive compatible in a fully decentralized setting, where computers an requesters have asymmetric utilities. The encoding of Marvel DC is available here [10].

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

	Marvel DC	Privacy Marvel DC
Setup()	2,000,000	$2,450,000 + h_{\psi} \cdot 150,000^{\dagger}$
Register()	80,000	82,000
Request()	$ 500,000 + n_{comp} \cdot 31,000 + 0.25 \text{ LINK}^{\dagger\dagger} $	$ \begin{vmatrix} 500,000 + n_{comp} \cdot 31,000 \\ + n_{comp} \cdot h_{\psi} \cdot 51,000^{\dagger\dagger\dagger} + 0.25 \text{ LINK}^{\dagger\dagger} \end{vmatrix} $
Response()	60,000	$63,000 +300,000^{\dagger\dagger\dagger\dagger}$
Reveal()	N/A	$80,000 + n_{reward} \cdot h_{\psi} \cdot 51,000^{\dagger\dagger\dagger}$
Finalize()	-	-
-No Arbitration	$80,000 + n_{reward} \cdot 14,000$	(results not revealed) 83,000 + 300,000 ††††
-Arbitration	N/A	$80,000 + n_{comp} \cdot 14,000$

Table 2: Gas costs of each contract call in Marvel DC and Privacy Marvel DC. †Cost of accessing Chainlink randomness. ††Cost to generate a once-off Merkle Tree which can be reused for all computations. ††† Cost to insert computer into copy of Merkle Tree. †††† Cost to verify proof of membership on-chain.

There are several considerations when calculating the cost of running Marvel DC on a blockchain. In the case where a computation has a single 32 byte 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 IPFS⁹, Layer-2 chains or even through an MPC protocol between computers and requesters (not necessarily the same entities involved in the computation). All of these suggestions require the use of checkpoints (such as those proposed in [6]), pieces of summary information identifying permanent states in an off-chain protocol which are published to the main-chain, updating the state of the main-chain.

In Table 2 we present the approximate gas costs for Marvel DC and Privacy Marvel DC on an Ethereum-compatible blockchain of the main operations for a requester and computers in a computation, given n_{comp} and a 256-bit result, with all messages published on-chain. This can be extended to l-dimensional

⁹https://ipfs.io/

results for any l>1. 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. We also include the cost of calling an on-chain randomness oracle in our calculation. This call needs to be made prior to the calculation of the indices for computation, and must be made when depositing rewards and escrow to ensure the requester cannot manipulate the selection of computers. To approximate the costs of running Privacy Marvel DC described in Section 6.3, we utilize n_{ψ} the maximum number of computers required to select in order to guarantee a majority are rational. We let $h_{\psi} = \lceil log_2(n_{\psi}) \rceil + 1$. This is the required height of the Merkle Tree to be used in the ZK-proofs for set-membership.

As an example, using current gas costs of 32 GWEI/gas and 1 GWEI = $\$0.00000335^{-10}$, for $n_{\psi} = 10$, and a computation with $n_{comp} = 10$, for the basic protocol, this is a per computer cost of 140,000 gas, or \$15, while a requesters cost is 950,000 gas, or \$102 + \$4.5 for the LINK¹¹ required to access the onchain randomness ¹². Deploying on a cheaper protocol like Harmony with gas costs of \$0.000000005 at the time of writing ¹³, this is an approximate cost of \$0.001 and \$4.505 respectively.

For Privacy Marvel DC on Harmony with $n_{reward}=5$, this gives $h_{\psi}=4$, and a gas cost of 950,000+ 2,040,000 for requesting, and 80,000+1,020,000 for revealing rewards, a total of 4,190,000 gas (\$0.02), plus the cost for accessing on-chain randomness (\$4.5), a total cost for the requester of \$4.52. For computers, the cost is 746,000 gas (\$0.004) in the optimistic execution, and 503,000 (\$0.0026) in the pessimistic, but only for one computer. This means both implementations of Marvel DC are suitable for immediate deployment on public

¹⁰https://etherscan.io/gastracker, Accessed: 03/04/2022

 $^{^{11}}$ https://docs.chain.link/docs/vrf-contracts, Accessed:03/04/2022

¹²https://www.coingecko.com/en/coins/chainlink, Accessed:03/04/2022

¹³https://explorer.harmony.one/, Accessed: 03/04/2022

blockchains.

7.2. Latency and Throughput

A direct comparison of our protocol to most existing distributed FL solutions with respect to throughput and latency is of limited benefit. This is because the the concept of time in decentralized systems (blocks in a blockchain) has no immediate parallel when centralized parties are required, as in [16, 1, 20, 19, 13, 27, 26].

Considering the protocols which allow for tokenized rewards in a decentralized setting (protocols which do not allow for tokenized rewarding have limited utility in a decentralized setting), this leaves only [25] and [24]. Using the terminology of our 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 6.1) 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 [25] lasts at least 4Tblocks to ensure workers are incentivized to submit at least one correct model update (using the terminology of [25], 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 [24] requires at least 3T blocks, as computers must commit to the data-set to be used, before submitting a computation result. All 3 protocols, including our own, are equipped to spawn arbitrarily many computations in parallel.

7.3. Necessary bounds on the number of computers per-computation

	$n_{comp} = 10$	25	50	100	250	500	1000
$\alpha = 0.45$	0.504	0.694	0.716	0.817	0.936	0.986	0.999
0.4	0.633	0.846	0.902	0.973	0.999	1	1
0.33	0.787	0.958	0.989	1	1	1	1
0.25	0.922	0.997	1	1	1	1	1
0.2	0.967	1	1	1	1	1	1
0.1	0.998	1	1	1	1	1	1
0.05	1	1	1	1	1	1	1
0.01	1	1	1	1	1	1	1

Table 3: The approximate probability, correct to 3 decimal places, of choosing a majority of rational computers given specific starting adversarial % of computers α (left column) and selected numbers of computers n_{comp} (top row) from a sufficiently large population of computers such that adversarial share represents the per-selection probability of selecting an adversarial computer throughout sampling.

To estimate the required value of n_{ψ} required to ensure a sufficiently probably majority of rational players, Table 3 serves as a starting point. We recommend conservative choices of adversarial share and probability of rational majority. For example, given an adversarial share of 20% of computers, any value of $n_{\psi} \geq 25$ suffices to ensure a majority of rational computers are chosen per-computation.

8. Conclusion

We present Marvel DC, a strong incentive compatible 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 incentivisation of rational computers in the presence of Byzantine computers, reputation-aware computer selection and privacy-enhancing techniques which can be bootstrapped to the core Marvel DC protocol to allow for the use of Marvel DC on computations revealing sensitive data about the computers participating in the protocol. 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.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We thank IEEE Fellow Josep Domingo-Ferrer and his group at Universitat Rovira i Vigili for many interesting and motivating conversations related to this paper.

Vitaes

Conor McMenamin



Conor is a Marie Curie Fellow, working as researcher at Nokia-Bell Labs undertaking a PhD in Universitat Pompeu Fabra (UPF). He holds a Bachelor of Science degree in Computational Thinking (2015), and a Master of Science degree in Cryptography (2019). His research focuses on the game-theoretic aspects of blockchain technology, both at the consensus and the application layer.

Vanesa Daza



Vanesa is an Associate Professor at UPF. She holds a degree and a PhD in Mathematics. Her research now focuses on designing cryptographic primitives and protocols to undertake security and privacy challenges on blockchain technologies. She has co-authored over 40 papers and is the co-inventor of two international patents. She has served as Associate Editor for two IEEE Transactions journals, several Program Committees and reviewer of international journals. She currently coordinates two H2020 projects and collaborates with several companies. Likewise, she has served at UPF in several roles, including chairing the Department of Information and Communication Technologies.

References

- [1] Xianglin Bao, Cheng Su, Yan Xiong, Wenchao Huang, and Yifei Hu. FLChain: A blockchain for auditable federated learning with trust and incentive. In 2019 5th International Conference on Big Data Computing and Communications (BIGCOM), pages 151–159, 2019.
- [2] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In 2014 IEEE Symposium on Security and Privacy, pages 459–474, 2014.
- [3] Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris Kolonelos. Zero-knowledge proofs for set membership: Efficient, succinct, modular. In Nikita Borisov and Claudia Diaz, editors, Financial Cryptography and Data Security, pages 393–414, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

- [4] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 118–128, Red Hook, NY, USA, 2017. Curran Associates Inc.
- [5] Vincenzo Botta, Daniele Friolo, Daniele Venturi, and Ivan Visconti. Shielded computations in smart contracts overcoming forks. In Nikita Borisov and Claudia Díaz, editors, Financial Cryptography and Data Security, pages 73–92, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.
- [6] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. https://arxiv.org/pdf/1710.09437, 2017. Retrieved: 19/04/2022.
- [7] Josep Domingo-Ferrer, Oriol Farràs, Sergio Martínez, David Sánchez, and Jordi Soria-Comas. Self-enforcing protocols via co-utile reputation management. *Information Sciences*, 367-368:159–175, 2016.
- [8] Josep Domingo-Ferrer, David Sánchez, Alberto Blanco-Justicia, and Najeeb Jebreel. Co-utile peer-to-peer decentralized computing. In 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), pages 31–40, 2020.
- [9] Lisa Eckey, Sebastian Faust, and Benjamin Schlosser. Optiswap: Fast optimistic fair exchange. In *Proceedings of the 15th ACM Asia Conference* on Computer and Communications Security, ASIA CCS '20, page 543–557, New York, NY, USA, 2020. Association for Computing Machinery.
- [10] Github. https://github.com/The-CTra1n/MarvelDC, 2022.
- [11] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, Advances in Cryptology
 EUROCRYPT 2016, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

- [12] Kobi Gurkan, Koh Wei Jie, and Barry Whitehat. Community proposal: Semaphore: Zero-knowledge signaling on Ethereum. https://github.com/appliedzkp/semaphore, 2020. Retrieved: 19/04/2022.
- [13] Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Blockchained on-device federated learning. *IEEE Communications Letters*, 24(6):1279–1283, 2020.
- [14] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacypreserving smart contracts. In 2016 IEEE Symposium on Security and Privacy (SP), pages 839–858, 2016.
- [15] Harry C. Li, Allen Clement, Edmund L. Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. Bar gossip. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, page 191–204, USA, 2006. USENIX Association.
- [16] Yuzheng Li, Chuan Chen, Nan Liu, Huawei Huang, Zibin Zheng, and Qiang Yan. A blockchain-based decentralized federated learning framework with committee consensus. *IEEE Network*, 35(1):234–241, 2021.
- [17] Chao Lin, Debiao He, Xinyi Huang, and Kim-Kwang Raymond Choo. OBFP: Optimized blockchain-based fair payment for outsourcing computations in cloud computing. *IEEE Transactions on Information Forensics and Security*, 16:3241–3253, 2021.
- [18] Ziyao Liu, Jiale Guo, Wenzhuo Yang, Jiani Fan, Kwok-Yan Lam, and Jun Zhao. Privacy-preserving aggregation in federated learning: A survey. https://arxiv.org/abs/2203.17005, 2022. Retrieved: 30/06/2022.
- [19] Chuan Ma, Jun Li, Ming Ding, Long Shi, Taotao Wang, Zhu Han, and H. Vincent Poor. When federated learning meets blockchain: A new distributed learning paradigm. https://arxiv.org/abs/2009.09338, 2020. Retrieved: 30/06/2022.

- [20] Umer Majeed and Choong Seon Hong. FLchain: Federated learning via mec-enabled blockchain network. In 2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS), pages 1–4, 2019.
- [21] Conor McMenamin, Vanesa Daza, and Matteo Pontecorvi. Achieving state machine replication without honest players. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, page 1–14, New York, NY, USA, 2021. Association for Computing Machinery.
- [22] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from Bitcoin. In 2013 IEEE Symposium on Security and Privacy, pages 397–411, 2013.
- [23] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '91, page 129–140, Berlin, Heidelberg, 1991. Springer-Verlag.
- [24] Timon Rückel, Johannes Sedlmeir, and Peter Hofmann. Fairness, integrity, and privacy in a scalable blockchain-based federated learning system. Comput. Netw., 202(C), jan 2022.
- [25] Kentaroh Toyoda and Allan N. Zhang. Mechanism design for an incentive-aware blockchain-enabled federated learning platform. In 2019 IEEE International Conference on Big Data (Big Data), pages 395–403, 2019.
- [26] Muhammad Habib ur Rehman, Khaled Salah, Ernesto Damiani, and Davor Svetinovic. Towards blockchain-based reputation-aware federated learning. In IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pages 183–188, 2020.
- [27] Jiasi Weng, Jian Weng, Jilian Zhang, Ming Li, Yue Zhang, and Weiqi Luo. Deepchain: Auditable and privacy-preserving deep learning with blockchain-based incentive. IEEE Transactions on Dependable and Secure Computing, 18(5):2438–2455, 2021.

[28] Yinghui Zhang, Robert H. Deng, Ximeng Liu, and Dong Zheng. Blockchain based efficient and robust fair payment for outsourcing services in cloud computing. *Information Sciences*, 462:262–277, 2018.