Demystifying incentives in the consensus computer

Loi Luu, Jason Teutsch, Raghav Kulkarni, Prateek Saxena
National University of Singapore
{loiluu, teutsch, raghav, prateeks}@comp.nus.edu.sg

ABSTRACT

Bitcoin and similar cryptocurrencies are a massive network of computational devices that maintain the robustness and correctness of the computation done in the network. Cryptocurrency protocols, including Bitcoin and the more recent Ethereum system, offer an additional feature that allows currency users to specify a "script" or contract which is executed collectively (via a consensus protocol) by the network. This feature can be used for many new applications of cryptocurrencies beyond simple cash transaction. Indeed, several efforts to develop decentralized applications are underway and recent experimental efforts have proposed to port a Linux OS to such a decentralized computational platform.

In this work, we study the security of computations on a cryptocurrency network. We explain why the correctness of such computations is susceptible to attacks that both waste network resources of honest miners as well as lead to incorrect results. The essence of our arguments stems from a deeper understanding of the incentive-incompatibility of maintaining a correct blockchain. We explain this via a ill-fated choice called the *verifier's dilemma*, which suggests that rational miners are well-incentivized to accept an unvalidated blockchain as correct, especially in next-generation cryptocurrencies such as Ethereum that are Turing-complete. To explain which classes of computation can be computed securely, we formulate a model of computation we call the consensus verifiability. We propose a solution that reduces the adversary's advantage substantially, thereby achieving near-ideal incentive-compatibility for executing and verifying computation in our consensus verifiability model. We further propose two different but complementary approaches to implement our solution in real cryptocurrency networks like Ethereum. We show the feasibility of such approaches for a set of practical outsourced computation tasks as case studies.

1. INTRODUCTION

Cryptocurrencies such as Bitcoin [1] are attracting massive investments in computing power, and the power consumed has been growing exponentially in recent years [2]. Bitcoin can be viewed as a large network of miners competing in a lottery that awards newly minted currency, called *Bitcoins*, in exchange for contributing computational resources to solutions of cryptographic puzzles, or *blocks*. Bitcoin miners collectively agree upon who receives the minted Bitcoins and which *transactions* to accept. This process of *consensus*, or agreement by majority, permanently records decisions in a public ledger called the *blockchain*. More than 50 cryptocurrencies use similar blockchain protocols. While the core blockchain mechanism has been used for establishing a public ledger of who-pays-whom transactions, it has features that go beyond this function. Specifically, the blockchain supports a light-weight *scripting* language, designed primarily to allow conditional transactions which can repurposed for other applications. Emerging cryptocurrencies now enable computation for applications such as financial back-offices, prediction markets, distributed computation (e.g. Gridcoin for BOINC [3]), and even a decentralized Linux OS [4].

In Bitcoin, a transaction defines a particular activity in the network, *e.g.*, sending Bitcoin between users. Transactions may include a script that specifies a validity condition. Figure 1 illustrates a basic transaction script which checks whether the payee in the transaction has the private key corresponding to the recipient's Bitcoin wallet address. More interestingly, next-generation cryptocurrencies such as Ethereum [5] have surfaced. Ethereum introduces a Turing-complete script language which makes it possible to encode arbitrary computation as scripts. In fact, the makers of Ethereum claim that their system can support arbitrary decentralized applications. We can thus view cryptocurrencies as a consensus protocol which involves a massive network of computing nodes collaborating to both execute and verify computational tasks. We refer to this network collectively as a *consensus computer*. We wish to characterize the classes of computation that users can trust a cryptocurrency-based consensus computer to execute and verify correctly.

Miners have two separate functions in the consensus computer: checking that blocks are correctly constructed, or *proof-of-work*, and checking the validity of transactions in each block. While verifying correct block construction requires a relatively small amount of work (two `SHA256` calculations), checking the validity of transactions contained in a block can take much more time for two reasons. First, the number of transactions per block may be large ($\approx$ 800 in Bitcion at the time of writing, and its capacity may soon be extended to support high transaction rates [6, 7]). Second, expressive transaction scripts in emerging cryptocurrencies such as Ethereum can require significant computational work to verify. These expressions create a new *verifier's dilemma* for miners — whether the miners should verify the validity of scripted transactions or accept them without verification. Miners are incentivized to verify all scripted transactions for the "common good" of the cryptocurrency so to speak. However, verifying scripts consumes computation resources and therefore delays honest miners in the

```
1 Input:
2   PreviousTX: ID of previous transaction
3   Index: 0
4   scriptSig: Sign(PubKey), PubKey
5
6 Output:
7   Value: 5000000000
8   scriptPubKey: %take Signature and PubKey as params
9     checkif Hash(PubKey) = Payee's ID,
10    checkif Sign(PubKey) is valid
```

Figure 1: Illustration of a simple transaction in Bitcoin. User's address is computed by hashing their public key. `scriptPubKey` is the script that defines how the payee claims the recieved Bitcoin.

race to mine the next block. We argue that this dilemma leaves open the possibility of attacks which result in unverified transactions on the blockchain. This means that some computation tasks outsourced to cryptocurrency-based consensus computers may not execute correctly.

We propose a new consensus computer primitive called *consensus verifiability* which generates correct solutions to puzzles in which the computational burden of verification is low. Consensus verifiability differs from verifiable computation on classical computers [8, 9, 10, 11, 12, 13, 14], in that it allows complete decentralization of computation — the puzzle giver need not trust individual verifiers nor the prover who provides the solution. Previous works implicitly assume that Bitcoin miners will verify and agree on correct transactions [15, 16, 17, 18, 19, 20, 21]. Our present model provides a formal explanation as to why this assumption holds and suggests potential constraints of consensus computation in cryptocurrencies with more expressive scripting languages. When the computational advantage of skipping verification is low, say ε, rational miners gain little by cheating. We call a system restricted to such primitives *ε-consensus verifiability* (ε-CV) and expect it to compute correctly.

We propose two mechanisms to realize ε-CV on Ethereum. Our first approach allows us to achieve correctness by splitting the computation into several smaller steps such that the consensus correctly verifies each step. This approach achieves exact correctness in results but with higher computational burden to the network. Our second mechanism allows for *approximate correctness*. Specifically, we allow the approximation gap between submitted and correct results to be tunable to a negligible quantity, at much lower computational cost. Whether one can design a distributed, cryptocurrency system which permits secure execution of a larger class of computations remains an interesting open problem, as is the problem of determining the class of puzzles whose solutions admit light-weight verification.

Contributions. In summary, our work makes the following contributions:

- *Verifier's dilemma and attacks.* We introduce a dilemma in which miners are vulnerable to attacks regardless of whether they verify a transaction or not. We further show that miners are incentivized to skip the verification and perform an attack to get more advantage in mining the next blocks.

- *Consensus verifiability model.* We formalize the computation and verification by consensus as a novel *consensus verifiability* model. We study the threat model and the assumption in which the computation performed by our consensus verifiability model is correct. We introduce an ε-consensus verifiability model which incentivizes rational miners to behave correctly.

- *Techniques to realize ε-CV.* We propose techniques to realize our ε-CV model in real cryptocurrency networks like Ethereum. We illustrate the practical utility of our techniques with examples from outsourced computation.

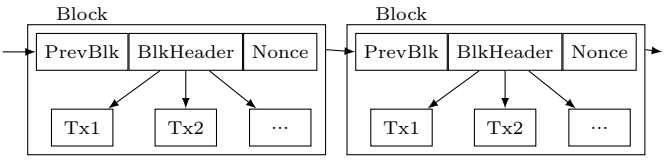

Figure 2: The Blockchain in popular cryptocurrencies like Bitcoin and Ethereum. Each block consists several transactions (Txs).

2. BACKGROUND: CRYPTOCURRENCIES

2.1 The consensus protocol

Most cryptocurrencies use a public peer-to-peer consensus protoco known as *Nakamoto consensus*, named after and introduced by the founder of Bitcoin [1], which does not require a central authority. At the heart of this protocol is a blockchain which acts as a public ledger and stores the complete transaction history of the network. The security of the blockchain is maintained by a cryptographic chain of puzzles (or blocks). Miners validate and approve transactions while generating, or *mining*, new blocks. Mining a new block rewards newly minted coins to one of the miners that demonstrates by consensus that it successfully solved the cryptographic puzzle. Figure 2 concisely illustrates the structure of the blockchain data structure.

The protocol uses consensus in two places to make the cryptocurrency robust. First, the network must agree on the rules to verify valid blocks and transactions. Second, the data of blockchain must be consistent across miners, so that everyone knows who owns what. Thus the blockchain acts as a base to verify which transactions are valid.

2.2 Transactions & Scriptability

A transaction in a cryptocurrency defines a particular activity in the network of that currency. For example, Figure 1 is a basic transaction in Bitcoin which transfers Bitcoins from sender to receiver. The `scriptPubKey` component (Line 8) allows the sender to define the receiver's address and on which condition he can spend the Bitcoins. The sender can dynamically program the `scriptPubkey` to support various use cases.

Transaction verification. In order to verify a transaction, one has to check whether the input provided in `scriptSig` satisfies the logic encoded in the `scritptPubkey` of the `PriviousTX`. For example, in Figure 1, miners will check if the receiver is the intended payee (on Line 9) and he indeed owns the payee's address (on Line 10). The verification of a transaction `TX` happens in two places, when:

- *A new transaction is broadcast* (step 1). When someone broadcasts `TX` to a miner, he verifies if the transaction is correct according to the latest blockchain. If so, he includes it in his block header to mine a new block, and propagates the transaction to his neighbors.

```
1 code:
2     if msg.datasize==2:
3         return msg.data[0] + msg.data[1]
```

Figure 3: An Ethereum contract that returns sum of two numbers.

- *A new block is broadcast* (step 2). When TX is included in a newly found block, everyone before accepting the block will check the correctness of all transactions contained in the block.

2.3 Incentivizing Correctness

While it is clear that Nakamoto consensus incentivizes a miner to mine new blocks with a reward of newly minted coins, the incentive for others to correctly verify the transactions is not generally understood. The folklore reasons for why miners verify transactions in the Bitcoin blockchain are:

- Verifying a transaction at both steps requires negligible additional work compared to mining a new block. For example, the Bitcoin transaction in Figure 1 has only 4 opcodes. Miners can do this extra check without expending any significant computational work. The cost of validation outweighs the miner's risk of having their earned Bitcoins in new blocks be discarded, if the invalidity is detected in the future.

- When receiving a new block, miners can accept it with or without verifying the included transactions. However, *most* miners want to maintain a cheap and correct system (the "common good"), so that the Bitcoin blockchain remains healthy and Bitcoins act as a store-of-value. Thus miners check the validity of a block's included transactions for free [1].

In this work, we study the financial incentives of users more carefully to understand when these folklore assumptions may fail and what happens when they do.

2.4 Ethereum—Turing-complete Scripting

Ethereum is a next generation cryptocurrency which enables more applications beyond those supported by Bitcoin. It provides a Turing-complete programming language in its design which equips users with a mechanism to express concrete conditional semantics for transactions.

Smart contract. A smart contract can embed many contractual clauses and make it expensive for anyone to disoblige or deviate from the contract after agreement [22]. In Ethereum, *smart contract* (or *contract*) is an independent entity staying in the blockchain. The smart contract has its own wallet address, balance, and storage space which is used to run a specified script. While Bitcoin only allows users to encode a stateless program, smart contracts support stateful programs. Users can trigger a contract by sending a transaction to the contract's wallet address. Once a transaction to a contract is included in the blockchain, everyone in the network is expected to execute the contract script to verify the contract execution and the contract state. Figure 3 is a simple contract which returns the sum of two numbers.

[1]Note that only the one who finds the block receives a transaction fee, not those who verify

Gas system. It may be obvious to some readers that having Turing-complete language in Ethereum script is a problem. More specifically, users can write a transaction/-contract script with long verification time to perform Denial-of-Service (DoS) attack on the network. For example, one can write a simple transaction to make the network enter an infinite loop while verifying the transaction. To prevent such attacks, Ethereum introduces the concept of *gas* [23]. The script is compiled into Ethereum opcodes while stored in the blockchain, and each opcode costs some predefined amount of *gas* (a form of transaction fee) charged to transaction sender. When a sender sends a transaction to activate a contract, he has to specify the available `gasLimit` that he supports for the execution. The gas is paid to a miner who verifies and includes the transactions in his block. Intuitively, the gas system seems to make it expensive for the attacker to perform DoS attack to the system. However, as we later show, this mechanism does not actually solve the DoS attack problem.

3. THE VERIFIER'S DILEMMA

In Section 2.3, we discussed the incentives for Bitcoin miners to verify transactions. We show that the same motivation argument does not carry over to Ethereum where users can create scripts for transactions/contracts which place arbitrary computational demands on miners. Transactions whose verification requires intensive computation introduce new security vulnerabilities, as Ethereum and Bitcoin share nearly identical incentive structures for mining blocks [5]. Indeed, the only difference implemented in Ethereum's deployed testnet includes a small reward for late, valid "uncle block" extensions.

In this section we present a *verifier's dilemma* in which the honest miners in the network decide whether to skip the verification of expensive transactions or to maintain the common good. To describe the verifier's dilemma, we first introduce a motivating smart contract which we will use throughout the paper.

3.1 Example of Outsourced Computation

Figure 4 shows a code snippet of a contract in Ethereum which allows a problem giver (G) to ask anyone to compute $A \times B$ where A, B are two matrices of size $n \times n$. In the `init` phase, G first sends an transaction which specifies n, A, B, and deposits the reward amount to the contract's wallet address. All information is public including the reward amount since everything is stored on the blockchain. If a prover (P), *i.e.*, a user is interested in solving the problem, wants to claim the solution to get the reward, he sends a transaction with the result matrix C to the verifier.

When other miners receive the transaction from the prover, they will verify if $C = A \times B$ by running the `code` phase in the contract script. G expects that the Nakamoto consensus protocol used in Ethereum will ensure that the result is correct due to the check on Lines 22–27, and only one prover will get paid due to the update of the contract's status on Line 30. Note that for the purpose of demonstration, we use an example in which verifying C requires to run the actual computation again. In practice, there are several problems that verifying whether a solution is correct is much easier than finding one, *e.g.*, solving a SAT instance, finding a hash inversion, breaking cryptography key and so on. Thus, G can create a contract to allow anyone to submit their so-

```
init:
    #Record the initiator, reward and data
    contract.storage[0]=msg.sender
    contract.storage[1]=0
    contract.storage[2]=msg.value
    #record the size n of the matrices
    contract.storage[3]=msg.data[0]
    #record the matrices A and B
    contract.storage[4] = msg.data[1]
    contract.storage[5] = msg.data[2]
    contract.storage[6] = 1 #status
code:
    if contract.storage[6] == 0:
        return (1)
    #data[C]: C is the result matrix
    if msg.datasize == 1:
        C = msg.data[0]
        n = contract.storage[3]
        A = contract.storage[4]
        B = contract.storage[5]
        # checking the result
        for i in range(n):
            for j in range(n):
                cell = sum([A[i][k] * B[k][j]
                    for k in range(n)])
                if cell != C[i][j]:
                    return (0)
        #if everything is fine, send the reward
        send(1000,msg.sender,contract.storage[2])
        contract.storage[6]=0 #update status
        contract.storage[7]=C #store result
        return (2)
```

Figure 4: Code snippet of a contract which allows an user can outsource a matrix multiplication problem. The contract will verify the correctness of the result before sending out the reward.

lution and rely on the network to verify the correctness as in Figure 4.

3.2 Attacks

When a transaction that asks miners to verify whether $C = A \times B$ appears in the network, miners have an option to either verify or not to verify. We show that the miners are susceptible to a *resource exhaustion attack* or a *incorrect transaction attack* depending on their choice.

P1: Resource exhaustion attack by problem givers. If miners honestly follow the protocol, they voluntarily verify all transactions in a new block which is broadcast to them (step 2 in Section 2.2). Thus, if an attacker broadcasts his expensive transactions to the network, other miners will have to spend a lot of power and time to verify all the transactions. To prevent such situations, Ethereum introduced a *gas* system to charge the transaction initiator (sender) some amount of money for each operation he wants verified.

However, the gas system does not prevent the attacker from creating and including those resource-intensive transactions in his newly minted block. This is because transaction fee (*i.e.*, gas) is collected by the block founder only (in step 1, Section 2.2). Thus, the attacker does not lose anything by adding his transactions to his own block. The other miners, on the other hand, have to spend a *significant* amount of time verifying those transactions (in step 2, Section 2.2) and get nothing in return. As a consequence, the attack not only exhausts other miners' resource, but also gives the attacker some time ahead of other miners in the race for the next block. Note that the attacker has complete freedom to prepare the transactions, so the difficulty of verifying the transaction script is in his control. Since gas charged in each transaction is credited to his account in step 1, his attack works at a zero fee (no loss).

As a concrete attack on the running example, the attacker first introduces the contract in Figure 4 with big matrices size, say $n = 1000$, and a small reward so that no one will attempt to solve it. In the second step, he includes a transaction with an arbitrary matrix C that he knows whether $C = A \times B$ before hand in all the blocks that he is mining. He also prepares enough gas to execute the contract script so that the transaction looks valid and normal. Other honest miners on receiving the block from the attacker spend time verifying the block and all the included transactions to see if the block is valid and move on to the next one in the chain. Since n is quite large, verifying a single transaction from the attacker will take significantly more time than normal transaction. The mining process of other miner will be delayed, while the attacker enjoys considerable advantage and has higher chance in finding the next valid blocks in the blockchain.

Remark 1. In Ethereum, the number of operations one includes in a new block is bounded by `oplimit`, which can be varied by a certain rate after every block by miners [5]. However, `oplimit` does not prevent the above attack since the attacker can increase `oplimit` to a large enough value after some blocks to include his resource-intensive transactions. We explain this in the Appendix A. Further, some may argue that attacker's blocks are likely to become stale, *i.e.*, other blocks get broadcast to the network faster, due to the long verification time. However, in [24], Miller *et al.* find that there are (approx. 100) influential nodes in the Bitcoin network which are more efficient in broadcasting blocks. An attacker can broadcast his block to those influential nodes to reduce the chance that his blocks getting staled *significantly*.

P2: Incorrect transaction attack by provers. Due to attack **P1**, rational miners have strong incentive to skip expensive transactions to compete for the race of the next block. The mechanism that drives the Bitcoin network (Section 2.3) to achieve a consensus does not hold in Ethereum. This is because verifying a transaction now may take much more time and affect the miners' mining speed.

As the result, the malicious prover P can include a transaction which has a wrong solution C. Since verifying $C = A \times B$ requires long time, rational miners are well-incentivized to accept it as correct without running the verification check. Thus, the result of a contract/transaction, although is derived from the consensus of the network, is incorrect and unreliable. The problem giver G wastes his money for an incorrect answer. Unlike in **P1**, G is a honest miner in this case. However his computational power is not enough to match the rational miners who are incentivized to skip the check.

It is clear that skipping the verification is problematic. First, P can include anything as the result of the contract execution, *e.g.*, sending others' money deposited in the contract's wallet to his wallet. Second, the health of the currency is affected since it is impossible to correctly verify who-owns-what. However, if the miners naïvely verify all the transactions, they are vulnerable to the **P1** attack.

3.2.1 Findings

The attacks **P1, P2** are not only specific to the running example, but are common challenges to any application that relies on the consensus protocol. From the *verifier's dilemma*, we establish the following findings and implications.

Claim 2 (Resource exhaustion attack)**.** In Turing-complete cryptocurrencies like Ethereum, the honest miners are vulnerable to amplified resource exhaustion attacks. Malicious miners can perform the attack with 0-fee and gain significant advantage in finding the next blocks. Further, the attack is applicable to most cryptocurrency once they become widely adopted, regardless of the applications running on top of the cryptocurrency. One can imagine the problem will be prominent in Bitcoin if the scalability of Bitcoin drives the block size up dramatically to, say, 1 GB per block. An attacker are incentivized to create such a huge block to waste others' resources as well as gain more advantage in the next block race.

Claim 3 (Ethereum protocol is not incentive compatible)**.** In Ethereum, miners have incentive to deviate from the correct behavior and skip the verification of transactions to gain more advantage in finding the next blocks. As a result, the main blockchain will become the unverified one.

Proof sketch. Due to the resource exhaustion attack, the time and computational resource required to verify all transactions in a block will expend significantly. For example in Figure 4, one can set n arbitrarily large to create a contract that costs, say, 30% of the computational power used in mining a block to verify its execution. That demand will slow down the mining process of miners and incentivize rational miners to skip verification in order to maintain their search speed in finding new blocks. Even thought the remaining honest miners still account for the majority of the computational power in the network, their effective power is lessened by the 30% extra effort they spend on verifying transactions. Thus with high probability the rational miners will find more blocks and get a longer blockchain by skipping the verification of these transactions. By Nakamoto consensus, the longer chain will be considered the main chain [2] and contain unverified transactions. □

3.2.2 *Verifier's dilemma in Bitcoin*

As the number of transactions increases per block on the Bitcoin network, the "common good" work required to verify all the transactions in a block approaches a nontrivial quantity. Indeed, some Bitcoin miners have already decided not to verify transactions. On July 4 this year a serious incident on the Bitcoin network was reported, wherein large pools extended blocks containing invalid transactions [25]. These pools mined on blocks without first verifying the block's transactions and caused a fork in the blockchain.

4. INCENTIVIZING CORRECTNESS

In this section, we study and address the design drawbacks of the consensus protocol to prevent the aforementioned **P1** and **P2** attacks. Our goal is to incentivize miners to verify all transactions in a new block broadcast to them. Typically, a correct consensus protocol should achieve the following property.

- *Correct behavior is incentivized.* The protocol should incentivize all users to honestly follow the protocol. Specifically, honest miners should suffer negligible disadvantage from dishonest or malicious miners.

We propose a more formal incentive model which generalizes beyond a specific cryptocurrency and can explain why the above properties are satisfied in Bitcoin network but not in Ethereum. We further propose our solution to make Ethereum an incentive-compatible network in this section.

4.1 Model: Consensus verifiability

We first define the *consensus verifiability* model which formalizes the verification process of transactions/contracts in any consensus protocol as following.

Definition 4 (Consensus verifiability)**.** A *consensus verifiability* protocol involves three parties.

- Problem giver (G): who needs a solution for his particular problem and provides a function f which verifies the correctness of a solution.
- Prover (P): who submits a solution s to claim the reward from G.
- Verifiers (miners or the *consensus computer*) (V): miners in the network execute $f(s)$ to decide whether s is a correct solution.

In Bitcoin, the verification function f is rather simple. For the transaction in Figure 1, the problem that a sender (G) asks is to determine whether a receiver is the intended payee. The solution s that the receiver (P) needs to provide is his public key signed by his private key. The miners (V) will execute a function f defined in `scriptPubKey` to determine if s is correct and P can spend the received amount in that transaction. Miners all try to find new blocks to get reward as newly minted coins. We let W_{blk} be the average work required to successfully mine a new block.

Let us denote by W_f the amount of work required to execute a verification function f. We define the advantage of rational miner as follows.

Definition 5 (Advantage for rational miners)**.** The advantage for a rational miner by skipping the verification of a transaction with verification function f is:

$$Adv(f) = W_f - W_{df} \tag{1}$$

where W_{df} is the work required by deviating from the honest protocol.

Generally $W_{df} = O(1)$, which is the cost of picking a random result in $\{0, 1\}$ or even $W_{df} = 0$ if the miners just answer a constant value. Based on Definition 5, we have the advantage that a dishonest miner can get by skipping the verification process in one block as:

$$Adv(Blk) = \sum_{i<=N} Adv(f_i) = \sum_{i<=N} W_{f_i} - O(1) \tag{2}$$

where N is the total number of transactions in a block Blk, and f_i is the verification function for the i-th transaction in Blk. Our threat model assumes an ε-rational miner as defined in Definition 6.

Definition 6 (ε-rational miners)**.** An ε-rational miner is one who honestly verifies the transactions in a block iff

$$Adv(Blk) \leq \varepsilon W_{blk},$$

and deviates from it otherwise.

[2] In Bitcoin, the protocol picks the blockchain which has more amount of work (the sum of the difficulties in blocks). However, it will be almost the same as comparing the lengths of both the blockchains

A ε-rational miner is incentivized to deviate from the honest protocol if the the work required to verify all transactions in a block is higher than a threshold εW_{blk}. Being dishonest helps him have a higher chance of finding the next blocks since others have to do a significant amount of work verifying transactions. On the other hand, if the work required is less, skipping the verification does not gain him much advantage (*e.g.*, lesser than a other practical factors like network latency) to make his blockchain the longest one. Doing that may even risk his new block value if other miners detect that the previous block includes invalid transactions.

Incentivizing correct consensus verifiability. Our solution for incentivizing miners to correctly execute f is to limit the amount of work required to verify all transactions in a block. Our goal is to provide an upper bound in the advantage that miners get by deviating from the honest protocol. That means honest miners are also guaranteed not to run long and expensive scripts while verifying the transactions. More specifically, we propose a new model called ε-consensus verifiability defined as follows.

Definition 7 (ε-consensus verifiability)**.** An *ε-consensus verifiability (ε-CV)* is a consensus verifiability protocol in which the verification functions cumulatively require successful miners to do *at most* εW_{blk} work per block.

From Definition 6, our ε-CV model is incentive-compatible w.r.t to ε-rational. More specifically, it incentivizes miners to behave correctly, *i.e.*, honestly verify all transactions in a block. Depending on how rational the miners in the network are, one can adjust ε accordingly to incentivize them, otherwise the network is not incentive-compatible.

In Bitcoin, a transaction requires only $O(1)$ work, (and $\varepsilon W_{blk} = 6000 \times O(1)$ [3]) since it only requires running basic arithmetic operations (*e.g.*, addition, multiplication, hashing) to execute. Executing those operations cost almost as much as generating a random answer in miners' physical machines.

4.2 Applying ε-CV model in Ethereum

We estimate the value of ε to be the largest amount of "common good" work that the majority of miners find acceptable. This value, however, depends on several factors including the real net-worth of applications on the currency, the network properties, the incentive mechanism in the cryptocurrency, and individual miner's beliefs about the currency's value. Estimating ε is a separate and interesting research problem.

We next describe how to support ε-CV based on Ethereum without requiring any major changes in its design. We further discuss which classes of computations can be run correctly on the existing cryptocurrency. While in general it is non-trivial to estimate the computation required by programs written in a Turing-complete language [27], the gas charged for a transaction is a reliable indicator of the amount of work required in that transaction script. To make our approach clearer, we define a gas function $G(x)$ as in Definition 8.

Definition 8 (Gas function)**.** The gas function $G(x)$ determines the maximum *gas* amount that a program can require to do x amount of work.

[3] 6000 is the maximum number of transactions that can be included in a block with the current block size [26]

Since Ethereum already specifies the gas value for each opcode in their design [23], computing $G(x)$ is relatively easy. Moreover, $G(x)$ can be computed once and used for all transactions. It only requires to update $G(x)$ again if the gas value is changed, or more opcodes are enabled.

We need only introduce a critical constraint on the transaction to make Ethereum an ε-CV. That is, miners should only accept to verify the transaction that has `gasLimit` bounded $G(\frac{\varepsilon W_{blk}}{N_0})$, where N_0 is the maximum number of transactions can be included in a block. Generally N_0 is fixed and represents the upper bound on the computational capacity of the network in one block time. We formally state our solution in Lemma 9.

Lemma 9 (Achieving ε-CV in Ethereum)**.** *Given a specific ε value, one can construct an ε-CV from Ethereum by limiting the* `gasLimit` *in every transaction by*

$$G\left(\frac{\varepsilon W_{blk}}{N_0}\right).$$

Proof. Let us denote W_{tx} as:

$$W_{tx} = \frac{\varepsilon W_{blk}}{N_0}.$$

Since the `gasLimit` is at most $G(W_{tx})$, W_{tx} is the upper bound on the amount of work required to verify a transaction. Thus, the work required in verifying all transactions in a block is no greater than εW_{blk}. By Definition 6, this an incentive-compatible strategy for ε-rational miners. □

There are certainly classes of computations that require more than W_{tx} and even εW_{blk} work to execute. While one can code such scripts in Ethereum, these puzzles fall outside the class of computations that our ε-CV model can guarantee to execute and verify correctly.

4.3 Supporting more applications on an ε-CV

We discuss two techniques for supporting puzzles which require greater than W_{tx} verification in Ethereum. One technique achieves correctness but sacrifices performance latency since it distributes the computation across multiple transactions and blocks that fit in the ε-CV model. This technique effectively amortizes the verification cost across multiple transactions and blocks in the blockchain. The second technique, on the other hand, sacrifices only exactness and achieves probabilistic correctness.

4.3.1 Exact consensus verifiability

We introduce our first technique in the context of exact solutions. Here we simply split the execution of the script into smaller steps such that verifying each step requires at most W_{tx} work. Once all the steps have been run and verified as separate scripts, the outcome is the same as the result one would obtain from correctly executing the original script. Although the total verification work that the ε-CV has to do is the same regardless of whether we use a single or multiple scripts, splitting the task guarantees a correct outcome because the ε-CV correctly verifies smaller. We illustrate an alternative implementation for outsourcing matrix multiplication in n^2 steps where W_{tx} is $O(n)$ in Figure 5 (instead of 1 transaction where W_{tx} is $O(n^3)$ as in Figure 4).

The execution of the contract in Figure 5 works as follows. P first submits C to the contract as the solution for $A \times B$. The verification happens across the next n^2 steps when P

```
1 init:
2     ...
3     contract.storage[7] == 0 #no. of rounds
4 code:
5     ...
6     if msg.datasize==1:
7         contract.storage[8]=msg.data[0] #store C
8         contract.storage[9]=msg.sender #store prover
9         return ("Submitted C")
10    #verify the result
11    elif msg.datasize == 0:
12        C = contract.storage[8]
13        i=contract.storage[7]/n
14        j=contract.storage[7]%n
15        contract.storage[7] += 1
16        cell = sum([A[i][k] * B[k][j]
17                    for k in range(n)])
18        if cell != C[i][j]:
19            return ("Invalid result")
20    #after n^2 checks have passed
21    #send the reward to the prover
22    if contract.storage[7] == n*n:
23        send_reward()
24        ....
```

Figure 5: Reimplement a contract which allows an user can outsource a matrix multiplication problem in a $O(n)$ consensus compuer.

sends n^2 transactions to the contract. Each transaction verifies a particular and different element $C_{i,j}$ of the submitted result. A counter stores the number of passed checks (Lines 13–15).

Advantage across multiple transactions. A careful reader may be concerned that a rational miners could recognize the global connection between the n^2 transactions and skip verifying all of them to gain an advantage. However, the advantage from skipping the verification of transactions in the i-th block only helps the miners find the $(i+1)$-th block faster, not other blocks in the blockchain. In other words, one cannot "save" the "unspent" advantage in skipping the verification in one block and use that in the future since the competition for finding a new block "restarts" after every block. Essentially, we amortize the advantage of the computation across multiple transactions and blocks such that at any given block, the advantage of rational miners is bounded by ϵW_{blk}. Thus our approach does not break the incentive compatibility of the ε-CV model.

Depending on the nature of the application, multiple steps can either happen in parallel or must be executed sequentially. For example in matrix multiplication, P can send simultaneously n^2 transactions. The reason is verifying $C_{i,j}$ does not rely on the correctness of $C_{i,j-1}$, and the order of verifying the two elements does not matter. On the other hand, if, say, $C_{i,j} = 2 \cdot C_{i,j-1}$, the prover would have to wait until $C_{i,j-1}$ gets verified before sending the next transaction which verifies $C_{i,j}$. In section 6, we illustrate these two scenarios with case studies.

4.3.2 Approximate consensus verifiability

Our second approach avoids the latency of sequential computation processes by employing probabilistic, light-weight verification. Will will guarantee the correctness of the solution to a certain (adjustable) extent while only having to check a small portion of the solution.

Definition 10 (δ-approximate verifiable)**.** Let us denote a problem $\mathsf{Z} : \{0,1\}^n \to \{0,1\}^m$, an input $x \in \{0,1\}^n$ and a claimed solution $y' \in \{0,1\}^m$ of $\mathsf{Z}(x)$. We say Z is δ-approximate verifiable if there exists a verification function f such that f accepts y' only if y' differs from $Z(x)$ in at most δ bits with a probability greater than p (say 0.99).

Our task is to encode f to ensure that if a solution y' is deemed correct only when y' does not differ much from the correct solution, *i.e.*, different in at most δ bits. Furthermore, in order to run f in a ε-CV model, f should not require more than W_{tx} work to finish. The high level insight to encode such a f is that by randomly sampling and checking an entry of the output, there is some probability that we can detect if the output is incorrect at that sample. If we find an incorrect sample, we can conclude that the submitted solution is wrong and reject the solution. The more samples we take, the better the probability of catching an error. Otherwise, the solution is close to a correct one with an overwhelming probability greater than p.

In Definition 10, we avoid defining too precisely the notion of "bits" and distance between two solutions since they vary slightly on specific encoding of the verification function f. One can translate "bits" as the positions in an array, or the pairs of elements, where the solution differs. In either case, the fraction of errors δ would change accordingly to the definition of "bit." For example, in the case of sorting, each "bit" would correspond to a pair (i,j) where $i < j$ and a bit in a solution y' differs from that in $\mathsf{Z}(x)$ if $y'[i] > y'[j]$.

We borrow the above sampling idea from the property testing literature [28]. In fact, previous works in property testing have shown that it is possible to check if an expected property represents in a solution for a large set of practical algorithms [28, 29]. Property testing differs from verifiable computation, however, in that the latter deals with general computations whereas the former can only consider decision problems. Decision problems are not interesting for consensus verifiability because without doing any work a prover P can simultaneously post two solutions, "0" and "1," and one of these is guaranteed to be correct. Thus puzzle givers cannot outsource work for decision problems to provers.

Typically, a probabilistic verifying function f must check the two following properties:

- *Property 1.* The provided solution differs in at most δ-bits from a solution that satisfies the property of the computation with probability greater than p.
- *Property 2.* The provided solution is computed from the given input x with probability greater than p.

Note that given a particular number of samples, the correctnesses that we can guarantee for *Property 1* and *Property 2* are different. For example, in the case of sorting a n-element array, n random samples may be sufficient to check whether the provided solution has the same elements as the given array with a 99% guarantee (*Property 2*). However, it may take more than n samples to fully check if the provided solution is sorted (*Property 1*). Thus, to achieve the overall 99% guarantee of correctness for both *Property 1* and *Property 2*, the number of samples one must take is the maximum of the numbers of samples to attain 99% guarantee of correctness for either of the checks. We provide details of how to construct a verification function f to guarantee both the above properties for sorting in Section 6.

5. IMPLEMENTATION

In this section we discuss the challenges while encoding a verification function f in our ε-CV model. We further discuss techniques to address those challenges.

5.1 Challenges in implementation

The presence of contract and Turing-complete language in Ethereum enables a verifiable computing environment in which users can ask anyone to solve their problem and get the results verified by the network. We have established how to encode those verification computations f in Section 4.2 to work with our ε-CV model. In fact, P can do the computation to arrive at the solution on his physical computer. G encodes the verification function f such that it takes P's solution and auxiliary data for the verification process. For example, if G asks P to sort an array, G can encode f to ask P to provide the sorted array (result) with the map between the result and the input. Our previous contract in Figure 5 is a concrete example where the matrix multiplication is outsourced to the network. We summarize the properties such a contract in our ε-CV model can achieve.

1. *Correctness.* G receives correct results.
2. *No prior trust.* No pre-established trust between P and G is required.
3. *No interaction.* No interaction between P and G is required in order to verify the result.
4. *Efficiency.* G and P have to do only a modest amount of work.
5. *Fairness.* P is rewarded for a valid solution.

Properties 1–4 are immediate when we encode the verification function f in our ε-CV model. Specifically, *correctness* is guaranteed since miners are incentivized to verify all computation. P and G do not need to trust or know each other, yet G cannot deviate after knowing the result. However, there are several challenges while implementing those verification function using smart contracts. In fact, simple smart contracts like the one in Figure 5 cannot guarantee *fairness* property due to one of the challenges that we describe below.

- **P3: Insecure relay attack.** Once P finds a solution C, he broadcasts a transaction having C to his neighbors and to the whole network. However, since the solution is in the plaintext, a rational neighbor node may copy C, delay or even discard the prover's transaction and forge a new transaction to claim the reward.

- **How to randomly sample in Ethereum?** One challenge to implement our approximate consensus verifiability is to randomly sample elements in a solution, given that Ethereum does not natively support a random generator operator. We want to make sure that the random generator is unbiased in seed selection, and that the seed is unpredictable. If it is biased or predictable, the correctness property may be violated since P can submit a solution which is correct only in the "bits" that will be sampled.

  Another constraint on a pseudo-random generator is that it should be consistent across the network, *i.e.*, return the same sample set to everyone. Otherwise, if an honest miner M verifies that a solution is correct with his samples, but the other miners see it incorrect since they have a different sets of samples. Although M is honest, every one rejects his block because with their sample sets M's transaction including his solution is invalid. It is unfair for M and problematic for the network as a whole since the correct behavior is probabilistic.

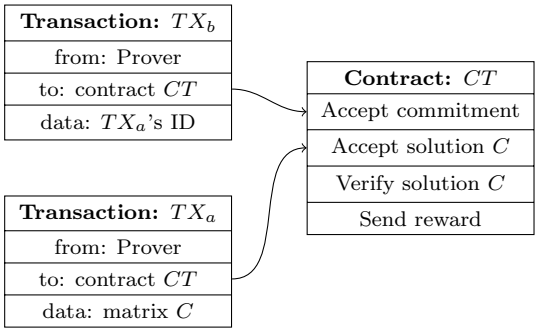

Figure 6: The commitment scheme in our ideal outsourced computation.

One naïve approach to defeat **P3** is to encrypt and sign the result, since the key management system already exists in Ethereum. Unfortunately this does not work since every miner needs to access to C in order to verify the transaction. In the next section, we devise a commitment scheme which helps us obtain *fairness* and discuss how to implement all the necessary components in our ε-CV protocol.

5.2 Construction

Achieving fairness via a commitment scheme. A commitment scheme provides two important features. First, it ensures that other users cannot see, thus steal P's solution and claim the reward. Second, P once commits a solution cannot later alter it. This is to prevent some prover from submitting a fake answer to win the race with others (since only one prover is paid for correct solution), then spending more time computing the correct solution.

Our commitment scheme leverages the one-way hash function, which is already used in current cryptocurrencies. Specifically, we ask P to prepare a transaction TX_a that he includes his solution, and a transaction TX_b to commit TX_a to the contract. P first sends TX_b, which includes TX_a's ID, to the contract to say that he has the solution and will send that in the next transaction having an ID as TX_a's ID. Once the contract accepts his commitment, he sends TX_a and proceeds further as in without having the commitment. We illustrate our commitment scheme in Figure 6.

Since an ID is computed by hashing a transaction data, other miners by observing TX_a's ID in TX_b are not able to construct TX_a to get C. Moreover, P cannot alter TX_a by a different solution since doing that will raise a conflict between the committed ID and the new one. In addition, once TX_b gets accepted and P broadcasts TX_a, the neighbors cannot claim the rewards with the solution observed in TX_a. It is because the contract is waiting for transaction TX_a which was committed before.

How to randomly sample in Ethereum? Our key idea is to leverage some data in the future blocks that miners do not know and cannot predict at the time of creating the transaction/contract. For instance, the randomness source can be the hash, timestamp or the founder's wallet address of a future block. Given a pseudo-random variable R con-

structed from any of those sources, we can generate n other pseudo-random numbers simply by taking a hash as:

$$\texttt{SHA256}(R \,||\, i)$$

in which $1 \leq i \leq n$, and $||$ is a concatenation operator.

Since the information of a block is public and consistent in the network, all miners when run the above pseudo-random generator will get the same set of samples, thus achieving consistency. Further, the information of block is unknown before it is mined by miners, coupling with the randomness of SHA256 function makes our pseudo-random generator fair.

6. CASE STUDIES

In this section, we exhibit several problems that can be solved by using our ε-*CV* model. We are interested in the problems that require high computational resource to verify if are encoded naïvely without using our techniques discussed in Section 4.2. The purpose of those examples is to illustrate the practicality of our techniques, and also describe how to encode f for various δ-approximate verifable problems. Our examples consists of several problems in diverse domains such as Graph Theory, Linear Algebra, Number Theory and simple operations like matrix multiplication and sorting. In several of these cases we are able to easily verify whether a solution is correct. However, for several interesting problems verifying the correctness of a solution appears to be elusive. We try to circumvent this difficulty by taking a recourse in an approximate verifiability. We illustrate how one can encode the verification function f to employ the light-weight verification and approximately verify the correctness of several problems. For the convenience of readers, we list our case studies in Table 1. In this section, we consider the basic operations as basic arithmetic operations, *e.g.*, addition, multiplication and comparison over 32-bit integers, unless otherwise stated.

6.1 With exact consensus verifiability

We show several applications that we can encode f to guarantee the correctness of a solution in an ε-*CV* model. We also discuss the potential high latency due to the need of distributing the computation to multiple transactions.

6.1.1 GCD of Large Numbers

This example computes the exact GCD of two large numbers, each of size n bits. The usual way to solve to the *greatest common divisor* problem on a classical computer goes via the Euclidean algorithm which takes work proportional to $O[\log^3(n)]$. We show that by encoding the verification function differently, one can verify the result in quadratic time $O[\log^2(n)]$. More interestingly, this example executes in a single transaction — that is, guarantees exactness with no additional latency. Specifically, the algorithm is as below.

1. G posts two integers m and n.
2. P posts five integers a, b, x, y, and z.
3. V checks that:
   (a) $ax = m$, $bx = n$,
   (b) $|y| < b$, $|z| < a$, and
   (c) $ay + bz = 1$,
   (d) If all of these checks succeed, then V accepts P's solution. Otherwise V rejects P's solution.

We claim that if both checks above succeed, then $\gcd(m, n) = x$. If (a) is satisfied, then clearly x divides m and y divides n. To see that x is the smallest such number, we appeal to Bézout's Identity which tells us that a and b are relatively prime iff there exist y and z satisfying (b) and (c). If some integer greater than x divided both m and n, then by uniqueness of factorization, x would also divide that integer, forcing a and b to have a nontrivial common factor. Our verification uses 10 arithmetic operations over inputs of size $\log(n)$.

6.1.2 Dot product

We compute the exact dot product of two vectors $(a_1, \ldots, a_n) \cdot (b_1, \ldots, b_n) = a_1b_1 + \cdots + a_nb_n$ by way of a sequence of transactions on the consensus computer. The puzzle giver G need not perform a single basic operation, while the miners perform no more than 3 basic operations per transaction. Each transaction will permit us to reduce the number of additions in the running sum by one. The puzzle giver G will post $\lceil n/2 \rceil$ transactions to the first block, $\lceil n/4 \rceil$ transactions to the second block, $\lceil n/8 \rceil$ to the third, and so on for a total of at most n transactions. For simplicity of presentation, we will assume that G knows the quantity n, however this assumption is not necessary. One could modify the protocol below so that we iterate over all indices i for which a_i and b_i are both defined (with possibly one extra pair $a_{n+1} = b_{n+1} = 0$ added in case n is odd), and similarly for the partial sums in the subsequent stages.

1. For each $i \leq \lceil n/2 \rceil$, G creates a puzzle transaction $T_{1,i}$ requesting the sum $a_ib_i + a_{i+1}b_{i+1}$.
2. For each i, P posts a number $s_{1,i}$ equal to the requested sum, in some permanent, public place.
3. For each i, V accepts P's solution iff $s_{1,i} = a_ib_i + a_{i+1}b_{i+1}$.

Subsequent stages k, for $k = 1$ up to the least k such that $2^k \geq n$ proceed similarly in a recursive way:

1. For each $i \leq \lceil \lceil \lceil n/2 \rceil /2 \rceil \ldots /2 \rceil$ (k 2's in this expression), G creates a puzzle transaction $T_{k,i}$ requesting the sum $s_{k-1,i} + s_{k-1,i+1}$.
2. For each i, P posts a number $s_{k,i}$ equal to the requested sum, in some permanent, public place.
3. V accepts P's solution iff $\forall i$ $s_{k,i} = s_{k-1,i} + s_{k-1,i+1}$.

By induction on k, if for each i P's solution is correct, then $\sum_i s_{k,i}$ is equal to the desired dot product. In the final stage, we obtain a single value which equals the dot product, and so our multi-stage protocol succeeds to compute the dot product. It is easy to verify that the work required in each transaction is $O(1)$.

Matrix multiplication.. We revisit the matrix multiplication problem in Figure 4 where P submit C as the result of $A \times B$. In Figure 5, we already show how to verify if a matrix solution C is correct by n^2 transactions, each runs in $O(n)$ steps. However, by using the above dot product contract n^2 times, one could verify C by transactions that require only $O(1)$ work.

6.2 With approximate consensus verifiability

Problems	Exact	Approx	# TXs	W_{tx}	#. Rounds
GCD	✓		$O(1)$	$O(1)$	1
Dot Product	✓		$O(n)$	$O(1)$	$O(\log n)$
Matrix Multiplication (Fig. 4)	✓		$O(1)$	$O(n^3)$	N/A
Matrix Multiplication (Fig. 5)	✓		$O(n^2)$	$O(n)$	N/A
Matrix Multiplication (Section 6.1)	✓		$O(n^3)$	$O(1)$	N/A
Matrix Multiplication		✓	$O(n)$	$O(1)$	$O(\log n)$
Sorting		✓	$O(1)$	$O(1)$	N/A
k-coloring		✓	$O(1)$	$O(1)$	N/A

Table 1: Case studies for outsourced computation in our ε-*CV* model. W_{tx} represents the amount of work required in each transaction. *#. Rounds* is the number of rounds required to have a solution verified if the transactions have to send in order (N/A means there is no such order).

As discussed earlier, some applications have solutions which are not easy or cheap to be verified. We show how to encode the verification function f for such applications to achieve probabilistic correctness with much smaller latency.

6.2.1 Approximate Sorting

We apply the direct sampling method so that with high probability the consensus will accept a correct solution but reject a solution with many errors. Let A denote the input array of $|A| = n$ elements. Let B denote the *claimed* output array of n elements and f denote a permutation on $\{1, \ldots, n\}$, representing the *claimed* sorting order.

Definition 11 (δ-approximate sorting)**.** For $\delta \geq 0$, we say that an array B (together with a permutation f) is a δ-approximate solution to sorting array A if:

- at most δ fraction of elements of B are incorrectly mapped by f, i.e., $A[i] \neq B[f(i)]$, and
- at most δn^2 pairs of B are out of order

We fix a $\delta > 0$ and let k denote the number of sample used by our protocol. The work required in the protocol below will be directly proportional to k. The protocol below guarantees that if the output is correctly sorted then it is accepted with high probability and if the output is *not* δ-approximately sorted then it is rejected with high probability. Thus we get the guarantees that if an output is accepted then it must be δ-approximately sorted with high confidence. The confidence can be made arbitrarily close to 1 at the cost of increasing the sample size.

1. G posts an array A of n elements.
2. P posts an array B of n elements and f, a permutation on $\{1, \ldots, n\}$.
3. V chooses k random indices $p_1, \ldots, p_k \leq |A|$ and k random pairs of indices $(q_1, r_1), \ldots, (q_k, r_k)$ and checks:
   (a) if: for all $i \leq k$, $A[p_i] = B[f(p_i)]$, and
   (b) if: for all $i \leq k$, if $q_i \leq r_i$ then $B[q_i] \leq B[r_i]$.
   (c) If both both the checks (a) and (b) succeed then V accepts P's solution. Otherwise V rejects.

Proof of Soundness of the Protocol: Let us first assume that P's solution (B, f) is a correct solution to sorting A, i.e., B has no errors in the bijection f and no pairs are out of order. In this case, check (a) and (b) will not fail as there are no errors. Thus the protocol will accept every correct solution. *Proof of Approximate Correctness of the Output:* On the other hand, if the array B differs from A in more than $\delta|A|$ places, then with probability at least δ, V will detect this error in (a), and if at least $\delta|A|^2$ pairs of elements in B are out of order, then again with probability at least δ we can detect this error in (b). V fails to detect an error only when both checks (a) and (b) succeed. The probability that (a) is satisfied for all i is equal to $(1-\delta)^k < e^{-\delta k}$, and similarly the probability that (b) is satisfied for all i is equal to $(1-\delta)^k < e^{-\delta k}$. Thus the probability that both of the checks succeed is the maximum of the two, which is still at most $e^{-\delta k}$. Thus if we choose k large enough such that $e^{-\delta k} < 1/100$ then we have the desired correctness guarantee, i.e., the protocol will reject the solution that is not δ approximate with probability at least 99%. Thus every solution accepted by protocol is δ-approximate with high probability. We note that it would suffice to choose $k > \frac{\log(1/100)}{\delta}$.

6.2.2 Approximate Matrix Multiplication

One can use the Dot Product in the previous section to verify Matrix Multiplication approximately. We want to guarantee that if our verification function f accepts C then at least $1 - \delta$ fraction of the entries of C match with $A \times B$ with high probability. The idea is to randomly pick a small number of entries of C and verify that they are computed correctly. For this check one may use the protocol for the Dot Product described in the previous subsection. It is easy to check that the number of samples is inversely proportional to the error parameter δ.

The number of random samples required by the above protocol to achieve the accuracy 99% will be $k > \log(1/100)/\delta$.

6.2.3 Approximate 2-Coloring

A *2-coloring* of a graph is an assignment of one of two specified colors to each of its nodes. An edge is colored *properly* if the two endpoints of the edge are assigned different color. Recall that a graph is 2-*colorable*, or *bipartite* if there a 2-coloring of its nodes such that every edge is properly colored. A 2-coloring is δ-approximate if at most δ fraction of the edges are *not* colored properly.

Using the sampling method, we design a protocol for the following decision problem: Does given A have an δ-approximate 2-coloring? Our protocol is inspired by the property testing algorithm for 2-coloring and its correctness relies on Szemeredi's Regularity Lemma [31]. The number of random samples required by the protocol to achieve the accuracy 99% is $50 \log(1/100) \frac{(\log 1/\delta)^4 \log\log(1/\delta)}{\delta}$. Below we let k denote the number of random samples.

1. G posts a graph A.
2. P posts either posts:
   (a) "yes" and a 2-coloring of A, or
   (b) "no" and an array of $O(\log^4(1/\delta) \log\log(1/\delta)/\delta)$

	Applicable Computation	Correctness	Fairness	No Priortrust	No Interaction	P's extra work	Setup cost
Thaler [8]	Regular	✓	✗	✗	✗	High	None
CMT [9]	Regular	✓	✗	✗	✗	High	None
Pepper [10]	Regular	✓	✗	✗	✓	High	Medium
Ginger [13]	Straight-line	✓	✗	✗	✓	High	Medium
Pinocchio [14]	Pure	✓	✗	✗	✓	High	Medium
Pantry [12]	Stateful	✓	✗	✗	✓	High	Medium
TinyRAM [11]	General loop	✓	✗	✗	✓	High	High
Kumaresan *et al.* [19, 21]	Limited	✗	✓	✓	✓	High	Medium
This work	Arbitrary ε-program	✓	✓	✓	✓	Low	Negligible

Table 2: Comparing IOC to previous work in verifiable computation. Some information is collected from [30].

edges from A.

3. V checks:
   (a) If P answered "yes," then V chooses k-random nodes $v_1, \ldots, v_k$ from A.
      i. V accepts P's solution if the subgraph induced on $v_1, \ldots, v_k$ is 2-colorable, and
      ii. V rejects otherwise.
   (b) If P answered "no," then
      i. V accepts P's solution if
         A. P's solution is an odd cycle, and
         B. all the edges of the odd cycle are present in A.
      ii. V rejects if either of the above conditions fails.

Correctness of the Protocol: The key fact we use, which follows from Szemeredi's Regularity Lemma, is that: a graph is *not* δ-approximate 2-colorable if and only if a random subset of $\widetilde{O}(1/\delta)$ nodes contains an odd cycle [32] with high probability. Thus if the graph is not δ-approximate 2-colorable then the local check done by our protocol suffices to detect this with high probability. On the other hand, if the graph is 2-colorable then no subset of $\widetilde{O}(1/\delta)$ nodes will contain odd cycle. Hence our protocol will correcty accept the solution. Thus our protocol computes correctly with high probability both when the answer is "yes" or "no"

Complexity of Verification: Note that if the graph is not δ-approximate 2-colorable then indeed we have a constant size witness for this. Hence the "no" answer from the prover has a light-weight verification. It remains to show that the "yes" answer from the prover also has an easy verification. This can be achieved by a sampling method similar to the one used for sorting. If more than δ fraction of the edges are not properly colored by P's 2-coloring, then we will catch a violated edge with high probability using $\widetilde{O}(1/\delta)$ samples.

In the "yes" instance, V does $O(k)$ basic operations to choose the random nodes and $O(k^2)$ comparisons to check that the subgraph inherits a 2-coloring. In the "no" instance, $O(k)$ basic operations are used to check whether P gave an odd cycle, and $O(k)$ basic operations to check it's presence in A for a total of $O(k)$ basic operations.

We note that to achieve 99% accuracy it suffices to choose the number of samples $k > \frac{50 \log(1/100) \log^4(1/\delta) \log\log(1/\delta)}{\delta}$.

Finally, we remark (without a proof) that our protocol can in fact be modified to verify approximate c-coloring as well for any constant c using [32].

7. RELATED WORK

Consensus protocol. Since the Nakamoto consensus protocol first appeared[1], numerous works have investigated alternative consensus algorithms [33, 34, 35, 36]. The problem we look at in this work is independent of the underlying consensus algorithm used in the network as the *verifier's dilemma* arises in any cryptocurrency that supports Turing-complete scripts.

Incentive compatibility in cryptocurrency. Apart from verifying blocks and transactions, mining new blocks is one major activity in cryptocurrencies. Block mining requires a huge computational resource, thus miners often join hands to mine together in a *pool*. Several previous works prove that pooled mining protocol is not incentive compatible by showing that miners and pools are susceptible to a block withholding attack [37, 38]. Our work also studies the incentive compatibility in cryptocurrency, but via the lens of the verification activity. We show that in network which allows Turing complete scripts, miners are vulnerable to a new class of attacks.

Verifiable computation. We compare our ε-*CV* protocol to previous work in verifiable computation in Table 2. We show that our protocol can achieve all the ideal properties that we mentioned in Section 5. One popular line of work uses classical computers for verification that involves prior trust establishment to guarantee that G does not deviate after knowing the solution [8, 9, 10, 11, 12, 13, 14]. Moreover, key exchange and interaction between P and G may be required in some protocols.

Since the rise of Bitcoin, a new line of work on repurposing the blockchain for other applications has been initiated [19, 21, 16, 20]. For example, in [19, 21], Kumaresan *et al.* studied how to run several interesting applications, including verifiable computation, on Bitcoin. Their technique is fairly complicated and relies on the assumption that all the computations done by consensus protocol will be correct. As we have shown, with expressive scripting languages (c.f. Ethereum), one can achieve what these previous works have done in Bitcoin with a single concise *smart contract*. However, as we pointed out in this paper, miners have incentive to deviate from the honest protocol in a new Turing-complete cryptocurrency. Thus, techniques used in [20, 21] may not guarantee the correctness of the computation. Further, our technique is different by leveraging a new design in Ethereum and adapting the property testing technique into the verifiable computation domain.

8. CONCLUSION

In this paper, we introduce a *verifier's dilemma* demonstrating that honest miners are vulnerable to attacks in

Turing-complete cryptocurrencies like Ethereum. We study the incentive structure in such cryptocurrencies and formalize an ε-*CV* model which guarantees the correctness of certain computations performed by the cryptocurrency network. Finally, we discuss how to implement our ε-*CV* model in Ethereum with various trade-offs in latency and accuracy. We consider it an interesting open problem to determine whether one can incentivize robust computations to execute correctly on a consensus computer by modifying the ε-*CV*'s underlying consensus mechanism.

Acknowledgment. We thank Frank Stephan, Shruti Tople, Pratik Sony, Ratul Saha, Virgil Griffith for useful discussions and feedback on the early version of the paper.

9. REFERENCES

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *bitcoin.org*, 2009.

[2] Karl J. O'Dwyer and David Malone. Bitcoin mining and its energy footprint. In *Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET*, pages 280–285, June 2014.

[3] Rob Halford. Gridcoin: Crypto-currency using berkeley open infrastructure network computing grid as a proof of work. http://www.gridcoin.us/images/gridcoin-white-paper.pdf, May 2014.

[4] Use case for factom: The world's first blockchain operating system (bos). http://kencode.de/projects/ePlug/Factom-Linux-Whitepaper.pdf, Feb 2015.

[5] Ethereum Foundation. Ethereum's white paper. https://github.com/ethereum/wiki/wiki/White-Paper, 2014.

[6] blockchain.info. Bitcoin average number of transactions per block. https://blockchain.info/charts/n-transactions-per-block.

[7] Gavin Andresen. Why increasing the max block size is urgent. http://gavinandresen.ninja/why-increasing-the-max-block-size-is-urgent, May 2015.

[8] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 71–89, 2013.

[9] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 90–112, 2012.

[10] Srinath T. V. Setty, Richard McPherson, Andrew J. Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *19th Annual Network and Distributed System Security Symposium, NDSS*, 2012.

[11] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. Cryptology ePrint Archive, Report 2013/507, 2013. http://eprint.iacr.org/.

[12] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 341–357, 2013.

[13] Srinath Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, pages 12–12, 2012.

[14] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 238–252, 2013.

[15] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Naryanan, Joshua A. Kroll, and Edward W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *IEEE Security and Privacy 2015*, May 2015.

[16] A Miller, A Juels, E Shi, B Parno, and J Katz. Permacoin: Repurposing bitcoin work for long-term data preservation. *IEEE Security and Privacy*, 2014.

[17] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. *IEEE Symposium on Security and Privacy*, 2013.

[18] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 459–474, Washington, DC, USA, 2014. IEEE Computer Society.

[19] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014*, pages 30–41, 2014.

[20] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, pages 443–458. IEEE Computer Society, 2014.

[21] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 421–439, 2014.

[22] Nick Szabo. The idea of smart contracts. http://szabo.best.vwh.net/smart_contracts_idea.html, 1997.

[23] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. http://gavwood.com/paper.pdf, 2014.

[24] Andrew Miller, James Litton, Andrew Pachulski, Neil Spring Neal Gupta, Dave Levin, and Bobby Bhattacharjee. Discovering Bitcoin's public topology and influential nodes. http://cs.umd.edu/projects/coinscope/coinscope.pdf, May 2015.

[25] Some miners generating invalid blocks. https://bitcoin.org/en/alert/2015-07-04-spv-mining, July 2015.

[26] Maximum transaction rate of Bitcoin. https://en.bitcoin.it/wiki/Maximum_transaction_rate, Jan 2014.

[27] Byron Cook, Andreas Podelski, and Andrey Rybalchenko. Proving program termination. *Commun. ACM*, 54(5):88–98, May 2011.

[28] Dana Ron. Property testing (a tutorial). http://www.eng.tau.ac.il/~danar/Public-pdf/tut.pdf, 2000.

[29] Oded Goldreich. Combinatorial property testing (a survey). *Electronic Colloquium on Computational Complexity (ECCC)*, 4(56), 1997.

[30] Michael Walfish and Andrew Blumberg. Verifying computations without reexecuting them: from theoretical possibility to near-practicality. http://eccc.hpi-web.de/report/2013/165/, July 2014.

[31] Endre Szemerédi. Regular partitions of graphs. In *Problèmes combinatoires et théorie des graphes (Colloq. Internat. CNRS, Univ. Orsay, Orsay, 1976)*, volume 260 of *Colloq. Internat. CNRS*, pages 399–401. 1978.

[32] Noga Alon and Asaf Shapira. Every monotone graph property is testable. *SIAM Journal on Computing*, 38(2):505–522, 2008.

[33] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, 1999.

[34] David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. https://www.stellar.org/papers/stellar-consensus-protocol.pdf, April 2015.

[35] Kwon Jae. Tendermint: Consensus without mining. http://jaekwon.com/2014/05/11/tendermint/, May 2014.

[36] King Sunny and Nadal Scott. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. http://peercoin.net/assets/paper/peercoin-paper.pdf, August 2012.

[37] Loi Luu, Ratul Saha, Inian Parameshwaran, Prateek Saxena, and Aquinas Hobor. On power splitting games in distributed computation: The case of bitcoin pooled mining. *The 28th IEEE Computer Security Foundations Symposium*, July 2015.

[38] Ittay Eyal. The miner's dilemma. In *The 36th IEEE Symposium on Security and Privacy*, SP '15. IEEE Computer Society, May 2015.

APPENDIX

A. `GASLIMIT` IN ETHEREUM

We explain that in Ethereum miners have more incentive to increase the `gasLimit`. The current design allows the miners to set the `gasLimit` for the next block once they find a block. However, the `gasLimit` of the next block cannot vary by 1/1024 times compared to the current `gasLimit`. This constraint is not mentioned any where publicly but in our private communication with the founder of Ethereum. This feature seems to mitigate our attack, however we explain why it is more likely that the `gasLimit` will reach to a large value that makes the resource exhaustion attack feasible. First, the attacker always has motivation to increase the `gasLimit` in order to conduct his attack. Second, the normal users in the network have the following incentives to extend the `gasLimit` value.

- Higher `gasLimit` value means higher transaction fee that miners can collect from a block.
- Block with higher `gasLimit` can support more applications, specifically the application that requires more gas to run. Thus, the value of the network and the underlying currency is more valuable, which benefits directly to the miners.
- As more and more applications are built on top of Ethereum, the `gasLimit` has to be increased correspondingly in order to improve the throughput of the network to support those applications. It is not practical to wait for, say, ten blocks to see a transaction get included in the blockchain due to the capacity of the block is too little, *i.e.*, the throughput is small.

In general, it is more likely that the `gasLimit` will get increased by 1/1024X after each block, either due to the incentive that drives rational miners or due to the attacker's purpose. Thus, our DoS attack, *i.e.*, resource exhaustion attack, is quite feasible in practice. Note that in Bitcoin, there is a hard limit in the size of the block, thus putting a cap in the number of transactions one can include in a block.