Fair Protocols for Verifiable Computations using Bitcoin and Ethereum

Dorsala Mallikarjun Reddy*†, V. N. Sastry*, Chapram Sudhakar†
*Institute for Development and Research in Banking Technology, Hyderabad, India
†National Institute of Technology, Warangal, India

***Abstract*—Outsourcing a computation has been a major research area in cryptography. A delegator (D) outsources a computation to a worker (W), who expects to get paid in return for delivering correct outputs. The delegator has to verify the output returned, to guard against malicious or malfunctioning worker. The worker may not trust the delegator to pay for computations performed by him. Blockchain systems like Bitcoin and Ethereum offer public verifiability of transactions generated in their networks. These transactions are of the form of a set of opcodes in Bitcoins or initiate execution of a contract in Ethereum, thus offering execution of small programs publicly. A fair protocol for verifiable computation between two parties D and W must provide the following guarantee: (1) Fast verification: The work performed to verify the correctness of output of a function is less than the work performed to compute function. (2) Pay to learn output: W obtains pay from D iff D received the correct output of the computation from W.**

In this work, we design two ideal functionalities $\mathcal{F}^*_{CRR}$ for single worker case and $\mathcal{F}^*_{CCRR}$ for multiple workers case. Existing methods do not allow Mutual revocation of deposit transaction which reduces the cost of script execution. Our ideal functionalities offer mutual revocation of the deposit transaction made by D. We show the realization of our ideal functionality $\mathcal{F}^*_{CRR}$ in Bitcoin, and we also discuss fair protocols for verifiable computations of GCD of two large numbers and optimization problems. As our final work, we show a fair protocol for proof-based verifiable computing schemes.

***Index Terms*—Blockchain; Bitcoin; Ethereum; Verifiable Computing**

I. INTRODUCTION

In the modern age of cloud computing, when a user outsources a computation to a cloud, he has to get a pay-per-use arrangement with the cloud provider. The cloud provider charges the user proportional to the effort involved in the computation. He must trust the cloud administrator, machine hardware, software and network all perform as expected. Many things can go wrong, particularly when the result of the computations tied to monetized entities such as smart contracts. The errors that are related to the correctness of the computation are deterred by using blockchain-based cryptocurrency systems. Throughout this paper, we consider a setting where a Delegator(D) outsources a computation f to a Worker(W) who expects to get paid in return for delivering correct outputs. A fair protocol for verifiable computation between two parties D and W must provide the following guarantee:

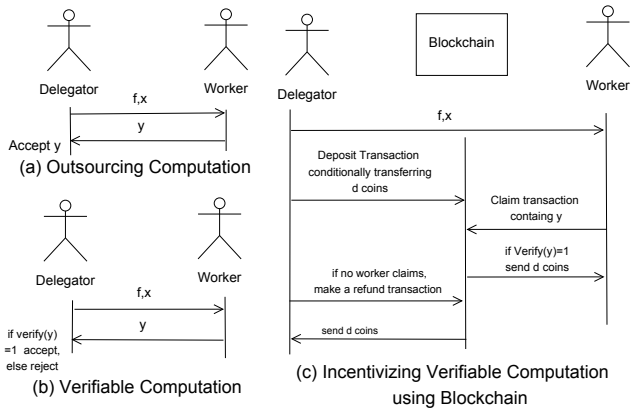

Fig. 1: Verifiable Computation vs Fair Verifiable Computation using Blockchain.

- **Fast verification**: The work performed to verify the correctness of output of f is less than the work performed to compute f .
- **Pay to learn output**: W obtains pay from D if and only if D received the correct output of the computation from W.

Figure 1, shows the comparison of outsourcing a computation, outsourcing a verifiable computation and incentivizing verifiable computation using a Blockchain. Bitcoin [1], first introduced by Nakamoto and other similar cryptocurrency applications uses Nakamoto consensus. Most of the consensus-based cryptocurrency applications have a peer-to-peer network, a blockchain, and a consensus protocol. The peer's in the network acts as either transaction generators or transaction validators(Miners). A distributed ledger a.k.a blockchain stores complete transaction history of the network and thus it can be used to verify the validity of transactions. The security of the blockchain is maintained as a part of consensus protocol by a chain of cryptographic puzzles(or blocks), solved by miners to change the state of the blockchain. The miners get the reward for solving puzzles in the form of newly minted coins. The structure of the nakamoto consensus based blockchain is shown in Figure 2. The design of most of the Blockchains focuses on who-pays-whom transactions, but it has features to go beyond this functionality. Specifically, most blockchain platform offers a scripting language, to allow conditional

2159-6190/18/$31.00 ©2018 IEEE
DOI 10.1109/CLOUD.2018.00107

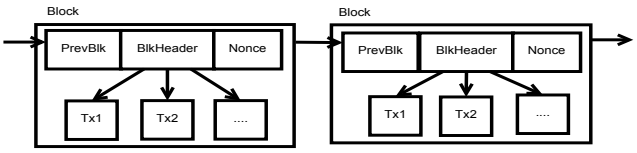

Fig. 2: The Blockchain in cryptocurrencies using Nakamoto Consensus protocol

transactions which can be repurposed for applications other than peer-to-peer payments.

Definition 1. A fair protocol for verifiable computation involves three parties.

- Delegator(D) : Who outsources a function f on input x, and also provides a function $f^{'}$ which verifies the correctness of the solution y. Upon, providing the correct solution for f a pre-announced reward is transferred from D to whoever produces a correct solution.
- Worker (W) : who presents a solution y to collect the reward from D.
- Miners (M) : who have a strong incentive to execute $f^{'}(y)$ and decide whether y is the correct solution for f or not.

The following properties of the Blockchain systems are used to obtain fairness in outsourced computation.

1) If the verification function is modeled as a transaction/-contract, then the correctness of output produced by the worker can be verified publicly.
2) Assuming that the decentralized consensus protocol is secure, the blockchain is trusted for correctness and availability of data it posses. Blockchain public ledgers provide perfect transparency, distributed verifiability, and immutability.
3) The result of the computation can be tied to a smart contract running on a blockchain, which can reward/penalize participants.

A. Our Contributions

1) We design two ideal functionalities $\mathcal{F}^{*}_{CRR}$ and $\mathcal{F}^{*}_{CCRR}$, which allows mutual revocation of deposit transaction between D and W.
2) We show the realization of $\mathcal{F}^{*}_{CRR}$ in Bitcoin.
3) We also design fair protocols for computing GCD of two large numbers, proof-based verifiable schemes using $\mathcal{F}^{*}_{CRR}$, and optimization problems using F^{*}_{CCRR}.

II. Related Work

Leveraging Blockchain for financial fairness. several researchers [2]–[7] have shown the usage of Bitcoin for applications other than peer-peer payments. The applications include decentralized auctions, markets, games such as poker, secure lottery, etc. The authors in [2] presented fair coin toss using Bitcoin without a trusted entity. The authors in [3], [4] has shown a method to design secure two-party protocols for functionalities that result in a "forced" financial transfer from one party to other using Bitcoin-based timed commitment scheme. Kumaresan et.al by using Bitcoin, developed a decentralized poker scheme in [7], designed fair protocols for secure multi-party computation and secure lottery in [5], investigated the incentivization of correct computations in [6]. Other Blockchains like Ethereum [8] and Hawk [9] with their rich scripting languages allow the creation of applications other than peer-to-peer payments. Another recent work by Kiayias et al. in [10] has shown protocols for fair and robust multi-party computations by using a global transaction ledger. Mostly, all previous works focus on multi-party computations like auctions and lottery; our work will focus on incentivizing verifiable computations. The works in [11]–[14] investigated incentive compatibility of Bitcoin. In [11], [12], the authors prove that mining through Bitcoin mining pools is not incentive compatible.

Verifiable computation. The verifiable computing techniques in [15]–[23] enables proving correctness of arbitrary circuits. Recent works like [15], [22] made verifiable computation practical and also achieves constant size proofs of correctness.

Fairness in verifiable computing. The concept of incentivizing verifiable computation using Bitcoin is first discussed by Kumaresan et al. in [6]. Their protocol uses an ideal functionality $\mathcal{F}^{*}_{exitCR}$ allows parties to agree to revoke the deposit transaction mutually. But, to realize $\mathcal{F}^{*}_{exitCR}$ in Bitcoin is practically not possible for the VC scheme in [15]. We show the realization of our ideal functionality $\mathcal{F}^{*}_{CRR}$ in Ethereum by assuming the existence of a pre-compiled contract computing bilinear pairings on Ethereum blockchain. Luu et al. in [24] design an ϵ-consensus computer and investigated miner's dilemma problem. They also investigated verifiable computations using several examples, but their verification functions are specific to the problems whereas we discuss incentivizing a more generalized approach.

TrueBit [25] is an application specially designed for incentivizing correct computations. TrueBit is an Ethereum smart contract developed in solidity. The core concept in TrueBit is the verification game played between Solver and challenger. TrueBit architecture is complex, and its verification game is resource intensive. Our approach is different from TrueBit as our protocols don't require any extra setting and use existing Bitcoin and Ethereum architectures. Recently, [26]–[28] also uses a smart contract-based approach to incentivize correct computations.

III. Preliminaries

Definition 2. Let $f : \{0,1\}^n \rightarrow \{0,1\}^m$ be a function to be computed on input $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$ be it's output, the correct computation of f is fairly incentivized , if

1) there exists a verification function $f^{'} : \{0,1\}^m \rightarrow \{0,1\}$ for f, on a public Blockchain.
2) the computational cost of $f^{'}$ is negligible when compared to f.

As an example we show the fair protocol for verifiable computation of GCD of large numbers in example 1.

Definition 3. Let $f : \{0,1\}^n \rightarrow \{0,1\}^m$ be a function to be optimized on set of inputs $x \in \{0,1\}^n$, and $y_1, ..., y_k \in \{0,1\}^m$ be the set of optimized solutions for f,the correct computation of f is fairly incentivized, if

1) there exists a verification function $f^{'} : \{0,1\}^m \rightarrow \{0,1\}$ for f, on a public Blockchain.
2) there exists a function $f^{''}$ such that $f^{''}(y_1, ..., y_k) \rightarrow y_i$ where $i \in \{1, ..., k\}$
3) the computational cost of $f^{'} + f^{''}$ is negligible when compared to f.

We discuss the fair protocol for verifiable computation of above function in example 2.

Verifiable Computation. The definition of public verifiable computation scheme [15] is as follows

Definition 4. A public verifiable computation scheme consists of set of three polynomial-time algorithms.

- $keygen(f, 1^\lambda) \rightarrow (ek_f, vk_f)$ A randomized key generation algorithm takes the function f to be outsourced and a security parameter outputs a public evalution key ek_f and a public verification key vk_f.
- $Compute(ek_f, x) \rightarrow (y, \pi_y)$. The worker uses ek_f and input x and outputs $y = f(x)$ and a proof π_y of $y's$ correctness.
- $Verify_{vk_f}(x, (y, \pi_y)) \rightarrow \{0,1\}$ Given vk_f, x, y and π_y this algorithm outputs 1 if $f(x) = y$ otherwise 0.

Definition 5. An incentivizable protocol for a public verification scheme between two parties D and W consists of three algorithms ($keygen$, $compute$, $verify$), where D runs $keygen$, W runs $compute$ and $verify$ algorithm is modeled as script on a public blockchain. The incentivizing protocol for verifiable computation of above function is described in Example 3.

The verification function is represented as scripts in Bitcoin or as contracts in Ethereum. The verifiers are miners who validate the scripts/contracts and include the transactions generated from the validations into a block and then into a distributed ledger. The verifiers in Bitcoin or ethereum can choose what transactions can go into a block; therefore a verifier may avoid the verification of a script/contract if it consumes too many resources. [24] has shown how much advantage a verifier can gain by avoiding the verification of a script. Bitcoin is a turing-incomplete language where we cannot use time consuming opcodes where as Ethereum, a turing complete language makes execution of time consuming opcodes costly by using the notation of gas.

coins. The $coins$ in our ideal functionalities is intended to capture cryptographic currencies contained in cryptographic wallets.

IV. Fair verifiable computing

In this section, we design ideal functionalities for single worker case and multiple workers case.

A. Single Worker Case

In single worker case, we assume the delegator and worker are known to each other but cannot trust each other. We first show ideal functionality $\mathcal{F}^*_{CRR}$ and its realization in Bitcoin.

1) Special Ideal Functionality($\mathcal{F}^*_{CRR}$): The special ideal functionality $\mathcal{F}^*_{CR}$ presented in [5] is simple and ideally captures the functioning of a distributed ledger which allows to conditionally ($\varnothing_{D,W}(\pi) = 1$) transfer coins from a delegator to a solver. One drawback is size and cost of the script($\varnothing$) to be evaluated by the miners. The miners will verify scripts without deviating from consensus protocol to maintain blockchain integrity as long as the effort needed for verification is small [24]. As the verification cost increases, the miners may skip the verification of a script; this may increase the risk of inclusion of invalid transactions into blockchain. This problem is known as Verifier's Dilemma [24], caused a Bitcoin fork in 2015 and one of the factors in denial-of-service attacks in Ethereum in 2016. Our new ideal functionality $\mathcal{F}^*_{CRR}$ formally described in Figure 3 allows both the delegator and worker to mutually revoke the deposit transaction made by delegator, thus reducing the cost of script execution. If the delegator doesn't agree to revocation, the worker can claim coins by revealing the proof in the claim phase. The script $\phi_{D,W}$ is evaluated to true only if either π_y or r is provided as input. Here, π_y is proof of correctness of y and r is the information required to revoke the deposit transaction.

$\mathcal{F}^*_{CRR}$ with session identifier sid, running with parties D and W, a parameter 1^λ, and an ideal adversary S proceeds as follows:

1) Deposit phase: Upon receiving the tuple Tx_d = ($deposit$, sid, $ssid$, D, W, $\varnothing_{D,W}$, τ, $coins(d)$) from D, record the message Tx_d and send it to all parties. Ignore any future deposit message with the same ssid from D to W.
2) Revoke phase: In round $\tau^{'} < \tau$ upon receiving the tuple Tx_r = ($revoke$, sid, $ssid$, D, W, $\varnothing_{D,W}$, τ, d, r) from W, check if (1) Tx_d with same ssid as Tx_r was recorded and (2) $\varnothing_{D,W}(r) = 1$. If both checks pass, send ($revoke$, sid, $ssid$, D, W, $\varnothing_{D,W}$, τ, $coins(d)$) to W, and delete the record Tx_d.
3) Claim phase: In round τ, upon receiving Tx_c=($claim$, sid, $ssid$, D, W, $\varnothing_{D,W}$, τ, d, π_y, y) from W, check if (1) Tx_d with same ssid as Tx_r was recorded, and (2) $\varnothing_{D,W}(\pi_y) = 1$. If both checks pass, send ($claim$, sid, $ssid$, D, W, $\varnothing_{D,W}$, τ, $coins(d)$) to W, and delete the record Tx_d.
4) Refund phase: In round $\tau+1$, if the record Tx_d was not deleted, then send ($refund$, sid, $ssid$, D, W, $\varnothing_{D,W}$, τ, $coins(d)$) to D, and delete the record tx_d

Fig. 3: Special ideal functionality $\mathcal{F}^*_{CRR}$

2) Realization of $\mathcal{F}^*_{CRR}$ using Bitcoin: The realization of $\mathcal{F}^*_{CRR}$ in Bitcoin is shown in Figure 4. We take the advantage

1) *Deposit phase.*
   a) D requests W for a fresh public key by sending $(deposit, sid, ssid, D, W, \tau)$.
   b) W responds by sending $(deposit\_ack, sid, ssid, D, W, \tau, pk_W)$ to D.
   c) D creates a Bitcoin transaction t_d that redeems d coins, controlled by it to an π output script: $\pi(\cdot) \triangleq OP\_CHECKSIG(pk_W, \cdot) \wedge (OP\_CHECKSIG(pk_D, \cdot) \vee \phi_W(\cdot))$. Where pk_D is the public key for which the secret key sk_D is known only to D and $\phi(\cdot)$ is an arbitary circuit i.e., a Bitcoin script.
   d) D prepares a transaction t_{rf} that takes $id_{CR} = SHA256d(t_d)$ as it's input script, has locktime of $\tau \cdot \tau^{'} \cdot \tilde{\tau}$ blocks, and spends the output of t_{rf} to some output script $\pi^{'}(\cdot)$ that it controls. Let b_c be the current height of the longest blockchain that D is aware of then the simplified form of t_{rf} is $t_{rf}^{simp} \triangleq ([id_{CR}, 1], [x, \pi^{'}], b_c + \tau \cdot \tau^{'} \cdot \tilde{\tau})$
   e) D sends $(deposit\_sign, sid, ssid, D, W, t_{rf}^{simp})$ to W.
   f) W responds by sending $(deposit\_sign\_ack, sid, ssid, D, W, sig_W = Sign_{sk_W}(t_{rf}^{simp}))$ to D.
   g) D checks if $Vrfy_{pk_W}(t_{rf}^{simp}, sig_W) = 1$, then broadcast t_d to Bitcoin network.
2) *Revoke phase*. After $b_{curr} + \hat{\tau} < b_{curr} + \tau \cdot \tau^{'} \cdot \tilde{\tau}$ blocks solved by Bitcoin network:
   a) W constructs a transaction t_r that takes $id_{CR} = SHA256d(t_d)$ as it's input and spends the output of t_d to output script $\pi^{''}(\cdot)$ that is controls. The simplified form of t_r is $t_r^{simp} \triangleq ([id_{CR}, 1], [x, \pi^{''}])$
   b) W sends $(revoke\_sign, sid, ssid, D, W, t_r^{simp})$ to D.
   c) D responds by sending $(revoke\_sign\_ack, sid, ssid, D, W, sig_D = Sign_{sk_D}(t_r^{simp}))$ to W.
   d) W checks if $Vrfy_{pk_D}(t_r^{simp}, sig_D) = 1$ then computes $Sign_{sk_W}(t_r^{simp})$ and combines it with $sign_D$ into an input script w_W that redeems txn_{CR} to output script $\pi^{''}(\cdot)$ that is controls. W then inserts w_W as an input script into t_r^{simp} and broadcasts the now complete t_r to Bitcoin network.
3) *Claim phase* After $\tau \cdot \tau^{'} \cdot \tilde{\tau}$ blocks solved by Bitcoin network:
   a) W broadcasts to Bitcoin network a transaction that redeems t_d to Output Script $\pi^{'}$ that it controls, by providing $Sign_{sk_W}(t_d^{simp})$ and revealing a witness w_W that satisfies $\phi_W(w_W) = 1$.
4) *Refund phase*. After $(\tau + 1) \cdot \tau^{'} \cdot \tilde{\tau} - \tilde{\tau}$ blocks solved by Bitcoin network:
   a) D computes $Sign_{sk_D}(t_{rf}^{simp})$ and combines it with sig_D into an input script(w_D) that redeems t_d, then inserts w_D into t_{rf}^{simp} and broadcasts the now complete t_{rf} to Bitcoin network.

Fig. 4: Realizing $\mathcal{F}_{CRR}^{*}$ in Bitcoin

of timelock, scripts and chaining of transactions in Bitcoin in order to realize $\mathcal{F}_{CRR}^{*}$. First, D creates a transaction t_d that takes d coins that it controls and can be redeemed according to "(D's signature and W's signature) OR (y such that $\phi_{D,W}(y) = 1$ AND W's signature)". D constructs one more transaction t_{rf}, which takes hash of t_d as an input, spends its output to an address controlled by it but has a timelock set in future and sends it to W. W signs t_{rf} and sends it back to D. W can not broadcast t_d as it sees only the hash of it. Now, D will broadcast t_d into Bitcoin network. When W gains enough confidence that t_d will not be reverted, it computes a witness y, such that $\phi_{D,W}(y) = 1$. W prepares a transaction t_r which takes t_d as input and its output is spent to the address controlled by it. W sends t_r to D along with y. If D is sure that $\phi_{D,W}(y) = 1$, he signs t_r and sends it W. W adds it signature to t_r, which is already signed by D, making the condition "D's signature and W's signature" as true. If D doesn't sign t_r, the honest W can claim the deposit by making a transaction t_c which takes t_d as input and satisfy the condition "W's signature and $\phi_{D,W}(y) = 1$". In round $\tau + 1$, D can recover its Deposit coins if W fails to claim. D will sign and broadcast t_{rf}, which is already signed by W satisfying the condition "D's signature and W's signature". The parameters $\tau^{'}$,$\tilde{\tau}$ correspond to the double-spending safety distance.

B. Multiple Workers Case

In this case the delegator announces a task on a public platform (like a public bulletin board), and the workers show their intent to solve the task. The special ideal functionality $\mathcal{F}_{CCRR}^{*}$ is formally described in Figure 5. At high level F_{CCRR}^{*} allows D to conditionally deposits $coins(d)$, which can be claimed by a W, who provides a solution to the D's problem. To reduce the execution cost F_{CCRR}^{*} allows a revocation of deposit without performing heavy execution and allow a W to challenge if D is cheating by falsely revoking the deposit transaction. Design of $\mathcal{F}_{CCRR}^{*}$ is complex than $\mathcal{F}_{CRR}^{*}$, because of several time requirements and many rounds. Workers must have to act according to the round numbers represented as $\{\tau_i, \tau_r, \tau_{ch}, \tau_c\}$ where workers have to show their intent before τ_i round, and D must send revoke message before τ_r and any challenge by W must be made before τ_{ch} and all the claim messages must be sent before τ_c. We have one more ideal functionality $\mathcal{F}_{resolve}^{*}$, which resolves and

announces winner in case of challenge of a revoke transaction. Design of $\mathcal{F}^*_{resolve}$ is straight forward and mostly depends on underlying application.

V. Examples

Example 1. GCD of two large numbers. There are many functions where verifying a solution takes less time when compared to finding the solution. Computing of GCD of two large numbers is one such function. We show the protocol for verifiable computation of GCD of two large numbers from [24].

1) D posts two integers m and n.
2) W posts five integers a, b, x, y, and z.
3) M checks that:
   a) $ax = m$, $bx = n$,
   b) $|y| < b$, $|z| < a$ and
   c) $ay + bz = 1$.
   d) if all these checks are succeed then M accepts W's solution, otherwise rejects.

If (a) is satisfied then x is a common factor. To show x is the greatest common divisor it is enough to show that a and b are relatively prime. To show a and b are relative primes, we use Bezout's identity, which states that a and b are relatively prime iff there exists two integers y and z such that $ay + bz = 1$. The verification has ten simple arithmetic operations and can be represented in Bitcoin scripting language[1]. A Bitcoin-based incentivizable scheme for computing GCD of two numbers between two parties D and W is given in Figure 6.

Example 2. Optimization Problems are resource intensive computations, and many a times perfect solution cannot be determined. We use our F^*_{CCRR} and design a fair protocol to incentivize computation of optimization problems. There can be multiple workers, who try to solve the problem and submits their solution. We assume the delegator also knows a function which determines the best solution out of all solutions submitted by workers.

1) D posts a function f for which the optimal solution has to be found in the input range $x \in [min, max]$.
2) W computes function using any optimization technique and outputs a value $y \in [min, max]$.
3) M follows the following steps
   a) if $y \in [min, max]$, then accepts y and compute $o = f(y)$, else reject.
   b) if more than one worker submits the solution, determine the best solution.

In figure 7, we show the fair protocol for incentivizing optimization problems using F^*_{CCRR}

[1]Although OP_MUL is disabled in Bitcoin, to simplify representation we use multiplication operator

$\mathcal{F}^*_{CCRR}$ with session identifier sid, running with parties D and $W_1, W_2, ..., W_n$, a parameter 1^λ, and an ideal adversary S proceeds as follows:

1) Deposit phase: Upon receiving the message Tx_d= ($deposit$, sid, $ssid$, D, ϕ, $\{\tau_i, \tau_r, \tau_{ch}, \tau_c, \tau\}$, $coins(d)$) from D, record the message Tx_d and send it to all parties. Ignore any future deposit message with same ssid.
2) Intent phase: In round $\tau < \tau_i$ Upon receiving the tuple Tx_{in} = ($intent$, sid, $ssid$, D, W_i, x, τ) from W_i, check if Tx_d with same $ssid$ as Tx_{in} was recorded, If yes record the message and send it to all parties.
3) Revoke phase: In round $\tau_i < \tau < \tau_r$ Upon receiving a message Tx_r = ($revoke$, sid, $ssid$, D, W_i, ϕ, x, τ) from D, check if Tx_i and Tx_d messages from W_i in Tx_r and D respectively with same $ssid$ are recorded. If both message are recorded then record Tx_r and send it to all parties. Ignore any future revoke message with same $ssid$ from D.
4) Challenge phase: In round $\tau_r < \tau < \tau_{ch}$ Upon receiving the message Tx_{ch} = ($challenge$, sid, $ssid$, D, W_i, π, x τ) from any W_i check if Tx_i and Tx_d messages from W_i in Tx_{ch} and D respectively with same $ssid$ are recorded. If both messages are recorded then record Tx_r and send it to all parties.
   In round $\tau > \tau_{ch} + 1$ if no Tx_{ch} is recorded and Tx_r is recorded then send ($reward$, sid, $ssid$, D, W_i, ϕ, $coins(d)$, τ) to W_i in Tx_r and delete the record Tx_d. Else, call the $\mathcal{F}^*_{resolve}$ functionality by sending x in Tx_r and Tx_{ch}. Upon receiving Tx_w=($winner$, sid, $ssid$, D, W_i, τ) from $\mathcal{F}^*_{resolve}$ send $(reward, sid, ssid, D, W_i, \tau, coins(d))$ to W_i in Tx_w and delete the record Tx_d.
5) Claim phase: In round $\tau < \tau_c$, Upon receiving the message Tx_c = $(claim, sid, ssid, D, W_i, \phi, x, \tau)$ from W_i check if Tx_i and Tx_d messages from W_i and D respectively with same $ssid$ are recorded. If both messages are recorded then record Tx_c and send it to all parties.
   In round $\tau > \tau_c$ call $F^*_{resolve}$ by sending all the x's from every W_i. Upon receiving Tx_w = ($winner$, sid, $ssid$, D, W_i, τ) send ($reward$, sid, $ssid$, D, W_i, x, τ, $coins(d)$) to W_i in Tx_w and delete the record Tx_d.
6) Refund phase: In round $\tau > \tau_c + 1$ if the record Tx_d was not deleted, then send $(refund, sid, ssid, D, \pi, \tau, coins(d))$ to D, and delete the record.

Fig. 5: Special ideal functionality $\mathcal{F}^*_{CCRR}$

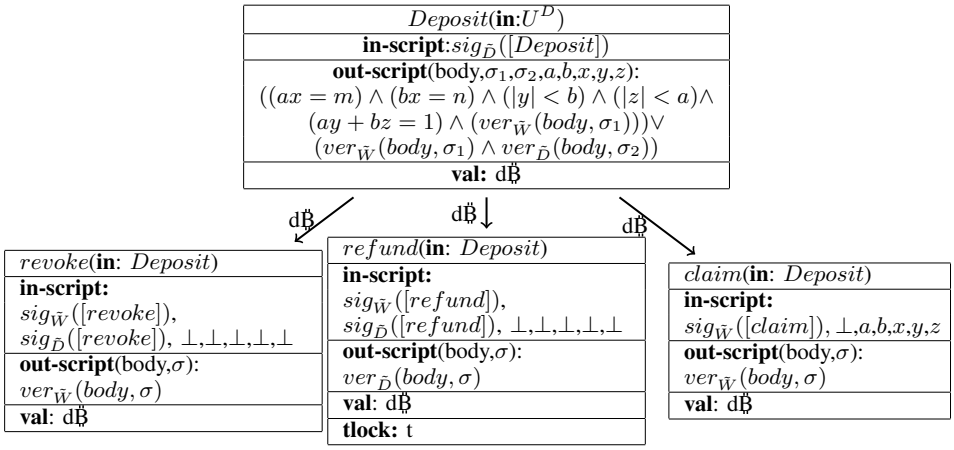

Fig. 6: Bitcoin scripts for the fair computation of GCD of two numbers. The scripts arguments, which are omitted are denoted by $\perp$.

Preliminaries: Let H be a collision resistant hash function. Let $y = 2x - (x^2)/16$ is function f to be maximized on input range $x \in [0, 31]$.

Protocol: D does the following

1) Announce the function and input range on a public platform.
2) sends Tx_d= ($deposit$, sid, $ssid$, D, ϕ, $\{\tau_i, \tau_r, \tau_{ch}, \tau_c, \tau\}$, $coins(d)$) to F^*_{CCRR}. Here ϕ is a circuit evaluated on input provided by every W_i.
3) Receive the solutions from all the workers and locally determine the winner and best solution and send Tx_r = ($revoke$, sid, $ssid$, D, W_i, π, x, τ) to F^*_{CCRR} and terminate.
4) If no corresponding claim message was received from F^*_{CRR}, then wait until round $\tau_c + 1$ to receive refund message from F^*_{CCRR} and terminate.

W does the following

1) If no corresponding Deposit message from F^*_{CCRR} was received on behalf of D, then terminate. Else find an x such that y obtains the maximum value.
2) compute $c_i = H(x)$.
3) send Tx_{in} = ($intent$, sid, $ssid$, D, W_i, c_i, τ) to F^*_{CCRR}.
4) send x to D.
5) Upon receiving Tx_r = ($revoke$, sid, $ssid$, D, W_i, π, x, τ) from $\mathcal{F}^*_{CCRR}$, if found D is cheating send Tx_{ch} = ($challenge$, sid, $ssid$, D, W_i, x, τ) to F^*_{CCRR} else terminate.
6) if no Tx_r message is received then send Tx_c = ($claim$, sid, $ssid$, D, W_i, x, τ)to $\mathcal{F}^*_{CCRR}$. $\mathcal{F}^*_{CCRR}$ evaluate π on all the inputs provided b every W_i and sends ($reward$, sid, $ssid$, D, W_i, τ, $coins(d)$) to the winning W_i.

Fig. 7: Fair protocol for incentivizing optimization problem using F^*_{CCRR}

The solidity code to incentivize the computation of maximizing the function $y = 2x - \{x^2\}/16$ [2] is given in Figure 8. D while creating the contract initializes all the time and reward parameters. Every W_i computes the solution and submits the hash of the solution by invoking *intent_and_commit* function. D after receiving all the solutions, locally runs the a function which determines the best solution and invoke the *revoke* function by sending the winner and best solution. If any worker feels that the delegator is cheating, then a worker can challenge by invoking the *challenge* function by sending his solution. Now, the contract will determine the best solution and rewards accordingly. If D is not willing to revoke, then every W_i can submit their solution by invoking *claim* function. Claim function determines the best solution and the winner is rewarded accordingly. If no worker is willing to claim the reward, the delegator can get his refund by invoking the *refund* function.

Example 3. In this example we show incentivizing Proof-based Verifiable Computing. We first discuss VC scheme[3] presented in [15].

1) D posts the evaluation key EK_F,Verification key VK_F and the input u
2) W posts the output y, and proof of correctness of y's computation π_y
3) M performs the following checks
   a) Divisibility check for the QAP
   b) Check that the linear combinations computed over $\mathcal{V}, \mathcal{W}, \mathcal{Y}$ are in appropriate spans
   c) Check that the same coefficients were used in each of the linear combinations over $\mathcal{V}, \mathcal{W}, \mathcal{Y}$

The checks in (a), (b) and (c) takes 3, 6 and 2 pairing operations respectively. Due to very limited instruction set,

[2]Currently, solidity does not supports operations on real values, we assume x has only integers values.

[3]Please refer [15] for complete protocol and it's implementation.

```
pragma solidity ^0.4.17;
contract opt{
address delegator;
uint reward;
address winner;
uint best;
bool status = false;
uint y;
uint _ITime;uint _RTime;uint _CHTime; uint _CLTime;
struct worker{    bytes32 WCValue; }
mapping(address => worker) workers;
constructor(uint iTime,uint rTime,uint chTime,uint
     clTime)payable {
   require(iTime<rTime && rTime < chTime && chTime <
        clTime);
   _ITime = now+iTime; _RTime = now+rTime; _CHTime =
        now+chTime; _CLTime = now+clTime;
   delegator=msg.sender;
   reward = msg.value; }
function intent_and_commit(bytes32 commitment) {
   require(now<_ITime);
   workers[msg.sender].WCValue = commitment;    }
function revoke(address winn,uint x) payable {
   require(now<_RTime && msg.sender==delegator);
   require(workers[winn].WCValue == sha3(x));
   winner = winn;
   y= 2*x-(x*x)/16;
   status=true;   }
function challenge(uint x) payable {
   require(now<_CLTime && msg.sender!=winner &&
        msg.value>1);
   uint penalty= msg.value;
   uint  cy = 2*x-(x*x)/16;
   if(cy>y)    {
      winner = msg.sender;
      msg.sender.transfer(penalty); }
   else {
      delegator.transfer(penalty); }
}
function get_reward() public {
   require(now>_CLTime && msg.sender == winner);
   winner.transfer(reward);
}
function claim(uint x) public {
   require(now<_CLTime && !status);
   uint yl = 2*x-(x*x)/16;
   if(yl>best) {
      best = y;
      winner = msg.sender; }
}
function sendreward() public payable {
   require(!status);
   status = true;
   winner.transfer(reward) ;
}
function refund() public payable {
   require(now>_CLTime && msg.sender==delegator &&
        !status);
   status=true;
   delegator.transfer(reward);
}
}
```

Fig. 8: Solidity code for incentivizing the computation of $y = 2x - \{x^2\}/16$

Preliminaries: Let H be a collision-resistant Hash function. Let $(keygen, compute, verify)$ be a verifiable computing scheme. Let f be a function to be computed on input u.

Protocol: D does the following

1) Runs $keygen(f, 1^\lambda) \rightarrow (ek_f, vk_f)$.
2) chooses a random number r and computes $c = H(r)$.
3) sends $Tx_d = (deposit, sid, ssid, D, W, \phi_{D,W}, \tau, coins(d))$ to F^*_{CRR}. Here $\phi_{D,W}$ is $verify$ algorithm to be evaluated on inputs provided by W. The circuit $\phi_{D,W}$ is evaluated to 1 either by revealing r or by providing the correct solution (y, π_y) by W.
4) If W sends a revoke request along with the output (y, π_y). verify it locally and if confident enough that the output is correct send r to W.
5) If no corresponding revoke or claim message was received from F^*_{CRR} on behalf of W , then wait until round $\tau + 1$ to receive refund message from F^*_{CRR} and terminate.

W does the following

1) If no corresponding Deposit message from F^*_{CRR} was received on behalf of D, then terminate.
2) Run $compute(EK_f, u) \rightarrow (y, \pi_y)$.
3) send a revoke request to D along with (y, π_y). if D sends r, and $c = H(r)$, prepare a revoke message Tx_r = $(revoke, sid, ssid, D, W, \phi_{D,W}, \tau, d, r)$ and send it to F^*_{CRR}. Receive the message $(revoke, sid, ssid, D, W, \phi_{D,W}, \tau, coins(d))$ from F^*_{CRR} and terminate.
4) if no response from D for the revoke request then send a claim message Tx_c = $(claim, sid, ssid, D, W, \phi_{D,W}, \tau, d, (y, \pi_y))$ to F^*_{CRR}. F^*_{CRR} evaluates $\phi_{D,W}$ on (y, π_y) if the result is 1, receive $(claim, sid, ssid, D, W, \phi_{D,W}, \tau, coins(d))$ from F^*_{CRR} and terminate.

Fig. 9: Fair Proof-based verifiable computing protocol using F^*_{CRR}

the bilinear pairing operations can not be implemented using Bitcoin scripting language, While the Ethereum Virtual Machine can make use of pairings in theory, they are currently too expensive. Not only expensive but these operations will exceed current Ethereum block gas limit. Kumaresan et.al [6] has shown incentivizing proof based verifiable computation using a distributed ledger. In our model we extend the notation given in [6], to minimize the verifiers work by permitting the delegator and solver to mutually revoke the deposit transaction. We assume that pairing operations can be performed on Ethereum virtual machine by using precompiled contracts, useful discussions can be found at [29]. The fair proof-based verifiable computing protocol using F^*_{CRR} is shown in 9.

VI. Conclusion and Future Work

In this paper, we design two new ideal functionalities $\mathcal{F}^*_{CRR}$ and $\mathcal{F}^*_{CCRR}$ and also show the realization of $\mathcal{F}^*_{CRR}$ in Bitcoin. We also show fair protocols for computing GCD

of two large numbers, optimization problems, and proof-based verifiable schemes. Our work leaves several interesting problems. First, in all existing models including this paper, the pay for computation is fixed, is it possible to design fair protocols for verifiable computation in which the pay is based on the resource usage of the worker. Second, our last approach uses verifiable scheme by Parno et al., is it possible to find a best verifiable computing scheme suitable for Blockchain systems like Bitcoin or Ethereum. The underlying consensus algorithm used in Bitcoin and Ethereum allows to verify small computations only, is it possible to develop a new consensus algorithm, which allows verifying complex computations in a fast and efficient way.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf, 2008.

[2] A. Back and I. Bentov, "Note on fair coin toss via bitcoin," *arXiv preprint arXiv:1402.3698*, 2014.

[3] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *IEEE Symposium on Security and Privacy (SP), San Jose, CS, USA, 2014*, pp. 443–458.

[4] M. Andrychowicz, S. Dziembowski, D. Malinowski, and Ł. Mazurek, "Fair two-party computations via bitcoin deposits," in *International Conference on Financial Cryptography and Data Security, Chirst Church, Barbodos, 2014*, pp. 105–121.

[5] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *International Cryptology Conference, Santa Barbara, CA, USA, 2014*, pp. 421–439.

[6] R. Kumaresan and I. Bentov, "How to use bitcoin to incentivize correct computations," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, Arizona, USA, 2014*, pp. 30–41.

[7] R. Kumaresan, T. Moran, and I. Bentov, "How to use bitcoin to play decentralized poker," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, Colorado, USA, 2015*, pp. 195–206.

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

[9] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 2016*, pp. 839–858.

[10] A. Kiayias, H.-S. Zhou, and V. Zikas, "Fair and robust multi-party computation using a global transaction ledger," in *Advances in Cryptology – EUROCRYPT 2016, Vienna, Austria, May 8-12, 2016*, M. Fischlin and J.-S. Coron, Eds.

[11] I. Eyal, "The miner's dilemma," in *2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 2015*, pp. 89–103.

[12] L. Luu, R. Saha, I. Parameshwaran, P. Saxena, and A. Hobor, "On power splitting games in distributed computation: The case of bitcoin pooled mining," in *2015 IEEE 28th Computer Security Foundations Symposium, Verona, Italy, 2015*, pp. 397–411.

[13] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Financial Cryptography and Data Security, Christ Church, Barbados, 2014*, pp. 436–454.

[14] J. A. Kroll, I. C. Davey, and E. W. Felten, "The economics of bitcoin mining, or bitcoin in the presence of adversaries," in *Workshop on the Economics of Information Security, June 2013*.

[15] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 2013*, pp. 238–252.

[16] S. T. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish, "Taking proof-based verified computation a few steps closer to practicality." in *USENIX Security Symposium, Bellevue, WA, USA, 2012*, pp. 253–268.

[17] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish, "Verifying computations with state," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles SOSP '13, Farminton, Pennsylvania, 2013*, pp. 341–357.

[18] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge," in *Advances in Cryptology – CRYPTO 2013, Santa Barbara, CA, USA, August 18-22, 2013*, pp. 90–108.

[19] S. T. Setty, R. McPherson, A. J. Blumberg, and M. Walfish, "Making argument systems for outsourced computation practical (sometimes)." in *19th Annual Network and Distributed System Security Symposium, NDSS, 2012*, pp. 253–268.

[20] G. Cormode, M. Mitzenmacher, and J. Thaler, "Practical verified computation with streaming interactive proofs," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference ITCS '12, Cambridge, Massachusetts, 2012*, pp. 90–112.

[21] J. 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*, pp. 71–89.

[22] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur, "Geppetto: Versatile verifiable computation," in *IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 2015*, pp. 253–270.

[23] M. Walfish and A. J. Blumberg, "Verifying computations without reexecuting them," *Communications of the ACM*, vol. 58, no. 2, pp. 74–84, jan 2015. [Online]. Available: http://doi.acm.org/10.1145/2641562

[24] S. Prateek, K. Raghav, L. Loi, and T. Jason, "Demystifying incentives in the consensus computer," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, Colorado, USA, 2015*, pp. 706–719.

[25] J. Teutsch and C. Reitwießner, "A scalable verification solution for blockchains," https://people.cs.uchicago.edu/~teutsch/papers/truebit.pdf, 2017.

[26] D. Harz, "Trust and verifiable computation for smart contracts in permissionless blockchains," Master's thesis, KTH, School of Information and Communication Technology (ICT), 2017.

[27] C. Dong, Y. Wang, A. Aldweesh, P. McCorry, and A. van Moorsel, "Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, Texas, USA*, ser. CCS '17. ACM, 2017, pp. 211–227. [Online]. Available: http://doi.acm.org/10.1145/3133956.3134032

[28] S. Jain, P. Saxena, F. Stephan, and J. Teutsch, "How to verify computation with a rational network," *arXiv preprint arXiv:1606.05917*, 2016.

[29] B. Vitalik and R. Christian, "Precompiled contracts for pairing function check," https://github.com/ethereum/EIPs/issues/197.