How to Use Bitcoin to Incentivize Correct Computations

Ranjit Kumaresan and Iddo Bentov

Technion, Haifa, Israel
{ranjit | idddo }@cs.technion.ac.il

ABSTRACT

We study a model of incentivizing correct computations in a variety of cryptographic tasks. For each of these tasks we propose a formal model and design protocols satisfying our model's constraints in a hybrid model where parties have access to special ideal functionalities that enable monetary transactions. We summarize our results:

- **Verifiable computation.** We consider a setting where a delegator outsources computation to a worker who expects to get paid in return for delivering correct outputs. We design protocols that compile both public and private verification schemes to support incentivizations described above.
- **Secure computation with restricted leakage.** Building on the recent work of Huang et al. (Security and Privacy 2012), we show an efficient secure computation protocol that monetarily penalizes an adversary that attempts to learn one bit of information but gets detected in the process.
- **Fair secure computation.** Inspired by recent work, we consider a model of secure computation where a party that aborts after learning the output is monetarily penalized. We then propose an ideal transaction functionality $\mathcal{F}^{\star}_{\mathrm{ML}}$ and show a constant-round realization on the Bitcoin network. Then, in the $\mathcal{F}^{\star}_{\mathrm{ML}}$-hybrid world we design a constant round protocol for secure computation in this model.
- **Noninteractive bounties.** We provide formal definitions and candidate realizations of noninteractive bounty mechanisms on the Bitcoin network which (1) allow a bounty maker to place a bounty for the solution of a hard problem by sending a single message, and (2) allow a bounty collector (unknown at the time of bounty creation) with the solution to claim the bounty, while (3) ensuring that the bounty maker can learn the solution whenever its bounty is collected, and (4) preventing malicious eavesdropping parties from both claiming the bounty as well as learning the solution.

All our protocol realizations (except those realizing fair secure computation) rely on a special ideal functionality that is not currently supported in Bitcoin due to limitations imposed on Bitcoin scripts. Motivated by this, we propose *validation complexity* of a protocol, a formal complexity measure that captures the amount of computational effort required to validate Bitcoin transactions required to implement it in Bitcoin. Our protocols are also designed to take advantage of optimistic scenarios where participating parties behave honestly.

CCS'14, November 3–7, 2014, Scottsdale, Arizona, USA.

http://dx.doi.org/10.1145/2660267.2660380.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

Keywords

Bitcoin; secure computation; verifiable computation; fair exchange; bounties

1. INTRODUCTION

We study a model of incentivizing correct computations in a variety of cryptographic tasks, namely verifiable computation, secure computation, fair computation, and bounty mechanisms. For each of these tasks we propose a formal model and design protocols satisfying our model's constraints in a hybrid model where parties have access to special ideal functionalities, e.g., $\mathcal{F}^{\star}_{\mathrm{CR}}$, $\mathcal{F}^{\star}_{f}$ [11], that enable monetary transactions. Below we explain each of the problems, provide motivation, discuss state-of-the-art, and outline our contributions.

Verifiable computation. Outsourcing computation has been a major area of research in cryptography. Recently several efficient schemes have been proposed [31]. Motivated by these developments, we consider a setting where a delegator outsources computation to a worker who expects to get paid in return for delivering correct outputs. Such settings may be useful for situations where the delegator is interested in a *pay per computation* model rather than a model where the delegator subscribes to cloud service to perform computations. We design protocols that compile both public and private verification schemes to support incentivizations described above.

Secure computation with restricted leakage. Protocols for secure computation anticipate worst-case behavior from malicious parties and typically make heavy use of expensive techniques that are meant solely to handle them. Recently, Huang et al. [23] (building on top of [29] proposed the "DualEx protocol" that restricts the amount of leakage to at most one bit even against malicious parties. While leaking a single bit might not sound too damaging, consider what happens when *multiple* secure evaluations are performed on the same data. This could be a server that is willing to allow computations on a database it holds, or on a set of master keys that it

possesses. In such scenarios leaking one bit of sensitive information per execution might indeed be catastrophic. While enhancements to DualEx protocol that allow (some restricted) detection of leakage, it is quite natural to expect that an adversary may interact with a server multiple times under different pseudonyms and in each interaction learn (with some probability) a sensitive bit of information.

Despite such severe consequences, we believe that sacrificing some leakage for (vastly) better efficiency is perhaps the way to go in the area of secure protocol design. For instance, consider state-of-the-art searchable symmetric key encryption schemes [12, 25, 30] that leak some information about access patterns to even *semihonest* adversaries. Indeed, the research direction in this area is to figure out "an acceptable balance between leakage and performance" [12]. Note that in contrast, the DualEx protocol leaks information only when parties deviate from the protocol. This justifies considering a model where a deviating party may be penalized if information leakage is detected. Specifically, we consider a model where a malicious party may attempt to learn one bit of information, but with the guarantee that if its cheating attempt is detected, then it is forced to pay a monetary penalty. We believe that this constitutes a practical way to enforce honest behavior in the DualEx protocol. We then construct a protocol that (for very large circuits) essentially has the same performance as the DualEx protocol in an optimistic scenario and yet allows for enforcing penalties. Our protocol is constructed in the $\mathcal{F}_{\mathrm{CR}}^{\star}$-hybrid model [11] and thus allows, assuming extended script support, for practical realization over the Bitcoin network. Our protocols provide a formal proof-of-concept that incentivizing secure computation to prevent leakage is indeed possible. We believe that our results provided added motivation for further research in designing secure protocols with restricted leakage—a relatively unexplored area—with the hope that these protocols can be incentivized to prevent leakage.

Fair secure computation. A major deficiency of secure computation is that, assuming a dishonest majority among participating parties, a corrupt party can always abort the protocol after learning the output while denying the output to honest parties. Such situations are highly undesirable if we want secure computation to be widely adopted in practice. Inspired by the recent works of [11, 4], we consider a model of secure computation where a party that aborts after learning the output is monetarily penalized. We then propose an ideal transaction functionality $\mathcal{F}_{\mathrm{ML}}^{\star}$ and show a constant-round realization on the Bitcoin network. Then, in the $\mathcal{F}_{\mathrm{ML}}^{\star}$-hybrid world we design a constant round protocol for secure computation in this model. Previous work [11] did not offer constant round implementations over Bitcoin.

Noninteractive bounties. Bitcoin users have been offering bounties that can be collected anonymously by anyone who solves an NP problem, such as SHA-1 and SHA-2 collisions [34]. The users who place the bounty expect to learn the preimages that cause the collision. The difficulty in realizing bounties arises from the fact that the identity of the user who solves the NP problem is unknown at the time the bounty is placed. This is a problem since other (malicious) nodes in the Bitcoin network could strip the witness and attempt to redeem the reward themselves. A recommendation outlined in [34] suggests that the user who claims the reward should generate the PoW block by herself. While this may very well be impractical, it still does not avoid the risk that other PoW miners will re-solve the block if the bounty is high enough and broadcast their own transaction that offers a higher fee to the Bitcoin miners.

Clearly, the above proposals for bounty mechanisms fall short of what one would expect from a bounty mechanism. We approach the problem by providing both formal definitions and candidate realizations of noninteractive bounty mechanisms on the Bitcoin network. The key constraints to keep in mind are that a noninteractive bounty mechanism must (1) allow a bounty maker to place a bounty for the solution of a hard problem by sending a single message, and (2) allow a bounty collector (unknown at the time of bounty creation) with the solution to claim the bounty, while (3) ensuring that the bounty maker can learn the solution whenever its bounty is collected, and (4) preventing malicious eavesdropping parties from both claiming the bounty as well as learning the solution.

Validation complexity. Our protocols and schemes are designed in a hybrid model where parties have access to an ideal transaction functionality, say $\mathcal{G}^{\star}$. The description of $\mathcal{G}^{\star}$ typically involves a conditional release of payment where the condition is formalized via a circuit ϕ. Our design of $\mathcal{G}^{\star}$ is certainly inspired by transaction functionalities supported by Bitcoin. In particular, the circuit ϕ corresponds to Bitcoin scripts that may be used to conditionally release payments. Unfortunately, heavy restrictions are imposed on the expressive power of Bitcoin scripts in the current Bitcoin system. Consequently some of our protocols cannot be implemented on the Bitcoin system today. On the one hand, we hope that our models and constructions offer compelling motivation to increase functionality of Bitcoin scripts. On the other hand, we propose *validation complexity*, a new complexity measure that attempts to capture the complexity of the Bitcoin script required in conditional transactions. Note that Bitcoin transactions need to be confirmed by the Bitcoin miners in order to append them to the public ledger. Thus, the miners need to first verify whether the transaction is valid. It is therefore natural to presume that miners levy an (additional) transaction fee that is proportional to the validation complexity of the transaction. As we will see, one of our main goals is to design protocols with low validation complexity.

Optimistic complexity. We use *optimistic* techniques for designing protocols to minimize the computation/communication/validation complexity of *honest executions* of our protocols, i.e., when all parties follow the protocol. As in prior work [26, 7, 33], our aim here is to design protocols that can "recognize the best cases and optimize for them, even in the midst of the protocol execution," [33] while guaranteeing security against worst-case behavior. Note that optimistic protocols are not intended to improve worse-case performance but are likely to offer meaningful gains in practice.

Bitcoin scripts and their limitations. Standard Bitcoin transaction currently blacklist many of the opcodes, primarily because of exploits in code that were not vetted carefully enough [1]. Even if all the opcodes will be whitelisted, it should be noted that the Bitcoin scripting language is not Turing complete, to avoid denial of service attacks. It is not enough to simply require a higher fee when the script size is bigger, because the risk of network DoS attacks implies that the nodes that propagate the transaction (and do not receive the fee) must verify it before re-broadcasting it. Hence, Bitcoin caps the transaction size, and bounds the verification time with a small polynomial function of the transaction size. Still, alternative protocol designs with Turing complete scripts are being considered, in particular with the Ethereum project [2]. Thus it is conceivable that in the future, richer forms of financial mechanisms will be used by Bitcoin and other cryptocurrencies, though all the users may have to pay somewhat larger fees as a result.

Other related work. The works of [21, 10] design a credit system where users are rewarded for good work and fined for cheating (assuming a trusted arbiter/supervisor in some settings). Fair secure computation with reputation systems was considered in [5]. Note that it has been claimed that reputation systems find limited appli-

cability because it is unclear how to define the reputation of new users [14].

2. PRELIMINARIES

A function $\mu(\cdot)$ is negligible in λ if for every positive polynomial $p(\cdot)$ and all sufficiently large λ's it holds that $\mu(\lambda) < 1/p(\lambda)$. A probability ensemble $X = \{X(a,\lambda)\}_{a\in\{0,1\}^*, n\in\mathbb{N}}$ is an infinite sequence of random variables indexed by a and $\lambda \in \mathbb{N}$. Two distribution ensembles $X = \{X(a,\lambda)\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y(a,\lambda)\}_{\lambda\in\mathbb{N}}$ are said to be computationally indistinguishable, denoted $X \stackrel{c}{\equiv} Y$ if for every non-uniform polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every $a \in \{0,1\}^*$,

$$|\Pr[D(X(a,\lambda)) = 1] - \Pr[D(Y(a,\lambda)) = 1]| \leq \mu(\lambda).$$

All parties are assumed to run in time polynomial in the security parameter λ. We prove security in the "secure computation with coins" (SCC) model proposed in [11]. Note that the main difference from standard definitions of secure computation [19] is that now the view of $\mathcal{Z}$ contains the distribution of coins. Let $\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(\lambda, z)$ denote the output of environment $\mathcal{Z}$ initialized with input z after interacting in the ideal process with ideal process adversary $\mathcal{S}$ and (standard or special) ideal functionality $\mathcal{G}_f$ on security parameter λ. Recall that our protocols will be run in a hybrid model where parties will have access to a (standard or special) ideal functionality $\mathcal{G}_g$. We denote the output of $\mathcal{Z}$ after interacting in an execution of π in such a model with $\mathcal{A}$ by $\text{HYBRID}^g_{\pi,\mathcal{A},\mathcal{Z}}(\lambda, z)$, where z denotes $\mathcal{Z}$'s input. We are now ready to define what it means for a protocol to SCC realize a functionality.

Definition 1. Let $n \in \mathbb{N}$. Let π be a probabilistic polynomial-time n-party protocol and let $\mathcal{G}_f$ be a probabilistic polynomial-time n-party (standard or special) ideal functionality. We say that π SCC realizes $\mathcal{G}_f$ with abort in the $\mathcal{G}_g$-hybrid model (where $\mathcal{G}_g$ is a standard or a special ideal functionality) if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ attacking π there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model such that for every non-uniform probabilistic polynomial-time adversary $\mathcal{Z}$,

$$\{\text{IDEAL}_{f,\mathcal{S},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^*} \stackrel{c}{\equiv} \{\text{HYBRID}^g_{\pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^*}. \quad \diamondsuit$$

Definition 2. Let π be a protocol and f be a multiparty functionality. We say that π securely computes f with penalties if π SCC-realizes the functionality $\mathcal{F}_f^\star$ according to Definition 1.

2.1 Standard primitives

Garbled circuits [35] allow two semihonest parties to compute an arbitrary function $f(x_1, x_2)$ that depends on their respective private inputs x_1 and x_2 while not leaking any information about their inputs beyond what is revealed by the function output. Our overview here follows the presentation in [23]; for more details see [27]. One party, acting as the circuit *generator*, produces a garbled circuit that is evaluated by the other party, known as the circuit *evaluator*. The result is an "encrypted" output, which can then be mapped to its actual value and revealed to either or both parties.

The basic idea is to transform a boolean circuit representing a function f into a garbled circuit that operates on labels (ie., cryptographic keys) instead of bits. We describe this transformation, denoted $\mathsf{Gb}(1^\lambda, f)$, below. Any binary gate g which has two input wires W_0, W_1 and output wire W_2 can be converted into a garbled gate. First, generate random labels w_i^0 and w_i^1 to contain 0 and 1 on each wire W_i. Then generate a truth table containing the four entries

$$\mathsf{Enc}_{w_0^{s_0}, w_1^{s_1}}(w_2^{g(s_0,s_1)})$$

for each $s_0, s_1 \in \{0,1\}$ (where s_0, s_1 denote the 1-bit signals on wires W_0, W_1, respectively), and randomly permute the table. This truth table is called a *garbled gate*. Observe that given the garbled gate and labels $w_0^{s_0}$ and $w_1^{s_1}$ it is possible to recover $w_2^{g(s_0,s_1)}$. That is given the labels that correspond to some set of input values for the entire circuit it is possible for the circuit evaluator to recover labels corresponding to the output of the circuit on those inputs. We denote this algorithm by Eval. If the circuit generator provides a way to map those labels back to bits, the circuit evaluator can recover the actual output.

Notation. We write a set of wire-label pairs as a matrix:

$$\mathbb{W} = \begin{pmatrix} w_1^0 & w_2^0 & \cdots & w_\ell^0 \\ w_1^1 & w_2^1 & \cdots & w_\ell^1 \end{pmatrix}.$$

A vector of wire labels is denoted as

$$\mathbf{w} = (w_1, w_2, \ldots, w_\ell).$$

If $v \in \{0,1\}^\ell$ is a string and $\mathbb{W}$ is a matrix as above, then we let

$$\mathbb{W}^v = (w_1^{v_1}, \ldots, w_\ell^{v_\ell})$$

be the corresponding vector of wire labels.

In some of our constructions we will use a seed-based garbling scheme Gb proposed in [22] that takes as input a security parameter λ, an explicitly specified seed ω to a pseudorandom generator (PRG) and a description of the function f and outputs the garbled circuit G. Note that in such a garbling scheme it is convenient to define two garbling functions iGb and oGb that generate wire-label pairs for only the input and output wires respectively. We use the notation $\mathbb{U} \leftarrow \mathsf{iGb}(1^\lambda, \omega, m)$ where $\mathbb{U}$ denotes the wire-label pairs for the input keys and m denotes the size of each party's input, and the notation $\mathbb{W} \leftarrow \mathsf{oGb}(1^\lambda, \omega, \ell)$ where $\mathbb{W}$ denotes that wire-label pairs for the output keys and ℓ denotes the size of the final output. Observe that iGb (resp. oGb) depend only on the input (resp. output) size of f and is otherwise independent of the description of f. The garbled gates are then computed using these wire labels exactly as in standard garbled circuits. To prove security of secure computation protocol based on garbled circuits, a special kind of garbled circuit $\hat{G}$ is computed by the simulator using algorithm $\mathsf{Fake}(1^\lambda, f)$. This algorithm outputs a "fake" garbled circuit, a random input $x' \in \{0,1\}^m$, wire labels $\mathbb{U}$ where positions corresponding to input x' are filled with random values while all other positions contain 0^λ, and output labels $\mathbf{w}$ such that $\mathsf{Eval}(\hat{G}, \mathbb{U}^{x'}) = \mathbf{w}$. Finally, we denote a garbling scheme by $(\mathsf{Gb}, \mathsf{Eval})$.

Verifiable computation. We provide the definition of *public verifiable computation* (taken from [31]).

Definition 3. A public verifiable computation scheme pubVC consists of a set of three polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{Compute}, \mathsf{Verify})$ defined as follows:

- $(ek_f, vk_f) \leftarrow \mathsf{KeyGen}(f, 1^\lambda)$: The randomized key generation algorithm takes the function f to be outsourced and security parameter λ; it outputs a public evaluation key ek_f, and a public verification key vk_f.
- $(y, \psi_y) \leftarrow \mathsf{Compute}(ek_f, u)$: The deterministic worker algorithm uses the public evaluation key ek_f and input u. It outputs $y \leftarrow f(u)$ and a proof ψ_y of y's correctness.

$\mathcal{F}_{\mathrm{CR}}^{\star}$ with session identifier sid, running with parties P_s and P_r, a parameter 1^{λ}, and adversary $\mathcal{S}$ proceeds as follows:

- *Deposit phase.* Upon receiving the tuple $(\mathsf{deposit}, sid, ssid, s, r, \phi_{s,r}, \tau, \mathsf{coins}(x))$ from P_s, record the message $(\mathsf{deposit}, sid, ssid, s, r, \phi_{s,r}, \tau, x)$ and send it to all parties. Ignore any future deposit messages with the same $ssid$ from P_s to P_r.
- *Claim phase.* In round τ, upon receiving $(\mathsf{claim}, sid, ssid, s, r, \phi_{s,r}, \tau, x, w)$ from P_r, check if (1) a tuple $(\mathsf{deposit}, sid, ssid, s, r, \phi_{s,r}, \tau, x)$ was recorded, and (2) if $\phi_{s,r}(w) = 1$. If both checks pass, send $(\mathsf{claim}, sid, ssid, s, r, \phi_{s,r}, \tau, x, w)$ to all parties, send $(\mathsf{claim}, sid, ssid, s, r, \phi_{s,r}, \tau, \mathsf{coins}(x))$ to P_r, and delete the record $(\mathsf{deposit}, sid, ssid, s, r, \phi_{s,r}, \tau, x)$.
- *Refund phase:* In round $\tau + 1$, if the record $(\mathsf{deposit}, sid, ssid, s, r, \phi_{s,r}, \tau, x)$ was not deleted, then send $(\mathsf{refund}, sid, ssid, s, r, \phi_{s,r}, \tau, \mathsf{coins}(x))$ to P_s, and delete the record $(\mathsf{deposit}, sid, ssid, s, r, \phi_{s,r}, \tau, x)$.

Figure 1: The special ideal functionality $\mathcal{F}_{\mathrm{CR}}^{\star}$.

- $\{0,1\} \leftarrow \mathsf{Verify}(vk_f, u, (y, \psi_y))$: Given the verification key vk_f, the deterministic verification algorithm outputs 1 if $f(u) = y$, and 0 otherwise.

The scheme pubVC should satisfy:

Correctness For any function f, it holds that

$$\Pr\left[\begin{array}{l}(ek_f, vk_f) \leftarrow \mathsf{KeyGen}(f, 1^{\lambda});\\ (y, \psi_y) \leftarrow \mathsf{Compute}(ek_f, u):\\ \qquad\qquad 1 = \mathsf{Verify}(vk_f, u, (y, \psi_y))\end{array}\right] = 1.$$

Soundness For any function f and any PPT $\mathcal{A}$ the following is negligible in λ:

$$\Pr\left[\begin{array}{l}\Pr[(u', y', \psi_y') \leftarrow \mathcal{A}(ek_f, vk_f):\\ \quad f(u') \neq y' \wedge 1 = \mathsf{Verify}(vk_f, u', (y', \psi_y'))\end{array}\right].$$

Efficiency KeyGen is assumed to be a one-time operation whose cost is amortized over many calculations, but we require that Verify is cheaper than evaluating f. ♢

2.2 Special ideal functionalities

Ideal functionality $\mathcal{F}_{\mathrm{CR}}^{\star}$ [11, 9, 28]. This special ideal functionality has found tremendous application in the design of multiparty fair secure computation and lottery protocols [11]. See Figure 1 for a formal description. At a high level, $\mathcal{F}_{\mathrm{CR}}^{\star}$ allows a sender P_s to *conditionally* send $\mathsf{coins}(x)$ to a receiver P_r. The condition is formalized as the revelation of a satisfying assignment (i.e., witness) for a sender-specified circuit $\phi_{s,r}(\,\cdot\,; z)$ (i.e., relation) that may depend on some public input z. Further, there is a "time" bound, formalized as a round number τ, within which P_r has to act in order to claim the coins. An important property that we wish to stress is that the satisfying witness is made *public* by $\mathcal{F}_{\mathrm{CR}}^{\star}$. In the Bitcoin realization of $\mathcal{F}_{\mathrm{CR}}^{\star}$, sending a message with $\mathsf{coins}(x)$ corresponds to broadcasting a transaction to the Bitcoin network, and waiting according to some time parameter until there is enough confidence that the transaction will not be reversed.

Secure computation with penalties. Loosely speaking, the notion of fair secure computation as considered in [11] guarantees:

- An honest party never has to pay any penalty.
- If a party aborts after learning the output and does not deliver output to honest parties, then *every* honest party is compensated.

$\mathcal{F}_f^{\star}$ with session identifier sid running with parties $P_1, \ldots, P_n$, a parameter 1^{λ}, and an ideal adversary $\mathcal{S}$ that corrupts parties $\{P_s\}_{s \in C}$ proceeds as follows: Let $H = [n] \setminus C$ and $h = |H|$. Let d be a parameter representing the safety deposit, and let q denote the penalty amount.

- *Input phase:* Wait to receive a message $(\mathsf{input}, sid, ssid, r, y_r, \mathsf{coins}(d))$ from P_r for all $r \in H$. Then wait to receive a message $(\mathsf{input}, sid, ssid, \{y_s\}_{s \in C}, \mathsf{coins}(hq))$ from $\mathcal{S}$.
- *Output phase:*
  - Send $(\mathsf{return}, sid, ssid, \mathsf{coins}(d))$ to P_r for all $r \in H$.
  - Compute $(z_1, \ldots, z_n) \leftarrow f(y_1, \ldots, y_n)$.
  - If $\mathcal{S}$ returns $(\mathsf{continue}, sid, ssid)$, then send $(\mathsf{output}, sid, ssid, z_r)$ to P_r for all $r \in H$, and send $(\mathsf{return}, sid, ssid, \mathsf{coins}(hq))$ to $\mathcal{S}$.
  - Else if $\mathcal{S}$ returns $(\mathsf{abort}, sid, ssid, \mathsf{coins}(t''hq))$, send $(\mathsf{extra}, sid, ssid, \mathsf{coins}(t'q))$ to P_r for all $r \in H$.

Figure 2: Idealizing secure computation with penalties $\mathcal{F}_f^{\star}$.

The functionality $\mathcal{F}_f^{\star}$ (cf. Figure 2) captures these requirements.

Ideal functionality $\mathcal{F}_f^{\star}$ [11]. In the first phase, the functionality $\mathcal{F}_f^{\star}$ receives inputs for f from all parties. In addition, $\mathcal{F}_f^{\star}$ allows the ideal world adversary $\mathcal{S}$ to deposit some coins which may be used to compensate honest parties if $\mathcal{S}$ aborts after receiving the outputs. Note that an honest party makes a fixed deposit $\mathsf{coins}(d)$ in the input phase. Then, in the output phase, $\mathcal{F}_f^{\star}$ returns the deposit made by honest parties back to them. If $\mathcal{S}$ deposited sufficient number of coins, then it gets a chance to look at the output and then decide to continue delivering output to all parties, or just abort, in which case *all* honest parties are compensated using the penalty deposited by $\mathcal{S}$. We note that our version of $\mathcal{F}_f^{\star}$ varies slightly from the one proposed in [11]. While they allowed $\mathcal{S}$ to deposit insufficient number of coins (i.e., less than hq), we do not. On the other hand, we do allow $\mathcal{S}$ to send extra coins to honest parties when it aborts. This somewhat unnatural step is required in order to make our construction in Section 5 simulatable.

3. VERIFIABLE COMPUTATION

Loosely speaking, an incentivizable protocol for verifiable computation between a delegator D and a worker W must provide the following guarantee:

- (Fast verification.) The amortized work performed by D for verification is less than the work required to compute f.
- (Pay to learn output.) W obtains $\mathsf{coins}(q)$ from D iff D received the correct output of the computation from W.

We start with a naïve solution in the $\mathcal{F}_{\mathrm{CR}}^{\star}$-hybrid model.

A naïve solution. D sends u, f to W, and then creates an $\mathcal{F}_{\mathrm{CR}}^{\star}$ transaction that allows W to claim its $\mathsf{coins}(q)$ if it reveals y such that $y = f(u)$. (More concretely, $\phi(w; (u, f)) = 1$ iff $w = f(u)$.) W computes $y = f(u)$ and claims $\mathsf{coins}(q)$ by providing $w = y$ to $\mathcal{F}_{\mathrm{CR}}^{\star}$.

Clearly, the above is sufficient to incentivize verifiable computation but the solution has obvious drawbacks when implemented in the Bitcoin network. Note that to validate the claim transaction each *miner* has to verify whether the witness provided was indeed valid. This means that each miner has to compute f in order to

confirm the validity of the transaction. This is clearly undesirable because (1) it puts a heavy load on the Bitcoin network and corresponds to heavy loss of resources, and (2) more philosphically, while D is expected to pay W for the computation of f, miners are now computing the same f essentially "for free".

Motivated by this, we now minimize the "validation complexity" of our protocol. First, we give a precise definition of $\mathcal{F}_{\mathrm{CR}}^{\star}$-validation complexity for a protocol in the $\mathcal{F}_{\mathrm{CR}}^{\star}$-hybrid model.

Definition 4. Let Π be a protocol among n parties $P_1, \ldots, P_n$ in the $\mathcal{F}_{\mathrm{CR}}^{\star}$-hybrid model. For circuit ϕ, let $|\phi|$ denote its circuit complexity. For a given execution of Π starting from a particular initialization Ω of parties' inputs and random tapes and distribution of coins, let $V_{\Pi,\Omega}$ (resp. $V_{\Pi,\Omega}^{\mathrm{hon}}$) denote the sum of all $|\phi|$ such that some honest party claimed an $\mathcal{F}_{\mathrm{CR}}^{\star}$ transaction by producing a witness for ϕ during an (resp. honest) execution of Π. Then the $\mathcal{F}_{\mathrm{CR}}^{\star}$-*validation complexity* of Π, denoted V_{Π}, equals $\max_{\Omega}(V_{\Pi,\Omega})$. The *optimistic* $\mathcal{F}_{\mathrm{CR}}^{\star}$-validation complexity of π, denoted V_{Π}^{opt} equals $\max_{\Omega} V_{\Pi,\Omega}^{\mathrm{hon}}$. $\diamondsuit$

Note that our definition extends naturally to capture the validation complexity of other transaction functionalities (e.g., $\mathcal{F}_{\mathrm{ML}}^{\star}$). Also, we simply say "(optimistic) validation complexity" instead of "(optimistic) $\mathcal{F}_{\mathrm{CR}}^{\star}$-validation complexity" in contexts where it is obvious that we are referring to the $\mathcal{F}_{\mathrm{CR}}^{\star}$-validation complexity. As mentioned in the Introduction, validation complexity of a transaction may justify the transaction fee that is required to validate it in the Bitcoin network.

Remark. Note that validating an $\mathcal{F}_{\mathrm{CR}}^{\star}$ transaction also requires verification of the designated sender's and receiver's signatures. Our justification for not accounting for this in the $\mathcal{F}_{\mathrm{CR}}^{\star}$-validation complexity as defined above is that such verifications are required even for standard transactions between two parties.

Incentivizing public verifiable computation. A natural approach to minimize the validation complexity would be to use a *public verifiable computation scheme* pubVC. Indeed we show how to compile an arbitrary public verifiable computation scheme into an incentivizable verifiable computation scheme. Perhaps the main difficulty in constructing such a compiler is the need to handle *malicious clients* in our setting. Note in contrast that in the standard setting of verifiable computation, the client is always assumed to be honest and security is required only against malicious server. To see why it is important to safeguard against a malicious client let us take a look at a naïve scheme based on any public verifiable computation scheme pubVC.

Naïve scheme based on pubVC. D runs $\mathsf{KeyGen}(f, 1^{\lambda})$ to generate (ek_f, vk_f), and then sends ek_f, u to W, and then creates an $\mathcal{F}_{\mathrm{CR}}^{\star}$ transaction that allows W to claim its $\mathsf{coins}(q)$ if it reveals (y, ψ_y) such that $\mathsf{Verify}(vk_f, u, (y, \psi_y)) = 1$. (More concretely, $\phi(w; (vk_f, u)) = 1$ iff $1 = \mathsf{Verify}(vk_f, u, w)$.) W runs $\mathsf{Compute}(ek_f, u)$ to obtain (y, ψ_y) and claims $\mathsf{coins}(q)$ by providing $w = (y, \psi_y)$ to $\mathcal{F}_{\mathrm{CR}}^{\star}$.

The main problem with the above solution is that a malicious D may not generate the verification key honestly, and thus an honest worker that computes $y \leftarrow f(u)$ is not guaranteed payment. Note however that in such a situation we may ask W to reveal y to D only if $w = (y, \psi_y)$ is such that $\phi(w) = 1$ for the ϕ obtained from $\mathcal{F}_{\mathrm{CR}}^{\star}$. Still the above solution is undesirable since a honest worker does perform the required the computation yet does not get paid by the delegator. This motivates the following condition:

- (Guaranteed pay on honest computation.) W obtains $\mathsf{coins}(q)$ from D if W followed the protocol honestly.

Note that standard secure computation protocol to jointly emulate KeyGen algorithm such that both D and W obtain (ek_f, vk_f) at the end of the protocol suffices to satisfy the condition above. Observe that the work performed by D for securely emulating KeyGen will be amortized over several executions.

Although the above modified scheme does minimize the validation complexity significantly, one may still wonder if further improvements are possible. Note that current state-of-the-art public verification schemes [31], although quite impressive relative to prior work, still require 288 bytes storage and 9ms to verify. That is, each miner would be required to spend 9ms to execute the verification algorithm in order to validate the $\mathcal{F}_{\mathrm{CR}}^{\star}$ transaction. We observe that in an optimistic scenario (where we assume both D and W are interested in reducing the validation complexity), it is possible to drive the validation complexity to zero. To do this, we first let D and W to interact as described above. Then, in a later phase, we simply let W reveal (y, ψ_y) to D. If $\mathsf{Verify}(vk_f, u, (y, \psi)) = 1$, then (honest) D pays W. Note that if D does not pay W, then W can always claim the $\mathcal{F}_{\mathrm{CR}}^{\star}$ transaction made by D. On the other hand, if D does pay W, then a malicious W may attempt to get paid twice by also claiming $\mathsf{coins}(q)$ from the original $\mathcal{F}_{\mathrm{CR}}^{\star}$ transaction. In order to avoid this double payment, we use a new ideal transaction functionality $\mathcal{F}_{\mathrm{exitCR}}^{\star}$. The description of our verifiable computation protocol (in the $\mathcal{F}_{\mathrm{exitCR}}^{\star}$-hybrid model) appears in the full version of our paper. We provide more details on the ideal transaction functionality $\mathcal{F}_{\mathrm{exitCR}}^{\star}$ and a candidate Bitcoin implementation below.

Ideal functionality $\mathcal{F}_{\mathrm{exitCR}}^{\star}$. Recall that $\mathcal{F}_{\mathrm{CR}}^{\star}$ (cf. Figure 1) allows a sender to conditionally send $\mathsf{coins}(x)$ to a receiver. However, $\mathcal{F}_{\mathrm{CR}}^{\star}$ does not allow parties to mutually agree to discard checking the condition to release payment. It is exactly this ability that our new ideal functionality $\mathcal{F}_{\mathrm{exitCR}}^{\star}$ offers. Specifically, $\mathcal{F}_{\mathrm{exitCR}}^{\star}$ allows parties to mutually agree to revoke the condition ϕ that releases payment. In addition there is a "time" bound, formalized as a round number τ_2 within which the revision has to occur. As in $\mathcal{F}_{\mathrm{CR}}^{\star}$, P_r must act within some round number τ_1 in order to claim $\mathsf{coins}(x)$ by revealing a witness for ϕ if the condition ϕ was not revoked.

To realize $\mathcal{F}_{\mathrm{exitCR}}^{\star}$ via Bitcoin, we need to modify the realization of $\mathcal{F}_{\mathrm{CR}}^{\star}$ (e.g., as in [11, Appendix F]) only slightly. The mechanism that we rely upon for tx_{refund} is the script in tx_{claim} that specifies that one of the ways to redeem tx_{claim} is by signing with secret keys that P_s and P_r hold. This allows tx_{refund} to be created by both parties signing a transaction that would be considered valid by Bitcoin nodes only if it is included in a future block (as specified by a *timelock* parameter). Hence, we only require an extra intermediate step after tx_{claim} was broadcast, in which, upon agreement, both parties will sign a transaction tx_{exit} that redeems tx_{claim} to P_r.

Incentivizing private verifiable computation. Perhaps the main concern about our previous scheme is that D's input u is made public on the Bitcoin network. This is because the verification algorithm $\mathsf{Verify}(vk_f, u, \cdot)$ is part of the Bitcoin script that each miner needs to verify before validating the transaction. A more desirable scheme would be one where D's input is kept private. However note that a malicious W is given access to D's input u, and hence always has the power to make u (or $f(u)$) public. Therefore, to make the problem more meaningful we will consider verifiable computation schemes which already preserve privacy against a malicious worker. Then one may ask whether it is possible to incentivize a verifiable computation that preserves input/output privacy of D. Indeed in the full version of our paper, we show somewhat surprisingly it is possible to incentivize verifiable computation schemes with *designated verifier* (i.e., in contrast to public verifi-

cation). However, this comes at a price. Specifically we no longer guarantee pay on honest computation. On the other hand, we show that it is possible to penalize a malicious worker that tries to execute the "rejection" attack (typically allowed by private verification schemes based on fully-homomorphic encryption). In such an attack, the malicious worker supplies incorrect proofs of computation and learns information depending on whether the honest delegator accepts its proof or not.

At a high level, our constructions use secure computation to emulate all algorithms except the Compute algorithm used by the worker. (Observe that the amortized complexity of D depends only on the input/output length of f and is otherwise independent of complexity of f.) An important issue is to ensure that parties (especially the delegator) provide consistent inputs across all these secure emulations. However, this is easily achieved by use of (one-time) message authentication codes (since the MAC verification happens inside a secure protocol). While securely emulating the Verify algorithm to secret share the final output of the computation between D and W if a successful proof was supplied, and then require D to make a $\mathcal{F}^{\star}_{\mathrm{CR}}$ deposit in order to learn the other secret share. This is achieved using techniques similar to the ones employed in [11]. We defer other details of the construction to the full version due to lack of space.

In summary, we provide two protocols that incentivize verifiable computation schemes (i.e., force D to pay to learn the output while denying payment for an incorrect output). The first scheme compiles any public verification scheme, guarantees pay on computation, but does not protect client privacy. The validation complexity equals the public verification complexity in the worst case and is zero in an optimistic scenario (due to use of $\mathcal{F}^{\star}_{\mathrm{exitCR}}$). The second scheme compiles the designated verifier scheme of [17], protects client privacy and also penalize malicious workers that supply invalid proofs, but does not guarantee pay on computation. The validation complexity of this protocol equals a hash invocation (with hash input equal to the length of output of the computation).

Remark. We note that similar techniques may be extended to allow penalizing deviations in publicly verifiable *covert* secure protocols [8, 6].

4. SECURE COMPUTATION

We focus on the DualEx protocol of Huang et al. [23] (which in turn is inspired by [29]). The protocol enjoys efficiency comparable to that of semihonest Yao garbled circuits protocol while guaranteeing that a malicious party can learn at most one bit of information about the honest party's input. Given that secure computation protocols require a high overhead due to use of cut-and-choose or zero-knowledge, the DualEx protocol offers an attractive alternative in scenarios where efficiency is the bottleneck.

We now provide a quick outline of the DualEx protocol. The high level idea is to let two parties P_1 and P_2 run two simultaneous instances of a *semihonest* garbled circuit protocol. In the first instance P_1 acts as the circuit constructor and P_2 acts as the circuit evaluator. In the second instance they swap roles. The key observation made in [29, 23] is that appending a secure equality test to the above step somewhat surprisingly results in a protocol that leaks at most a single bit of information about an honest party's input to a malicious party. Furthermore, several enhancements to the DualEx are possible. For instance, it is possible to design a variant that releases output to the parties only if the equality test passes. In such a scenario, a cheating adversary does so only at the expense of not learning the actual output. [23] also experimentally validate the superior efficiency of the DualEx protocol.

$\mathcal{F}^{\star}_{f,\mathrm{leak}}$ with session identifier sid, running with parties P_1 and P_2, a parameter 1^{λ}, and an ideal adversary $\mathcal{S}$ that corrupts P_i for $i \in \{1,2\}$ proceeds as follows. Let $j \in \{1,2\} \setminus \{i\}$. Let d be a parameter representing the safety deposit, and let q denote the penalty amount.

- *Input phase.* Honest P_j sends its input $(\mathsf{input}, sid, ssid, x_j, \mathsf{coins}(d))$. $\mathcal{S}$ sends input $(\mathsf{input}, sid, ssid, x_i, L, \mathsf{coins}(q))$ on behalf of P_i, where $x_1, x_2 \in \{0,1\}^{\ell}$, and $L : \{0,1\}^{\ell} \to \{0,1\}$.
- *Output phase.*
  - Send $(\mathsf{return}, sid, ssid, \mathsf{coins}(d))$ to P_j.
  - Compute $z \leftarrow f(x_1, x_2)$ and $y \leftarrow L(x_j)$.
  - If $y = 0$ send message $(\mathsf{abort}, sid, ssid)$ to $\mathcal{S}$ and $(\mathsf{penalty}, sid, ssid, \mathsf{coins}(q))$ to P_j, and terminate.
  - Else send message $(\mathsf{output}, sid, ssid, z, y)$ to $\mathcal{S}$.
  - If $\mathcal{S}$ returns $(\mathsf{continue}, sid, ssid)$ then set $z' = z$ and $q' = 0$, and send $(\mathsf{return}, sid, ssid, \mathsf{coins}(q))$ to $\mathcal{S}$. Else if $\mathcal{S}$ returns $(\mathsf{abort}, sid, ssid)$ set $z' = \perp$ and $q' = q$.
  - Send $(\mathsf{output}, sid, ssid, z', \mathsf{coins}(q'))$ to P_j.

Figure 3: The leaky functionality with penalty $\mathcal{F}^{\star}_{f,\mathrm{leak}}$.

Ideal functionality $\mathcal{F}^{\star}_{f,\mathrm{leak}}$. In the first phase, the functionality $\mathcal{F}^{\star}_{f,\mathrm{leak}}$ receives inputs from both parties. In addition $\mathcal{F}^{\star}_{f,\mathrm{leak}}$ allows the ideal world adversary $\mathcal{S}$ to deposit $\mathsf{coins}(q)$ and specify a "leakage function" denoted L. Note that an honest party makes a fixed deposit $\mathsf{coins}(d)$ in the input phase which is returned to it in the output phase. The functionality first computes the output z using inputs received from both parties, and also computes the output y of the leakage function on the honest party's input. If the output of the leakage function equals 0 (without loss of generality), then the honest party is compensated by $\mathsf{coins}(q)$. On the other hand if output of the leakage function is 1, then this goes "undetected." The ideal functionality $\mathcal{F}^{\star}_{f,\mathrm{leak}}$ also penalizes corrupt parties that abort on learning the output.

High level overview. As observed in [23], the attacks a malicious party may use against a DualEx protocol can be grouped into three main types: *selective failure*, in which the attacker constructs a circuit that fails along some execution paths and attempts to learn about the party's private inputs from the occurrence of failure, *false function*, in which the attacker constructs a circuit that implements function that is different from f, and *inconsistent inputs*, in which the attacker provides different inputs to the two executions. The DualEx protocol mitigates all of the attacks in a elegant way and allows a malicious party to learn at most one bit of information.

Our main observation is that attacks due to selective failure or inconsistent inputs can be prevented using techniques whose efficiency depends only on the input/output length and is otherwise independent of the circuit size of the function to be evaluated. Motivated by this observation, we design our protocol to narrow down the one-bit leakage to be launched via the *false function* attack. We then use standard techniques to penalize false function attacks.

Detailed overview. Our starting point is the observation that leakage in the DualEx protocol is detected only at the equality test ("secure validation") step. More precisely, in the event of detected leakage, the equality test simply fails. We take advantage of this in the following way: (1) letting P_1, P_2 exchange hash values $h_1 = \mathsf{H}(r_1), h_2 = \mathsf{H}(r_2)$ of random strings r_1, r_2 ahead of the equality step; (2) letting P_1, P_2 make $\mathcal{F}^{\star}_{\mathrm{CR}}$ transactions that release $\mathsf{coins}(q)$ to the other party if it reveals the preimage to both hash

Input from P_1: m, x_1, ω_1.
Input from P_2: m, x_2, ω_2.

Output to both P_1 and P_2:

- Create $\mathbb{U}_1 \leftarrow \mathsf{iGb}(1^\lambda, \omega_1, m)$ and $\mathbb{U}_2 \leftarrow \mathsf{iGb}(1^\lambda, \omega_2, m)$.
- Compute $g'_1 = \mathsf{com}(\omega_1; \rho_1)$ and $g'_2 = \mathsf{com}(\omega_2; \rho_2)$ where ρ_1, ρ_2 are picked uniformly at random.
- Output $(\mathbb{U}_2^{x_1 \| x_2}, g'_2, \rho_1)$ to P_1 and $(\mathbb{U}_1^{x_1 \| x_2}, g'_1, \rho_2)$ to P_2.

Figure 4: Secure key transfer subroutine KT.

Input from P_1: $\ell_1 = \ell, \mathbf{w}_1, \omega_1, \rho_1, r_1, h_2, g'_2$.
Input from P_2: $\ell_2 = \ell, \mathbf{w}_2, \omega_2, \rho_2, r_2, h_1, g'_1$.

Output to both P_1 and P_2:

- If $\ell_1 \neq \ell_2$ or $\mathsf{H}(r_1) \neq h_1$ or $\mathsf{H}(r_2) \neq h_2$ or $\mathsf{com}(\omega_1; \rho_1) \neq g'_1$ or $\mathsf{com}(\omega_2; \rho_2) \neq g'_2$, output bad and terminate.
- Create $\mathbb{W}_1 \leftarrow \mathsf{oGb}(1^\lambda, \omega_1, \ell)$ and $\mathbb{W}_2 \leftarrow \mathsf{oGb}(1^\lambda, \omega_2, \ell)$.
- Check if $\exists v_1, v_2 \in \{0,1\}^\ell$ such that $\mathbb{W}_1^{v_1} = \mathbf{w}_2$ and $\mathbb{W}_2^{v_2} = \mathbf{w}_1$. If the check fails output bad and termiante.
- Check if $\exists v \in \{0,1\}^\ell$ such that $\mathbb{W}_1^v = \mathbf{w}_2$ and $\mathbb{W}_2^v = \mathbf{w}_1$. If check fails output (r_1, r_2). Else, output v.

Figure 5: Secure validation subroutine SV.

values; and (3) augmenting the equality step to let parties to also input r_1, r_2 such that (r_1, r_2) is revealed iff the equality test fails.

Unfortunately, the above idea turns out to be naïve mainly because although the equality test detects leakage it does not quite help in identifying the deviating party (that must then be penalized). Perhaps even more severely, a malicious party that simply supplies junk input to the equality test can easily learn (r_1, r_2) and then deny honest party from learning this output. (This is possible since the equality step is implemented using a *unfair* secure computation protocol.) This results in a honest party losing its coins to the malicious party.

These obstacles lead us to design a more sophisticated secure validation subroutine. Specifically, we enforce that parties indeed supply the correct output keys by using a very specific garbling scheme proposed in [22]. At a high level, using a seed (for a PRG) we generate the parties' output keys *in situ* thereby preventing a malicious party from learning (r_1, r_2) by supplying junk input. Unfortunately this does not prevent other attacks. Specifically, a malicious party may provide legitimate output keys and yet fail the equality test (e.g., by providing inconsistent keys thereby producing different outputs). This necessitates the use of a protocol for "secure computation with penalties" (cf. Figure 2) to implement the secure validation step. Now a malicious party may abort the secure validation step after learning (r_1, r_2) but in this case it is forced to pay a penalty. Our secure validation subroutine SV is described in Figure 5. Our protocol is constructed in the $\mathcal{F}^\star_{\mathsf{SV}}$-hybrid model. A bonus side-effect of working in the $\mathcal{F}^\star_{\mathsf{SV}}$-hybrid model is that our protocol guarantees fairness (in the sense of [11]).

Although, we have resolved the "fairness" problem, we are still left with the possibility that a malicious party may force the output of $\mathcal{F}^\star_{\mathsf{SV}}$ to be (r_1, r_2) by simply providing inconsistent inputs. To resolve this attack, we employ a sophisticated key transfer subroutine (Figure 4) that generates the parties' input keys *in situ* and further distributes keys based on the parties' inputs (i.e., subsuming the oblivious transfer step). All of the above steps now ensure that information leakage can happen only due to false function attacks.

Inputs: P_1, P_2 respectively hold inputs $x_1, x_2 \in \{0,1\}^m$.
Preliminaries. Let $(\mathsf{com}, \mathsf{dec})$ be a perfectly binding commitment scheme. Let NP language $\mathcal{L}$ be such that $u = (a, b) \in \mathcal{L}$ iff there exists α, β such that $a = \mathsf{Gb}(1^\lambda, f, \alpha)$ and $b = \mathsf{com}(\alpha; \beta)$. Let $(\mathcal{K}, \mathcal{P}, \mathcal{V})$ be a non-interactive zero knowledge scheme for $\mathcal{L}$. Let $\mathsf{crs} \leftarrow \mathcal{K}(1^\lambda)$ denote the common reference string. Let H be a collision-resistant hash function.
Protocol: For each $i \in \{1,2\}$, P_i does the following: Let $j \in \{1,2\}, j \neq i$.

1. Pick ω_i at random and compute $G_i \leftarrow \mathsf{Gb}(f, \omega_i)$.
2. Send $(\mathsf{input}, sid, ssid, (m, x_i, \omega_i))$ to $\mathcal{F}_{\mathsf{KT}}$. If the output from $\mathcal{F}_{\mathsf{KT}}$ is abort, terminate. Else let output equal $(\mathbb{U}'_j, g'_j)$.
3. Send G_i to P_j and receive G_j from P_j.
4. Compute $\mathbf{w}_i \leftarrow \mathsf{Eval}(G_j, \mathbb{U}'_j)$.
5. Choose random r_i and send $h_i = \mathsf{H}(r_i)$ to P_j.
6. Let $X_i = (G_j, g'_j, h_j)$, and let $\phi_i(w; X_i) = 1$ iff $w = (\alpha, \beta)$ such that $\mathcal{V}(\mathsf{crs}, (G_i, g'_i), \alpha) = 1$ and $\mathsf{H}(\beta) = h_j$. Send $(\mathsf{deposit}, sid, ssid, i, j, \phi_i(\cdot; X_i), \tau, \mathsf{coins}(q))$ to $\mathcal{F}^\star_{\mathrm{CR}}$.
7. If no corresponding $\mathsf{deposit}$ message was received from $\mathcal{F}^\star_{\mathrm{CR}}$ on behalf of P_j, then wait until round $\tau + 1$ to receive refund message from $\mathcal{F}^\star_{\mathrm{CR}}$ and terminate.
8. Send $(\mathsf{input}, sid, ssid, (\ell_i, \mathbf{w}_i, \omega_i, r_i, h_j, g'_j), \mathsf{coins}(d))$ to $\mathcal{F}^\star_{\mathsf{SV}}$. Let z_i denote the output received from $\mathcal{F}^\star_{\mathsf{SV}}$. Do: (1) If $z_i = \perp$, then terminate. (2) Else if $z_i = z$, then output z and terminate. (3) Else if $z_i = (r_1, r_2)$, then compute $\pi_i \leftarrow \mathcal{P}(\mathsf{crs}, (G_i, g'_i), \omega_i)$ and send $(\mathsf{claim}, sid, ssid, j, i, \phi_j, \tau, q, (\pi_i, r_j))$ to $\mathcal{F}^\star_{\mathrm{CR}}$, receive $(\mathsf{claim}, sid, ssid, j, i, \phi_j, \tau, \mathsf{coins}(q))$ and terminate.

Figure 6: Realizing $\mathcal{F}^\star_{f,\mathrm{leak}}$.

Recall that the equality test does not help in identifying the deviating party. On the other hand, a false function attack can be readily detected by simply asking the parties to prove in *zero-knowledge* (ZK) that they computed the garbled circuit correctly. Thus, we ask the $\mathcal{F}^\star_{\mathrm{CR}}$ transaction to release $\mathsf{coins}(q)$ to the other party if it reveals the preimage to both hash values and also provides a ZK proof that its garbled circuit was constructed correctly. (Observe that ZK proofs are required to ensure privacy of honest inputs.)

All of the above ideas still need to be integrated together with great care to ensure that the protocol is as secure as the DualEx protocol of [23]. Our protocol is described in Figure 6.

Efficiency. Note that in an optimistic setting, i.e., when both parties follow the protocol, there is no need for any party to compute a NIZK proof (whose cost is proportional to the circuit size of f), no $\mathcal{F}^\star_{\mathrm{CR}}$ transactions are claimed, and thus optimal validation complexity is simply hash verification (required in $\mathcal{F}^\star_{\mathsf{SV}}$ [11]). It is easy to see that for very large circuits with $|f| \gg m + \ell$, the optimal computation/communication complexity is essentially the same as that of the DualEx protocol. In practical instantiations, it is desirable to instantiate the PRG used for generating the garbled circuit via a cryptographic hash function as described in Section 2. Also, one may use NIZKs constructed in [18] to support very fast verification and have very short size (e.g., 7 group elements from a bilinear group). In Appendix A we formally prove:

Theorem 1. *Let $f : \{0,1\}^m \times \{0,1\}^m \to \{0,1\}^\ell$ and λ be a computational security parameter. Assume that collision-resistant hash functions, perfectly binding commitment schemes, and non-interactive zero knowledge (NIZK) arguments exist for NP. Then assuming that Gb is a secure garbling scheme as in [22], there ex-*

ists a protocol in the $(\mathcal{F}_{\mathrm{OT}}, \mathcal{F}_{\mathrm{CR}}^{\star})$-hybrid model that SCC realizes $\mathcal{F}_{f,\mathrm{leak}}^{\star}$ (cf. Definition 1) and has the following properties:

- *Its optimistic communication/computation complexity is $2 \cdot |\mathsf{Gb}(1^{\lambda}, f, \cdot)| + \mathrm{poly}(k, m, \ell)$ where $|\mathsf{Gb}(1^{\lambda}, f, \cdot)|$ denotes the output length of Gb (i.e., size of the garbled circuit), and the optimistic validation complexity is $O(1)$ hash verifications.*
- *Its worst case validation complexity equals the complexity of NIZK verification in addition to $O(1)$ hash verifications.*

5. FAIR COMPUTATION

In this section, we show how to design fair protocols that are more round-efficient than prior constructions [11]. Our efficiency gains are due to use of a new Bitcoin transaction functionality which we formalize as an ideal functionality below.

Ideal functionality $\mathcal{F}_{\mathrm{ML}}^{\star}$. The purpose of $\mathcal{F}_{\mathrm{ML}}^{\star}$ (cf. Figure 7) is to allow n parties to jointly lock their coins in an atomic fashion, where each party P_i commits to a statement of the following kind: "Before round τ, I need to reveal a witness w_i that satisfies $\phi_i(w_i) = 1$, or else I will forfeit my security deposit of x coins." Hence $\mathcal{F}_{\mathrm{ML}}^{\star}$ satisfies the following:

- The atomic nature of $\mathcal{F}_{\mathrm{ML}}^{\star}$ guarantees that either all the n parties agreed on the circuits $\phi_i(\cdot)$, the limit τ, and the security deposit amount x, or else none of the coins become locked.
- Each corrupt party who aborts after the coins become locked is forced to pay $\mathsf{coins}(\frac{x}{n-1})$ to each honest party.
- If P_i reveals a correct w_i then w_i becomes public to everyone.
- The limit τ prevents the possibility that a corrupt party learns the witness of an honest party, and then waits for an indefinite amount of time before recovering its own coins amount.

The Bitcoin realization of $\mathcal{F}_{\mathrm{ML}}^{\star}$ is presented in Figure 10. The parameter $\tilde{\tau}$ denotes the double-spending safety distance, and the parameter τ' denotes how many $\tilde{\tau}$ intervals exist in a single "Bitcoin round". See [11, Appendices G and F] for technical Bitcoin details.

Given $\mathcal{F}_{\mathrm{ML}}^{\star}$, the following theorem is easy to prove.

Theorem 2. *Assuming the existence of one-way functions, for every n-party functionality f there exists a protocol that* securely computes *f* with penalties *in the $(\mathcal{F}_{\mathrm{OT}}, \mathcal{F}_{\mathrm{ML}}^{\star})$-hybrid model. Further, the protocol requires $O(1)$ rounds, a single invocation of $\mathcal{F}_{\mathrm{ML}}^{\star}$, and each party deposits $(n-1)$ times the penalty amount.*

Proof sketch. The protocol proceeds in two stages. In the first stage, parties run a (unfair) secure computation protocol in the $\mathcal{F}_{\mathrm{OT}}$-hybrid model that accepts input y_i, then computes $z \leftarrow f(y_1, \ldots, y_n)$, and then uses the $\mathsf{pubNMSS}$ primitive [11], which essentially additively shares z into $\mathsf{sh}_1, \ldots, \mathsf{sh}_n$, and then for every $j \in [n]$, computes (honest binding) commitments Tag_j on share with the corresponding decommitment Token_j. At the end of this stage, each P_j obtains $(\mathsf{AllTags}, \{\mathsf{Token}_j\}_{j\in[n]})$ where $\mathsf{AllTags} = \{\mathsf{Tag}_i\}_{i\in[n]}$. In the second stage, parties run a protocol for "fair reconstruction" of the shares.

Note that our first stage is exactly the same as in [11]. While they use $\mathcal{F}_{\mathrm{CR}}^{\star}$ to implement the second stage, we use $\mathcal{F}_{\mathrm{ML}}^{\star}$. Let $\phi_j(\mathsf{Token}_j; \mathsf{Tag}_j) = 1$ iff Token_i is a valid decommitment of Tag_j. Recall that $\{\mathsf{Tag}_j\}_{j\in[n]}$ are public, hence the relations ϕ_j can be specified by anyone, but the corresponding witness Token_j is known only to P_j. Given this, the protocol is quite straightforward. Let d be a deposit parameter (which we will set later). Let $D_i = (d, \phi_1, \ldots, \phi_n, \tau)$ for every $i \in [n]$. Each party sends $(\mathsf{lock}, sid, ssid, i, D_i, \mathsf{coins}(d))$ to $\mathcal{F}_{\mathrm{ML}}^{\star}$. If they receive abort from $\mathcal{F}_{\mathrm{ML}}^{\star}$, then they abort the protocol. Else, in round τ, each P_j sends $(\mathsf{redeem}, sid, ssid, j, \mathsf{Token}_j)$ to $\mathcal{F}_{\mathrm{ML}}^{\star}$, and receives back $\mathsf{coins}(d)$. If in round τ party P_i received $(\mathsf{redeem}, sid, ssid, j, \mathsf{Token}_j)$ for P_j, then it extracts the shares from each token, and reconstructs z, and terminates the protocol. Else it proceeds to round $\tau + 1$ and collects messages $(\mathsf{payout}, sid, ssid, j, i, \mathsf{coins}(d'))$ for each j for which P_i does not possess Token_j. This completes the description of the protocol. The protocol has a fairly straightforward simulation and follows ideas in [11]. Due to space limitations, we omit the simulation. □

Efficiency. In contrast to the constructions of [11] where the n parties broadcast $O(n)$ messages in $O(n)$ "Bitcoin rounds", with $\mathcal{F}_{\mathrm{ML}}^{\star}$ the parties broadcast $O(n^2)$ messages in $O(1)$ rounds. Note that if all parties are honest then $\mathcal{F}_{\mathrm{ML}}^{\star}$ requires only $O(n)$ transactions on the Bitcoin ledger, though $O(n^2)$ transaction data and $O(n^2)$ signatures (to assure compensations after the τ limit) are still needed.

5.1 Bitcoin protocol enhancement proposal

In [3, Section 3.2], the authors propose to modify the Bitcoin protocol so that in order to create a transaction tx_{new} that redeems an unspent output i of an earlier transaction tx_{old}, this output will be referenced in tx_{new} via $(\mathrm{SHA256d}(tx_{\mathsf{old}}^{\mathsf{simp}}), i)$ instead of $(\mathrm{SHA256d}(tx_{\mathsf{old}}), i)$. In other words, the id of tx_{old} shall be derived from the simplified form $tx_{\mathsf{old}}^{\mathsf{simp}}$, i.e., the form that excludes the input scripts which are required for tx_{old} to become valid. One important advantage of [3, Section 3.2] is allowing a user to commit coins on condition that another transaction would become valid, by referencing the simplified form of that other transaction. This enables users to have more rich kinds of contracts, and in particular it enables $\mathcal{F}_{\mathrm{ML}}^{\star}$. There is also a disadvantage, which is that we lose some of the expressive power that Bitcoin scripts currently allow. For example, suppose that P_1 can redeem an unspent output by revealing a witness w or w' (e.g. preimages of hardcoded hashed values $H(w), H(w')$). When P_1 broadcasts a transaction that redeems that output, and its transaction is added to the blockchain, the simplified id hash will not express whether P_1 revealed w or w'. Therefore, if P_2 and P_3 have some contract that depend on the witness that P_1 revealed, they may not be able to settle their contract since there would be plausible deniability that P_1 broadcast the other witness.

Our proposal here is to enhance [3, Section 3.2] and get the best of both worlds, by still using $\mathrm{SHA256d}(tx_{\mathsf{old}})$ as the id of tx_{old} for the Merkle tree in which the transaction tx_{old} resides, but using $\mathrm{SHA256d}(tx_{\mathsf{old}}^{\mathsf{simp}})$ to refer to tx_{old} in the transaction tx_{new}, i.e., the output that tx_{new} spends shall be specified as $(\mathrm{SHA256d}(tx_{\mathsf{old}}^{\mathsf{simp}}), i)$. This way, the PoW computations on the root of the Merkle tree to which $\mathrm{SHA256d}(tx_{\mathsf{old}})$ belongs will commit to the witness that redeemed tx_{old}, thus the disadvantage is eliminated. Let us note that inserting $\mathrm{SHA256d}(tx_{\mathsf{old}}^{\mathsf{simp}})$ as the id of tx_{old} in the UTXO set (i.e. a tree of the currently unspent outputs that Bitcoin nodes maintain) would commonly not even require an extra SHA256d invocation, since $\mathrm{SHA256d}(tx_{\mathsf{old}}^{\mathsf{simp}})$ has to be computed when verifying the signature of tx_{old} for the first time.

To realize $\mathcal{F}_{\mathrm{ML}}^{\star}$ with the current Bitcoin protocol, in step (6) of Figure 10 the parties need to run any *unfair* secure MPC protocol to obtain $id_{\mathsf{lock}} = \mathrm{SHA256d}(tx_{\mathsf{lock}})$. To elaborate, the input of each P_i for this MPC is $inp_i = \mathsf{Sign}_{sk_i}(tx_{\mathsf{lock}}^{\mathsf{simp}})$, and the output to all parties is $\mathrm{SHA256d}(tx_{\mathsf{lock}}^{\mathsf{simp}}, inp_1, \ldots, inp_n)$. This MPC can be unfair because the inputs $\{inp_i\}_{i=1}^{n}$ remain private, hence the coins cannot become locked until step (11) of Figure 10 executes.

6. NON-INTERACTIVE BOUNTIES

Our model consists of a bounty maker denoted M and a set of parties $P_1, P_2, \ldots, P_N$ (denoting parties in the Bitcoin network).

$\mathcal{F}^{\star}_{\mathrm{ML}}$ with session identifier sid, running with parties $P_1, \ldots, P_n$ and a parameter 1^{λ}, proceeds as follows:

- *Lock phase.* Wait to receive $(\mathsf{lock}, sid, ssid, i, D_i = (x, \phi_1, \ldots, \phi_n, \tau), \mathsf{coins}(x))$ from each P_i and record $(\mathsf{locked}, sid, ssid, i, D_i)$. Then if $\forall i, j : D_i = D_j$ send message $(\mathsf{locked}, sid, ssid)$ to all parties and proceed to the *Redeem phase*. Otherwise, for all i, if the message $(\mathsf{locked}, sid, ssid, i, D_i)$ was recorded then delete it and send message $(\mathsf{abort}, sid, ssid, i, \mathsf{coins}(x))$ to P_i, and terminate.
- *Redeem phase.* In round τ: upon receiving a message $(\mathsf{redeem}, sid, ssid, i, w_i)$ from P_i, if $\phi_i(w_i) = 1$ then delete $(\mathsf{locked}, sid, ssid, i, D_i)$, send $(\mathsf{redeem}, sid, ssid, \mathsf{coins}(x))$ to P_i and $(\mathsf{redeem}, sid, ssid, i, w_i)$ to all parties.
- *Payout phase.* In round $\tau + 1$: For all $i \in [n]$: if $(\mathsf{locked}, sid, ssid, i, D_i)$ was recorded but not yet deleted, then delete it and send the message $(\mathsf{payout}, sid, ssid, i, j, \mathsf{coins}(\frac{x}{n-1}))$ to every party $P_j \neq P_i$.

Figure 7: The ideal functionality $\mathcal{F}^{\star}_{\mathrm{ML}}$.

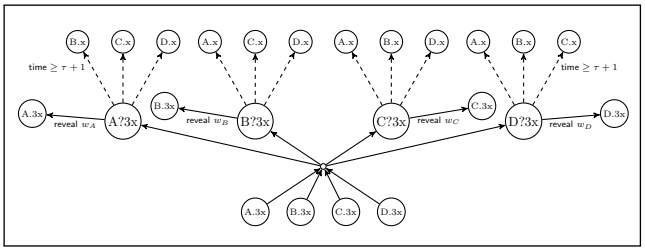

Figure 8: Illustration of the $\mathcal{F}^{\star}_{\mathrm{ML}}$ Functionality.

M holds a relation Φ_x (defining a NP language $\mathcal{L}$) and wishes to learn a witness w such that $\Phi_x(w) = 1$. In return for the knowledge of the witness M is willing to pay $\mathsf{coins}(q)$ to a party $C \in \{P_1, \ldots, P_N\}$ that finds the witness. We stress that at the time of bounty creation, the identity of the bounty collector is unknown. Informally, the properties that we want to guarantee from our bounty collection problem are:

- (Noninteractive.) The bounty maker M publishes a single message to the network and remains passive otherwise.
- (Race-free soundness.) If there exists at most one party C that knows the witness, then no party other than C or M can claim the bounty except with probability negligible in λ.
- (Correctness and privacy.) An honest C holding valid witness will claim the bounty except with probability negligible in λ. In this case, only M learns the witness.

For simplicity we assume that there exists exactly one such bounty collector C. Furthermore, we assume that the bounty maker is honest (alternatively we can ask M to give a ZK proof that its published message corresponds to a bounty). We assume that some small fraction of the Bitcoin miners are malicious. Therefore, if the witness is made public on the Bitcoin network, this may in turn result in a scenario where parties "race" to claim the bounty. Afterall in such a situation, there is nothing that distinguishes C from any other party. Formally, we define a *noninteractive private bounty mechanism* as a four-tuple of algorithms $(\mathsf{Make}, \mathsf{Coll}, \mathsf{Ver}, \mathsf{Ext})$:

1. $(t_m, \omega) \leftarrow \mathsf{Make}(1^{\lambda}, \Phi_x)$. The bounty maker with input ϕ uses private randomness ω and runs Make to generate t_m.
2. $t_c \leftarrow \mathsf{Coll}(w, t_m)$. The bounty collector with a witness w such that $\phi(w) = 1$ uses algorithm Coll to generate t_c.
3. $\{0, 1\} \leftarrow \mathsf{Ver}(t_m, t_c)$. Upon receiving a claim t_c the miners use Ver to determine whether the claim is valid.
4. $w \cup \perp \leftarrow \mathsf{Ext}(\omega, t_c)$. The bounty maker runs the algorithm Ext using the message t_c. The output of the algorithm is either w such that $\Phi_x(w) = 1$ or $\perp$.

The scheme should satisfy:

Correctness (with guaranteed extraction) For any x, w such that if $x \in L$ (i.e., $\Phi_x(w) = 1$):

$$\Pr\left[\begin{array}{c} (t_m, \omega) \leftarrow \mathsf{Make}(\phi, 1^{\lambda}); \\ t_c \leftarrow \mathsf{Coll}(w, t_m) \end{array} : \begin{array}{c} 1 = \mathsf{Ver}(t_m, t_c) \bigwedge \\ w = \mathsf{Ext}(\omega, t_c) \end{array}\right] = 1.$$

Extractability There exists a simulator $\mathsf{Sim} = (S_1, S_2)$ such that for all PPT adversaries $\mathcal{E}$ and all poly q there exists a PPT extractor E and a poly p, such that for all auxiliary inputs z and for all $x \in \{0,1\}^*$ the following holds:

$$\left| \begin{array}{l} \Pr\left[\begin{array}{l} (t_m, \omega) \leftarrow \mathsf{Make}(\Phi_x, 1^{\lambda}); \\ t_c \leftarrow \mathsf{Coll}(w, t_m) \end{array} : 1 = \mathcal{E}(t_m, t_c)\right] = 1 \\ -\Pr\left[\begin{array}{l} (t_m, \mathsf{st}) \leftarrow S_1(\Phi_x, 1^{\lambda}); \\ t_c \leftarrow S_2(t_m, \mathsf{st}) \end{array} : 1 = \mathcal{E}(t_m, t_c)\right] = 1 \end{array} \right| \geq 1/q(|x|)$$

$$\implies \Pr[E(x, z) = w : \Phi_x(w) = 1] \geq 1/p(|x|).$$

The extractability condition above essentially formalizes the privacy property which in turn helps in satisfying the "race-free soundness" property. The condition effectively states that an adversary does not learn any information about the witness w even after obtaining both the bounty maker's message and the collector's message. More precisely, an adversary can distinguish between the simulated messages (where the simulator does not use the witness at all) and the actual messages generated by M and C, only if it already knew the witness. This leads us to a contradiction since we assumed that only C knows the witness. Our definitions are inspired by definitions of *witness encryption*, a powerful cryptographic primitive that excellently fits to our scenario.

Definition 5 (Witness encryption [16, 20])**.** A witness encryption scheme for an $\mathbf{NP}$ language L (with corresponding witness relation ϕ) consists of the following two polynomial-time algorithms:

Encryption. The algorithm $\mathsf{Enc}(1^{\lambda}, x, m)$ takes as input a security parameter 1^{λ}, an unbounded-length string x, and a message $m \in \{0, 1\}$, and outputs a ciphertext ψ.

Decryption. The algorithm $\mathsf{Dec}(\psi, w)$ takes as input a ciphertext ψ and an unbounded-length string w, and outputs a message m or the symbol $\perp$.

These algorithms satisfy:

- **Correctness.** For any security parameter λ, for any $m \in \{0, 1\}$, and for any $x \in L$ such that $\phi(w; x)$ holds, we have that

$$\Pr[\mathsf{Dec}(\mathsf{Enc}(1^{\lambda}, x, m), w) = m] = 1. \qquad \diamondsuit$$

Definition 6 (Extractable security [20])**.** A witness encryption scheme for a language $\mathcal{L} \in \mathbf{NP}$ is secure if for all PPT adversaries A and all poly q there exists a PPT extractor E and a poly p, such that for all auxiliary inputs z and for all $x \in \{0,1\}^*$ the following holds:

$$\Pr\left[\begin{array}{c} b \leftarrow \{0,1\}; \psi \leftarrow \mathsf{Enc}(1^{\lambda}, x, b) \\ : A(x, \psi, z) = b \end{array}\right] \geq 1/2 + 1/q(|x|)$$

$$\implies \Pr[E(x, z) = w : \phi(w; x) = 1] \geq 1/p(|x|).$$

A starting point is to let the bounty maker create a witness encryption ψ of a signing key sk and create a Bitcoin transaction t that allows a party to claim the bounty only if it possesses sk. Clearly, a party holding the witness is able to decrypt ψ and using sk is able to transfer the bounty to a different address $\texttt{addr}$ of its choice. Note however that the above solution does not allow the bounty maker to learn the witness! Alternatively, if the bounty maker modifies t such that the bounty can be claimed only if a party can produce w such that $\Phi_x(w) = 1$, then this appears to solve the problem. Unfortunately, this idea turns out to be naïve since a party C that claims the transaction reveals the witness which when made public allows malicious miners to decrypt ψ, recover the signing key and then claim the bounty to an address $\texttt{addr}'$ of its choice. In other words, this leads to a network race between the legitimate collector C and malicious nodes on the Bitcoin network.

What we need is a mechanism that simultaneously allows a legitimate collector to claim the bounty while allowing the bounty maker to extract a valid witness. We present a novel solution to this problem via use of garbled circuits. In our construction the bounty maker broadcasts the following: (1) witness encryption ψ of the signing key sk, (2) a garbled circuit computing $\Phi_x(\cdot)$, (3) witness encryption ψ' of the input labels $\mathbb{U}$ corresponding to GC, (4) the output label w_1 of GC corresponding to the value 1, and (5) a transaction that releases the bounty to a party that possesses sk and supplies input labels that evaluates GC to produce the output label w_1. Clearly, a legitimate collector can claim the bounty by decrypting ψ, ψ' and the revealing the input labels w' corresponding to witness w. Further, since the bounty maker knows all input labels, it can obtain the witness using w'. On the other hand, the privacy property of the garbling scheme ensures that a malicious miner that obtains w', GC still does not have any information about the actual witness w. Although the miners can copy the value w' and claim it as their own, a network race is avoided because they are unable to forge a signature without knowing the signing key. Our bounty mechanism is presented in Figure 9. We formally prove:

Theorem 3. *Let λ be a computational security parameter. Assuming the existence of extractable witness encryption, an existentially unforgeable secure signature scheme* $(\mathsf{SigKeyGen}, \mathsf{Sig}, \mathsf{SigVer})$*, and a secure garbling scheme* $(\mathsf{Gb}, \mathsf{Eval})$*, there exists a noninteractive private bounty mechanism for NP language $\mathcal{L}$ with relation $\Phi_x(\cdot)$ for $x \in \mathcal{L}$ whose validation complexity equals the complexity of* $\mathsf{Eval}(\mathsf{Gb}(1^\lambda, \Phi_x), \cdot)$ *plus the complexity of* SigVer.

Proof sketch. We rely on the semantic security of the extractable witness encryption scheme as well as the existential unforgeability of the signature scheme. Specifically, we consider a simulator that upon receiving input Φ_x for $x \in \mathcal{L}$ does the following:

- Compute $(\hat{GC}, \hat{r}, \hat{\mathbb{U}}, \hat{\mathbf{w}}) \leftarrow \mathsf{Fake}(1^\lambda, \Phi_x)$.
- Let $\hat{w} \in \hat{\mathbf{w}}$. Generate $(pk, sk) \leftarrow \mathsf{SigKeyGen}(1^\lambda)$.
- Compute $\hat{\psi} = \mathsf{Enc}(1^\lambda, x, 0^\lambda)$ and $\hat{\psi}' = \mathsf{Enc}(1^\lambda, x, \hat{\mathbb{U}})$.
- Generate $\hat{\sigma} = \mathsf{Sig}_{sk}(r')$ for random r' and $\hat{w}' = \hat{\mathbb{U}}^{\hat{r}}$.
- Output $t_m = (\Phi_x, \hat{\psi}, \hat{\psi}', pk, \hat{GC}, \hat{h})$ and $t_c = (\hat{\sigma}', \hat{w}')$.

We then construct a series of games starting from the real transcript and ending up with the simulated transcript. In the first set of games we replace ψ by $\hat{\psi}$, and then we replace one-by-one the input labels in $\mathbb{W}$ with encryptions of 0 in a way that ultimately ends up in transforming $\mathbb{U}$ to have a structure similar to $\hat{\mathbb{U}}$. In the second set, we replace the actual garbled circuit GC and the legitimate input labels w' by their faked counterparts $\hat{GC}, \hat{w}'$. By the security of the garbling scheme we have that the adversary's advantage in second set of games is negligible in λ. Therefore, an adversary that has $1/\mathrm{poly}$ advantage in distinguishing real transcripts from simulated transcripts must have $1/\mathrm{poly}$ advantage in distinguishing between the real transcript and the last of the first set of games. Then by appealing to the extrability of witness encryption schemes, we can derive an extractor who succeeds in guessing the witness with $1/\mathrm{poly}$ probability. Finally observe that the privacy of the scheme and the security of the signature scheme taken together suffice to show that our mechanism provides race-free soundness. □

Let $(\mathsf{Enc}, \mathsf{Dec})$ be a witness encryption scheme for $\mathcal{L}$ with witness relation Φ_x. Let $(\mathsf{SigKeyGen}, \mathsf{Sig}, \mathsf{SigVer})$ be an existentially unforgeable secure signature scheme. Let $(\mathsf{Gb}, \mathsf{Eval})$ be a secure garbling scheme. The bounty protocol proceeds as follows:

- M with input Φ_x executes $\mathsf{Make}(1^\lambda, \Phi_x, \omega)$ for random ω:
  - Generate $(pk, sk) \leftarrow \mathsf{SigKeyGen}(1^\lambda)$.
  - Generate $(GC, \mathbb{U}, \mathbb{W}) \leftarrow \mathsf{Gb}(1^\lambda, \Phi_x; \omega)$.
  - Compute $\psi = \mathsf{Enc}(1^\lambda, x, sk)$ and $\psi' = \mathsf{Enc}(1^\lambda, x, \mathbb{U})$.
  - Let $(w_0, w_1) = \mathbb{W}$. Set $t_m = (\Phi_x, \psi, \psi', pk, GC, w_1)$.
- C holding w such that $\Phi_x(w) = 1$ executes $\mathsf{Coll}(w, t_m)$:
  - Compute $sk \leftarrow \mathsf{Dec}(\psi, w)$ and $\mathbb{U} \leftarrow \mathsf{Dec}(\psi', w)$.
  - Set $t_c = (\sigma = \mathsf{Sig}_{sk}(\texttt{addr}), w' = \mathbb{U}^w)$.
- Miners execute $\mathsf{Ver}(t_m, t_c)$:
  - Parse $t_m = (\Phi_x, \psi, \psi', GC, w_1)$ and $t_c = (\tilde{\sigma}, \tilde{w}')$.
  - Output 1 iff $\mathsf{SigVer}(pk, \tilde{\sigma}) = 1 \bigwedge \mathsf{Eval}(GC, \tilde{w}') = w_1$.
- M executes $\mathsf{Ext}(\phi, \omega, t_c)$:
  - Parse $t_c = (\sigma', w')$. If $\mathsf{SigVer}(pk, \sigma') = 0$, output $\perp$.
  - Output $\hat{w}$ s.t. $\mathbb{U}^{\hat{w}} = w'$. If no such $\hat{w}$ exists output $\perp$.

Figure 9: A noninteractive Bitcoin bounty mechanism.

Remark. Extractable witness encryption is a heavy assumption [15] and is quite inefficient in practice (cf. [13]). We sketch a *heuristic* construction to replace use of witness encryption that works for certain languages. For e.g., assume that $x \in \mathcal{L}$ iff x is a RSA modulus. Let $\Phi_x(w) = 1$ iff $w = (p, q)$ such that both p and q are prime and $x = p \cdot q$. Our key observation is that we can replace the witness encryption scheme simply by any RSA encryption scheme with RSA modulus x. Note that knowing the factorization of the RSA modulus x readily allows decryption.

Bounties via time-locked puzzles. We now sketch a noninteractive *nonprivate* bounty mechanism that still enjoys race-free soundness. To do this, we use a time-lock puzzles scheme [32]. Such a scheme allows the bounty maker M to generate a time-locked encryption $sk' = \mathsf{puzz}(sk, t)$, so that it should take approximately time t for anyone besides M to compute sk from sk' (even allowing parallel computations). The time-lock scheme allows M to generate sk' in time that is orders of magnitude shorter than t, hence M can estimate which t_0 implies that the puzzle would take e.g. ≥ 30 minutes to solve at the year in which computing the witness w is likely to be feasible, and use $\psi = \mathsf{Enc}(1^\lambda, x, \mathsf{puzz}(sk, t_0))$ for the witness encryption scheme. This way, C would have a head start of t_0 over other parties, and is therefore likely to win the race because its transaction will be buried under enough PoW blocks. Depending on the complexity of $\Phi_x(\cdot)$, this bounty protocol may be realizable with the current Bitcoin standard scripts.

7. CONCLUSIONS

In this paper we have shown that a variety of cryptographic primitives can be incentivized in order to encourage honest behavior by participants. We believe that our constructions offer compelling motivation to change the state-of-affairs. Our work leaves a number of open questions some of which are mentioned below.

- **Verifiable computation.** Is it possible to develop a formal model to incentivize based on the resource usage of the worker in private verification schemes?
- **Fair computation.** Is it possible to design a protocol that needs only $O(1)$ rounds and $O(n)$ transactions in the worst case?
- **Secure computation with leakage.** Is it possible to come up with a general methodology to design highly efficient secure computation protocols that guarantee restricted leakage? Can such protocols be incentivized?

Acknowledgments

This work was supported by funding received from European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 259426 and 240258. The second author would like to thank Alex Mizrahi for useful discussions and contributions to Section 5.1.

8. REFERENCES

[1] Bitcoin wiki: CVEs. https://en.bitcoin.it/wiki/CVEs#CVE-2010-5141.
[2] G. Andresen. Turing complete language vs non-turing complete. https://bitcointalk.org/index.php?topic=431513.20#msg4882293.
[3] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Fair two-party computations via the bitcoin deposits. In *First Workshop on Bitcoin Research, FC*, 2014.
[4] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. In *IEEE Security and Privacy*, 2014.
[5] Gilad Asharov, Yehuda Lindell, and Hila Zarosim. Fair and efficient secure multiparty computation with reputation systems. In *Asiacrypt (2)*, pages 201–220, 2013.
[6] Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In *Asiacrypt*, pages 681–698, 2012.
[7] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. In *Eurocrypt*, 1998.
[8] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *4th Theory of Cryptography Conference — TCC 2007*, volume 4392 of *LNCS*, pages 137–156. Springer, February 2007.
[9] S. Barber, X. Boyen, E. Shi, and E. Uzun. Bitter to better - how to make bitcoin a better currency. In *FC*, 2012.
[10] Mira Belenkiy, Melissa Chase, C. Christopher Erway, John Jannotti, Alptekin Kupcu, and Anna Lysyanskaya. Incentivizing outsourced computation. In *NetEcon*, pages 85–90, 2008.
[11] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *ePrint 2014/129*, 2014.
[12] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Crypto (1)*, 2013.
[13] J.-S. Coron, T. Lepoint, and M. Tibouchi. Practical multlinear maps over the integers. In *Crypto (1)*, 2013.
[14] E. Friedman and P. Resnick. The social cost of cheap pseudonyms. In *Journal of Economics and Management Strategy*, pages 173–199, 2000.
[15] S. Garg, C. Gentry, S. Halevi, and D. Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. In *ePrint 2013/860*.
[16] S. Garg, C. Gentry, A. Sahai, and B. Waters. Witness encryption and its applications. In *STOC*, 2013.
[17] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology — Crypto 2010*, 2010.
[18] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *Eurocrypt*, 2013.
[19] Oded Goldreich. Foundations of cryptography - vol. 2. 2004.
[20] S. Goldwasser, Y.T̃. Kalai, R.Ã. Popa, V. Vaikuntanathan, and N. Zeldovich. How to run turing machines on encrypted data. In *Crypto (2)*, pages 536–553, 2013.
[21] Philippe Golle and Ilya Mironov. Uncheatable distributed computations. In David Naccache, editor, *Cryptographers' Track — RSA 2001*, volume 2020 of *LNCS*, pages 425–440. Springer, April 2001.
[22] V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In *Advances in Cryptology — Eurocrypt 2008*.
[23] Y. Huang, J. Katz, and D. Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *IEEE Security and Privacy*, pages 272–284, 2012.
[24] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In *Advances in Cryptology — Crypto 2008*, pages 572–591, 2008.
[25] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *CCS*, pages 875–888.
[26] L. Lamport. Fast paxos, 2005. MSR-TR-2005-112.
[27] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology.*, 22(2):161–188, 2009.
[28] G. Maxwell. Zero knowledge contingent payment. 2011. https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment.
[29] P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *PKC 2006*.
[30] V. Pappas, B. Vo, F. Krell, S.-G. Choi, V. Kolesnikov, S. Bellovin, A. Keromytis, and T. Malkin. Blind seer: A scalable private dbms. In *IEEE Security and Privacy*, 2014.
[31] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE S&P*, 2013.
[32] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, MIT, 1996.
[33] A. Rosen and A. Shelat. Optimistic concurrent zero knowledge. In *Advances in Cryptology — Asiacrypt 2010*.
[34] P. Todd. Reward offered for hash collisions for sha1, sha256, ripemd160. https://bitcointalk.org/index.php?topic=293382.0, 2013.
[35] Andrew Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

APPENDIX

A. PROOF OF THEOREM 2

Consider the protocol in Figure 6. Using the protocol of [11], we have that implementing $\mathcal{F}_{\mathsf{SV}}^{\star}$ in the $(\mathcal{F}_{\mathrm{OT}}, \mathcal{F}_{\mathrm{CR}}^{\star})$-hybrid model has validation complexity the $O(1)$ hash verifications. Given this, it is easy to see that its optimistic communication/computation/validation complexity and the worst case validation complexity is exactly as stated in the theorem. In the rest of the proof we give details about the simulation in the $(\mathcal{F}_{\mathsf{KT}},\mathcal{F}_{\mathsf{SV}}^{\star},\mathcal{F}_{\mathrm{CR}}^{\star})$-hybrid model. Note that $\mathcal{F}_{\mathsf{KT}}$ can be realized in the $\mathcal{F}_{\mathrm{OT}}$-hybrid model [24] and that $\mathcal{F}_{\mathsf{SV}}^{\star}$ can be realized in the $(\mathcal{F}_{\mathrm{OT}},\mathcal{F}_{\mathrm{CR}}^{\star})$-hybrid model [11]. Assume that adversary $\mathcal{A}$ corrupts P_i. Let P_j denote the honest party. We sketch the simulation below. Let $(\mathcal{S}_1, \mathcal{S}_2)$ be the NIZK simulator.

- $\mathcal{S}$ computes $(\mathsf{crs}, t) \leftarrow \mathcal{S}_1(1^{\lambda})$.
- $\mathcal{S}$ computes $(\hat{G}_j, \hat{x}, \hat{\mathbb{U}}, \hat{\mathbf{w}}) \leftarrow \mathsf{Fake}(1^{\lambda}, f)$.

- Acting as $\mathcal{F}_{\mathsf{KT}}$, $\mathcal{S}$ next obtains (m, x_i, ω_i) from $\mathcal{A}$. If $\mathcal{A}$ sends abort, then $\mathcal{S}$ terminates the simulation. Else it specifies output of $\mathcal{F}_{\mathsf{KT}}$ as $(\hat{\mathbb{U}}, g'_j = \mathsf{com}(\mathbf{0}; \hat{\rho}_j))$ where $\hat{\rho}_j$ is picked uniformly at random.
- Acting as P_j, $\mathcal{S}$ sends $\hat{G}_j$ to $\mathcal{A}$ and receives G_i from $\mathcal{A}$.
- $\mathcal{S}$ chooses random r_j and sends $h_j = \mathsf{H}(r_j)$ to $\mathcal{A}$ and receives h_i from $\mathcal{A}$.
- Acting as $\mathcal{F}^{\star}_{\mathsf{SV}}$, $\mathcal{S}$ obtains $(\mathsf{input}, sid, ssid, (\ell_i, \mathbf{w}_i, \tilde{\omega}_i, \tilde{r}_i, \tilde{h}_j, \tilde{g}'_j), \mathsf{coins}(q))$ from $\mathcal{A}$. If $\tilde{\omega}_i \neq \omega_i$ or $\mathsf{H}(\tilde{r}_i) \neq h_i$ or $\tilde{h}_j \neq h_j$ or $\tilde{g}'_j \neq g'_j$ or $\mathbf{w}_i \neq \hat{\mathbf{w}}$, then set $z_i = \mathsf{bad}$.
- Using the extracted ω_i, $\mathcal{S}$ computes $\mathbb{U}_i, \mathbb{W}_i$. $\mathcal{S}$ specifies the following as the "leakage" function $L_{x_i}(x_j)$:
  - Compute $\mathbf{w}_j = \mathsf{Eval}(G_i, \mathbb{U}_i^{x_1 \| x_2})$.
  - Compute $v \in \{0,1\}^{\ell}$ such that $\mathbf{w}_j = \mathbb{W}_i^v$.
  - If $v = f(x_1, x_2)$ return 1, else return 0.
- If z_i is not already set to bad, $\mathcal{S}$ sends $(\mathsf{input}, sid, ssid, x_i, L, \mathsf{coins}(q))$ to $\mathcal{F}^{\star}_{f,\mathrm{leak}}$.
- If $(\mathsf{abort}, sid, ssid)$ is received from $\mathcal{F}^{\star}_{f,\mathrm{leak}}$, then set $z_i = (r_1, r_2)$.
- $\mathcal{S}$ acting as $\mathcal{F}^{\star}_{\mathsf{SV}}$ sets output as z_i and delivers output messages to $\mathcal{A}$.
- If $z_i = (r_1, r_2)$, then $\mathcal{S}$ computes a simulated NIZK argument $\hat{\pi} \leftarrow \mathcal{S}_2(\mathsf{crs}, (G_j, g'_j), t)$, and acting as $\mathcal{F}^{\star}_{\mathrm{CR}}$ sends message $(\mathsf{claim}, sid, ssid, i, j, \phi_i, \tau, q, (\hat{\pi}, r_j))$ to $\mathcal{A}$.
- If at any stage $\mathcal{S}$ acting as $\mathcal{F}^{\star}_{\mathrm{CR}}$ receives message $(\mathsf{claim}, sid, ssid, j, i, \phi_j, \tau, q, w)$ from $\mathcal{A}$ and it holds that $\phi_j(w) = 1$, then it outputs fail and terminates the simulation.

We first prove that conditioned on $\mathcal{S}$ not outputting fail, the simulation is indistinguishable from the protocol execution throught a series of hybrid execution. Let Hyb_0 denote the protocol execution. In Hyb_1, we change the NIZK argument with a simulated argument. Indistinguishability of Hyb_0 and Hyb_1 follows from the zero-knowledge property of NIZKs. In Hyb_2, we compute $g'_j = \mathsf{com}(\mathbf{0})$ (i.e., commitment on the all-zero string) instead of $\mathsf{com}(\omega_j)$. Indistinguishability of Hyb_1 and Hyb_2 follows from the hiding property of the commitment scheme. In Hyb_3, we compute $G_j, \mathbb{U}'_j$ using Fake (instead of Gb and iGb). Indistinguishability of Hyb_2 and Hyb_3 follows from the security of the garbling scheme Gb. It is easy to see that Hyb_3 is identical to the simulated execution.

It remains to be shown that the probability that $\mathcal{S}$ outputs fail is negligible in λ. We consider two cases. Suppose the output $z_i = z$. In this case, observe that $\mathcal{S}$ outputs fail iff $\mathcal{A}$ produces r' such that $\mathsf{H}(r') = h_j$. It then follows from the collision-resistance of H that such an event happens with negligible probability. On the other hand suppose the output $z_i = (r_1, r_2)$. In this case, $\mathcal{S}$ outputs fail iff $\mathcal{A}$ provides a valid proof that $(G_i, g'_i) \in \mathcal{L}$. By the soundness property of NIZK, except with negligible probability there exists ω, ρ such that $g'_i = \mathsf{com}(\omega; \rho)$ and $G_i \leftarrow \mathsf{Gb}(1^{\lambda}, f, \omega)$. Now suppose $G_i \neq \mathsf{Gb}(1^{\lambda}, f, \omega_i)$ (where ω_i was extracted via $\mathcal{F}_{\mathsf{KT}}$), then this means that g'_i can be opened to both ω as well as ω_i and thus we have a contradiction since we assumed a perfectly binding commitment scheme. On the other hand, if $G_i = \mathsf{Gb}(1^{\lambda}, f, \omega_i)$, then essentially $\mathcal{A}$ has executed the protocol honestly. It can then be easily verified that if $z_i \neq \mathsf{bad}$, then z_i will not be of the form (r_1, r_2) either. This completes the proof.

Lock phase.

1. Every P_i holds a public key pk_i for which only it knows the corresponding secret key sk_i, scripts $\{\phi_j\}_{j=1}^n$, locktime value $\tau_0 = \tau \cdot \tau' \cdot \tilde{\tau}$, and an unspent output (id_i, t_i) of $p = x(n-1)$ coins that it controls (i.e. specified as a pair of transaction id and output index).
2. For $i \in [n-1]$, P_i sends $(\mathsf{lock\_init}, i, (id_i, t_i), pk_i)$ to P_n.
3. P_n creates the simplified transaction $tx^{\mathsf{simp}}_{\mathsf{lock}}$ that spends the n inputs $\big[(id_1, t_1), \ldots, (id_n, t_n)\big]$ to n outputs $\big[(p, \pi_1) \ldots, (p, \pi_n)\big]$, where $\pi_i(w, s_1, \ldots, s_n) \triangleq (\texttt{OP\_CHECKSIG}(pk_1, s_1) \wedge \ldots \wedge \texttt{OP\_CHECKSIG}(pk_n, s_n)) \vee (\texttt{OP\_CHECKSIG}(pk_i, s_i) \wedge \phi_i(w) = 1)$.
4. P_n sends $(\mathsf{lock\_prepare}, tx^{\mathsf{simp}}_{\mathsf{lock}})$ to all parties.
5. Every P_i ensures that for all $j \in [n]$, the j^{th} output (y_j, π_j) of $tx^{\mathsf{simp}}_{\mathsf{lock}}$ has $y_j = p$ and π_j incorporates ϕ_j accordingly.
6. Let $id_{\mathsf{lock}} \leftarrow \text{SHA256d}(tx^{\mathsf{simp}}_{\mathsf{lock}})$. **Note:** this is justified due to Section 5.1, and the reader is referred to Section 5.1 for an alternative that works with the current Bitcoin protocol.
7. Every P_i creates a simplified transaction $tx^{\mathsf{simp}}_{\mathsf{pay}:i}$ that has locktime τ_0 and spends the input (id_{lock}, i) to $n-1$ outputs $\big[(x, \psi^1_i(\cdot) = \texttt{OP\_CHECKSIG}(pk_1, \cdot)), \ldots, (x, \psi^n_i(\cdot) = \texttt{OP\_CHECKSIG}(pk_n, \cdot))\big]$ excluding $(x, \psi^i_i(\cdot))$, and sends $(\mathsf{payback}, i, tx^{\mathsf{simp}}_{\mathsf{pay}:i}, ps_{i,i} = \mathsf{Sign}_{sk_i}(tx^{\mathsf{simp}}_{\mathsf{pay}:i}))$ to all parties.
8. Every P_i ensures that all the locktime values of $\{tx^{\mathsf{simp}}_{\mathsf{pay}:j}\}_{j \in [n] \setminus \{i\}}$ equal τ_0. Otherwise P_i aborts.
9. Every P_i computes $n-1$ signatures $S_i = \{ps_{i,j} = \mathsf{Sign}_{sk_i}(tx^{\mathsf{simp}}_{\mathsf{pay}:j})\}_{j \in [n] \setminus \{i\}}$, and sends the message $(\mathsf{payback\_ack}, i, S_i)$ to all parties.
10. Every P_i extracts $\{pk_j\}_{j \in [n] \setminus \{i\}}$ from $tx^{\mathsf{simp}}_{\mathsf{lock}}$ and ensures that $\mathsf{Vrfy}_{pk_j}(tx^{\mathsf{simp}}_{\mathsf{pay}:k}, ps_{j,k}) = 1$ for all $j \in [n] \setminus \{i\}$ and all $k \in [n] \setminus \{i\}$. Otherwise P_i aborts.
11. Every P_i computes $sig_i = \mathsf{Sign}_{sk_i}(tx^{\mathsf{simp}}_{\mathsf{lock}})$. For $i \in [n-1]$, P_i sends the message $(\mathsf{lock\_finalize}, i, sig_i)$ to P_n.
12. P_n transforms $tx^{\mathsf{simp}}_{\mathsf{lock}}$ to tx_{lock} by injecting each sig_j as the script that redeems the input (id_j, t_j), and broadcasts the now valid transaction tx_{lock} to the Bitcoin network.

Redeem phase.

13. Every P_i waits until $\tilde{\tau}$ PoW blocks extend the block in which tx_{lock} resides, and then broadcasts to the Bitcoin network a transaction that spends the i^{th} output of tx_{lock} by signing with sk_i and revealing w_i that satisfies $\phi_i(w_i) = 1$.
14. Until τ_0 blocks have been solved by the Bitcoin network, every P_i listens on the network and waits until for all $j \in [n] \setminus \{i\}$, P_j redeems the j^{th} output of tx_{lock} and thereby reveals w_j that satisfies $\phi_j(w_j) = 1$.

Payout phase. After $(\tau+1)\tau'\tilde{\tau} - \tilde{\tau}$ blocks have been solved:

15. Every P_i checks for each $j \in [n] \setminus \{i\}$ whether the j^{th} output of tx_{lock} can still be spent. If so, P_i injects the signatures $\{ps_{j,k}\}_{k=1}^n$ into $tx^{\mathsf{simp}}_{\mathsf{pay}:j}$, and broadcasts the now valid transaction $tx_{\mathsf{pay}:j}$ to the Bitcoin network.

Figure 10: Realizing $\mathcal{F}^{\star}_{\mathrm{ML}}$ in Bitcoin.