Improvements to Secure Computation with Penalties

Ranjit Kumaresan
MIT
Cambridge, Massachusetts
vranjit@mit.edu

Vinod Vaikuntanathan
MIT
Cambridge, Massachusetts
vinodv@mit.edu

Prashant Nalini Vasudevan
MIT
Cambridge, Massachusetts
prashvas@mit.edu

ABSTRACT

Motivated by the impossibility of achieving fairness in secure computation [Cleve, STOC 1986], recent works study a model of fairness in which an adversarial party that aborts on receiving output is forced to pay a mutually predefined monetary penalty to every other party that did not receive the output. These works show how to design protocols for *secure computation with penalties* that tolerate an arbitrary number of corruptions.

In this work, we improve the efficiency of protocols for secure computation with penalties in a hybrid model where parties have access to the "claim-or-refund" transaction functionality. Our first improvement is for the ladder protocol of Bentov and Kumaresan (Crypto 2014) where we improve the dependence of the script complexity of the protocol (which corresponds to miner verification load and also space on the blockchain) on the number of parties from quadratic to linear (and in particular, is completely independent of the underlying function). Our second improvement is for the see-saw protocol of Kumaresan et al. (CCS 2015) where we reduce the total number of claim-or-refund transactions and also the script complexity from quadratic to linear in the number of parties.

We also present a 'dual-mode' protocol that offers different guarantees depending on the number of corrupt parties: (1) when $s < n/2$ parties are corrupt, this protocol guarantees fairness (i.e., either all parties get the output or none do), and (2) when $t > n/2$ parties are corrupt, this protocol guarantees fairness with penalties (i.e., if the adversary gets the output, then either the honest parties get output as well or they get compensation via penalizing the adversary). The above protocol works as long as $t + s < n$, matching the bound obtained for secure computation protocols in the standard model (i.e., replacing "fairness with penalties" with "security-with-abort" (full security except fairness)) by Ishai et al. (SICOMP 2011).

Keywords: Bitcoin, secure computation, fairness.

1. INTRODUCTION

Protocols for secure multiparty computation [29, 15, 8, 11] allow a set of mutually distrusting parties to carry out a distributed computation without compromising on privacy of inputs or correctness of the end result. Despite being a powerful tool, it is known that secure computation protocols do not provide fairness or guaranteed output delivery when a majority of the parties are dishonest [12].[1] Addressing this deficiency is critical if secure computation is to be widely adopted in practice, especially given the current interest in practical secure computation. Several workarounds have been proposed in the literature to counter adversaries that may decide to abort, possibly depending on the outcome of the protocol (see [28, 4, 23, 17]). In this work, we are interested in the workaround proposed in [24, 23, 7] where an adversarial party that aborts on receiving output is forced to pay a mutually predefined monetary penalty to every other part that did not receive the output. In practice, such mechanisms would be effective if the compensation amount is rightly defined. While the original works [24, 23, 7] depended on e-cash systems, recent works [5, 2, 9, 21, 1, 22, 19] have shown how to use a decentralized digital currency (like Bitcoin) to design protocols for secure computation in the penalty model.

Our contributions in a nutshell. We present two efficiency improvements to secure computation with penalties.

- We improve the total size of the transactions (more concretely "script complexity" defined below) used in secure computation with penalties for single stage computations. Let f denote the function being computed. Prior work by Bentov and Kumaresan [9] required transactions of total size $O(n^2|z|)$ where n is the number of parties and $|z|$ is the size of the output of f. We present a protocol that requires transactions of total size $O(n\lambda)$ where λ is the security parameter, and is independent of f. This protocol works in the programmable random oracle model.
- We improve the number of transactions and the total size of the transactions used in secure computation with penalties for multiple stage computations. Prior scheme by Kumaresan et al. [22] had the number of transactions grow quadratically in the number of parties n. We present a protocol whose number of transactions grows linearly in n. In particular, this implies that the n-party poker protocol (among other applications) as described in [22] can be implemented using only linear number of transactions.

Additionally, we present the following qualitative improvement to secure computation with penalties.

- We present a "dual mode" protocol (alternatively, "best-of-both-worlds" protocol) that offers different guarantees depending on the number of corrupt parties: (1) when $s < n/2$

CCS'16, October 24 - 28, 2016, Vienna, Austria

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4139-4/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2976749.2978421

[1]Fairness guarantees that if one party receives output then all parties receive output. Guaranteed output delivery ensures that an adversary cannot prevent the honest parties from computing the function.

parties are corrupt, this protocol guarantees fairness (i.e., either all parties get the output or none do), and (2) when $t > n/2$ parties are corrupt, this protocol guarantees fairness with penalties (i.e., if the adversary gets the output, then either the honest parties get output as well or they get compensation via penalizing the adversary). The above protocol works as long as $t + s < n$, matching the bound obtained for secure computation protocols in the standard model (i.e., replacing "fairness with penalties" with "security-with-abort" (full security except fairness)) by [18].

Next, we discuss the model and efficiency metrics for *secure computation with penalties*. We follow the model used in [9, 21, 22] where parties are assumed to have access to the "claim-or-refund" transaction functionality (discussed below). This functionality can be implemented in Bitcoin (subject to limitations discussed below) or in Ethereum.

Claim-or-refund transaction functionality. In [9, 22], protocols for secure computation with penalties are designed in a hybrid model where parties have access to an ideal transaction functionality called the *claim-or-refund transaction functionality* [9, 6, 26]. This functionality, denoted as $\mathcal{F}^*_{\mathrm{CR}}$, takes care of handling "money/coins" and allows protocols to be designed independently of the Bitcoin ecosystem. $\mathcal{F}^*_{\mathrm{CR}}$ implements the following functionality: (1) it accepts a deposit of $\mathsf{coins}(q)$, a Boolean circuit ϕ, and a time-limit τ from a designated sender S; and (2) waits until time τ to get a witness w from a designated receiver R such that $\phi(w) = 1$; and (3) if such a witness was received within time τ transfers $\mathsf{coins}(q)$ to R; (4) else returns $\mathsf{coins}(q)$ back to S.

Three features of $\mathcal{F}^*_{\mathrm{CR}}$ explain its importance: (1) $\mathcal{F}^*_{\mathrm{CR}}$ can be very efficiently implemented in Bitcoin [9, 6, 26] or in Ethereum, (2) $\mathcal{F}^*_{\mathrm{CR}}$ provides an abstraction which makes protocols designed in the $\mathcal{F}^*_{\mathrm{CR}}$-hybrid model robust to changes in the Bitcoin architecture, and (3) $\mathcal{F}^*_{\mathrm{CR}}$ is "complete" for secure computation with penalties [9, 22]. Protocols for secure computation with penalties designed in the $\mathcal{F}^*_{\mathrm{CR}}$-hybrid model work as long as $\mathcal{F}^*_{\mathrm{CR}}$ is implemented. Such an implementation need not be tied to Bitcoin, i.e., Bank of America, Paypal, etc. could, in principle, support $\mathcal{F}^*_{\mathrm{CR}}$ transactions. Each of the latter provides services by relying on its own network for providing consistency of its "ledger" and at this level, the underlying mechanics is not very different from Bitcoin.

Practical relevance of our contributions. Next, we discuss the cost of secure computation with penalties in the $\mathcal{F}^*_{\mathrm{CR}}$-hybrid model so that our contributions can be better understood. A protocol for secure computation with penalties in the $\mathcal{F}^*_{\mathrm{CR}}$-hybrid model typically involves an *sequence* of $\mathcal{F}^*_{\mathrm{CR}}$ transactions. The following metrics capture the costs of such a protocol:

- *The total number of calls to* $\mathcal{F}^*_{\mathrm{CR}}$. This captures the number of Bitcoin transactions that need to be broadcasted. Recall that each transaction stays in the blockchain forever.
- *The maximum/total size of the Boolean circuits ϕ employed in the sequence.* We refer to this as the "script complexity" of the protocol since this goes inside the script of the $\mathcal{F}^*_{\mathrm{CR}}$ Bitcoin transaction. Script complexity captures the load on the Bitcoin network and the verification time for the miners and SPV nodes. Larger ϕ also translates to larger scripts and consequently larger transactions and transmitting them across the network would also become a bottleneck.[2]
- *The maximum/total amount of deposits (i.e., money) made to* $\mathcal{F}^*_{\mathrm{CR}}$. This captures the amount of collateral that each party needs to input during the protocol (and will remain unusable till the protocol completes). Recall that each honest party would regain its deposit at the end of the protocol.
- *The maximum time-limit τ used in the sequence.* This captures the total time to completion for the protocol. Recall that each transaction takes roughly 1 hour to be confirmed on the Bitcoin blockchain.

[2]We denote scripts as circuits (not as RAM programs) and their size is proportional to their running time.

Our goal is to minimize these costs as much as possible. We refer to the first and second costs as "on-chain" costs since these costs are shared by the Bitcoin miners and affect the Bitcoin system as a whole. See also the discussion in [25]. We refer to the third and the fourth costs as "off-chain" costs since these are costs borne by the parties running the secure computation protocol. Note that the above arguments are valid for any other alt-coin as well. Anyway, in this work, we focus on improving the "on-chain" costs since we believe these to be the major bottleneck. Our protocols make heavy use of secure computation protocols but these are done off-chain (except in our construction for the reactive setting), so their costs are only shared between the participants and not by the Bitcoin system. Finally, we note that in the non-reactive case our results provide a improvement in the script complexity over [9] while keeping all other parameters equal. On the other hand, in the reactive case, our results improve the script complexity relative to [22] but perform worse in the total/max deposit expected. While [22]'s deposits grow linearly in the number of stages of the reactive protocol, ours grows quadratically. (Note that the remaining parameters are equal in both constructions.) Note that this is consistent with our motivation of relieving the burden on the Bitcoin system and off-loading more burden on the specific participants of the secure computation with penalties protocol.

Important notes and caveats about the model. Our model is essentially the standard model used in secure computation literature except that we allow working in the $\mathcal{F}^*_{\mathrm{CR}}$-hybrid model. (More formal description of the model can be found in the next section and also in [9].) While this model is Bitcoin-inspired, it is Bitcoin-independent. Currently, there are several important limitations about implementing $\mathcal{F}^*_{\mathrm{CR}}$ in Bitcoin. For instance, the scripts that can go inside a Bitcoin transaction (specifically, the value ϕ in an $\mathcal{F}^*_{\mathrm{CR}}$ transaction) are very limited—not all scripts are currently supported. There are also ongoing issues about malleability of transactions and how it affects $\mathcal{F}^*_{\mathrm{CR}}$ implementation (see discussion in [3]). Newer and simpler implementations of $\mathcal{F}^*_{\mathrm{CR}}$ namely via `OP_CHECKLOCKTIMEVERIFY` have been suggested and accepted. The bottomline is that the Bitcoin code is highly volatile. This is main reason why we follow the model in [9, 21, 22] and work in an idealized model (i.e., by abstracting the $\mathcal{F}^*_{\mathrm{CR}}$ transaction as an ideal functionality) with the hope of providing techniques and results that are resistant to the frequent changes to the Bitcoin code. Furthermore, since the limitations in the Bitcoin realization are by no means fundamental (as evidenced by Ethereum that proposes to realize all types of transaction functionalities), our constructions also have practical value both in the Bitcoin system and also elsewhere. To summarize, our results on secure computation with penalties work on Bitcoin (or an alt-coin or using a bank/trusted party) as long as the underlying $\mathcal{F}^*_{\mathrm{CR}}$ transactions are implementable in Bitcoin (or the corresponding alt-coin or a bank/trusted party). At least one alt-coin, namely Ethereum, supports programmable contracts with no limitations on scripts and thus can be used to implement our protocols.

Related work. We discussed the relation between our work and the works of [9, 22]. The works of [5, 6] construct 2-party lottery protocols using Bitcoin scripts which essentially implement $\mathcal{F}^*_{\mathrm{CR}}$.

Other notable works which are not in the $\mathcal{F}^*_{\text{CR}}$ model include the works of [2, 1, 20, 19, 21]. The works of [20, 19] use a more powerful transaction functionality which implements a ledger/blockchain to implement "smart contracts" and fair secure computation (under the penalties notion). Hawk [20] also provides financial privacy which our protocols do not provide. Ethereum supports programmable transactions and smart contracts from scratch. We wish to emphasize that protocols constructed in the $\mathcal{F}^*_{\text{CR}}$-hybrid model can be easily cast into protocols in any of the above models. Also, we make an explicit distinction between the off-chain costs and the on-chain costs which is not always captured in other works. For instance, in Ethereum, the entire smart contract (or the function) is put on the blockchain, and in a naïve construction, every miner is involved in the computation of the function as well as the state changes associated with executing the contract. These are exactly the type of burdens on the miners that we are trying to relieve via use of (possibly expensive) off-chain mechanisms (e.g., secure computation).

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 $X \overset{c}{\equiv} Y$ 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 [9]. Note that the main difference from standard definitions of secure computation [14] is that (1) the model treats coins as atomic entities that are fungible and cannot be duplicated, (2) the adversary cannot produce/destroy coins; only the environment $\mathcal{Z}$ has such powers, and (3) the view of $\mathcal{Z}$ contains the distribution of coins.[3] 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 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 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 $\mathcal{Z}$,*

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

[3]Note that typically in the simulation, coins are being exchanged between the ideal functionality, the simulator, and the adversary that the simulator is simulating. By (1) and (2) above, it follows that the simulator cannot send coins to the ideal functionality that it had already sent to the adversary (and vice versa).

$\mathcal{F}^\star_f$ with session identifier sid, security parameter 1^λ, penalty amount q, running with parties $P_1, \ldots, P_n$, and 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.

- *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 $(z_1, \ldots, z_n) \leftarrow f(y_1, \ldots, y_n)$.
  - Send message $(\mathsf{output}, sid, ssid, \{z_s\}_{s\in C})$ to $\mathcal{S}$.
  - If $\mathcal{S}$ returns $(\mathsf{continue}, sid, ssid, H_{\mathsf{out}})$, then send $(\mathsf{output}, sid, ssid, z_r)$ to P_r for all $r \in H$, and send $(\mathsf{payback}, sid, ssid, \mathsf{coins}((h - |H_{\mathsf{out}}|)q))$ to $\mathcal{S}$, and send $(\mathsf{extrapay}, sid, ssid, \mathsf{coins}(q))$ to P_r for each $r \in H_{\mathsf{out}}$.
  - Else if $\mathcal{S}$ returns $(\mathsf{abort}, sid, ssid)$, send $(\mathsf{penalty}, sid, ssid, \mathsf{coins}(q))$ to P_r for all $r \in H$.

Figure 1: Secure (non-reactive) computation with penalties $\mathcal{F}^\star_f$.

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

Throughout this paper, we deal only with static adversaries. Also, unless otherwise stated, we deal with the case where a majority of parties are dishonest.

2.1 Ideal Functionalities

Secure computation with penalties—non-reactive case. Loosely speaking, our notion of fair secure computation 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.

Ideal functionality $\mathcal{F}^\star_f$ for the non-reactive case [9, 22, 21, 1]. See Figure 1 for a formal description. In the first phase, the functionality $\mathcal{F}^\star_f$ receives inputs for f from all parties. In addition, $\mathcal{F}^\star_f$ allows the ideal world adverary $\mathcal{S}$ 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, $\mathcal{F}^\star_f$ returns the deposit made by honest parties back to them. If insufficient number of coins are deposited, then $\mathcal{S}$ does not obtain the output, yet may potentially pay penalty to some subset of the honest parties. 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 $\mathcal{S}$.

Secure computation with penalties—reactive case. Loosely speaking, our notion of fair reactive secure computation guarantees:

- An honest party never has to pay any penalty.
- If a party aborts after the computation has started (even though it may not learn outputs of all stages of the reactive computation) and does not deliver output to honest parties, then *every* honest party is compensated.

$\mathcal{F}_f^\star$ with session identifier sid, security parameter 1^λ, penalty amount q, running with parties $P_1, \ldots, P_n$, and 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. Initialize $\mathsf{state}_0 := \mathsf{NULL}$ and $\mathsf{flag} := 1$. Note that $f = (f_1, \ldots, f_\rho)$ is a reactive function composed of ρ stages.

- *Deposit phase:* Wait to receive a message $(\mathsf{deposit}, sid, ssid, r, \mathsf{coins}(d)$ from P_r for all $r \in H$. Then wait to receive $(\mathsf{deposit}, sid, ssid, \mathsf{coins}(hq))$ from $\mathcal{S}$.
- *Computation phase:* For each $\ell = 1, \ldots, \rho$, do:
  - Wait to receive a message $(\mathsf{input}, sid, ssid, r, x_{\ell,r})$ from P_r for all $r \in H$.
  - If $\mathcal{S}$ sends $(\mathsf{abort}, sid, ssid, \{\mathsf{coins}(q_r)\}_{r\in H})$, send $(\mathsf{penalty}, sid, ssid, \mathsf{coins}(q + q_r))$ to P_r for all $r \in H$, set $\mathsf{flag} := 0$, and terminate phase.
  - Else if $\mathcal{S}$ sends $(\mathsf{input}, sid, ssid, \{x_{\ell,s}\}_{s\in C})$, set $x_\ell = (x_{\ell,1}, \ldots, x_{\ell,n})$.
  - Compute $(z_\ell, \mathsf{state}_\ell) \leftarrow f_\ell(x_\ell; \mathsf{state}_{\ell-1})$, and parse z_ℓ to obtain $(z_{\ell,1}, \ldots, z_{\ell,n})$.
  - Send $(\mathsf{output}, sid, ssid, \{z_{s,\ell}\}_{s\in C})$ to $\mathcal{S}$.
  - If $\mathcal{S}$ returns $(\mathsf{continue}, sid, ssid)$, then send $(\mathsf{output}, sid, ssid, z_{\ell,r})$ to P_r for all $r \in H$.
  - Else if $\mathcal{S}$ sends $(\mathsf{abort}, sid, ssid, \{\mathsf{coins}(q_r)\}_{r\in H})$, send $(\mathsf{penalty}, sid, ssid, \mathsf{coins}(q + q_r))$ to P_r for all $r \in H$, set $\mathsf{flag} = 0$, and terminate phase.

Figure 2: Secure (reactive) computation with penalties $\mathcal{F}_f^*$.

Ideal functionality $\mathcal{F}_f^\star$ for the reactive case [9, 22, 21, 1]. It is very similar to the description in Figure 1 except the simulator is required to produce $\mathsf{coins}(hq)$ up front in a deposit stage. Also, now the computation proceeds in stages, with each stage delivering output, and aborts after every stage are penalized. See Figure 2 for a formal description. Note that $f = (f_1, \ldots, f_\rho)$ is a reactive function composed of ρ stages. The variable state_i captures the state of the reactive computation and the variable flag indicates whether there has been an abort.

Ideal functionality $\mathcal{F}_{\rm CR}^\star$ [9, 6, 26]. This special ideal functionality has found tremendous application in the design of multiparty fair secure computation and lottery protocols [9]. We elaborate more on the definition of the ideal functionality $\mathcal{F}_{\rm CR}^*$ below. See Figure 3 for a formal description. At a high level, $\mathcal{F}_{\rm CR}^\star$ 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. Note that the satisfying witness is made *public* by $\mathcal{F}_{\rm CR}^\star$. Bitcoin, or alternatively a cryptocurrency that supports time-locks and scripts, can be used to realize $\mathcal{F}_{\rm CR}^*$ [9, 6, 26]. In the Bitcoin realization of $\mathcal{F}_{\rm CR}^*$, 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. We denote an $\mathcal{F}_{\rm CR}^\star$ transaction where sender P_s asks receiver P_r for a witness for a predicate ϕ in exchange for $\mathsf{coins}(q)$ with deadline round τ by:

$$P_s \quad \xrightarrow[q,\tau]{\phi} \quad P_r$$

$\mathcal{F}_{\rm CR}^\star$ 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 $\mathsf{deposit}$ 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 3: The special ideal functionality $\mathcal{F}_{\rm CR}^\star$.

We formally define script complexity of a protocol consisting of a sequence of $\mathcal{F}_{\rm CR}^*$ deposits.

DEFINITION 3 (SCRIPT COMPLEXITY [21]). *Let Π be a protocol among n parties $P_1, \ldots, P_n$ in the $\mathcal{F}_{\rm CR}^*$-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 $V_{\Pi,\Omega}$ denote the sum of all $|\phi|$ such that some honest party claimed an $\mathcal{F}_{\rm CR}^*$ transaction by producing a witness for ϕ during an execution of Π. Then the* script complexity *of Π, denoted V_Π, equals $\max_\Omega\,(V_{\Pi,\Omega})$.* $\diamondsuit$

Note: Often while denoting the script complexity of a scheme, we ignore $\mathrm{poly}(\lambda)$ factors and focus on the dependence on the number of parties n, the function f, and the round complexity r of an arbitrary n-party protocol. This is purely for the sake of clarity and to better present our improvements. Also, we extensively use the random oracle model. We chose to do this (even when standard model alternatives exist in some cases) for the sake of clarity, consistency, and to better present our main ideas. Throughout this paper, we denote the random oracles as Hash and Hash'. The latter is a programmable random oracle while the former is non-programmable. If security under standard assumptions is desired, then Hash can be replaced by an honest-binding commitment [13, 9, 22]. However, this substitution might affect some of our theorems (notably, the one related to compact ladder). Also, we focus on the SCC model and while our protocols do not require rewinding in the proofs wrt coins or cryptographic primitivies, there are subtle issues while trying to guarantee universal composability in the programmable random oracle model. See [10] for more extensive discussion.

Remark on cash distribution. The works of [2, 1, 22] gave protocols for secure cash distribution with penalties (SCD) in the two-party and multiparty settings. Although we do not discuss this in the paper due to space limitations, we note that the mechanism to add cash distribution to our reactive protocols is straightforward and thus, our protocols can be directly upgraded (using a generic compiler) to handle secure cash distribution with penalties. Since SCD models stateful reactive functionalities it provides a way to securely implement *smart contracts* (as discussed in [22, 20]) in a

Roof deposits. For each $j \in [n-1]$ simultaneously:

$$P_j \xrightarrow[q,\tau_n]{\phi_n} P_n \qquad (\mathsf{Tx}_{n,j})$$

Ladder deposits. For $i = n-1$ down to 1 (one-by-one):

$$P_{i+1} \xrightarrow[i\cdot q,\tau_i]{\phi_i} P_i \qquad (\mathsf{Tx}_i)$$

Figure 4: Deposits in the ladder mechanism [9].

decentralized setting, and consequently captures a wide variety of applications such as games, auctions, markets, etc.

Remark. We give the strongest possible theorem statements corresponding to our results. That is, we may present constructions in the random oracle model (for the sake of clarity and to avoid cumbersome notation) even when there a construction that is based on one-way functions. However our theorems will be stated as assuming the existence of one-way functions if that's indeed the case. Also, we present the planted ladder mechanism as one that takes an r-round protocol as a parameter and discuss efficiency in the text as a function of r. Since in order to implement reactive MPC with penalties it suffices to use a constant-round protocol, i.e., $r = O(1)$, our theorem statement ignores r when describing the efficiency. Likewise, sometimes we state our theorems as operating in the $\mathcal{F}_{\mathrm{OT}}$-hybrid model (i.e., with ideal oblivious transfer). Recall that OT implies MPC unconditionally.

3. THE LADDER MECHANISM

Since our efficiency improvements come from modifying the ladder mechanism of [9], we briefly describe this mechanism. Figure 5 contains a description of the ladder mechanism. Each party P_i enters the ladder mechanism holding some private input y_i. For the ladder mechanism to be useful to construct a protocol for secure computation with penalties, the predicates $\{\phi_i\}_{i\in[n]}$ (which are parameters to the ladder mechanism) need to satisfy the following two informal properties:

- **Extensibility.** It is possible to satisfy ϕ_{i+1} (and claim Tx_{i+1}) using y_{i+1} and a witness that satisfies ϕ_i.
- **Unforgeability.** It is computationally infeasible to satisfy ϕ_{i+1} given only $\{y_j\}_{j\neq i}$ and witnesses to $\{\phi_j\}_{j<i}$ (in particular without a witness to ϕ_i).

We now proceed to describe the ladder mechanism. The mechanism is a sequence of $\mathcal{F}^*_{\mathrm{CR}}$ deposits split into two phases. In the first phase, known as the *roof deposits*, all parties except P_n make an $\mathcal{F}^*_{\mathrm{CR}}$ deposit to P_n for an amount of $\mathsf{coins}(q)$ and with predicate ϕ_n. If all roof deposits are made, then parties enter the second phase, known as the *ladder deposits*. Here first P_n makes an $\mathcal{F}^*_{\mathrm{CR}}$ deposit to P_{n-1} for an amount of $\mathsf{coins}((n-1)q)$ and with predicate ϕ_{n-1}. Likewise, P_{n-1} makes an $\mathcal{F}^*_{\mathrm{CR}}$ deposit to P_{n-2} for an amount of $\mathsf{coins}((n-2)q)$ and with predicate ϕ_{n-2}. This continues all the way down to P_2 who makes an $\mathcal{F}^*_{\mathrm{CR}}$ deposit to P_1 for an amount of $\mathsf{coins}(q)$ and with predicate ϕ_1. The sequence of $\mathcal{F}^*_{\mathrm{CR}}$ deposits in the ladder mechanism are claimed in reverse. First, P_1 claims Tx_1. Then P_2 claims Tx_2, and so on until finally P_n claims all $\mathsf{Tx}_{n,j}$ for $j \neq n$. This sequence of claims is enabled by the *extensibility property* described above. Next, we briefly describe how aborts are typically handled in the ladder mechanism. If some party aborts during the *roof deposit phase*, then parties terminate immediately (and wait for $\mathcal{F}^*_{\mathrm{CR}}$ deposits to be refunded—this is guaranteed by the *unforgeability property* above), and in particular do not make any ladder deposits. If some party P_{i+1} did not make its $\mathcal{F}^*_{\mathrm{CR}}$ deposit to P_i in the *ladder deposit phase*, then (1) each P_j for $j \leq i$ does not make its ladder deposit, and (2) each P_j (for $j \neq n$) claims Tx_j iff P_{j-1} claimed Tx_{j-1}, and (3) P_n claims $\mathsf{Tx}_{n,j}$ for all $j \neq n$ iff P_{n-1} claimed Tx_{n-1}. We formally present the framework for the ladder mechanism that we described above in Figure 5. This framework takes as parameters a protocol Init, a set of predicates $\{\phi_i\}_{i\in[rn]}$, and procedures Extend and Recon.

The ladder mechanism is parameterized by a protocol Init, predicates $\phi_1,\ldots,\phi_n$, and procedures Extend and Recon.

Initialization. Parties $P_1,\ldots,P_n$ run Init with their respective inputs $x_1,\ldots,x_n$ to obtain respective outputs $y_1,\ldots,y_n$. If there is an abort in this step such that some parties did not obtain their outputs, then all parties terminate and output $\perp$.

Roof deposits. For each $j \in [n-1]$ simultaneously:

$$P_j \xrightarrow[q,\tau_n]{\phi_n} P_n \qquad (\mathsf{Tx}_{n,j})$$

Handling aborts. If a party P_j does not make an $\mathcal{F}^*_{\mathrm{CR}}$ deposit to P_n as above, then each party P_i terminates the protocol and wait to collect refund from $\mathsf{Tx}_{n,i}$ (for $i \neq n$).

Ladder deposits. For $i = n-1$ down to 1 (one-by-one):

$$P_{i+1} \xrightarrow[i\cdot q,\tau_i]{\phi_i} P_i \qquad (\mathsf{Tx}_i)$$

Handling aborts. If a party P_{i+1} does not make an $\mathcal{F}^*_{\mathrm{CR}}$ deposit to P_i as above, then (1) each party P_j for $j \leq i$ does not make its ladder deposit (i.e., Tx_{j-1}) and waits to collect refund from $\mathsf{Tx}_{n,j}$ (for $j \neq n$), and (2) each party P_j for $j > i$ continues on to the ladder claim phase.

Claims. P_1 claims Tx_1 using witness $\alpha_1 = \mathsf{Extend}(1,\perp;y_1)$. For $i = 1$ to $n-1$ (one-by-one), at time τ_i:

- If P_i claimed Tx_i, then let α_i be the witness satisfying ϕ_i. P_{i+1} computes $\alpha_{i+1} \leftarrow \mathsf{Extend}(i+1,\alpha_i;y_{i+1})$. If $i+1 \neq n$, then P_{i+1} claims Tx_{i+1} using witness α_{i+1}. If $i+1 = n$, then P_n claims $\mathsf{Tx}_{n,j}$ for all j using witness α_n.
- If P_i did not claim Tx_i, then P_{i+1} terminates the protocol and waits to collect refund from $\mathsf{Tx}_{n,i+1}$ if $i+1 \neq n$.

Output. If $\mathsf{Tx}_{n,j}$ was claimed by P_n for some j, then let α_n be the witness satisfying ϕ_n. Each party P_i outputs $z_i = \mathsf{Recon}(\alpha_n;y_i)$ and terminates the protocol. If no $\mathsf{Tx}_{n,j}$ was claimed, then each party outputs $\perp$.

Figure 5: Protocol framework for the ladder mechanism.

Usefulness for secure computation with penalties. Given the above, we briefly provide some intuition on why the ladder mechanism is useful for secure computation with penalties. Let $i+1 < n$. First, note that $\mathsf{Tx}_{n,i+1}$ locks up $\mathsf{coins}(q)$ belonging to P_{i+1}. Now, if Tx_i is claimed, then P_{i+1} would have lost $\mathsf{coins}(i\cdot q)$ to P_i. However, by the extensibility property, P_{i+1} will be able to claim Tx_{i+1} and obtain $\mathsf{coins}((i+1)\cdot q)$ from P_{i+2}. Claiming Tx_{i+1} will release the witness for the predicate ϕ_{i+1} to all other parties. The main assertion is thus the following: when a party P_{i+1} releases a witness for the predicate ϕ_{i+1}, then at this point it has neither gained nor lost coins. Now suppose, P_n (either corrupt or honest)

Protocol Init. Parties $P_1, \ldots, P_n$ with their MPC inputs $x_1, \ldots, x_n$ run an MPC protocol that

- computes $z \leftarrow f(x_1, \ldots, x_n)$;
- n-out-of-n secret shares z into $z_1, \ldots, z_n$;
- samples $\omega_1, \ldots, \omega_n$ at random from $\{0,1\}^\lambda$;
- for $j \in [n]$: computes $h_j = \mathsf{Hash}(z_j \| \omega_j)$;
- for $i \in [n]$: outputs $y_i = z_i \| \omega_i$ and $\{h_j\}_{j \in [n]}$ to party P_i.

Predicates. Note that the predicates have $\{h_j\}_{j\in[n]}$ hard-coded in them. Define ϕ_i as follows:

$$\phi_i\left(\beta_1 \| \cdots \| \beta_i; \{h_j\}_{j \le i}\right) = (\mathsf{Hash}(\beta_1) \overset{?}{=} h_1) \bigwedge \cdots \bigwedge (\mathsf{Hash}(\beta_i) \overset{?}{=} h_i)$$

Procedure $\mathsf{Extend}(i+1, \alpha_i; y_{i+1})$: Output $\alpha_{i+1} = \alpha_i \| y_{i+1}$.

Procedure $\mathsf{Recon}(\alpha_n; y_i)$: Parse α_n as $\beta_1 \| \cdots \| \beta_n$. Parse each β_i as $z'_i \| \omega'_i$. Output $\bigoplus_{j\in[n]} z'_j$.

Figure 6: Parameters to the ladder mechanism from [9].

does not reveal a witness to ϕ_n, then all the roof deposits $\mathsf{Tx}_{n,j}$ get refunded. Thus, the following is true: If

1. P_{i+1} released a witness for the predicate ϕ_{i+1}, and
2. P_n did not release a witness for the predicate ϕ_n by time τ_n,

then party P_{i+1} would ultimately stand to gain $\mathsf{coins}(q)$ at the end of the protocol.

Bentov-Kumaresan parameters for the ladder mechanism to get MPC with penalties for a non-reactive function f. Refer to Figure 6 for a description of the parameters for the ladder mechanism used in [9]. The protocol Init is an MPC among the parties that computes the function output and secret shares among the parties. Then it commits to the shares using additional randomness. (In the figure, we used a hash function Hash modeled as a random oracle. It can safely be replaced by honest-binding commitments as in [9].) The parties get commitments to all shares and the decommitment corresponding to their index. The predicates $\phi_{i_{i\in[n]}}$ are defined in the following way: ϕ_i has $\{h_j\}_{j\le i}$ hard-coded in it, and requires the hash preimages of these values. Extensibility is readily guaranteed (as shown in Figure 6), and unforgeability is guaranteed via the random oracle (alternatively, binding property of the commitment). Finally, the procedure Recon takes the witness to ϕ_n, i.e., all the shares, and XORs them to obtain the final output. This completes the description of the parameters for the ladder mechanism of [9] that yields MPC with penalties for a non-reactive function f. It is easy to see that the script complexity of the above protocol grows quadratic in n.

4. COMPACT LADDER

In this section, we improve the efficiency of secure computation with penalties in the non-reactive setting. Recall that the ladder protocol of [9] required script complexity $\Omega(n^2|z|)$ where $|z|$ is the size of the output of the non-reactive function. We are going to improve this to $O(n\lambda)$, and in particular make it completely independent of the function.

Protocol Init. Parties $P_1, \ldots, P_n$ with their MPC inputs $x_1, \ldots, x_n$ run an MPC protocol that

- samples random $k_1, \ldots, k_n$ from $\{0,1\}^\lambda$;
- for $j \in [n]$: computes $h_j = \mathsf{Hash}(k_1 \oplus \cdots \oplus k_j)$;
- computes $e \leftarrow f(x_1, \ldots, x_n) \oplus \mathsf{Hash}'(k_1 \oplus \cdots \oplus k_n)$;
- for $i \in [n]$: outputs $y_i = k_i$, $\{h_j\}_{j\in[n]}$, and e to party P_i.

Predicates. Note that the predicate ϕ_i has h_i hard-coded in it. Define ϕ_i as follows:

$$\phi_i\left(\alpha_i; h_i\right) = (\mathsf{Hash}(\alpha_i) \overset{?}{=} h_i)$$

Procedure $\mathsf{Extend}(i+1, \alpha_i; y_{i+1})$: Output $\alpha_{i+1} = \alpha_i \oplus y_{i+1}$.

Procedure $\mathsf{Recon}(\alpha_n; y_i)$: Output $e \oplus \mathsf{Hash}'(\alpha_n)$.

Figure 7: Compact ladder parameters.

First ideas—using hybrid encryption. A standard technique (also appearing in [16]) is to use an MPC to encrypt the actual output of the computation and then to secret share the keys among parties and allow to reconstruct the encryption key in a fair manner (in our case with the ladder mechanism). This has the immediate impact of reducing the script complexity to $O(n^2\lambda)$. In particular, the i-th predicate ϕ_i will have size $O(i\lambda)$ (ignoring $\mathrm{poly}(\lambda)$ factors). One caveat is that for our strong security notion, we will need to use a programmable random oracle in order to achieve equivocation properties in the simulation proof.[4] (In contrast, [16] only considered standalone security and could do away with semantic security of the encryption scheme.)

Compact ladder. To further reduce the script complexity to $O(n\lambda)$, we have a simple tweak to the parameters of the ladder mechanism. The main idea to have the size of the i-th predicate ϕ_i be $O(\lambda)$. To to do this, we run an MPC protocol in Init that generates secret shares $k_1, \ldots, k_n$ of a key k that is used to encrypt the output of the computation. The MPC protocol also generates hash images of the following form: $h_j = \mathsf{Hash}(k_1 \oplus \cdots \oplus k_j)$. We then define the predicates as $\phi_i(\alpha_i; h_i) = (\mathsf{Hash}(\alpha_i) \overset{?}{=} h_i)$. That is, the predicate ϕ has the value h_i hard-coded in it and only takes a λ-bit input. This is the main idea. A formal description of the parameters appears in Figure 7.

Clearly, the script complexity of the resulting protocol, i.e., the sum of the sizes of all predicates, is $O(n\lambda)$. It remains to be shown that the predicates satisfy the extensibility as well as the unforgeability property. Extensibility follows from the fact that given a witness α_i to the predicate ϕ_i and the secret value k_{i+1}, the witness to the predicate ϕ_{i+1} can be obtained as $\alpha_i \oplus k_{i+1}$. Unforgeabiility follows from the use of the random oracle. More concretely, given the values $k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_n$ and the hash values $\mathsf{Hash}(k_1), \ldots, \mathsf{Hash}(k_1 \oplus \cdots \oplus k_n)$, it is computationally infeasible to find a hash preimage of any of $\mathsf{Hash}(k_1 \oplus \cdots \oplus k_i), \ldots, \mathsf{Hash}(k_1 \oplus \cdots \oplus k_n)$ or even distinguish them from random. In the formal simulation proof we also need to argue about the timing of various events (e.g., interaction with the ideal functionality, the adversary, etc.), distribution of coins, and

[4]Alternatives in the standard model include use of non-committing encryption but here we would not be able to get the desirable efficiency [27].

equivocation of the final output. We state our theorem and provide a proof sketch below and defer the full proof to the full version.

THEOREM 1. *Assuming the existence of a programmable random oracle, there exists a protocol in the $(\mathcal{F}_{\text{OT}}, \mathcal{F}^*_{\text{CR}})$-hybrid model that* SCC-realizes $\mathcal{F}^*_f$ *for any non-reactive function* f. *Furthermore, the protocol makes only* $O(n)$ *calls to* $\mathcal{F}^*_{\text{CR}}$ *and has script complexity* $O(n\lambda)$.

Proof sketch. The simulator $\mathcal{S}$ invokes the simulator of the MPC protocol in the initialization phase and generates a random string e as the encryption, random $k_1, \ldots, k_n$ and distributes hash images on these values to the adversary $\mathcal{A}$ and also gives the value k_i to each corrupt P_i. $\mathcal{S}$ also extracts inputs of the corrupt parties in the previous step. However, it does not directly send these inputs to the ideal functionality $\mathcal{F}^*_f$ since it may also needs to extract and submit $\mathsf{coins}(hq)$ to the ideal functionality (especially in the case when P_n is corrupt). Acting as $\mathcal{F}^*_{\text{CR}}$, $\mathcal{S}$ simulates the deposit phases of the protocol. Note that some of the deposits might not be made by $\mathcal{A}$. $\mathcal{S}$ emulates the honest parties to $\mathcal{A}$ exactly as described in the protocol (including how they react to deposits that were not made). When the claim phase starts and it's the turn of an honest party P_j to claim Tx_j, $\mathcal{S}$ will use the value $k_1 \oplus \cdots \oplus k_j$. It also receives corrupt parties' claim witnesses via $\mathcal{A}$. Additionally, during the claim phase, $\mathcal{S}$ will need to produce coins whenever $\mathcal{A}$ claims them. Suppose P_i is corrupt and it produces a valid witness α to claim Tx_i. (For the time being, assume $i \neq n$; we'll take care of this case later.) To simulate this, $\mathcal{S}$ needs to produce $\mathsf{coins}(i \cdot q)$.

Ideally, we would ask $\mathcal{S}$ to get these coins via Tx_{i-1}, i.e., $\mathsf{coins}((i-1)q)$ and $\mathsf{coins}(q)$ from P_i's roof deposit $\mathsf{Tx}_{n,i}$. However, it is not clear that Tx_{i-1} was even made by P_i. Importantly, we are able to make the following claim: if Tx_{i-1} was not made by P_i but P_i can produce valid witnesses to claim Tx_i, then for every $j < i$, party P_j is corrupt. To prove the claim, we appeal to the unforgeability property of the predicates in the following way. Note that according to the protocol description, if Tx_{i-1} was not made, then for $j < i$ an honest party P_j would not make Tx_{j-1}. Since it never made a deposit Tx_{j-1}, this in turn means that it would never reveal a valid witness for Tx_j (note: honest P_j claims Tx_j iff Tx_{j-1} is claimed). Then it follows from the unforgeability property that without a valid witness for Tx_j it is computationally infeasible to provide a valid witness to Tx_i. Now we are ready to describe the simulation of the claims $\mathsf{Tx}_1, \ldots, \mathsf{Tx}_i$ (whichever were made). For transactions Tx_j with $j < i$ that were claimed, we simply use corrupt P_{j+1}'s deposit of $\mathsf{coins}(jq)$ to give to corrupt P_j who's claiming it. For transaction Tx_i, we use $\mathsf{coins}(q)$ from the roof deposit of each of $P_1, \ldots, P_i$. Recall P_{i+1} is honest, and if P_{i+1} made deposit Tx_i it must necessarily hold that all parties (in particular, $P_1, \ldots, P_i$) made roof deposits of $\mathsf{coins}(q)$ each. The above strategy takes care of parties that try to claim a deposit Tx_i without make the deposit Tx_{i-1}. In the rest of the proof, we assume that every P_i that tries to claim deposit Tx_i has made the deposit Tx_{i-1}. This case is handled easily: we ask $\mathcal{S}$ to get these coins via Tx_{i-1}, i.e., $\mathsf{coins}((i-1)q)$ and $\mathsf{coins}(q)$ from P_i's roof deposit $\mathsf{Tx}_{n,i}$ (out of a total of $\mathsf{coins}(q)$). The above strategy works only for $i \neq n$. We first complete the discussion assuming P_n is honest and then discuss the other case later. Suppose P_n is honest. We have already addressed how to handle coins during the claims. The only remaining thing to address is when to ask $\mathcal{S}$ to contact $\mathcal{F}^*_f$. Let i_c be the largest index such that P_{i_c} is corrupt. $\mathcal{S}$ will contact the ideal functionality when Tx_{i_c} is claimed by P_{i_c} since it is clear that $\mathcal{A}$ will no longer abort the protocol. $\mathcal{S}$ would then get the actual output z from the ideal functionality, and program Hash' to return $z \oplus e$ (where e is the random string $\mathcal{S}$ distributed to the parties initially). Note that this is indistinguishable from the real execution as long as $\mathcal{A}$ did not query on the string $k_1 \oplus \cdots \oplus k_n$—an event that happens with negligible probability. This completes the simulation when P_n is honest.

When P_n is corrupt, we need to show how to handle claims made by P_n and also how to handle the interaction with the ideal functionality. Actually, when Tx_{n-1} is deposited by P_n, $\mathcal{S}$ will take $\mathsf{coins}(hq)$ out of these $\mathsf{coins}((n-1)q)$ (note: $(n-1) \geq h$) to deposit to the ideal functionality along with the inputs to the computation (extracted in the very first step). $\mathcal{S}$ would then get the actual output from the ideal functionality. Let i_h be the largest index such that P_{i_h} is honest. $\mathcal{S}$ will actually do the above steps of giving $\mathsf{coins}(hq)$ to $\mathcal{F}^*_f$ right before time τ_{i_h} so that it can obtain the actual output z and program Hash' such that $\mathsf{Hash}'(k_1 \oplus \cdots \oplus k_n) = e \oplus z$. This ensures that remaining honest messages are such that the view of the adversary is indistinguishable from the real execution. Also, $\mathcal{S}$ would ask $\mathcal{F}^*_f$ to deliver the output to all honest parties (because the corrupt parties did not abort the computation at all) and would get $\mathsf{coins}(hq)$ back from the ideal functionality which it can then use to handle P_n's claim of $\mathsf{Tx}_{n,j}$ for all j since it now has exactly $\mathsf{coins}((n-1)q)$. This completes the simulation when P_n is honest. $\square$

5. PLANTED LADDER MECHANISM

In this section, we improve the efficiency of secure computation with penalties in the reactive setting. In the reactive setting, we have to deal with a multi-round protocol and guarantee its completion or guarantee compensation to all honest parties. Previous work by Kumaresan et al. [22] showed the "see-saw protocol" for the reactive setting which is essentially an adaptation of the ladder mechanism of [9] but that guarantees that all honest parties are compensated no matter where the abort happens. However, to do this, they required $\Omega(n^2)$ $\mathcal{F}^*_{\text{CR}}$ transactions and a script complexity of $\Omega(n^2 T)$ where T is the size of the transcript of an n-party secure computation protocol that implements the reactive function f with security-with-abort. We are going to improve both the script complexity to $O(nT)$ and the number of $\mathcal{F}^*_{\text{CR}}$ calls to $O(n)$.

First ideas—tweaking the ladder mechanism. The main idea behind the improvement is a simple set of tweaks to the ladder mechanism that preserves the structure of the mechanism and yet suffices for the reactive setting. We first provide a rough sketch of the idea, then discuss the problem with the approach, and then proceed to give the fix.

Note that since we have to handle multi-round protocols we let parties take turns to send messages until the entire protocol is completed. More concretely, suppose we have an r-round protocol π, then we essentially have r copies of the original ladder mechanism of [9]. The k-th such copy (from the bottom) corresponds to the k-th round messages (there are n of them, one from each party) of π. It is important to note that in the k-th copy of the ladder mechanism, party P_{i+1} makes a deposit of $\mathsf{coins}((n(k-1)+i)q)$. (Note how when $k = 1$, this is the same as the ladder mechanism in [9].) That is, the amount deposited keeps growing across the r copies of the ladder eventually ending in the last deposit of $\mathsf{coins}((nr-1)q)$.

The best way to understand the planted ladder mechanism is to view the whole deposit sequence as a *single* ladder mechanism among rn virtual parties $P'_1, \ldots, P'_{rn}$, where $P'_i \equiv P_{i \bmod n}$. The main observation is that the ladder mechanism already guarantees compensation to parties who have made their move (i.e., revealed their share or the next message of the protocol) in case the protocol is aborted. Thus, if the protocol is aborted *after* the first copy of the ladder deposits have been claimed (i.e., the first round of the pro-

Roof deposits. For each $j \in [n-1]$ simultaneously:

$$P_j \xrightarrow[(r+1)\cdot q, \tau_{(r+1)n}]{\phi_{rn} \wedge \phi'_n} P_n \qquad (\mathsf{Tx}_{n,j})$$

Ladder deposits. For $i = rn - 1$ down to 1 (one-by-one):

$$P_{i+1 \bmod n} \xrightarrow[(n+i)\cdot q, \tau_{n+i}]{\phi_i \wedge \phi'_n} P_{i \bmod n} \quad (\mathsf{Tx}_{n+i})$$

Underground deposits. For $i = n$ down to 1 (one-by-one):

$$P_{i+1 \bmod n} \xrightarrow[i \cdot q, \tau_i]{\phi'_i} P_i \qquad (\mathsf{Tx}_i)$$

Figure 8: Deposits in the planted ladder mechanism.

tocol has been completed), it follows that each honest party would be compensated! In fact, by the time the k-th copy of the ladder deposits are claimed, each honest party would have $\mathsf{coins}(kq)$ as compensation in the case the protocol is aborted in this phase. (Note that each party P_j for $j \neq n$ would have deposited $\mathsf{coins}(rq)$ as the roof deposit. That is, if the protocol is completed, then no party gains or loses money.)

The problem with the idea above is that aborts within the first copy of the ladder mechanism will not guarantee compensation to all honest parties (i.e., not all parties would have gotten $\mathsf{coins}(q)$ as compensation).

Dummy witnesses and underground deposits. To fix this problem, we use a simple idea: have the first copy of the ladder mechanism reveal *dummy witnesses* which are independent of the actual computation. That is, now there are $(r+1)$ copies of the ladder mechanism: the first copy is a special set of transactions that we call "underground" deposits, while the remaining r of them correspond to the actual r-round protocol. The deposits are shown in Figure 8.

The underground deposits essentially serve to bootstrap the computation. Aborts during the underground claim phase may not compensate all honest parties, but this is not a problem since the messages corresponding to the actual protocol have not been sent by anyone. This amounts to aborting the protocol even before it started and is still a fair protocol since neither the honest parties nor the adversary obtain any information about the actual protocol. Additional ideas ensure that the underground deposits will be claimed before any of the deposits in the r copies of the ladder are claimed. This ensures that aborts in any of the r copies will result in each honest party being compensated by the adversary.

A framework for the planted ladder mechanism. We present a simple framework for the planted ladder mechanism in Figure 9. Just like the framework for the ladder protocol, this framework takes as parameters a protocol Init, a set of predicates $\{\phi_i\}_{i\in[rn]}$, and procedures Extend and Recon. As before we will crucially rely on the extensibility and unforgeabilty property of the predicates $\{\phi_i\}_{i\in[rn]}$ (and also of the underground predicates $\{\phi'_i\}_{i\in[n]}$). In the initialization phase, parties run an MPC protocol that sets up the dummy witnesses $k_1, \ldots, k_n$ and computes the Hash on XORs of the prefixes of the sequence of dummy witnesses. This part is similar to our compact ladder mechanism. (Again, one can replace Hash with an honest-binding commitment as in [22].) Each party P_i receives the value k_i and the images of $k_1, \ldots, k_n$ under Hash. In addition parties run Init which is a parameter of the planted ladder mechanism. The dummy witnesses define the underground predicates ϕ'_n as: $\phi'_i(\alpha_i; h_i) = (\mathsf{Hash}(\alpha_i) \overset{?}{=} h_i)$. After this, parties enter the three deposit phases: Roof, Ladder, and $\mathsf{Underground}$.

We already mentioned that the predicates for the underground deposits are $\phi'_1, \ldots, \phi'_n$. The predicates for the Roof and Ladder deposits are $\phi_j \wedge \phi'_n$ for $j \in [rn]$. The extra predicate ϕ'_n is to ensure that the underground deposits are claimed before any of the ladder/roof deposits are claimed. Other than these, there are essentially no differences between the ladder mechanism and the planted ladder mechanism. One thing to note is that we sometimes interpret Recon as a protocol (instead of a procedure). This will be relevant to the dual mode protocol that we describe in the next section.

Intuition. The best way to understand the planted ladder mechanism is to view the whole deposit sequence as a *single* ladder mechanism among $(r+1)n$ virtual parties $P'_1, \ldots, P'_{(r+1)n}$, where $P'_i \equiv P_{i \bmod n}$.

PROPOSITION 2. *Honest parties do not lose money.*

Proof sketch. This actually follows from the extensibility property of the predicates $\{\phi'_i\}$ and $\{\phi_i\}$. More concretely, every time an honest party's ladder/underground deposit, say Tx_j is claimed, the honest party $P_{j+1 \bmod n}$ will always be able to claim Tx_{j+1}. Since claiming Tx_{j+1} gives $\mathsf{coins}((j+1)q)$ to $P_{j+1 \bmod n}$ and since it would have lost only $\mathsf{coins}(jq)$ when $P_{j \bmod n}$ claimed Tx_j, it follows that $P_{j+1 \bmod n}$ never loses money during the ladder claim phase i.e., before time $\tau_{(r+1)n-1}$.

Now suppose there is some Tx_{j+1} which $P_{j+1 \bmod n}$ is unable to claim. Then by the *unforgeability* property of the predicates it holds that transactions Tx_k for $k > j+1$ cannot be claimed by $P_{k \bmod n}$. In particular, the roof deposit $\mathsf{Tx}_{n,j+1}$ cannot be claimed. On the other hand, if $P_{j+1 \bmod n}$ was able to claim all deposits made to it, then it follows that it would have gained $\mathsf{coins}((r+1)q)$ during the course of the claims (i.e., $\mathsf{coins}(q)$ for each of the $r+1$ stages). This is also the maximum amount that it can lose during the roof claim by P_n. This therefore means that honest parties will never lose money. $\square$

PROPOSITION 3. *Suppose for some i, party $P_{i \bmod n}$ is honest and revealed a witness to the predicate ϕ_i during the execution of the protocol. Then either the protocol is completed (i.e., P_n reveals a witness to the predicate ϕ_{rn}) or all honest parties get compensated.*

Proof sketch. By the unforgeability property of $\{\phi'_i\}_{i\in[n]}$, it follows that an honest party P_i reveals a witness to ϕ_i only if each honest party P_j revealed k_j in the underground claim phase. This is because for a party to claim the ladder deposit it needs to produce a witness that satisfies ϕ'_n which is possible only if each honest P_j revealed k_j. For the rest of the argument, we assume that all honest parties revealed the underground dummy witnesses.

Now suppose there is an abort. Let $P_{i+1 \bmod n}$ be an honest party such that (1) all honest parties P_j with $j \bmod n \leq i \bmod n$ could claim Tx_j for $j < i$, but (2) P_{i+1} could not claim Tx_{i+1}. From the above it follows that party P_i must be corrupt. Also by the unforgeability property, we have that Tx_j for $j > i$ cannot be claimed by $P_{j \bmod n}$ since honest $P_{i+1 \bmod n}$ did not reveal its witness α_{i+1}. In particular, this means that the roof deposits will be refunded back to all honest parties. Also all deposits Tx_j made by honest $P_{j \bmod n}$ such that $j > i$ will also be refunded back to P_{j+1}. Finally, recall that all honest parties were able to reveal their dummy witnesses, therefore it must hold that $i+1 > n$.

Now let us analyze how much money each honest party possesses at the end of the protocol. If $P_{j+1 \bmod n}$ is honest and

The planted ladder mechanism is parameterized by a protocol Init, predicates $\{\phi_i\}_{i\in[rn]}$, procedures Extend and Recon.

Note: In the following: $P_i \equiv P_{i \bmod n}$. Also: $n \bmod n \triangleq n$.

Initialization. Parties $P_1, \ldots, P_n$ run an MPC protocol that

- samples random $k_1, \ldots, k_n$ from $\{0,1\}^\lambda$;
- for $j \in [n]$: computes $h_j = \mathsf{Hash}(k_1 \oplus \cdots \oplus k_j)$;
- for $i \in [n]$: outputs $\{h_j\}_{j\in[n]}$ and k_i to P_i.

In addition, parties $P_1, \ldots, P_n$ run Init with their respective inputs $x_1, \ldots, x_n$ to obtain respective outputs $y_1, \ldots, y_n$. If there is an abort (either in Init or the protocol above) and some parties did not obtain their outputs, then all parties terminate and output $\bot$. The predicates ϕ'_i have h_i hard-coded in them:

$$\phi'_i\,(\alpha_i; h_i) = (\mathsf{Hash}(\alpha_i) \overset{?}{=} h_i)$$

Roof deposits. For each $j \in [n-1]$ simultaneously:

$$P_j \xrightarrow[(r+1)\cdot q,\,\tau_{(r+1)n}]{\phi_{rn}\wedge\phi'_n} P_n \qquad (\mathsf{Tx}_{n,j})$$

Handling aborts. If P_j does not make deposit $\mathsf{Tx}_{n,j}$, then each P_i terminates the protocol and waits to collect refund.

Ladder deposits. For $i = rn - 1$ down to 1 (one-by-one):

$$P_{i+1} \xrightarrow[(n+i)\cdot q,\,\tau_{n+i}]{\phi_i\wedge\phi'_n} P_i \qquad (\mathsf{Tx}_{n+i})$$

Handling aborts. If P_{i+1} does not make deposit Tx_{n+i}, then each P_j terminates the protocol and waits to collect refunds.

Underground deposits. For $i = n$ to 1 (one-by-one):

$$P_{i+1} \xrightarrow[i\cdot q,\,\tau_i]{\phi'_i} P_i \qquad (\mathsf{Tx}_i)$$

Handling aborts. If P_{i+1} does not make deposit Tx_i, then (1) each P_{j+1} for $j < i$ does not make deposit Tx_j and waits to collect refunds, while (2) each P_{j+1} for $j \geq i$ continues on to the claim phase.

Underground claims. P_1 claims Tx_1 using witness $\alpha'_1 = k_1$. For $i = 1$ to $n-1$ (one-by-one), at time τ_i:

- If P_i claimed Tx_i, then let α'_i be the witness satisfying ϕ'_i. Then P_{i+1} claims Tx_{i+1} using witness $\alpha'_i \oplus k_{i+1}$.
- If P_i did not claim Tx_i, then each P_{j+1} terminates the protocol and waits to collect refunds.

Ladder/roof claims. Let $X_i = (x_i, \omega_i, y_i)$ where x_i, y_i respectively are inputs, outputs of P_i from Init and ω_i is P_i's private randomness. For $i > n$, let $X_i \equiv X_{i \bmod n}$. P_1 claims Tx_{n+1} using witness $\alpha_1 \leftarrow \mathsf{Extend}(1, \bot, X_1)$ at time τ_{n+i}:

- If P_i claimed Tx_i say using witness α_i, then P_{i+1} computes $\alpha_{i+1} = \mathsf{Extend}(i+1, \alpha_i, X_{i+1})$ If $i + 1 \neq rn$, then P_{i+1} claims Tx_{i+1} using witness $(\alpha_{i+1}, \alpha'_n)$, else P_{i+1} claims $\mathsf{Tx}_{n,j}$ for all j using witness (α_{rn}, α'_n).
- If P_i did not claim Tx_i, then P_{i+1} terminates the protocol and waits to collect refunds.

Output. Parties run protocol Recon to generate output.

Figure 9: Framework for the planted ladder mechanism.

its ladder deposit Tx_j is claimed, then by the extensibility property $P_{j+1 \bmod n}$ can claim Tx_{j+1}. The amounts in the mechanism are such that at time τ_{j+1}, P_{j+1} would have an additional $\mathsf{coins}((j+1)q) - \mathsf{coins}(jq) = \mathsf{coins}(q)$ for every Tx_j it claimed, i.e., an additional $\mathsf{coins}(q)$ for every completed round. Since (1) at least one round was completed (note: $i + 1 > n$), and (2) all deposits Tx_j for $j > i$ were refunded, it follows from above that each honest party gets at least $\mathsf{coins}(q)$ as compensation. $\square$

Parameterizing the planted ladder mechanism to get MPC with penalties for a reactive function f. Let π' be an n-party r-round publicly verifiable MPC protocol realizing f with security-with-abort. Let π' be described by next message functions $\{\mathsf{nmf}'_j\}_{j\in[rn]}$ and transcript validation functions $\{\mathsf{tv}'_j\}_{j\in[rn]}$. We would want to derive the parameters Extend, Recon and the predicates ${\phi_\mathsf{i}}_{i\in[n]}$ from the protocol π'. The predicates ${\phi_\mathsf{i}}_{i\in[n]}$ could be defined in terms of the transcript validation functions. While extensibility is readily guaranteed via the next message function, the unforgeability property is not always clear (especially with respect to first round messages). However, adding the unforgeability property to π' is possible via a simple transformation.

The idea is to ask parties to sign every message sent as part of π' under their public key. That is, the transcript TT_j of the transformed protocol consisting of the first j messages would be of the form $(\mu_1 \| \sigma_1) \| \cdots \| (\mu_j \| \sigma_j)$, where for each i, the value μ_i is the message that would have been sent in π' and the value σ_i is a signature under party $P_{i \bmod n}$'s public key on the message μ_i. We denote the transformed protocol as π. It is easy to see that Extend can be defined in terms of the next message function of π. Likewise, Recon may be defined as the procedure by which parties generate the output in the protocol π. Also, we define Init as the protocol in which parties generate fresh public-key/signing-key pairs and distribute them via a local broadcast channel (i.e., among the n parties). See Figure 10 for a formal description of the parameters.

Note that we have abstracted away several features of f and focus on it as one single function for which we need to guarantee protocol termination. We already gave intuition as to why the planted ladder mechanism ensures protocol completion as long as the predicates $\{\phi_i\}_{i\in[n]}$ satisfy both the extensibility and the unforgeability properties. In the formal simulation proof we also need to argue about the timing of various events (e.g., interaction with the ideal functionality, the adversary, etc.), distribution of coins, and equivocation of the final output. We state our theorem and provide a proof sketch below and defer the full proof to the full version.

THEOREM 4. *Assume the existence of enhanced trapdoor permutations. Let f be a reactive function that has a constant-round protocol UC-realizing it with security-with-abort, the size of whose transcript is T. Then there exists a protocol in the $\mathcal{F}^*_{\mathrm{CR}}$-hybrid model that* SCC*-realizes $\mathcal{F}^*_f$ which requires only $O(n)$ calls to $\mathcal{F}^*_{\mathrm{CR}}$ and whose script complexity is $O(nT)$.*

Proof sketch. The simulator $\mathcal{S}$ acts as $\mathcal{F}^*_{\mathrm{CR}}$ and also uses the simulator $\mathcal{S}_\pi$ of the constant-round protocol π (after undergoing the transformation described above). $\mathcal{S}$ invokes the simulator of the MPC protocol in the initialization phase and generates random $k_1, \ldots, k_n$ and distributes commitments (or hash images) to the adversary $\mathcal{A}$ and also gives the value k_i to each corrupt P_i. $\mathcal{S}$ then invokes the simulator of Init and distributes fresh public-keys belonging to honest parties and receives the public-keys of the corrupt parties from $\mathcal{A}$. Then acting as $\mathcal{F}^*_{\mathrm{CR}}$, $\mathcal{S}$ emulates the three deposit phases of the protocol. Note that some of the deposits might not be made by $\mathcal{A}$. $\mathcal{S}$ emulates the honest parties to $\mathcal{A}$ exactly as described

Protocol Init. Parties $P_1, \ldots, P_n$ each locally run KeyGen of a digital signature scheme $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ to generate fresh verification-key signing-key pairs. Let (vk_j, sk_j) be the key pair corresponding to P_j. Each P_j then broadcasts vk_j to all other parties. At the end, each P_j outputs the set of all public keys $\{vk_i\}_{i\in[n]}$ and its private signing key sk_j.

The parameters Extend, Recon, and $\{\phi_i\}_{i\in[rn]}$ are defined via the outputs of Init and a protocol π described below.

Protocol π. Let π' be an n-party r-round publicly verifiable MPC protocol realizing f (possibly a reactive function) with security-with-abort. Let π' be described by next message functions $\{\mathsf{nmf}'_j\}_{j\in[rn]}$ and transcript validation functions $\{\mathsf{tv}'_j\}_{j\in[rn]}$. Let $X'_i := (x_i, \omega_i)$ and $X_i := (x_i, \omega_i, (\{vk_j\}_{j\in[n]}, sk_i))$ where x_i, ω_i respectively are private inputs, randomness of P_i. For $i > n$, let $X'_i \equiv X'_{i \bmod n}$ and $X_i \equiv X_{i \bmod n}$. Let π, described by next message functions $\{\mathsf{nmf}_j\}_{j\in[rn]}$ and transcript validation functions $\{\mathsf{tv}_j\}_{j\in[rn]}$, be obtained from π' in the following way:

1. $\mathsf{nmf}_j(\mathtt{TT}_{j-1}, X_j)$ parses $\mathtt{TT}_{j-1} = (\mu_1\|\sigma_1)\|\cdots\|(\mu_{j-1}\|\sigma_{j-1})$ and computes $\mu_j \leftarrow \mathsf{nmf}'_j(\mu_1\|\cdots\|\mu_{j-1}, X'_j)$, and outputs $\mathtt{TT}_{j-1}\|\mu_j\|\mathsf{Sign}(\mu_j, sk_j)$.
2. $\mathsf{tv}_j(\mathtt{TT}_j, \{vk_i\}_i)$ parses $\mathtt{TT}_j = (\mu_1\|\sigma_1)\|\cdots\|(\mu_j\|\sigma_j)$, outputs $\mathsf{tv}'_j(\mu_1\|\cdots\|\mu_j) \wedge \bigwedge_{j'} \mathsf{Verify}(vk_{j' \bmod n}, \sigma_{j'})$.

Predicates. $\phi_i(\alpha) \equiv \mathsf{tv}_i(\alpha; \{vk_j\}_j)$.

Procedure $\mathsf{Extend}(i, \alpha; X_i) \equiv \mathsf{nmf}_i(\alpha, X_i)$.

Procedure Recon. If $\mathsf{Tx}_{n,j}$ was claimed for some j, then output the output of π', else output $\perp$.

Figure 10: Parameterizing the planted ladder mechanism to get MPC with penalties for a reactive function f.

in the protocol (including how they react to deposits that were not made).

When the claim phase starts and it's the turn of an honest party to send out a message in π, $\mathcal{S}$ invokes $\mathcal{S}_\pi$ to generate this message. It also receives corrupt parties' messages in π (via $\mathcal{A}$) and feeds them to $\mathcal{S}_\pi$. $\mathcal{S}$ essentially uses $\mathcal{S}_\pi$ to extract inputs of the corrupt parties. However, it does not directly send these inputs to the ideal functionality $\mathcal{F}^*_f$ since it also needs to extract and submit $\mathsf{coins}(hq)$ to the ideal functionality. Also, $\mathcal{S}$ will use $\mathcal{S}_\pi$ to generate messages according to the protocol to claim deposits as done by the honest parties in the real execution (Note: this is where we use the extensibility property indirectly via $\mathcal{S}_\pi$). Note that $\mathcal{S}_\pi$ does not need the actual output of the execution until the last honest party message (i.e., it can simulate honest messages until then).

Additionally, during the claim phase, $\mathcal{S}$ will need to produce coins whenever $\mathcal{A}$ claims them. Suppose $P_{i \bmod n}$ is corrupt and it produces a valid witness (α, α'_n) to claim Tx_i. (For the time being, assume $i \neq (r+1)n$; we'll take care of this case later.) To simulate this, $\mathcal{S}$ needs to produce $\mathsf{coins}(i \cdot q)$. Ideally, we would ask $\mathcal{S}$ to get these coins via Tx_{i-1}, i.e., $\mathsf{coins}((i-1)q)$ and $\mathsf{coins}(q)$ from $P_{i \bmod n}$'s roof deposit $\mathsf{Tx}_{n,i \bmod n}$ (out of a total of $\mathsf{coins}((r+1)q)$). However, it is not clear that Tx_{i-1} was even made by $P_{i \bmod n}$. Importantly, we are able to make the following claim: if Tx_{i-1} was not made by $P_{i \bmod n}$ but $P_{i \bmod n}$ can produce valid witnesses to claim Tx_i, then for every $j < i$, party $P_{j \bmod n}$ is corrupt. Note that the above claim implies that $i < n$, since we assume that there is at least one honest party.

To prove the claim, we appeal to the unforgeability property of the transcript validation function of π in the following way. Note that according to the protocol description, if Tx_{i-1} was not made, then for $j < i$ an honest party $P_{j \bmod n}$ would not make Tx_{j-1}. Since it never made a deposit Tx_{j-1}, this in turn means that it would never reveal a valid witness for Tx_j (Note: honest $P_{j \bmod n}$ claims Tx_j iff Tx_{j-1} is claimed). Then it follows from the unforgeability property that without a valid witness for Tx_j it is computationally infeasible to provide a valid witness to Tx_i. Now we are ready to describe the simulation of the claims $\mathsf{Tx}_1, \ldots, \mathsf{Tx}_i$ (whichever were made). For transactions Tx_j with $j < i$ that were claimed, we simply use corrupt P_{j+1}'s deposit of $\mathsf{coins}(jq)$ to give to corrupt P_j who's claiming it. For transaction Tx_i, we use $\mathsf{coins}(q)$ from the roof deposit of each of $P_1, \ldots, P_i$. Recall P_{i+1} is honest, and if P_{i+1} made deposit Tx_i it must necessarily hold that all parties (in particular, $P_1, \ldots, P_i$) made roof deposits of $\mathsf{coins}((r+1)q)$ each. The above strategy takes care of parties that try to claim a deposit Tx_i without making the deposit Tx_{i-1}.

In the rest of the proof, we assume that every P_i that tries to claim deposit Tx_i has made the deposit Tx_{i-1}. This case is handled easily: we ask $\mathcal{S}$ to get these coins via Tx_{i-1}, i.e., $\mathsf{coins}((i-1)q)$ and $\mathsf{coins}(q)$ from $P_{i \bmod n}$'s roof deposit $\mathsf{Tx}_{n,i \bmod n}$ (out of a total of $\mathsf{coins}((r+1)q)$). The above strategy works only for $i \bmod n \neq n$. We first complete the discussion assuming P_n is honest and then discuss the other case later. Suppose P_n is honest. We have already addressed how to handle coins during the claims. The only remaining thing to address is when to ask $\mathcal{S}$ to contact $\mathcal{F}^*_f$. Let i_c be the largest index such that $P_{i_c \bmod n}$ is corrupt. $\mathcal{S}$ will contact the ideal functionality when Tx_{i_c} is claimed by P_{i_c}. In fact, when Tx_{i_c-1} is deposited by P_{i_c}, $\mathcal{S}$ will take $\mathsf{coins}(hq)$ out of these $\mathsf{coins}((i_c-1)q)$ (note: $i_c > n > h$) to deposit to the ideal functionality along with the inputs to the computation (extracted via $\mathcal{S}_\pi$). $\mathcal{S}$ would then get the actual output from the ideal functionality, feed this to $\mathcal{S}_\pi$ and get the remaining honest messages such that the view of the adversary is indistinguishable from the real execution. Also, $\mathcal{S}$ would ask $\mathcal{F}^*_f$ to deliver the output to all honest parties (because the corrupt parties did not abort the computation at all) and would get $\mathsf{coins}(hq)$ back from the ideal functionality which it can then use to handle P_{i_c}'s claim of Tx_{i_c}. This completes the simulation when P_n is honest.

When P_n is corrupt, we need to show how to handle claims made by P_n and also how to handle the interaction with the ideal functionality. The claims made by P_n are handled easily: if Tx_i for $i \bmod n = n$ is claimed by P_n, then we supply $\mathsf{coins}(iq)$ to P_n by taking it out of $\mathsf{coins}((i-1)q)$ deposited by P_n in Tx_{i-1} and $\mathsf{coins}(q)$ taken out of $\mathsf{coins}(((r+1)n-1)q)$ deposited by P_n in $\mathsf{Tx}_{(r+1)n-1}$. Note that while handling ladder claims, a total of $\mathsf{coins}(rq)$ have been removed from $\mathsf{coins}(((r+1)n-1)q)$ deposited by P_n in $\mathsf{Tx}_{(r+1)n-1}$. This leaves $\mathsf{coins}(((r+1)n-1)q) - \mathsf{coins}(rq) = \mathsf{coins}(rnq + nq - q - rq) = \mathsf{coins}((r+1)(n-1)q)$. Actually, when $\mathsf{Tx}_{(r+1)n-1}$ is deposited by P_n, $\mathcal{S}$ will take $\mathsf{coins}(hq)$ out of these remaining unused $\mathsf{coins}((r+1)(n-1)q)$ (note: $(r+1)(n-1) > h$ since $r > 0$) to deposit to the ideal functionality along with the inputs to the computation (extracted via $\mathcal{S}_\pi$). $\mathcal{S}$ would then get the actual output from the ideal functionality, feed this to $\mathcal{S}_\pi$ and get the remaining honest messages such that the view of the adversary is indistinguishable from the real execution. Also, $\mathcal{S}$ would ask $\mathcal{F}^*_f$ to deliver the output to all honest parties (because the corrupt parties did not abort the computation at all) and would get $\mathsf{coins}(hq)$ back from the ideal functionality which it can then use to handle P_n's claim of $\mathsf{Tx}_{n,j}$ for all j since it now has exactly $\mathsf{coins}((r+1)(n-1)q)$!

The one additional point to note is the equivocation of honest party's messages so that the view of the adversary is indistinguishable from the real execution. Let i_h be the largest index such that $P_{i_h \bmod n}$ is honest. $\mathcal{S}$ will actually do the above steps of giving $\mathsf{coins}(hq)$ to $\mathcal{F}_f^*$ right before time τ_{i_h} so that it can obtain the output and feed this to $\mathcal{S}_\pi$ to produce the correct view. This completes the simulation when P_n is honest. $\square$

6. DUAL MODE PROTOCOL

We now show an application of our planted ladder mechanism. This application is motivated by the fact that if $s < n/2$ parties are corrupt, then standard secure computation protocols [8, 15] obtain guaranteed output delivery (and, in particular, fairness) and one does not have to resort to the notion of secure computation with penalties. Additionally, note that a single corrupt party P_n in the ladder mechanism can deny the output to all honest parties. Ideally, one would want a "dual mode" protocol (alternatively, "best-of-both-worlds" protocol) where

1. if $s < n/2$ parties are corrupt, then the protocol guarantees fairness; and
2. if $t > n/2$ parties are corrupt, then the protocol guarantees fairness under the penalties notion.

Ishai et al. [18] considered a weaker version of the above where they relaxed the second requirement to obtain a protocol that guarantees "security-with-abort" which is the standard security notion for secure computation in the presence of a dishonest majority [14]. They showed (1) a negative result that precludes such "best-of-both-worlds" protocol when the parameters s, t are such that $s + t \geq n$, and (2) a positive result with an explicit construction of such a protocol when $s + t < n$ (in fact, their protocol achieves guaranteed output delivery when at most s parties are corrupt). Clearly, "security-with-abort" is weaker than "security-with-penalties" since the latter implies the former. Thus, we cannot hope for positive results in the regime where $s + t \geq n$. In this section, we show that the regime where $s + t < n$ indeed supports a "best-of-both-worlds" protocol as described above.

Our protocol. Our first observation is that restarts are required in [18] in order to guarantee output delivery, and can be avoided in our case since we only require fairness. Then to make the script complexity independent of the complexity of evaluating f, a natural idea is to follow the template of [9] and to use an off-chain secure computation protocol and make the on-chain complexity depend only on the size of the output of f. Our off-chain MPC protocol actually computes the output and performs a $(t + 1)$-out-of-n secret sharing of the output. This ensures that at the end of the MPC protocol, the adversary corrupting $t > n/2$ parties does not obtain any information about the output. Parties then proceed to run the planted ladder mechanism using essentially the parameters of the original ladder mechanism of [9]. (That is, they reveal the secret shares one-by-one in the ladder claim phase after revealing the dummy witnesses in the underground claim phase.)

We note that it is important to use the planted ladder mechanism as the ladder mechanism does not guarantee compensation to honest parties that did not reveal their secret share (say, in the event that the protocol was aborted in the middle). This is especially important as the adversary requires only one additional secret share in order to compute the output. On the other hand, the planted ladder mechanism guarantees that either the protocol is completed or *all* honest parties obtain a compensation. That is, either all parties get the output or all honest parties get compensated. This takes care of the case when $t \geq n/2$ are corrupt as in this case we get fairness with penalties.

Protocol Init. Parties $P_1, \ldots, P_n$ with their MPC inputs $x_1, \ldots, x_n$ run an MPC protocol that

- computes $z \leftarrow f(x_1, \ldots, x_n)$;
- $(t + 1)$-out-of-n secret shares z into $z_1, \ldots, z_n$;
- samples $\omega_1, \ldots, \omega_n$ at random from $\{0, 1\}^\lambda$;
- for $j \in [n]$: computes $h_j = \mathsf{Hash}(z_j \| \omega_j)$;
- for $i \in [n]$: outputs $y_i = z_i \| \omega_i$ and $\{h_j\}_{j \in [n]}$ to party P_i.

Predicates For $i \in [n]$, predicate ϕ_i has $\{h_j\}_{j \leq i}$ hard-coded:

$$\phi_i\left(\beta_1 \| \cdots \| \beta_i; \{h_j\}_{j \leq i}\right) = \left(\mathsf{Hash}(\beta_1) \overset{?}{=} h_1\right) \bigwedge \cdots \bigwedge \left(\mathsf{Hash}(\beta_i) \overset{?}{=} h_i\right)$$

Procedure $\mathsf{Extend}(i, \alpha; y_i) \equiv \alpha \| y_i$.

Protocol Recon. If each party has already obtained at least $t + 1$ shares via the ladder mechanism, then they all reconstruct the final output and terminate the protocol. On the other hand, suppose less than $t + 1$ shares were disclosed during the execution of the ladder mechanism, then each honest party P_i broadcasts y_i. If at least $t + 1$ shares were broadcasted, then parties now use these shares to reconstruct the output z, else they output $\perp$.

Figure 11: Parameterizing the planted ladder mechanism to get the dual mode protocol.

For the case when $s < n - t$ parties are corrupted, we need to obtain complete fairness. Since there are $n - s \geq t + 1$ honest parties, we simply ask the honest parties to broadcast their shares at the end of the protocol. Recall that $t + 1$ shares are sufficient to reconstruct the secret. Note that an honest party would broadcast its share only after obtaining a compensation. Therefore, when $t > n/2$ parties are corrupt, we obtain fairness with penalties, and when less than $n - t$ parties are corrupt we obtain complete fairness. The script complexity of the above protocol would be $O(n^2|z|)$ since we use the mechanism of [9]. Alternatively we could use the compact ladder mechanism assuming a programmable random oracle to get a protocol whose script complexity is $O(n\lambda)$. The formal protocol is described in Figure 11. (Note that Hash can be safely replaced by honest-binding commitments as in [9].) Due to space limitations, the proof of the following theorem is deferred to the full version.

THEOREM 5. *Assume the existence of one-way functions. Let s, t be such that $s < n/2$, $t \geq n/2$, and $s + t < n$. Then there exists a protocol in the $(\mathcal{F}_{\mathrm{OT}}, \mathcal{F}^*_{\mathrm{CR}})$-hybrid model that simultaneously (1) SCC-realizes $\mathcal{F}_f^*$ when t parties are corrupted, and (2) realizes $\mathcal{F}_f$ with complete fairness when s parties are corrupted. Additionally, the protocol makes only $O(n)$ calls to $\mathcal{F}^*_{\mathrm{CR}}$ and the script complexity of the protocol is $O(n^2|z|)$ where $|z|$ denotes the size of the output of f. (This can be reduced to $O(n\lambda)$ assuming a programmable random oracle.)*

7. CONCLUSIONS

The results in this paper are motivated by the distinction between on-chain and off-chain complexity of secure computation with penalties. Specifically, we design protocols for secure computation with penalties which have lower script complexity (i.e., on-chain complexity) relative to prior works [9, 22]. Our two main efficiency improvements are: (1) an improvement to the script complexity of secure computaion with penalties in the non-reactive setting from $O(n^2|z|)$ to $O(n\lambda)$ (in particular, making the complexity independent of the function), and (2) an improvement to the script complexity (and also number of $\mathcal{F}^*_{\rm CR}$ calls) of secure computation with penalties in the reactive setting from $O(n^2T)$ to $O(nT)$. Both these results provide a quadratic to linear reduction in the dependence on the number of parties n and thus are likely to be useful in practice. One major open question in this area is to remove/lessen the dependence of the script complexity in the reactive setting on T, i.e., the size of the transcript of the MPC protocol realizing the reactive function.

Another contribution of this work is providing a general framework for the ladder mechanism which in fact makes the protocols easier to describe. Additionally, we use this framework to design a protocol that offers "best-of-both-worlds" security in the sense that it offers (1) complete fairness when up to s ($< n/2$) parties are corrupted, and (2) fairness with penalties when up to t ($> n/2$) parties are corrupted. Our protocol works in the regime where $s + t < n$ which is essentially the only regime where positive results are possible [18]. Open questions in this area include a lower bound on the script complexity and the number of $\mathcal{F}^*_{\rm CR}$ transactions required to implement secure computation with penalties tolerating $t < n$ corruptions.

8. REFERENCES

[1] 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.

[2] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. In *IEEE Security and Privacy*, 2014.

[3] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. On the malleability of bitcoin transactions. In *Financial Cryptography Workshops*, pages 1–18, 2015.

[4] N. Asokan, V. Shoup, and M. Waidner. Optimistic protocols for fair exchange. In *ACM CCS*, pages 7–17, 1997.

[5] A. Back and I. Bentov. Note on fair coin toss via bitcoin. http://arxiv.org/abs/1402.3698, 2013.

[6] S. Barber, X. Boyen, E. Shi, and E. Uzun. Bitter to better - how to make bitcoin a better currency. In *FC*, 2012.

[7] M. Belenkiy, M. Chase, C. Erway, J. Jannotti, A. Kupcu, A. Lysyanskaya, and E. Rachlin. Making p2p accountable without losing privacy. In *Proc. of WPES*, 2007.

[8] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM Press, May 1988.

[9] I. Bentov and R. Kumaresan. How to use bitcoin to design fair protocols. In *Crypto (2)*, pages 421–439, 2014.

[10] R. Canetti, A. Jain, and A. Scafuro. Practical uc security with a global random oracle. In *CCS*, pages 597–608, 2014.

[11] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *20th Annual ACM Symposium on Theory of Computing*. ACM Press, May 1988.

[12] R. Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369, 1986.

[13] Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In Cyril Gavoille and Pierre Fraigniaud, editors, *30th ACM Symposium Annual on Principles of Distributed Computing*, pages 179–186. ACM Press, June 2011.

[14] Oded Goldreich. Foundations of cryptography - vol. 2. 2004.

[15] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*. ACM Press, 1987.

[16] S. Dov Gordon, Yuval Ishai, Tal Moran, Rafail Ostrovsky, and Amit Sahai. On complete primitives for fairness. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 91–108. Springer, February 2010.

[17] S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 157–176. Springer, May 2010.

[18] Y. Ishai, J. Katz, E. Kushilevitz, Y. Lindell, and E. Petrank. On achieving the "best of both worlds" in secure multiparty computation. In *SIAM J. Comput. Vol. 40, No. 1*, pages 122–141.

[19] A. Kiayias, H-S. Zhou, and V. Zikas. Fair and robust multi-party computation using a global transaction ledger. In *Eurocrypt*, pages 705–734, 2016.

[20] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE Security and Privacy*, 2016.

[21] R. Kumaresan and I. Bentov. How to use bitcoin to incentivize correct computations. In *CCS*, pages 30–41, 2014.

[22] R. Kumaresan, T. Moran, and I. Bentov. How to use bitcoin to play decentralized poker. In *CCS*, 2015.

[23] Alptekin Küpçü and Anna Lysyanskaya. Usable optimistic fair exchange. In Josef Pieprzyk, editor, *Topics in Cryptology – CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 252–267. Springer, March 2010.

[24] Andrew Y. Lindell. Legally-enforceable fairness in secure two-party computation. In Tal Malkin, editor, *Topics in Cryptology – CT-RSA 2008*, volume 4964 of *Lecture Notes in Computer Science*, pages 121–137. Springer, April 2008.

[25] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena. Demystifying incentives in the consensus computer. In *CCS*, pages 706–719, 2015.

[26] G. Maxwell. Zero knowledge contingent payment. 2011. https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment.

[27] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, August 2002.

[28] Benny Pinkas. Fair secure two-party computation. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 87–105. Springer, May 2003.

[29] Andrew Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

Acknowledgements

Research supported in part by NSF Grants CNS-1350619 and CNS-1414119, Alfred P. Sloan Research Fellowship, Microsoft Faculty Fellowship, the NEC Corporation, a Steven and Renee Finn Career Development Chair from MIT, and an MIT Translational Fellowship. This work was also sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Research Office under contracts W911NF-15-C-0226.