zkHawk: Practical Private Smart Contracts from MPC-based Hawk

Aritra Banerjee
ADAPT Centre
School of Comp Sci & Stats
Trinity College Dublin
Dublin, Ireland
abanerje@tcd.ie

Michael Clear
School of Comp Sci & Stats
Trinity College Dublin
Dublin, Ireland
clearm@tcd.ie

Hitesh Tewari
School of Comp Sci & Stats
Trinity College Dublin
Dublin, Ireland
htewari@tcd.ie

***Abstract*—Cryptocurrencies have received a lot of research attention in recent years following the release of the first cryptocurrency Bitcoin. With the rise in cryptocurrency transactions, the need for smart contracts has also increased. Smart contracts, in a nutshell, are digitally executed contracts wherein some parties execute a common goal. The main problem with most of the current smart contracts is that there is no privacy for a party's input to the contract from either the blockchain or the other parties. Our research builds on the Hawk project that provides transaction privacy along with support for smart contracts. However, Hawk relies on a special trusted party known as a manager, which must be trusted not to leak each party's input to the smart contract. In this paper, we present a practical private smart contract protocol that replaces the manager with an MPC protocol such that the function to be executed by the MPC protocol is relatively lightweight, involving little overhead added to the smart contract function, and uses practical sigma protocols and homomorphic commitments to prove to the blockchain that the sum of the incoming balances to the smart contract matches the sum of the outgoing balances.**

***Index Terms*—Hawk, Private Smart Contracts, Multi-Party Computation**

I. Introduction

Cryptocurrencies are generally decentralized and based on a public distributed ledger called a blockchain. Many cryptocurrencies that followed Bitcoin, including Ethereum, do not offer transaction privacy. In a nutshell, transaction privacy hides the origin of a transaction (the sending party), along with the amounts transacted, such that a third party inspecting the blockchain cannot track the flow of money between accounts. Zcash [1] and Monero [2] are two popular cryptocurrencies that provide transaction privacy. A major limitation of these cryptocurrencies is that they do not support smart contracts. In short, a smart contract is a program that determines a set of deposits and withdrawals to/from a set of (anonymous) accounts. A smart contract is created by a set of parties to facilitate a common goal. We call these set of parties the *participant*s of the smart contract. The absence of support for smart contracts in existing cryptocurrencies with transaction privacy inspired the Hawk project [3]. Hawk obtains transaction privacy with support for smart contracts, albeit with the trust assumption of a special type of entity known as a *manager*. The manager sees the inputs of each participant in the smart contract and must be explicitly trusted not to leak them. There have been other recent developments in privacy where the smart contract execution is done off-chain like in Arbitrum [4] which used the concept of Virtual Machines (VMs) along with the trust assumption of a *manager* as well. But Arbitrum provided only partial privacy, i.e, the input of the parties are hidden from blockchain but not from each other. The goal of this research is to build upon Hawk so that along with support for transaction privacy and smart contracts, we additionally obtain privacy for the participant's inputs (i.e. they are hidden from the blockchain and each other), without the trust assumption of the *manager*.

Our starting point is an observation made by the authors of the Hawk protocol [3], namely that the manager can be replaced by running a multi-party computation (MPC) protocol between the parties. However the authors of the Hawk paper point out that this approach would be presently impractical. Indeed it can be readily seen that by applying MPC to the design of Hawk as it is incurs the prohibitively expensive overhead of executing a zk-SNARK proof [5], [6] within an MPC program (a circuit in practice). We therefore propose to remove the zk-SNARK proof from the MPC program. However this leaves us with a considerable challenge - How do we prove to the blockchain that the sum of the in-going account balances in a smart contract is equal to the sum of the outgoing balances? As a solution, we will compute a relatively practical proof of knowledge proof within the MPC program such that all parties contribute to the witness. The resulting effect is to guarantee that the difference between the sum of the input balances and the sum of the outgoing balances is zero. An MPC protocol [7] enables a collection of parties to interact with each other in several "rounds" of communication in order to compute a function f and learn the output $y = f(x_1, x_2, \ldots, x_n)$ where x_i is party $i's$ input. Such that even if up to t parties are malicious (for some collusion tolerance t); they cannot learn the other parties' inputs i.e. they are kept secret. We distinguish between two types of input/output privacy, which are:

This publication has emanated from research conducted with the financial support of Science Foundation Ireland grants 13/RC/2106 (ADAPT) and 17/SP/5447 (FinTech Fusion). This work was also supported in part by Science Foundation Ireland grant 13/RC/2094 (Lero).

- **Weak Input/Output Privacy** is defined such that the execution of a smart contract does not reveal any parties' inputs/outputs to the public. But the inputs/outputs of each party are not hidden from each other.
- **Strong Input/Output Privacy** is defined such that parties' inputs/outputs are not leaked to both the public and to each of the other parties.

Furthermore, we distinguish between two types of PSC evaluation: non-interactive PSC (NI-PSC) and interactive PSC (I-PSC). In the former, the blockchain executes the smart contract function and no interaction is needed between the contract participants. In contrast, interactive PSC involves executing the smart contract function off-chain by all parties interacting for the purpose of executing an MPC protocol to compute the smart contract function. This is the setting primarily explored in this paper. We do however briefly touch upon non-interactive PSC in Section V but we leave further development to future work.

We consider a cryptocurrency that supports transaction privacy. More precisely, we expect, for example, to inherit a blockchain that implements the $\mathsf{Blockchain}_{\mathsf{cash}}$ program in [3] and a user program that implements the $\mathsf{UserP}_{\mathsf{cash}}$ protocol in [3] i.e. the operations of mint and pour from ZCash/Hawk that provides transaction privacy. To avoid implementing these operations in this work, we interface with such a construction through a commitment scheme that is defined by the PSC evaluation protocol. First, we formalize an (interactive) PSC evaluation protocol in this paper as shown in Section II. In the formal definitions sections we also specify the set of input coins $\mathsf{coin}_1, \ldots, \mathsf{coin}_n$ which are commitments to hidden values $\$\mathsf{val}_1, \ldots, \$\mathsf{val}_n$ respectively. Next, we construct a concrete PSC evaluation protocol as shown in Section III that allows the contract participants to evaluate a smart contract function off-chain that yields a new set of coins $\mathsf{coin}'_1, \ldots, \mathsf{coin}'_n$ that hide the output values $\$\mathsf{val}'_1, \ldots, \$\mathsf{val}'_n$ from the smart contract such that the blockchain can be sure that $\sum_{i\in[n]} \$\mathsf{val}'_i = \sum_{i\in[n]} \$\mathsf{val}_i$. Finally, we prove our protocol t-secure for collusion tolerance $t < n$ assuming the hardness of the discrete logarithm problem in the random oracle model (Section IV). Although this security notion is weaker than security in the universal composability framework, it shows our protocol resists a wide variety of attacks from a malicious adversary that can *statically* corrupt $t < n$ parties. We intend to prove full security in the UC framework [8] in future work.

One of the primary goals of our protocol is to minimize the computation overhead of the MPC program to ensure practicality. With this in mind, we use simple and lightweight cryptographic tools such as standard sigma protocols that can be computed efficiently. We also exploit a simple trick that involves decomposing the output coin commitment into commitments to its individual bits, then for each bit, sending both candidate commitments (a commitment to a zero and a commitment to a one) in random order to the blockchain, and choosing between the candidates using a simple multiplexer circuit in the MPC program based on the output value of the coin. Therefore, this idea simultaneously functions as a range proof and a way to reduce computation in the MPC program (we can use SPDZ [9]) to a simple multiplexer circuit. The associated cost of this approach is the requirement of two NIZK proofs for every bit. These proofs are however not computed within the MPC program so the approach remains practical.

A. Example of a Private Smart Contract

Consider a sealed bid auction as a Private Smart Contract as shown in Figure 1. In a sealed-bid auction, one of the participants is the seller and the other participants are bidders. Each bidder submits a bid and the bidder with the highest bid wins the auction. All bids are kept private.
Consider a function f that implements this smart contract. Suppose there are k bidders, and in total $n = k + 1$ parties (recall the seller is one of the parties). The seller is designated as the first party. We define:

- $f((\$\mathsf{seller}, \cdot), (\$\mathsf{bidder}_1, \cdot), \ldots, (\$\mathsf{bidder}_k, \cdot))$
  - $\$\mathsf{highest} \leftarrow 0$
  - $\mathsf{winner} \leftarrow 0$
  - For $i \in \{1, \ldots, k\}$:
    - If $\$\mathsf{bidder}_i > \$\mathsf{highest}$
      - $\$\mathsf{highest} \leftarrow \$\mathsf{bidder}_i$
      - $\mathsf{winner} \leftarrow i$
  - $\$\mathsf{seller}' \leftarrow \$\mathsf{seller} + \$\mathsf{highest}$
  - $\$\mathsf{bidder}'_{\mathsf{winner}} \leftarrow 0$
  - For $i \in \{1, \ldots, k\} \setminus \{\mathsf{winner}\}$:
    - $\$\mathsf{bidder}'_i \leftarrow \$\mathsf{bidder}_i$
  - Return $(\$\mathsf{seller}', \$\mathsf{bidder}'_1, \ldots, \$\mathsf{bidder}'_k, \mathsf{out} := \mathsf{winner})$

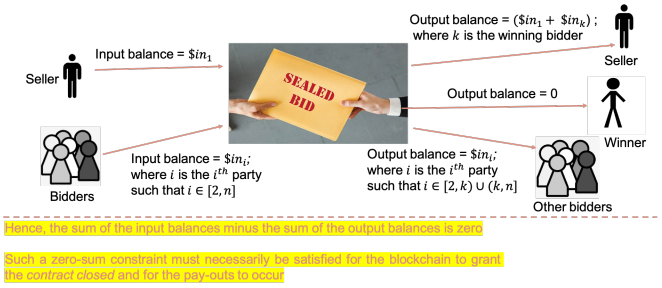

Fig. 1. Example of a Private Smart Contract: A Sealed Bid Auction

B. Overview of Interactive Private Smart Contracts

Private Smart Contract (PSC) execution involves three phases as shown in Figure 2. The first, which we call the *freeze* phase (a term borrowed from Hawk), is where each participant sends its input coin to the blockchain. The blockchain "freezes" the coins so that they cannot be spent and records them for use later. The second phase is *computation*. This occurs off-chain and involves the parties running an MPC protocol between them. The third phase is what we call *finalization* where at least one party must notify the blockchain

of the result of the MPC program. There are a set of output coins resulting from smart contract execution (one for each party). The blockchain has to check that the sum of the input coins is equal to the sum of the output coins. Once this is asserted, the blockchain can award the output coin to each party; we call this *contract closure*[1].

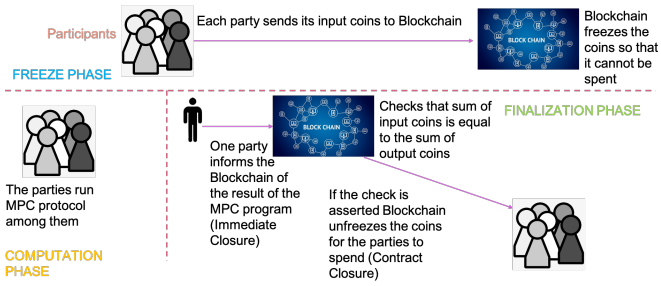

Fig. 2. How Interactive Private Smart Contracts will Work

II. Formal Definitions

We distinguish amounts of currency with a leading "$" (dollar sign) symbol before the associated variable name or literal value to enhance readability. We define the set of valid currency values $\mathbb{V}$ as the set of non-negative integers less than a strict upper bound L. More formally, we have that $\mathbb{V} := \{\$\mathsf{val} \in \mathbb{Z} : 0 \leq \$\mathsf{val} < L\}$. For convenience, we choose L as a power of 2 i.e. we let $L = 2^\ell$ for some positive integer ℓ.

Definition 1: An n-party smart contract is an n-ary function $f : (\mathbb{V} \times \{0,1\}^*)^n \to (\mathbb{V}^n \times \{0,1\}^*) \cup \{\perp\}$ satisfying the following property:

- For all choices of $\$\mathsf{val}_1, \ldots, \$\mathsf{val}_n \in \mathbb{V}$ and $\mathsf{in}_1, \ldots, \mathsf{in}_n \in \{0,1\}^*$, one of the following statements is true
  1) $t = \perp$ (failure)
  2) $t = (\$\mathsf{val}'_1, \ldots, \$\mathsf{val}'_n, \mathsf{out}) \wedge$
  $$\sum_{i \in [n]} \$\mathsf{val}'_i - \sum_{i \in [n]} \$\mathsf{val}_i = 0 \text{ (zero-sum constraint)}$$

  where $t = f((\$\mathsf{val}_1, \mathsf{in}_1), \ldots, (\$\mathsf{val}_n, \mathsf{in}_n))$.

A. Coins

Following on from Zcash [1] and Hawk [3], we define a *coin* as a commitment to some value $\$\mathsf{val}$ with randomness r using some binding and hiding commitment scheme Com. We write this as $\mathsf{coin} = \mathsf{Com}(\$\mathsf{val}; r)$. We assume the inclusion of a mechanism to provide transaction privacy in a manner such as Zcash and Hawk, but we endeavor to sidestep repeating the details here by defining an appropriate interface between such a mechanism for transaction privacy and a PSC evaluation protocol. We achieve this by assuming that the value associated with the input coin coin_i of each party $\mathcal{P}_i$ is private; that is, hidden from the blockchain and the other parties. Execution of the smart contract produces an output coin coin'_i for each party $\mathcal{P}_i$ whose value is also hidden from the blockchain and the other parties. A particular PSC evaluation protocol, defined momentarily, specifies the commitment scheme that it uses to bind a value $\$\mathsf{val}$ to a coin using randomness r.

[1]A contract is said to be closed when the zero-sum constraint is evaluated to be satisfied and the payouts are made.

B. PSC Evaluation Protocol

We now present the notion of a private smart contract (PSC) evaluation protocol. This is a protocol that consists of a blockchain program, which handles three types of messages $(\mathsf{freeze}, \cdot)$, $(\mathsf{compute}, \cdot)$ and $(\mathsf{finalize}, \cdot)$ and a collection of user parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$. We defer to the full version a formal generalized definition of a PSC evaluation protocol using the blockchain model of cryptography [3] in the universal composability (UC) [8] framework. Our definition here is more specialized as it specifically addresses the case of interactive PSC, which is what our protocol targets in this paper. In interactive PSC, computation of the smart contract function occurs off-chain, most likely using a multi-party computation protocol. Several elements of our definition are inspired by the blockchain model [3] but our model here is effectively "stripped-down" and tailored to the task in hand. We require a special trusted incorruptible entity $\mathcal{M}$ that is defined as follows. If $\mathcal{M}$ receives a message (input, F, x_i) from every party $\mathcal{P}_i$ for $i \in [n]$, then it privately computes $y = F(x_1, \ldots, x_n)$ and sends y to every party $\mathcal{P}_i$ with the message (output, F, y). Note that there is an authenticated and private channel between $\mathcal{M}$ and every party $\mathcal{P}_i$. Such an entity $\mathcal{M}$ allows us to model a correct and secure idealized MPC protocol.

Definition 2: A PSC evaluation protocol π (in the interactive setting) is a tuple (Com, B, U) where Com is a commitment scheme used to establish input and output coins, $B = (\mathsf{Init}, \mathsf{Freeze}, \mathsf{Finalize})$ is the blockchain program (modelled as a tuple of stateful PPT algorithms) and $U = (\mathsf{Init}, \mathsf{Create}, \mathsf{Freeze}, \mathsf{PrepareCompute}, \mathsf{Finalize})$ is the user program. The program B is passed to the blockchain program wrapper B', defined in Figure 3, to produce a blockchain functionality. The program U is passed to the user program wrapper U', defined in Figure 4 to specify user behavior during execution of the protocol π.

The fundamental correctness condition expected of a PSC evaluation protocol $\pi := (\mathsf{Com}, B, U)$ is as follows. Let f be an n-ary smart contract function as defined in Definition 1. For any $\$\mathsf{val}_1, \ldots, \$\mathsf{val}_n \in \mathbb{V}$ and $\mathsf{in}_1, \ldots, \mathsf{in}_n \in \{0,1\}^*$ such that $((\$\mathsf{val}'_1, \ldots, \$\mathsf{val}'_n), \mathsf{out}) \leftarrow f((\$\mathsf{val}_1, \mathsf{in}_1), \ldots, (\$\mathsf{val}_n, \mathsf{in}_n))$ for $\$\mathsf{val}'_1, \ldots, \$\mathsf{val}'_n \in \mathbb{V}$ and $\mathsf{out} \in \{0,1\}^*$. Then we say that π correctly executes π if the following experiment terminates and outputs 1 with all but negligible probability.

Experiment $\mathsf{Exper}_\pi(f, P, (\$\mathsf{val}_1, \mathsf{in}_1), \ldots, (\$\mathsf{val}_n, \mathsf{in}_n))$:
- Run blockchain process $\mathcal{B}$ with program $B'(B)$
- Run $\mathcal{P}_i$ with program $U'(U)$ for $i \in [n]$
- Send $(\mathsf{freeze}, f, P, \$\mathsf{val}_i, \mathsf{in}_i)$ to $\mathcal{P}_i$ for $i \in [n]$
- Wait until $(\mathsf{id}, P, P, \mathsf{compute}) \in \mathcal{B}.\mathsf{Contracts}$
- Send $(\mathsf{compute}, \mathsf{id})$ to $\mathcal{P}_i$ for $i \in [n]$
- Wait until $(\mathsf{id}, P, P, \mathsf{finalized}) \in \mathcal{B}.\mathsf{Contracts}$
- Return $\sum \$\mathsf{val}'_i = \sum \$\mathsf{val}_i$ where $(\$\mathsf{val}'_i, \cdot) \in \mathcal{P}_i.\mathsf{Coins}$

Blockchain program wrapper $B'(B)$:

Init:
Contracts $\leftarrow \{\}$
Call B.Init() (i.e. Call program B)
Send blockchain state to $\mathcal{A}$.

Freeze: Upon receiving msg $:= (\mathsf{freeze}, \mathsf{id}, P, \cdot)$ from $\mathcal{P}$:
Send $(\mathsf{msg}, \mathcal{P})$ to $\mathcal{A}$
Assert $\mathcal{P} \in P$
If $(\mathsf{id}, \cdot, \cdot, \cdot) \notin$ Contracts:
Contracts $\leftarrow$ Contracts $\cup \{(\mathsf{id}, P, P' := \{\}, \mathsf{freeze})\}$
Assert $(\mathsf{id}, \bar{P}, P', t := \mathsf{freeze}) \in$ Contracts
Assert $\bar{P} = P$
Assert $\mathcal{P} \notin P'$
If B.Freeze(msg, $\mathcal{P}$) $= 1$: (i.e. Call program B)
If $P' \cup \{\mathcal{P}\} = \bar{P}$:
$t \leftarrow$ compute
Replace $(\mathsf{id}, \bar{P}, P', \mathsf{freeze})$ in Contracts
with $(\mathsf{id}, \bar{P}, P' \cup \{\mathcal{P}\}, t)$
Send blockchain state to $\mathcal{A}$

Compute: Computation is performed off-chain

Finalize: Upon receiving msg $:= (\mathsf{finalize}, \mathsf{id}, \cdot)$ from $\mathcal{P}$:
Send $(\mathsf{msg}, \mathcal{P})$ to $\mathcal{A}$
Assert $(\mathsf{id}, P, P', \mathsf{compute}) \in$ Contracts
If B.Finalize(msg) $= 1$:
Replace $(\mathsf{id}, P, P', \mathsf{compute})$ in Contracts
with $(\mathsf{id}, P, P', \mathsf{finalized})$
Send blockchain state to $\mathcal{A}$

Fig. 3. Blockchain program wrapper $B'(B)$ for PSC evaluation. Note that $\mathcal{A}$ is the adversary.

User program wrapper $U'(U)$:

Init:
Computations $\leftarrow \{\}$
Coins $\leftarrow \{\}$
Call U.Init() (i.e. Call program U)

Freeze: Upon receiving message $(\mathsf{freeze}, f, P, \$\mathsf{val}, \mathsf{in})$:
$r \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{Com}}}$
coin $\leftarrow \mathsf{Com}(\$\mathsf{val}; r)$
Coins $\leftarrow$ Coins $\cup \{(\$\mathsf{val}, r)\}$
id $\leftarrow$ U.Create(f, P)
msg $\leftarrow$ U.Freeze(id, $(\$\mathsf{val}, r)$, in)
Send msg to blockchain

Compute: Upon receiving message $(\mathsf{compute}, \mathsf{id})$:
$(\hat{f}, P, x) \leftarrow$ U.PrepareCompute(id)
Computations $\leftarrow$ Computations $\cup (\hat{f}, P, \mathsf{id})$
Send $(\mathsf{input}, \hat{f}, P, x)$ to $\mathcal{M}$

Finalize: Upon receiving msg $:= (\mathsf{output}, \hat{f}, P, y)$ from $\mathcal{M}$:
Assert $(\hat{f}, P, \mathsf{id}) \in$ Computations
$(\mathsf{msg}, \mathsf{coin}'_1, \ldots, \mathsf{coin}'_n, \mathsf{out}, (\$\mathsf{val}', r')) \leftarrow$ U.Finalize(id, y)
Coins $\leftarrow$ Coins $\cup \{(\$\mathsf{val}', r')\}$
Send msg to blockchain
Remove $(\hat{f}, P, \mathsf{id})$ from Computations

Fig. 4. User program wrapper $U'(U)$ for PSC evaluation.

To clarify on the experiment above, when a process $\mathcal{X}$ is run with a program X, the initialization procedure X.Init is executed first and then the control is transferred to a message loop where the program waits for incoming messages which are dispatched to their appropriate handlers. We denote by $\mathsf{transcript}_{\pi,\mathcal{S},\mathcal{R}}$ the sequence of messages sent from $\mathcal{S}$ to $\mathcal{R}$ during the execution of protocol π. Of particular interest is $\mathsf{transcript}_{\pi,\mathcal{B},\mathcal{A}}$ where $\mathcal{B}$ is the blockchain and $\mathcal{A}$ is the adversary. In fact, in our security definition, which is presented next, the objective is to simulate the protocol and produce a transcript that is computationally indistinguishable from that produced in the real world.

C. Security

We now present a security definition that is strictly weaker than security in the UC framework, which we leave to an extended paper. The security definition we describe in this section is modelled on the typical simulation-based security notion for MPC. However, there are some notable differences. Firstly, the adversary $\mathcal{A}$ interacts with a blockchain, which is modelled as an entity that is trusted for availability and correctness which shares its internal state with the adversary. Secondly, in the real world, the contract participants $\mathcal{P}_i$ interact privately with a trusted third party $\mathcal{M}$ that cannot be corrupted by the adversary nor can the adversary access the contents of the private channels between $\mathcal{M}$ and $\mathcal{P}_i$.

Let $I = \{i_1, \ldots, i_t\}$ be the set of indices of the $t \leq n$ corrupted parties. Additionally, we let $\bar{I} = [n] \setminus I$ be the set of indices of the honest parties. Let $f : (\mathbb{V} \times \{0,1\}^*)^n \rightarrow (\mathbb{V}^n \times \{0,1\}^*) \cup \{\perp\}$ be an n-party smart contract function as defined in Definition 1. We denote by $\vec{\mathbf{x}}$ a vector of *inputs* from all n parties to the protocol; that is, we have $\vec{\mathbf{x}} := (x_1 := (\$\mathsf{val}_1, \mathsf{in}_1), \ldots, x_n := (\$\mathsf{val}_n, \mathsf{in}_n))$. Let π be an interactive PSC evaluation protocol as defined in Definition 2.

1) Real World: We now give the experiment for the real world execution of π in the presence of an adversary $\mathcal{A}$.

Experiment $\mathsf{REAL}_{\pi,\mathcal{A},I}(f, P, \vec{\mathbf{x}})_{\mathcal{D}}$:
Run blockchain process $\mathcal{B}$ with program $B'(B)$
Run $\mathcal{P}_i$ with program $U'(U)$ for $i \in \bar{I}$
Parse $(\$\mathsf{val}_i, \mathsf{in}_i) \leftarrow x_i$ for $i \in \bar{I}$
Send $(\mathsf{freeze}, f, P, \$\mathsf{val}_i, \mathsf{in}_i)$ to $\mathcal{P}_i$ for $i \in \bar{I}$
Wait until $(\mathsf{id}, P, \cdot, \cdot) \in \mathcal{B}$.Contracts
Send $(\mathsf{compute}, \mathsf{id})$ to $\mathcal{P}_i$ for $i \in \bar{I}$
Output $\mathcal{D}(\mathsf{transcript}_{\pi,\mathcal{B},\mathcal{A}})$

Furthermore, the blockchain executes blockchain program $B'(B)$. Recall from the definition of B' that the blockchain sends its internal state to $\mathcal{A}$ after it handles every message.

2) Ideal World: In the ideal world, a simulator $\mathcal{S}$ interacts with an ideal functionality $\mathcal{F}$ that executes the smart contract function f. All parties send their inputs to $\mathcal{F}$, then the function f is computed and the result returned to $\mathcal{S}$. The goal of simulation in the ideal world is for $\mathcal{S}$ to simulate the adversary's view in the real world i.e. it must produce a

Idealized Entity $\mathcal{M}$:

Init:
$\mathsf{Inputs} \leftarrow \{\}$

Input: Upon receiving $(\mathsf{input}, F, P, x)$ from $\mathcal{P}$:
Assert $\mathcal{P} \in P$
Remove $(F, P, \mathcal{P}, \cdot)$ from Inputs if it exists
$\mathsf{Inputs} \leftarrow \mathsf{Inputs} \cup (F, P, \mathcal{P}, x)$
If $(F_j, P_j, \mathcal{P}_j, x_j) \in \mathsf{Inputs} \wedge$
$F_j = F \wedge P_j = P$ for all $\mathcal{P}_j \in P$:
$y \leftarrow F(x_1, \ldots, x_{n'})$ where $n' = |P|$
Send $(\mathsf{output}, F, P, y)$ to $\mathcal{P}_j$ for all $\mathcal{P}_j \in P$

Fig. 5. Idealized entity $\mathcal{M}$ that privately and correctly executes a designated function on inputs supplied by a set of parties, which are kept private.

transcript that is computationally indistinguishable from the transcript generated in the real world. We denote by $\vec{\mathbf{x}}_J$ for $J \subseteq [n]$ the sub-vector of $\vec{\mathbf{x}}$ containing the components of $\vec{\mathbf{x}}$ at every index in J.

Experiment $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},I}(f, P, \vec{\mathbf{x}}))_{\mathcal{D}}$:
$\mathsf{transcript} \leftarrow \mathcal{S}^{\mathcal{F}_{f,P}(\vec{\mathbf{x}}_{\bar{I}}, \cdot)}(f, P, \vec{\mathbf{x}}_I)$
Output $\mathcal{D}(\mathsf{transcript})$

Definition 3: A PSC evaluation protocol π is t-secure if for all $I \subseteq [n]$ with $|I| \leq t$, all polynomial-time computable smart contract functions f, all polynomial-sized $\vec{\mathbf{x}} \in (\{0,1\}^*)^n$ and all subsets of participants $P \subseteq \{\mathcal{P}_i\}_{i \in [n]}$, then for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that for any PPT distinguisher $\mathcal{D}$ it holds that

$$\Big|\mathsf{Pr}\Big[\mathsf{REAL}_{\pi,\mathcal{A},I}(f, P, \vec{\mathbf{x}})_{\mathcal{D}} \to 1\Big] - \mathsf{Pr}\Big[\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},I}(f, P, \vec{\mathbf{x}})_{\mathcal{D}} \to 1\Big]\Big| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{negl}(\lambda)$ is a negligible function in the security parameter λ.

III. Our PSC Evaluation Protocol

The protocol we present ensures both *contract closure* and *immediate closure*[2]. Our first idea is to instantiate the commitments used for coins with Pedersen commitments [10]. Let $\mathbb{G}$ be a finite cyclic group of prime order p. Let g and h be two generators of $\mathbb{G}$. Then a Pedersen commitment is defined as

$$\mathsf{Com}(x; r) = g^x \cdot h^r$$

Consider input coins $\{\mathsf{coin}_j = \mathsf{Com}(\$\mathsf{val}_j; r_j)\}_{j \in [n]}$ and output coins $\{\mathsf{coin}'_j = \mathsf{Com}(\$\mathsf{val}'_j; s_j)\}_{j \in [n]}$. Then by the homomorphic property of Pedersen commitments [10], [11], we have

$$\prod_{j \in [n]} \mathsf{coin}'_j / \mathsf{coin}_j = h^{\sum_{j \in [n]} s_j - r_j}$$

[2]A protocol is said to support immediate closure if the blockchain can instigate contract closure after receiving a single accepted finalization message.

if and only if $\sum_{j \in [n]} \$\mathsf{val}'_j - \$\mathsf{val}_j = 0$. Our protocol leverages the Schnorr sigma protocol [12] to prove knowledge of the discrete logarithm in base h of $\prod_{j \in [n]} \mathsf{coin}'_j / \mathsf{coin}_j$. Note that this sigma protocol can be transformed into a non-interactive proof of knowledge in the random oracle model via the Fiat-Shamir heuristic [13].

A. Initialization

We define the algorithms B.Init and U.Init before we proceed any further. In the blockchain program, we initialize a set called FrozenCoins that will contains coins that are frozen. Therefore, we have

- B.Init():
  - $\mathsf{FrozenCoins} \leftarrow \emptyset$
  - $\mathsf{FreezeRecords} \leftarrow \emptyset$
- U.Init():
  - $\mathsf{Identifiers} \leftarrow \emptyset$
  - $\mathsf{SecretElems} \leftarrow \emptyset$

A set of m participants decide to create a private smart contract by specifying an m-ary smart contract function f along with a set of participant identities P (i.e. their pseudonyms in the underlying cryptocurrency). For simplicity, we often assume that P contains all participants we have defined in our security definition i.e. $m = n$ and $P = \{\mathcal{P}_i\}_{i \in [n]}$ but this is without loss of generality since it is easy to see that any subset of $m \leq n$ participants can be supported. We assume that each party gives their consent to the smart contract function f that is used; that is, each party has knowledge of the code of f, inspects it and agrees that it is fair. Specifying and validating the correctness of f is beyond the scope of this paper and literature abounds on this. For example, we would assume that each party contributes its own preconditions and post-conditions to the smart contract code in order to establish consensus. Although we do not aim to keep f itself private in this paper, the astute reader will observe that it is straightforward to achieve this with our protocol. We now specify the U.Create algorithm which takes as input a smart contract function f and a set of participants P and returns a unique identifier for this pair. Let H be a collision-resistant hash function, modelled as a random oracle in the security analysis (as we shall see later), that maps an arbitrary-length string to a uniformly random string $\{0,1\}^\lambda$ where λ is the security parameter.

- U.Create(f, P):
  - $\mathsf{id} \leftarrow H(f \parallel P)$
  - $\mathsf{Identifiers} \leftarrow \mathsf{Identifiers} \cup \{(\mathsf{id}, f, P)\}$
  - Return id.

Much of what we have defined thus far is "boilerplate" and possibly common to any PSC evaluation protocol. Next, we define the algorithms used in the freeze phase.

B. Freezing Coins

In the freeze phase of the protocol, we employ the sigma protocol from [14] that proves in zero knowledge that a commitment commits to 0 or 1. We use its transformation

(in the random oracle model) into a NIZK via the Fiat-Shamir heuristic [13], and denote the scheme by $\mathsf{BNIZK} := (\mathsf{BNIZK.Prove}, \mathsf{BNIZK.Verify})$. We have already established that $\mathbb{V}$ denotes the set of valid amounts of currency. Here we ordain the upper limit L of a valid amount of currency to be 2^ℓ such that a valid amount of currency lies in the range $[0, 2^\ell)$ for some ℓ. In addition, we decompose an output coin coin' into a product of ℓ commitments corresponding to the binary decomposition of the output value. For each $k \in \{0, \ldots, \ell-1\}$, the party generates two commitments, one of them commits to zero and the other commits to one. Importantly, it permutes the order of these commitments when it sends them to the blockchain so that the blockchain does not know whether the first or the second one commits to zero etc. More precisely, for each $k \in \{0, \ldots, \ell-1\}$, we sample a bit $b_k \xleftarrow{\$} \{0,1\}$ and compute $c^{(k,b_k)}$ as a commitment to b_k and $c^{(k,1-b_k)}$ as a commitment to $1-b_k$. We prove in zero knowledge that both commitments commit to a bit using BNIZK and sends both commitments along with the proofs to the blockchain. These steps take place in the $\mathsf{U.Freeze}$ algorithm, which we now formally define:

- $\mathsf{U.Freeze}(\mathsf{id}, (\$\mathsf{val}, r), \mathsf{in})$:
  - Assert $(\mathsf{id}, f, P) \in \mathsf{Identifiers}$
  - $\mathsf{coin} \leftarrow \mathsf{Com}(\$\mathsf{val}; r)$
  - For $k \in \{0, \ldots, \ell-1\}$:
    - $b_k \xleftarrow{\$} \{0,1\}$
    - $s^{(k,0)} \xleftarrow{\$} \mathbb{Z}_p$
    - $s^{(k,1)} \xleftarrow{\$} \mathbb{Z}_p$
    - $c^{(k,b_k)} \leftarrow \mathsf{Com}(b_k; s^{(k,b_k)})$
    - $c^{(k,1-b_k)} \leftarrow \mathsf{Com}(1-b_k; s^{(k,1-b_k)})$
    - $\pi^{(k,b_k)} \leftarrow \mathsf{BNIZK.Prove}(c^{(k,b_k)}, (s^{(k,b_k)}, b_k))$
    - $\pi^{(k,1-b_k)} \leftarrow \mathsf{BNIZK.Prove}(c^{(k,1-b_k)}, (s^{(k,1-b_k)}, 1-b_k))$
  - $\mathsf{SecretElems} \leftarrow \mathsf{SecretElems} \cup \{(\mathsf{id}, \$\mathsf{val}, r, \mathsf{in}, \{(c^{(k,0)}, c^{(k,1)}, s^{(k,0)}, s^{(k,1)})\}_{0\le k<\ell})\}$
  - $\mathsf{msg} \leftarrow (\mathsf{freeze}, \mathsf{id}, P, \mathsf{coin}, \{((c^{(k,b_k)}, \pi^{(k,b_k)}), (c^{(k,1-b_k)}, \pi^{(k,1-b_k)}))\}_{0\le k<\ell})$
  - Return msg

where id is the contract identifier, P is the set of contract participants i.e. $\{\mathcal{P}_i\}_{i\in[n]}$.

The blockchain responds by executing the following steps, which are part of the $\mathsf{B.Freeze}$ algorithm that we now define:

- $\mathsf{B.Freeze}(\mathsf{msg}, \mathcal{P})$:
  - Parse msg as $(\mathsf{freeze}, \mathsf{id}, P, \mathsf{coin}, \{((c^{(k,b_k)}, \pi^{(k,b_k)}), (c^{(k,1-b_k)}, \pi^{(k,1-b_k)}))\}_{0\le k<\ell})$
  - For $k \in \{0, \ldots, \ell-1\}$:
    1) Assert $\mathsf{BNIZK.Verify}(c^{(k,b_k)}, \pi^{(k,b_k)})$
    2) Assert $\mathsf{BNIZK.Verify}(c^{(k,1-b_k)}, \pi^{(k,1-b_k)})$
    3) $C^{(k)} \leftarrow \{c^{(k,b_k)}, c^{(k,1-b_k)}\}$
  - $\mathsf{FreezeRecords} \leftarrow \mathsf{FreezeRecords} \cup \{(\mathsf{id}, \mathcal{P}, \mathsf{coin}, \{C^{(k)}\}_{0\le k<\ell})\}$
  - $\mathsf{FrozenCoins} \leftarrow \mathsf{FrozenCoins} \cup \{\mathsf{id}, \mathsf{coin}\}$
  - Return 1

C. Computation

Consider a smart contract function $f : (\mathbb{V} \times \{0,1\}^*)^n \to \mathbb{V}^n \times \{0,1\}^*$. Let $\$\mathsf{val}_i$ and in_i be the input currency value and input string of party $\mathcal{P}_i$ respectively.
All parties obtain the tuples $(\mathsf{id}, \mathcal{P}_i, \mathsf{coin}_i, \{C_i^{(k)}\}_{0\le k<\ell})$ from the blockchain.

We now define the following function $\hat{f}$. We use the notation $\$\mathsf{val}[k]$ to denote the k-th bit of $\$\mathsf{val}$.

MPC Function $\hat{f}$

$\hat{f}((\$\mathsf{val}_1, r_1, \mathsf{in}_1, (c_1^{(k,0)}, c_1^{(k,1)}, s_1^{(k,0)}, s_1^{(k,1)})_{0\le k<\ell}),$
$\ldots, (\$\mathsf{val}_n, r_n, \mathsf{in}_n, (c_n^{(k,0)}, c_n^{(k,1)}, s_n^{(k,0)}, s_n^{(k,1)})_{0\le k<\ell}))$
$((\$\mathsf{val}'_1, \ldots, \$\mathsf{val}'_n), \mathsf{out}) \leftarrow$
$f((\$\mathsf{val}_1, \mathsf{in}_1), \ldots, (\$\mathsf{val}_n, \mathsf{in}_n))$
For all $j \in [n]$:
For all $k \in \{0, \ldots, \ell-1\}$:
$c_j^{(k)} \leftarrow c_j^{(k,\$\mathsf{val}'_j[k])}$
$r \leftarrow \sum_{j\in[n]}(\sum_{0\le k<\ell} 2^k \cdot s_j^{(k,\$\mathsf{val}'_j[k])} - r_j)$
Compute proof π with Schnorr protocol with knowledge r and base h
Return $(((c_1^{(k)})_{0\le k<\ell}, \ldots, (c_n^{(k)})_{0\le k<\ell}), \pi, \mathsf{out})$

Fig. 6. Definition of the function that is evaluated by the MPC protocol. This function is built from f, the smart contract function.

We can observe two things here. Firstly, all parties must collaborate to produce an accepting proof π, which serves to validate the zero-sum constraint. The second point is that symmetric encryption of the $\$\mathsf{val}'_i$ is not needed since the value $\$\mathsf{val}'_i$ can be read by $\mathcal{P}_i$ (and only $\mathcal{P}_i$) from $((c_i^{(k)})_{0\le k<\ell}$.

Now we define the $\mathsf{U.PrepareCompute}$ algorithm:

- $\mathsf{U.PrepareCompute}(\mathsf{id})$:
  - Assert $(\mathsf{id}, f, P) \in \mathsf{Identifiers}$
  - Assert $(\mathsf{id}, \$\mathsf{val}, r, \mathsf{in}, \{(c^{(k,0)}, c^{(k,1)}, s^{(k,0)}, s^{(k,1)})\}_{0\le k<\ell}) \in \mathsf{SecretElems}$
  - Fetch $(\mathsf{id}, \mathcal{P}_i, \mathsf{coin}_i, \{C_i^{(k)}\}_{0\le k<\ell})$ from $\mathcal{B}.\mathsf{FreezeRecords}$ for each $\mathcal{P}_i \in P$ (recall we assume that $|P| = n$)
  - Define function $\hat{f}$ as in Figure 6 using function f.
  - $x \leftarrow (\$\mathsf{val}, r, \mathsf{in}, (c^{(k,0)}, c^{(k,1)}, s^{(k,0)}, s^{(k,1)})_{0\le k<\ell})$
  - Return $(\hat{f}, P, x)$

D. Finalization

Since this protocol supports immediate closure, a valid finalization message from a single party is sufficient to instigate contract closure. The MPC protocol is executed between the parties during the computation phase and it concludes with a final output y that is accessible to all the parties. Therefore, we now define the algorithm, namely $\mathsf{U.Finalize}$, that processes this y and produces a finalize message to be

sent to the blockchain along with extracting all output coins $\mathsf{coin}'_1, \ldots, \mathsf{coin}'_n$.

- U.Finalize(id, y):
  - Assert $(\mathsf{id}, f, P) \in \mathsf{Identifiers}$
  - Assert $(\mathsf{id}, \$\mathsf{val}, r, \mathsf{in}, \{(c^{(k,0)}, c^{(k,1)}, s^{(k,0)}, s^{(k,1)})\}_{0 \le k < \ell}) \in \mathsf{SecretElems}$
  - Fetch $(\mathsf{id}, \mathcal{P}_i, \mathsf{coin}_i, \{C_i^{(k)}\}_{0 \le k < \ell})$ from $\mathcal{B}$.FreezeRecords for each $\mathcal{P}_i \in P$ (recall we assume that $|P| = n$)
  - Parse y as $(((c_1^{(k)})_{0 \le k < \ell}, \ldots, (c_n^{(k)})_{0 \le k < \ell}), \pi, \mathsf{out})$ For all $j \in [n]$:
    1) For all $k \in \{0, \ldots, \ell - 1\}$:
       a) Assert $c_j^{(k)} \in C_j^{(k)}$
    2) $\mathsf{coin}'_j \leftarrow \prod_{0 \le k < \ell} (c_j^{(k)})^{2^k}$
  - $c \leftarrow \prod_{j \in [n]} \mathsf{coin}'_j / \mathsf{coin}_j$
  - Assert $\mathsf{Schnorr.Verify}((c, h), \pi)$
  - Let i' be the index of *this* party
  - For $k \in \{0, \ldots, \ell - 1\}$:
    1) If $c_{i'}^{(k)} = c_{i'}^{(k,0)}$:
       $*$ $v_k \leftarrow 0$
       Else:
       $*$ $v_k \leftarrow 1$
  - $\$\mathsf{val}' \leftarrow \sum_{0 \le k < \ell} v_k \cdot 2^k$
  - $r' \leftarrow \sum_{0 \le k < \ell} s_{i'}^{(k, v_k)} \cdot 2^k$
  - Fetch $\{\mathsf{coin}_{i'}\}$ from $\mathcal{B}$.FrozenCoins for $\mathcal{P}_{i'} \in P$
  - $\mathsf{coin}'_{i'} \leftarrow \mathsf{Com}(\$\mathsf{val}', r')$
  - Replace $\mathsf{coin}_{i'}$ in $\mathcal{B}$.FrozenCoins with $\mathsf{coin}'_{i'}$
  - $\mathsf{msg} \leftarrow (\mathsf{finalize}, \mathsf{id}, ((c_1^{(k)})_{0 \le k < \ell}, \ldots, (c_n^{(k)})_{0 \le k < \ell}), \mathsf{out}, \pi)$
  - Return $(\mathsf{msg}, \mathsf{coin}'_1, \ldots, \mathsf{coin}'_n, \mathsf{out}, (\$\mathsf{val}', r'))$

The message that is returned in the algorithm above is sent by the contract participant to the blockchain, which processes it in the following algorithm B.Finalize:

- B.Finalize(msg):
  - Parse msg as $(\mathsf{finalize}, \mathsf{id}, ((c_1^{(k)})_{0 \le k < \ell}, \ldots, (c_n^{(k)})_{0 \le k < \ell}), \mathsf{out}, \pi)$
  - Assert $(\mathsf{id}, \mathcal{P}_i, \mathsf{coin}_i, \{C_i^{(k)}\}_{0 \le k < \ell}) \in \mathsf{FreezeRecords}$ for each $\mathcal{P}_i \in P$ (recall we assume that $|P| = n$)
  - For all $j \in [n]$:
    1) For all $k \in \{0, \ldots, \ell - 1\}$:
       a) Assert $c_j^{(k)} \in C_j^{(k)}$
    2) $\mathsf{coin}'_j \leftarrow \prod_{0 \le k < \ell} (c_j^{(k)})^{2^k}$
  - $c \leftarrow \prod_{j \in [n]} \mathsf{coin}'_j / \mathsf{coin}_j$
  - Return $\mathsf{Schnorr.Verify}((c, h), \pi)$

IV. Security Analysis

In this section we give a proof of security for our PSC evaluation protocol $\pi := (\mathsf{Com}, B, U)$ defined in the previous section. More precisely, we prove that π is t-secure for all $t < n$ in accordance with Definition 3 assuming the hardness of the discrete logarithm problem in the random oracle model. We must construct a simulator $\mathcal{S}$ that uses an adversary $\mathcal{A}$ to simulate a transcript that is computationally indistinguishable to any PPT distinguisher $\mathcal{D}$ from the transcript produced in the real world execution of π. To correctly make use of $\mathcal{A}$ in the simulation, we must correctly simulate $\mathcal{A}$'s view which requires us to simulate the honest parties, the blockchain and $\mathcal{M}$. The simulator is permitted access to an ideal functionality $\mathcal{F}$ that computes the smart contract function. Note that the current version of our protocol, for simplicity, does not provide verification for the smart contract output string out nor does our correctness condition check its veracity. We note that it is straightforward to modify the protocol such that the output string is included in the Schnorr proof such that the resulting proof serves to encompass a "vote" amongst all participants of the correctness of out since the MPC program is assumed to be correct.

Theorem 4: Assuming a t-secure MPC protocol, our PSC evaluation protocol is t-secure for all $t < n$ assuming the hardness of the discrete logarithm problem in the random oracle model.

Proof: We prove the theorem via a hybrid argument. Our goal is to move to a hybrid where we no longer depend on the inputs of the honest parties i.e. $(x_i := (\$\mathsf{val}_i, \mathsf{in}_i))_{i \in \bar{I}}$. Recall that I denotes the set of indices of the corrupt parties whereas $\bar{I}$ denotes the set of indices of the honest parties. The reductions underlying indistinguishability of the various hybrids are relatively straightforward and for brevity sake, we omit them here.

Hybrid 0: This is the real system.

Hybrid 1: The change we make in this hybrid is to simulate $\mathcal{M}$ and its output y that it sends to all parties. We receive the $(\mathsf{input}, \cdot)$ messages from the adversary for the corrupted parties and obtain the inputs of the corrupted parties. In this hybrid, we still have access to the inputs of the honest parties. We obtain y be computing the function $\hat{f}$ in Figure 6 and send it to all parties as an $(\mathsf{output}, \cdot)$ message, therefore simulating $\mathcal{M}$. This hybrid is distributed identically to the previous hybrid.

Hybrid 2: The change we make in this hybrid is to the Schnorr proof π component of y (i.e. the output of $\hat{f}$). Instead of computing the proof normally, we use the simulator for the NIZK. Therefore, we no longer need the randomness r_i and $s_i^{(k,b)}$ to generate π and it can still be generated even when the zero-sum condition does not hold. The hybrids are indistinguishable from the zero-knowledge property of the Schnorr protocol.

For $i \in \bar{I}$ and $k \in \{0, \ldots, \ell - 1\}$:

Hybrid 3, i, k: In this hybrid, we do not compute the proofs $\pi_i^{(k,0)}$ and $\pi_i^{(k,1)}$ using BNIZK.Prove but instead use the simulator for BNIZK. Indistinguishability of the hybrids follows form the zero-knowledge property of BNIZK.

For $i \in \bar{I}$ and $k \in \{0, \ldots, \ell - 1\}$ and $b \in \{0, 1\}$:

Hybrid 4, i, k, b: In this hybrid, we compute the commitment $c_i^{(k,b)}$ as a commitment to a uniformly random element of $\mathbb{Z}_p$. Indistinguishability of the hybrids from the perfectly blinding property of Pedersen commitments.

For $i \in \bar{I}$:

Hybrid 5, i: In this hybrid, we change the input coin commitment coin_i to a commitment to a uniformly random element of $\mathbb{Z}_p$. Indistinguishability of the hybrids from the perfectly blinding property of Pedersen commitments.

Hybrid 6: In this hybrid, we change how we compute $(\$\mathsf{val}'_i)_{i\in I}, \mathsf{out})$ by instead calling the ideal functionality $\mathcal{F}$ and sending it the inputs of the corrupted parties, which have been obtained from $\mathcal{A}$. This hybrid is distributed identically to the previous hybrid. This hybrid has no dependence on the inputs and outputs of the honest parties and hence we obtain a simulator that uses the adversary $\mathcal{A}$ to produce a transcript that is computationally indistinguishable from the real world (Hybrid 0). The result follows. ■

V. Conclusion and Future Work with NI-PSC

The protocol presented above facilitates Interactive Private Smart Contracts (I-PSC) wherein an Interactive protocol (MPC) between the parties is executed replacing the manager in Hawk in order to determine the outcome of the smart contract and reach consensus on the payout distribution. At least one of the parties must then notify the blockchain of the outcome so that the blockchain can instigate contract closure. A disadvantage of I-PSC is that all parties must remain online for computation to proceed. A more desirous and powerful alternative is a protocol facilitating Non-Interactive Private Smart Contracts (NI-PSC) wherein the blockchain performs the computation of the smart contract while retaining input privacy; that is, its computation is oblivious to the inputs. Parties need not be online during computation nor are they required to interact. Furthermore, there is no possibility for aborts as in the interactive case and the execution of smart contracts can be scheduled for specified times and use public information such as stock prices, a situation that may lead to aborts in the interactive case e.g. a stock price that negatively affects a party's position in a contract may encourage that party to abort the protocol. In a NI-PSC, only one round of interaction with the blockchain is needed to execute a private smart contract, namely the freeze round. In the freeze round, each party sends a message containing (1). a unique identifier id that identifies the contract (e.g. a hash of the contract function, execution number and set of participants); (2). the set of participants P; (3). the party's input coin; and (4). protocol-specific information. The round is complete when the blockchain receives a valid freeze message with matching (id, P) for every $\mathcal{P} \in P$. The blockchain can then independently compute the output of the smart contract and instigate contract closure.

A. Realizing NI-PSC

NI-PSC can be realized with a primitive known as Non-Interactive MPC (NI-MPC). We assume we are in the PKI (public key infrastructure) model where the parties know each other's authentic public key. Then, in NI-MPC, each party need only send one message, which contains its input and the function to be evaluated. Given the messages from all parties, the desired function can be computed on the parties' inputs. Choosing the function f' described in our protocol above as the function computed by NI-MPC would fully realize NI-PSC. Known constructions of NI-MPC rely on indistinguishability obfuscation (iO) [15], [16], which has not been realized from standard assumptions. Basing NI-MPC on standard assumptions is a stubbornly difficult open problem because NI-MPC for general functions implies iO. Since "pure" NI-PSC is so difficult to achieve under standard assumptions, we intend to consider various relaxations as part of future work in order to give constructions under standard assumptions.

References

[1] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 459–474.

[2] A. Miller, M. Möser, K. Lee, and A. Narayanan, "An empirical analysis of linkability in the monero blockchain," *arXiv preprint arXiv:1704.04299*, 2017.

[3] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016, pp. 839–858.

[4] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1353–1370.

[5] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 781–796.

[6] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2010, pp. 321–340.

[7] R. Canetti, U. Feige, O. Goldreich, and M. Naor, "Adaptively secure multi-party computation," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 639–648.

[8] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 136–145.

[9] T. Araki, A. Barak, J. Furukawa, M. Keller, Y. Lindell, K. Ohara, and H. Tsuchida, "Generalizing the spdz compiler for other protocols," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 880–895.

[10] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*. Springer, 1991, pp. 129–140.

[11] J. Groth, "Homomorphic trapdoor commitments to group elements." *IACR Cryptol. ePrint Arch.*, vol. 2009, p. 7, 2009.

[12] C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *Conference on the Theory and Application of Cryptology*. Springer, 1989, pp. 239–252.

[13] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 186–194.

[14] J. Groth and M. Kohlweiss, "One-out-of-many proofs: Or how to leak a secret and spend a coin," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 253–280.

[15] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (im) possibility of obfuscating programs," in *Annual international cryptology conference*. Springer, 2001, pp. 1–18.

[16] P. Ananth, A. Jain, and A. Sahai, "Indistinguishability obfuscation from functional encryption for simple functions," *Eprint*, vol. 730, p. 2015, 2015.