# Multiverse of HawkNess: A Universally-Composable MPC-based Hawk Variant*

Aritra Banerjee[1] and Hitesh Tewari[2]

[1] ADAPT Centre, School of Computer Science and Statistics,
Trinity College Dublin, Dublin, Ireland
abanerje@tcd.ie
[2] School of Computer Science and Statistics,
Trinity College Dublin, Dublin, Ireland
htewari@tcd.ie

**Abstract.** The evolution of Smart contracts in recent years inspired a crucial question: Do smart contract evaluation protocols provide the required level of privacy when executing contracts on the Blockchain? The Hawk (IEEE S&P '16) paper introduces a way to solve the problem of privacy in smart contracts by evaluating the contracts *off-chain*, albeit with the trust assumption of a *manager*. To avoid the partially trusted manager altogether, a novel approach named zkHawk (IEEE BRAINS '21) explains how we can evaluate the contracts privately off-chain using a multi-party computation (MPC) protocol instead of trusting said manager. This paper dives deeper into the detailed construction of a variant of the zkHawk protocol titled *V-zkHawk* using formal proofs to construct the said protocol and model its security in the universal composability (UC) framework (FOCS '01). The V-zkHawk protocol discussed here does not support immediate closure, i.e, all the parties ($n$) have to send a message to inform the blockchain that the contract has been executed with corruption allowed for up to $t$ parties, where $t < n$. In the most quintessential sense, the V-zkHawk is a variant because the outcome of the protocol is similar (i.e., execution of smart contract via an MPC function evaluation) to zkHawk, but we modify key aspects of the protocol essentially creating a small trade-off (removing immediate closure) to provide UC (stronger) security. The V-zkHawk protocol leverages joint Schnorr signature schemes, encryption schemes, Non-Interactive Zero-Knowledge Proofs (NIZKs), and commitment schemes with Common Reference String (CRS) assumptions, MPC function evaluations, and assumes the existence of asynchronous, authenticated broadcast channels. We achieve malicious security in a dishonest majority setting in the UC framework.

**Keywords:** zkHawk · Hawk · MPC · V-zkHawk · NIZKs · Universal Composability

* This work is done as a part of first author's PhD research. 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).

## 1 Introduction

Smart contracts have existed in the literature prior to their integration with cryptography and in cryptocurrencies. One of the oldest examples of smart contracts is the vending machine [48]. The implementation of early vending machines dates back to the $1^{st}$ century AD in Egypt during the reign of the Roman Emperor Augustus Caesar. However, modern vending machines as we know them today did not come into existence until the early 1880s. Fast-forward to the 21st century, interest in smart contracts as a cryptographic application has risen dramatically in the past decade. It all started with the 2013 [13] Ethereum paper that introduced smart contract as a decentralized application and integrated the concept of smart contract code evaluations in cryptocurrencies. Ethereum focused on evaluating contracts on the public blockchain which resulted in a non-private smart contract setting. This made users apprehensive about including contracts containing sensitive information onto the Ethereum smart contracts. In terms of transaction privacy, Zcash [47] became one of the leading cryptocurrencies to provide a private, scalable, and fast coin transfer mechanism which took the concept of privacy leaps and bounds ahead of Bitcoin. This was due to the significant improvement in the domain of zk-SNARKs [3, 31, 32] (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge).

The idea of Hawk [38] stemmed from uniting the concepts of transaction privacy and the execution of smart contracts. Hawk achieves transaction privacy by evaluating contracts partially *off-chain* with the trust assumption of a *manager*. The manager is minimally trusted and can only be trusted to evaluate the smart contract and provide the correct output. The manager cannot be trusted to maintain the security or privacy of the contract execution. Kosba et al. [38] observed that in Hawk the manager can be replaced by running an MPC protocol between the parties. However, they pointed out that this approach would be currently impractical. Indeed, it can be easily seen that by applying MPC to the design of Hawk as it is incurs the prohibitively expensive overhead of executing a zk-SNARK proof [31,32] within an MPC program (a circuit in practice). Hence, we propose to remove the zk-SNARK proof from the MPC program [4]. However, this leaves us with a considerable challenge - How do we prove to the blockchain that the sum of the incoming balances in the smart contract is equal to the sum of the outgoing balances? As a solution, in V-zkHawk we compute a relatively practical KDK (Kursawe, Danezis and Kohlweiss) [40] 2-round MPC sum check with broadcasting and allow signatures to prove within an MPC function such that all parties contributed to the contract execution. The resulting effect guarantees that the difference between the sum of the output balances and the sum of the input balances is zero.

Recent implementations of MPC protocols including the MP-SPDZ framework [36] have proven to be quite efficient while proving security against malicious security models. Starting from Yao's protocols in the early 1980s [50] to the GMW protocol in the late 80s [45] to more recent protocols like Danish Sugar beet auction [6] or SPDZ [25] or MACE [1] for fault tolerance, the MPC journey has been profound. The applications of MPC grows each day and hence

the motivation to apply MPC to solve the inherent privacy issues for evaluating a private smart contract.

**A Simple Private Smart Contract:** For a clearer understanding on how a private smart contract works let us consider the example of Crowdfunding (See Figure 1). Consider the smart contract function $\hat{f}$ as the lending platform. The lenders $(len_1, \ldots, len_k)$ and the borrower $(bo)$ are the *participants* $(n)$. The loan amount required and lending amounts by each lender are kept as secret. Let the required loan amount be $\$\mathsf{x}$ and $\$\mathsf{bo} \leftarrow 0$.

### 1.1 Our Contributions:

In this paper, we present V-zkHawk which is a variant of the zkHawk [4] protocol such that the smart contracts are evaluated off-chain without the trust assumption of a *manager*. This can be achieved using an MPC [6, 11, 25] function to evaluate the smart contract and send the output back to the relevant party off-chain. Such a protocol guarantees Strong Input/Output privacy [3]. We provide formal proofs for the construction of the V-zkHawk protocol. The proofs and theorems in this paper are modeled in the UC [14] framework. We define the ideal functionalities for our real world protocols that also interacts with a simulator (mimicking an adversary in real world) such that no PPT (Probabilistic Polynomial Time) environment can distinguish between ideal and real worlds. Our work utilizes commitment schemes which is impossible to prove UC secure in the standard model [18]. Hence, we will be employing the CRS model [5, 14] with a secure broadcast protocol for the setup/preprocessing stage. The CRS and broadcast protocols also helps NIZK protocols realize the NIZK functionalities in the ideal world. We define security against malicious (active) adversaries in the dishonest majority setting.

### 1.2 Existing Private Smart Contract Protocols

We have already described above the motivation behind Hawk and how it works. In this section, we will discuss a few more contemporary Private Smart Contract (PSC) protocols apart from Hawk [38].

- **zkay**: zkay [49] extends on Ethereum smart contracts to allow users to share encrypted data on the blockchain. This is not an off-chain protocol but rather works on private data on-chain to prove that data is correctly encrypted and that the smart contract executions are correct.
- **Arbitrum**: Arbitrum [35], unlike Ethereum, uses virtual machines (VMs) to implement smart contracts. Each party can create smart contract functionality by writing a code that the VMs then implement off-chain. Only verifiable digital signatures are needed to ensure that the parties have agreed on the

[3] Strong Input/Output privacy is defined such that parties' inputs/outputs are not leaked to both the public and to each other throughout the contract execution.

**Smart Contract Function $\hat{f}$**

- $\hat{f}(\$\mathsf{x}, (\$\mathsf{len}_1), \ldots, (\$\mathsf{len}_n), \$\mathsf{bo})$
  - $\$\mathsf{len}_x \leftarrow \sum_{i \in [n]} \$\mathsf{len}_i$
  - If $\$\mathsf{len}_x > \$\mathsf{x}$
    - $\$\mathsf{bo} \leftarrow \$\mathsf{len}_x$
    - For $i \in \{1, \ldots, n\}$
      - $\$\mathsf{len}_i \leftarrow 0$

  **Contract Closure:**
  - Return $\$\mathsf{len}_i$ to $len_i$
  - Send $\$\mathsf{bo}$ to $bo$

**Fig. 1.** Example of a Crowdfunding Private Smart Contract function $\hat{f}$

VMs functionality. This ensures that the contract is executed off-chain. Similar to Hawk, Arbitrum also relies on a manager who is one of the parties to monitor the behaviour of the VMs. It also relies on an honest majority setting for privacy guarantees.

- **Kachina**: Kachina [37], a more recent PSC protocol that models its security in the UC framework. This provides a more private and secure PSC evaluation for Zcash privacy-preserving payment system, but it does not bode well with a dishonest majority in a malicious setting.
- **Zether**: Zether [12], being a retro-fitted privacy-preserving smart contract protocol for currency can be utilized only in places like sealed-bid auctions or crowdfunding. This protocol cannot be utilised in non-monetary smart contract applications like e-Voting, Rock-Paper-Scissors, etc.
- **Shielded Computations in Smart Contracts**: Recent work by V. Botta et al. [10] leverages on-chain MPC protocols for executing smart contracts by forking blockchains like Ethereum. It works for both honest and dishonest majority setting.
- **ShadowEth**: ShadowEth [51] utilizes the Trusted Execution Environment (TEE) to generate private smart contract evaluations for public blockchains like Ethereum. It utilizes the Intel SGX [23] hardware enclave to implement the protocol that creates an isolated secure environment running parallel to the OS.

### 1.3 Outline of the Paper

The paper starts with an introduction of smart contracts, zkHawk, contemporary private smart contract protocols and what we are trying to achieve via V-zkHawk. In Section 2 we define the notations used throughout the paper along with definition of cryptographic primitives used in the V-zkHawk protocol and a general overview of the UC framework. In Section 3 we elaborate on the

construction of V-zkHawk protocol along with the subroutines that are running inside the protocol such that an adversary can corrupt a party running a specific subroutine or protocol. In Section 4 we define the ideal functionalities for the protocols defined in Section 3 and simulate its security using a simulator in the UC framework. Finally, in Section 5 we conclude that the security achieved via the proposed V-zkHawk smart contract protocol in UC secure [14] against *malicious*, *static* adversaries for a dishonest majority and how we can further refine this protocol in the Global UC (GUC) [2,17] model along with considering further optimisations to the efficiency of the protocol.

## 2 Preliminaries

Let $\lambda \in \mathbb{N}$ be the security parameter. We denote the value $[m]$ as an iteration of values from 1 to $m$. $\mathsf{Com}$ is a commitment scheme used to establish input and output coin commitments. $A \underset{C}{\approx} B$ denotes $A$ is computationally indistinguishable from $B$ to a $\mathsf{PPT}$ observer (here *environment*).

Let $L$ be an NP-language and $R$ be a binary witness relation in $L$ such that $L = \{x \mid \exists\ w : R(x, w) = 1\}\ \forall\ (x, w) \in R$ where $x$ is the statement to be proved and $w$ is the witness.

**Definition 1 (Non-Interactive Zero Knowledge Proofs (NIZKs)).** *A NIZK proof $\pi$ is a tuple of algorithms* $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ *such that:*

$\mathsf{CRSGen}(1^\lambda)$ *: This algorithm generates a common reference string* $\mathsf{CRS}$ *using a security parameter $\lambda$ as input.*

$\mathsf{Prove}(\mathsf{CRS}, x, w)$*: This algorithm generates a proof $\phi$ from the common reference string* $\mathsf{CRS}$*, $x$ and $w$.*

$\mathsf{Verify}(\mathsf{CRS}, x, \phi)$*: This algorithm generates an output $b \in \{0, 1\}$ leveraging the proof $\phi$,* $\mathsf{CRS}$ *and $x$.*

We require our NIZKs to follow *perfect completeness* and *perfect soundness* for the V-zkHawk protocol. Since, we are dealing with static adversaries we need not worry about adaptive *non-erasure* or adaptive *witness-indistinguishability* properties of NIZKs. For brevity sake we will not go too much into details of these properties in this paper and defer the reader for further reading on NIZKs to [28,33].

Let us consider a signature scheme $\Sigma_{sig} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ such that the secret key elements belong in some group $(\mathbb{X}, +)$ and public key elements belong in some group $(\mathbb{Y}, \cdot)$. We require the signature scheme leveraged in V-zkHawk (Schnorr [43,44] or BLS [8,9] or ECDSA [34]) to be secret key to public key homomorphic. This concept was introduced in [27].

**Definition 2 (Secret to Public Key Homomorphic).** *We can say that a signature scheme $\Sigma_{sig}$ provides secret to public key homomorphism, if $\exists$ a map $\delta$: $\mathbb{X} \rightarrow \mathbb{Y}$ such that:*

- $\forall\ s_k, s'_k \in \mathbb{X}$ *it satisfies the relation* $\delta(s_k + s'_k) = \delta(s_k) \cdot \delta(s'_k)$
- $\forall\ (s_k, p_k) \leftarrow \mathsf{KeyGen}$*, it satisfies the relation* $p_k = \delta(s_k)$

### 2.1 Universal Composability

$\mathcal{A}$ is the adversary in the real world which interacts with the real parties $\mathcal{P}$ and $\mathcal{S}$ is the simulator in the ideal world imitating the adversary in the real world. Further, we denote $\mathcal{F}(.)$ as the functionality in the ideal world interacting with the dummy parties $\mathcal{P}'$ and the simulator $\mathcal{S}$. $\mathcal{Z}$ is a $\mathsf{PPT}$ environment which interacts with the ideal world protocol $\phi$ and real world protocol $\pi$ as well as $\mathcal{A}$ and $\mathcal{S}$. In the UC framework, the environment $\mathcal{Z}$ interacts twice with the protocol execution: Firstly, before the execution of the protocol where it can provide inputs to the parties and the adversary and Secondly, when the protocol execution is terminated, it collects the outputs from the adversary and the parties. Then, it outputs a single bit which is basically $\mathcal{Z}$ specifying whether it thinks it interacted with $\pi$ or $\mathcal{F}$. We define a $\mathsf{UC-secure}$ protocol as follows:

**Definition 3.** *An n-party (where $n \in \mathbb{N}$) protocol $\pi$ is* $\mathsf{UC-secure}$ $\forall$ $\mathcal{Z}$ *if for any* $\mathcal{A}$ $\exists$ $\mathcal{S}$*:*

- *The protocol* $\pi$ $\mathsf{UC-emulates}$ $\phi$*, i.e,* $\mathsf{UC-IDEAL}_{\phi,\mathcal{S},\mathcal{Z}} \underset{C}{\approx} \mathsf{UC-REAL}_{\pi,\mathcal{A},\mathcal{Z}}$
- *The protocol* $\pi$ $\mathsf{UC-realizes}$ $\mathcal{F}$

Informally, we say that a protocol $\pi$ **securely emulates** a protocol $\phi$ if for any given input the probability that $\mathcal{Z}$ will output 1 after interacting with $\pi$ and $\mathcal{A}$ will differ negligibly to the probability that $\mathcal{Z}$ will output 1 after interacting with $\phi$ and $\mathcal{S}$. We also define that $\pi$ $\mathsf{UC-realizes}$ $\mathcal{F}$ if given the adversary $\mathcal{A}$ and $\mathcal{Z}$ interacts at any point during the execution of the protocol, $\mathcal{Z}$ still cannot tell the difference whether it is interacting $\mathcal{A}$ and $\pi$ or $\mathcal{S}$ and $\mathcal{F}$.

**Note:** We write $\pi$ $\mathsf{UC-realizes}$ $\mathcal{F}$ instead of $\pi$ $\mathsf{UC-realizes}$ $\phi$ because all the parties in $\phi$ are dummy parties $\mathcal{P}'$ and they just forward their input and output to $\mathcal{F}$ for secure computation. Hence, we replace $\phi$ with $\mathcal{F}$ for most notations as $\mathcal{Z}$ is basically interacting with $\mathcal{F}$ and not the other dummy parties in $\phi$.

From [14,16], we borrow the knowledge of Interactive Turing Machines (ITMs) which we leverage in the V-zkHawk protocol. Our computation consists of several instances of ITMs $\mathcal{M}_1, \ldots, \mathcal{M}_n$ for $n$-parties that can write on the externally writable tapes of each other. These are usually commands to activate a Turing machine or run a certain protocol. An ITI is an instance of an ITM. To help distinguish among different protocol executions in a system, we assume that the contents of the identity tape of each party consists of two fields, namely a Session ID ($\mathsf{sid}$) and a Party ID ($\mathsf{pid}$). We defer the reader to [14,16,19] for more detailed understanding of ITMs and ITIs.

### 2.2 Smart Contract

In zkHawk/V-zkHawk, a *coin* is an anonymized marker of an amount of currency used by the blockchain. A coin can be spent in a transaction such that it cannot be linked to its owner. It has an associated value that is hidden. A coin is realized with a commitment scheme; that is, a coin $\mathsf{coin} = \mathsf{Com}(\$\mathsf{val}, r)$ is a commitment

to a value of currency $\$\mathsf{val}$ with randomness $r$. The randomness $r$ is only known to the coin's owner and is needed to spend the coin.

A coin has also a unique serial number $\mathsf{sn}$ which is computed by a party $\mathcal{P}_i$ with its randomly sampled PRF secret key $\mathsf{sk}_{\mathsf{PRF}}$ as $\mathsf{sn} \leftarrow \mathsf{PRF}^{sn}_{\mathsf{sk}_{\mathsf{PRF}}}(\mathsf{pid}||\mathsf{coin})$. The serial number does not reveal the party that owns the coin. Assurance that the same PRF key is used at all times by $\mathcal{P}_i$ is provided in the zero-knowledge proofs by asserting that the public PRF key associated with $\mathcal{P}$'s identity, which we write as $\mathsf{pk}_{\mathsf{PRF}} := \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}}}(0)$.

$\mathsf{Contracts}$ stores the set of Session IDs (SIDs) of executed ITMs, the party IDs (PIDs) and the phase message (freeze, compute or finalize). We can also imagine this as a set of different message that denotes which stage the zkHawk protocol is at and which parties have executed which phase of the protocol. $\mathsf{Coins}$ denote the set of input and output coin commitments. $\mathsf{Computations}$ denote which parties (PIDs) have executed the given session (SID) of the MPC function evaluating the smart contract function. $\mathsf{SpentCoins}$ consist of a set of serial number of coins that has been frozen by the blockchain. $\mathsf{FrozenCoins}$ is a user side set that consists of the coin commitments that has been frozen by the blockchain.

Suppose a set of $n$ parties $\mathcal{P} := \{\mathcal{P}_i\}_{i\in[n]}$[4] wish to execute a smart contract function $f$. Let $P := \{\mathsf{pid}_1, \cdots, \mathsf{pid}_n\}$ be the set of party IDs (PIDs) for each $\{\mathcal{P}_i\}_{i\in[n]}$.

**Blockchain as a special protocol participant:** We denote the blockchain process as a protocol participant $\mathcal{B}$ running a Turing machine ($\mathsf{ITM}_{\mathcal{B}}$). In V-zkHawk, we assume that the Blockchain takes the role of an Honest Verifier (HV) and that it is publicly verifiable. To reduce complexity we also assume that an adversary $\mathcal{A}$ can read the messages on the blockchain but cannot corrupt the blockchain (i.e, write on the incoming or outgoing tape of the $\mathsf{ITM}_{\mathcal{B}}$). The Blockchain takes no part in the smart contract execution, hence we will omit considering blockchain within the set $\mathcal{P}$ as well refrain from assigning a $\mathsf{pid}$ to $\mathcal{B}$. Since, no sensitive information is present on the blockchain and contract execution occurs *off-chain* among the parties we guarantee strong privacy.

Deriving from [4], we define the set of valid votes $\mathbb{V}$ as the set of non-negative integers less than a strict upper bound $X$. Mathematically, $\mathbb{V} := \{\$\mathsf{val} \in \mathbb{Z} : 0 \leq \$\mathsf{val} < X\}$.

**Assumption:** We choose $X$ as a power of 2 i.e. we let $X = 2^\ell$ for some positive integer $\ell$.

**Definition 4.** *An $m$-party smart contract is an $m$-ary function $f : (\mathbb{V}\times\{0,1\}^*)^m \rightarrow (\mathbb{V}^m \times \{0,1\}^*) \cup \{\perp\}$ satisfying the following property:*

- *For all choices of $\$\mathsf{val}, \ldots, \$\mathsf{val}_m \in \mathbb{V}$ and $\mathsf{in}_1, \ldots, \mathsf{in}_m \in \{0,1\}^*$, one of the following statements is true*
  1. $z = \perp$

[4] We assume there is a total order on the pseudonyms $\mathcal{P}_i$ so that a unique index may be assigned to each party. Without loss of generality and for simplicity, when we write $\mathcal{P}_i$, we assume its index is $i$.

2. $z = (\$\mathsf{val}'_1, \ldots, \$\mathsf{val}'_n, \mathsf{out}) \wedge$

$$\sum_{i \in [m]} \$\mathsf{val}'_i - \sum_{i \in [m]} \$\mathsf{val}_i = 0 \textit{ (zero-sum constraint)}$$

*where* $z = \hat{f}((\$\mathsf{val}_1, \mathsf{in}_1), \ldots, (\$\mathsf{val}_m, \mathsf{in}_m))$.

## 3 Variant zkHawk (V-zkHawk)

In this paper, we elaborate and modify the construction of the zkHawk [4] paper suggested by Banerjee et al. to design a variant zkHawk (V-zkHawk) protocol. zkHawk as explained earlier is a PSC protocol leveraging the Hawk protocol along with employing MPC functions to execute the contract instead of a partially trusted *manager*. Specifically, we will describe how the contracts are getting executed leveraging an MPC [6, 25, 40, 42] function and its security in the UC framework [14] with commitments, signatures, symmetric encryption, NIZK arguments, assumption of CRS models and broadcast channels. To do this, we will broadly divide the security into the real world and the ideal world scenario with any $\mathsf{PPT}$ environment $\mathcal{Z}$.

The MPC function defined in V-zkHawk protocol uses the classical concept of secret sharing [24, 39, 42] to share the secret inputs among the parties and a 2-round broadcast [40] for summation thus checking zero-sum constraint. This ensures that we achieve input privacy and the contracts are executed securely using an MPC function.

To adhere to the task at hand and for brevity, we will not provide a detailed mathematical proof of the inner workings of the MPC function although the reader can look into any recent developments of MPC protocols which implies secret sharing [6, 25]. We also observe that we can use Function Secret Sharing (FSS) scheme [11] to share the MPC function among the different participants of the smart contract.

**Assumption**: We aim to provide security for *malicious, static* adversaries with dishonest majority in the UC model.

**Definition 5.** *The V-zkHawk protocol* $\pi$ *is a tuple* $(\mathsf{Com}, B, U)$ *where* $B = (\mathsf{Preprocess}, \mathsf{Freeze}, \mathsf{Finalize})$ *is the blockchain program (modelled as a tuple of stateful PPT algorithms) and* $U = (\mathsf{Preprocess}, \mathsf{Freeze}, \mathsf{Compute}, \mathsf{Finalize})$ *is the user program. The program* $B$ *is passed to the blockchain program wrapper* $\mathcal{G}$*, defined in Figure 2, to produce a blockchain functionality. The program* $U$ *is passed to the user program wrapper* $\mathcal{H}$*, defined in Figure 3 to specify user behavior during execution of the protocol* $\pi$*.*

On a high level, the goal of the V-zkHawk protocol for a set of participants is to take an input (E.g., an amount of currency or votes) from each participant, evaluate the smart contract (*off-chain*) using MPC function and output the results of each of the party such that the input/output values remain hidden from the blockchain as well other contract participants throughout the execution.

Blockchain program wrapper $\mathcal{G}(B)$ :

**Preprocess:**
- $\mathsf{Contracts} \leftarrow \{\}$
- CRS generation and broadcast
- Send blockchain state to $\mathcal{A}$ (read only).

**Freeze:** $\mathcal{Z}$ activates the Blockchain $\mathcal{B}$
- Upon receiving $\mathsf{msg} := (\mathsf{freeze}, \mathsf{sid}, P, \cdot)$ from $\mathcal{P}_i$:
- Send $(\mathsf{msg}, \mathsf{pid})$ to $\mathcal{A}$ (read only)
- Assert $\mathsf{pid} \in P$
- If $(\mathsf{sid}, \cdot, \cdot, \cdot) \notin \mathsf{Contracts}$:
  - $\mathsf{Contracts} \leftarrow \mathsf{Contracts} \cup \{(\mathsf{sid}, P, P' := \{\}, \mathsf{freeze})\}$
- Assert $(\mathsf{sid}, P, P', t := \mathsf{freeze}) \in \mathsf{Contracts}$
- Assert $\mathsf{pid} \notin P'$
- If $\mathsf{B.Freeze}(\mathsf{msg}, \mathsf{pid}) = 1$: (i.e. Call program $B$)
  - If $P' \cup \{\mathsf{pid}\} = P$:
    - $t \leftarrow \mathsf{compute}$
  - Replace $(\mathsf{id}, P, P', \mathsf{freeze})$ in $\mathsf{Contracts}$
    - with $(\mathsf{id}, P, P' \cup \{\mathsf{pid}\}, t)$
- Send blockchain state to $\mathcal{A}$ (read only)
- Control goes back to $\mathcal{Z}$

**Finalize:** $\mathcal{Z}$ activates the Blockchain $\mathcal{B}$
- Upon receiving $\mathsf{msg} := (\mathsf{finalize}, \mathsf{sid}, \cdot)$ from $\mathcal{P}$:
- Send $(\mathsf{msg}, \mathsf{pid})$ to $\mathcal{A}$ (read only)
- Assert $(\mathsf{sid}, P, P', \mathsf{compute}) \in \mathsf{Contracts}$
- If $\mathsf{B.Finalize}(\mathsf{msg}, \mathsf{pid}) = 1$:
  - If $P' \cup \{\mathsf{pid}\} = P$:
    - $t \leftarrow \mathsf{finalized}$
    - Replace $(\mathsf{sid}, P, P', \mathsf{compute})$ in $\mathsf{Contracts}$
    - with $(\mathsf{sid}, P, P', \mathsf{finalized})$
- Send blockchain state to $\mathcal{A}$ (read only)
- Control goes back to $\mathcal{Z}$

**Fig. 2.** Blockchain program wrapper $\mathcal{G}(B)$ for V-zkHawk evaluation.

### 3.1 Preprocessing Phase

V-zkHawk protocol relies on the two-round MPC protocol for summation by Kursawe, Danezis and Kohlweiss (KDK) [40]. In the discrete logarithm setting, let $\mathbb{G}$ be a finite cyclic group of prime order $p$ and let $g$ be a generator of $\mathbb{G}$. In the *preprocessing phase* (a term borrowed from SPDZ [25]), a party $\mathcal{P}_i$ samples a uniformly random $v_i \in \mathbb{Z}_p$. $\mathcal{P}_i$ then broadcasts $u_i \leftarrow g^{v_i} \in \mathbb{G}$ (as a part of the CRS). We note that the $u_i$ and $v_i$ acts a public key and private key for

User program wrapper $\mathcal{H}(U)$ :

**Preprocess:**
- $\mathsf{Computations} \leftarrow \{\}$
- $\mathsf{Coins} \leftarrow \{\}$
- $\mathsf{FrozenCoins} \leftarrow \{\}$
- CRS generation and broadcast

**Freeze:** $\mathcal{Z}$ activates party $\mathcal{P}_i$ or $\mathcal{A}$ (if corrupt)
- Upon receiving message $(\mathsf{freeze}, f, P, \$\mathsf{val}, \mathsf{in})$:
- $r \xleftarrow{\$} \{0,1\}^{\ell}$
- $\mathsf{coin} \leftarrow \mathsf{Com}(\$\mathsf{val}, r)$
- $\mathsf{Coins} \leftarrow \mathsf{Coins} \cup \{\mathsf{coin}\}$
- $\mathsf{FrozenCoins} \leftarrow \mathsf{FrozenCoins} \cup \{(\mathsf{sid}, \mathsf{coin})\}$
- $\mathsf{msg} \leftarrow \mathsf{U.Freeze}(\mathsf{sid}, \mathsf{coin}, r, \mathsf{in})$
- Send $\mathsf{msg}$ to blockchain
- If $\mathcal{P}_i$ was activated
  - Control goes back to $\mathcal{Z}$
- If $\mathcal{A}$ was activated
  - Activate $\mathcal{P}_i$ or return control to $\mathcal{Z}$

**Compute:** $\mathcal{Z}$ activates party $\mathcal{P}_i$ or $\mathcal{A}$ (if corrupt)
- Upon receiving message $(\mathsf{compute}, \mathsf{sid})$:
- Define MPC function $f'$ with input parameters $x$
- $\mathsf{Computations} \leftarrow \mathsf{Computations} \cup (f', P, \mathsf{sid})$
- Send $(\mathsf{input}, f', P, x)$ to $\mathsf{U.Compute}(\mathsf{sid})$

**Finalize:** Upon receiving $\mathsf{msg} := (\mathsf{output}, f', P, y)$ from $\mathsf{U.Compute}(\mathsf{sid})$:
- Assert $(f', P, \mathsf{sid}) \in \mathsf{Computations}$
- Assert $(\mathsf{sid}, \mathsf{coin}) \in \mathsf{FrozenCoins}$
- $(\mathsf{msg}, \mathsf{coin}'_1, \ldots, \mathsf{coin}'_n, \mathsf{out}, (\$\mathsf{val}', r')) \leftarrow \mathsf{U.Finalize}(\mathsf{sid}, y)$
- $\mathsf{Coins} \leftarrow (\mathsf{Coins} \setminus \{\mathsf{coin}\}) \cup \{(\$\mathsf{val}', r')\}$
- Send $\mathsf{msg}$ to blockchain
- Remove $(f', P, \mathsf{sid})$ from $\mathsf{Computations}$
- Remove $(\mathsf{sid}, \mathsf{coin})$ from $\mathsf{FrozenCoins}$
- If $\mathcal{P}_i$ was activated
  - Control goes back to $\mathcal{Z}$
- If $\mathcal{A}$ was activated
  - Activate $\mathcal{P}_i$ or return control to $\mathcal{Z}$

**Fig. 3.** User program wrapper $\mathcal{H}(U)$ for V-zkHawk evaluation.

the signature scheme to be used in V-zkHawk. Also, we observe that in this signature scheme setting the secret to public key homomorphic (as explained

in Definition 2) property is satisfied. In the *finalization phase* specifically, $\mathcal{P}_i$ first computes $t_i := \prod_{j<i} u_j^{-1} \cdot \prod_{i<j\leq n} u_j$ and then sends $z_i := t_i^{v_i} \cdot g^{m_i}$ to the blockchain via secure side channels, where $m_i$ is $\mathcal{P}_i$'s input to the summation. Then evaluating the product $\prod_{i\in[n]} z_i$ gives $g^{\sum_{i\in[n]} m_i}$; that is, the sum of the $m_i$ is in the exponent and Pollard's lambda algorithm can be used to extract it. In our usage of KDK, the expected sum will be zero. Therefore, we check whether $\prod_{i\in[n]} z_i \stackrel{?}{=} 1$.

In this phase, the parties compute the CRS (Figure 4) and send the CRS to each other via a UC broadcast protocol [30]. We assume the presence of *authenticated, asynchronous* broadcast channels among the parties and the blockchain. The broadcast protocol $\Pi_{ASBC}$ is defined in Figure 5.

$\Pi_{CRS}$ **- CRS Generation Protocol**

**CRS Generation:**

$v_i \xleftarrow{\$} \mathbb{Z}_p$
$u_i \leftarrow g^{v_i}$
$\beta_i \xleftarrow{\$} \{0,1\}^{4\ell}$
$K_i \leftarrow \mathsf{KeyGen}_{Sym}(1^\lambda)$
$sk_{\mathsf{PRF}i} \xleftarrow{\$} \mathbb{Z}_p$
$pk_{\mathsf{PRF}i} \leftarrow \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}i}}(0)$

**Output:**

The common reference string is declared as $\mathsf{CRS} = (u_i, pk_{\mathsf{PRF}i}, \beta_i)_{i\in[n]}$

**Fig. 4.** UC CRS generation protocol

### 3.2 Freeze phase

The first phase of the protocol is freeze phase. In order to execute a smart contract, each participant must freeze its input coin to the contract so that it cannot be spent.

After the CRS broadcast message to each of the parties, the freeze phase is initiated by the parties in $\mathcal{P}$ as in Figure 6 message which generates a freeze to be sent to the blockchain.

As in [4], we generate the coin commitments based on the Pederson commitment scheme [46]. Let $h_1$ and $h_2$ be two generators of finite cyclic $\mathbb{G}$ of prime order $p$ as stated above. Let the value to be committed by $c$ and the blinding factor (or randomness here) be $r$. Then a Pedersen commitment is defined as

$$\mathsf{Com}(x, r) = h_1^x \cdot h_2^r$$

**Broadcast Protocol** $\Pi_{ASBC}$

**Input:**
$\mathcal{P}_i$ receives activation (broadcast, CRS) from $\mathcal{Z}$
**Broadcast:**
$\mathcal{P}_i$ sends CRS to all parties.
Upon receiving $\mathsf{CRS}_i$ from $\mathcal{P}_i$, $\mathcal{P}_k$ sends the value $\mathsf{CRS}_i$ to all other parties.
$\mathcal{P}_k$ waits to receive a message from every party (other than $\mathcal{P}_i$).
Each party outputs (broadcast, pid, CRS) where $\mathsf{CRS} = (u_i, pk_{\mathsf{PRF}i}, \beta_i)_{i\in[n]}$

**Fig. 5.** UC Broadcast protocol for the CRS

We chose Pederson commitments because the commitments generated follow the perfectly hiding and computationally binding properties. Each commitment commits to a bit $b \in \{0,1\}$ as $\mathsf{UCC}_{\mathsf{OneTime}}$ in [18] and leverages the CRS to commit and reveal the commitments.

The blockchain on receiving this message instantiates the blockchain freeze protocol (as in Figure 7) to block/freeze coin commitments (by extension their serial numbers) that are being sent to the blockchain. This ensures that the coin is not double spent and once a party commits a coin to the smart contract they cannot change the value of the coin mid-execution or re-use the coin.

User anonymity is maintained in the freeze phase as we use NIZK (which are zero-knowledge) proofs to ensure whether a particular serial number of a coin commitment exists on the SpentCoins set or not. Thus, no information about $r$ or $\mathsf{sk}_{\mathsf{PRF}}$ is revealed during the proof and finding which commitment correspond to a particular transaction from SpentCoins and Coins is like inverting the PRF function which is assumed to be infeasible.

### 3.3 Computation phase

The computation phase is where the smart contract function gets executed using an MPC function. Our approach permits the MPC function to be carried out efficiently between the participants of the contract. In particular, the MPC program does not have to compute an expensive zk-SNARK proof as in Hawk [38]. Consider a smart contract function $f : (\mathbb{Z} \times \{0,1\}^*)^n \to \mathbb{Z}^n \times \{0,1\}^*$. Let $\$\mathsf{val}_i$ and $\mathsf{in}_i$ be the input currency value and input string of party $\mathcal{P}_i$ respectively. Applying $f$, we have, $((\$\mathsf{val}'_1, \ldots, \$\mathsf{val}'_n), \mathsf{out}) \leftarrow f((\$\mathsf{val}_1, \mathsf{in}_1), \ldots, (\$\mathsf{val}_n, \mathsf{in}_n))$.

All parties obtain the tuple $(\mathsf{sid}, P, \mathsf{pid}, u_i, \mathsf{coin}_i)$ from the blockchain. There must be a tuple for each $\mathsf{pid} \in P$ in order for execution to proceed i.e. it is the case that $\mathsf{sid} \in \mathsf{Contracts}$.

In this phase, each party $\mathcal{P}_i$ generates its own output coin $\mathsf{coin}'_i$ after it learns its output value $\$\mathsf{val}'_i$. There are two underlying key ideas for this protocol. The

$U.Freeze()$ **- User Function**

**Input:**

$\mathcal{P}_i$ receives $\mathsf{CRS}$ from broadcast protocol $\Pi_{ASBC}$
Parse $\mathsf{CRS}$

**Compute:**

$\mathcal{P}_i$ computes $r_i \xleftarrow{\$} \{0,1\}^\ell$
$\mathsf{coin}_i \leftarrow \mathsf{Com}(\$\mathsf{val}_i, r_i)$
Generates $\pi_i \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)$ where the statement to prove $x$ is $(\mathsf{coin}_i, \mathsf{sn}_i)$ and witness $w$ is $sk_{\mathsf{PRF}}$ for the relation that checks
  Assert $\mathsf{sn}_i = \mathsf{PRF}^{sn}_{\mathsf{sk}_{\mathsf{prf}}}(\mathsf{pid}||\mathsf{coin}_i)$

**Output:**

$\mathcal{P}_i$ sends a message to the blockchain of the form $(\mathsf{freeze}, \mathsf{sid}, \mathsf{pid}, u_i, \mathsf{coin}_i, \mathsf{sn}_i, \pi_i, \mathsf{CRS})$

**Fig. 6.** The User function for the freeze phase in V-zkHawk

$B.Freeze()$ **- Blockchain Function**

**Input:**

Blockchain receives $(\mathsf{freeze}, \mathsf{sid}, \mathsf{pid}, u_i, \mathsf{coin}_i, \mathsf{sn}_i, \pi_i, \mathsf{CRS})$ from party $\mathcal{P}_i$

**Compute:**

Assert $\mathsf{coin}_i \in \mathsf{Coins}$
Assert $\mathsf{sn}_i \notin \mathsf{SpentCoins}$
Assert $\mathsf{NIZK.Verify}(\mathsf{CRS}, x, \pi_i)$ where the statement to be verified $x$ is $(\mathsf{coin}_i, \mathsf{sn}_i)$
Add $\mathsf{sn}_i$ to $\mathsf{SpentCoins}$

**Output:**

Store $(\mathsf{sid}, P, \mathsf{pid}, u_i, \mathsf{coin}_i)$
If a tuple has been stored for every $\mathcal{P}_j \in P$
  Add $\mathsf{sid}$ to $\mathsf{Contracts}$.

**Fig. 7.** The Blockchain function for the freeze phase in V-zkHawk

first is that the MPC function generates and outputs the output coin for each party. The second idea is that the MPC function computes a "joint" signature (with the public keys of all parties) on the set of output coins. As a result, an honest party can inform the blockchain of the set of jointly-signed new coins.

The blockchain can detect if a malicious party sends a coin with a different value than its correctly assigned output value and therefore such an action can be appropriately penalized (explained later). However the extra complexity of the MPC function means that its computation is considerably more expensive.

We therefore consider some optimizations. In the discrete logarithm setting, we observe that for aggregate multi-signature Schnorr schemes without random oracles [43], a signature with the modular sum of a set of secret keys is a signature associated with the modular product of the set of corresponding public keys (Definition 2) and that aggregation in V-zkHawk takes place during signing. Hence, a single $\sum_{sig}$ computation suffices to perform a "joint" signature to validate the contract execution in V-zkHawk. For V-zkHawk, we require the signature scheme to support unforgeability as in EUF-CMA and that it is universally composable according to [15]. We defer the reader to Figure 1 of [15] for the ideal $\mathcal{F}_{Sig}$ functionality that holds in the UC model. We can also optimise using elliptic pairing signature schemes like aggregate multi-signature BLS [8] or compact BLS [7]. The compute function is defined as in Figure 8.

We first define the MPC function $f'$ in this phase:

- $f'((K_1, \$\mathsf{val}_1, s_1, v_1, \mathsf{in}_1), \ldots, (K_n, \$\mathsf{val}_n, s_n, v_n, \mathsf{in}_n))$:
  - $((\$\mathsf{val}'_1, \ldots, \$\mathsf{val}'_n), \mathsf{out}) \leftarrow f((\$\mathsf{val}_1, \mathsf{in}_1), \ldots, (\$\mathsf{val}_n, \mathsf{in}_n))$
  - For all $j \in [n]$:
    - $\mathsf{coin}'_j \leftarrow \mathsf{Com}(\$\mathsf{val}'_j; s_j)$
  - $\sigma \leftarrow \sum_{\mathsf{sig}}.\mathsf{Sign}(\sum_{j\in[n]} v_j, \mathsf{coin}'_1 \parallel \cdots \parallel \mathsf{coin}'_n)$
  - Return $((\psi_1 := \mathsf{Enc}(K_1, \$\mathsf{val}'_1), \ldots, \psi_n := \mathsf{Enc}(K_n, \$\mathsf{val}'_n)), (\mathsf{coin}'_1, \ldots, \mathsf{coin}'_n), \sigma, \mathsf{out})$

*U.Compute*() **- User Function**

**Input:**

Party $\mathcal{P}_i$ receives $(\mathsf{compute}, \mathsf{sid}, P, \mathsf{pid}, u_i, \mathsf{coin}_i)$ from the blockchain

**Compute:**

$\mathcal{P}_i$ computes $s_i \xleftarrow{\$} \{0,1\}^\ell$
The parties in $\mathcal{P}$ jointly compute an MPC function:
$((\psi_1, \ldots, \psi_n), (\mathsf{coin}'_1, \ldots, \mathsf{coin}'_n), \sigma, \mathsf{out})$
$\leftarrow \mathsf{MPC}(f', (K_1, \$\mathsf{val}_1, s_1, v_1, \mathsf{in}_1), \ldots, (K_n, \$\mathsf{val}_n, s_n, v_n, \mathsf{in}_n))$

**Output:**

Return $((\psi_1, \ldots, \psi_n), (\mathsf{coin}'_1, \ldots, \mathsf{coin}'_n), \sigma, \mathsf{out})$

**Fig. 8.** The User function for the computation phase in V-zkHawk

### 3.4 Finalization Phase

In the V-zkHawk protocol, a single honest party sending a finalize message, which contains the jointly-signed coins $(\mathsf{coin}'_1, \ldots, \mathsf{coin}'_n)$, prevents a malicious party $\mathcal{P}_i$ from convincing the blockchain to accept a different coin generated by $\mathcal{P}_i$. However in order for contract closure to take place, all parties must send a finalize message since the blockchain requires a $z_i$ (with associated NIZK proof) from each party to validate the zero-sum constraint (as shown in Figure 9). A finalize message is said to be *accepted* from party $\mathcal{P}_i$ if the blockchain stores the tuple $(\mathsf{sid}, \mathsf{pid}, z_i, \mathsf{coin}'_i)$ (see **Output** part in Figure 10).

### 3.5 Incentives for honest and financial penalties for malicious parties

To incentivize parties to send a finalize message that is accepted by the blockchain, we can impose a financial penalty on parties which fail to send such a message within a certain time span. A blockchain can derive a measure of discrete units of time (e.g.: each block in the blockchain is one time unit). We can therefore specify a timeout for contract closure. We define the following behavior. When the timeout is reached and contract closure has not occurred, only the frozen coins $\{\mathsf{coin}_j\}_{j\in[n]}$ belonging to the parties who had a finalize message accepted are refunded. Hence, a malicious party $\mathcal{P}_i$ who fails to send such a message loses the entire value of its frozen coin $\mathsf{coin}_i$. There is however one special case where this outcome serves to inadequately incentivize malicious party $\mathcal{P}_i$ from sending a finalize message. This is the case where $\mathcal{P}_i$ loses the entire value of its input coin in accordance with the contract i.e. $\$\mathsf{val}'_i = 0$. To resolve this problem, we can introduce the precondition of *deposits*. More precisely, when a party freezes its input coin, it must prove as part of the NIZK proof sent to the blockchain that the value of its input coin is greater than some fixed threshold (the deposit amount). A party $\mathcal{P}_i$'s deposit is refunded if one of the following conditions is met: (1) contract closure is reached; (2) the timeout is reached, an insufficient number of finalize messages have been accepted, and an accepted finalize message has been received from party $\mathcal{P}_i$. Therefore, even in the event of a contract deciding an output value of $\mathsf{val}'_i = 0$ for party $\mathcal{P}_i$, there is still an incentive for $\mathcal{P}_i$ to participate (i.e. the return of its deposit).

## 4 Ideal Functionalities and UC Security Analysis

In this section, we discuss the UC security of the V-zkHawk protocol designed in the section above. The task at hand is to design the different functionalities $\mathcal{F}(.)$ which can be emulated any protocol $\Pi$ that are used in V-zkHawk construction. Thereby also mimicking an adversary $\mathcal{A}$ using a simulator $\mathcal{S}$ in ideal V-zkHawk construction. In line with UC security, the environment $\mathcal{Z}$ provides the input to the parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and can interact with $\mathcal{A}$ and $\mathcal{S}$ anytime during the protocol execution. Let $J = \{j_1, \ldots, j_n\}$ be the set of all parties executing the zk-Hawk protocol and $I = \{i_1, \ldots, i_t\}$ be the set of indices of the $t \leq n$ corrupted

$U.Finalize()$ **- User Function**

**Input:**

Party $\mathcal{P}_i$ receives $((\psi_1, \ldots, \psi_n), (\mathsf{coin}'_1, \ldots, \mathsf{coin}'_n), \sigma, \mathsf{out})$ from the MPC function evaluation

**Compute:**

$\mathcal{P}_i$ computes:

$\$\mathsf{val}'_i \leftarrow \mathsf{Dec}(K_i, \psi_i)$

$\mathsf{coin}'_i \leftarrow \mathsf{Com}(\$\mathsf{val}'_i; s_i)$

$t_i := \prod_{j<i} u_j^{-1} \cdot \prod_{i<j\leq n} u_j$

$z_i := t_i^{v_i} \cdot g^{\$\mathsf{val}'_i - \$\mathsf{val}_i}$

Generates $\pi'_i \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)$ where the statement to prove $x$ is $(\mathsf{coin}_i, \mathsf{coin}'_i, u_i, (u_j)_{j\in[n]}, z_i)$ with the witness $w$ $(\$\mathsf{val}_i, \$\mathsf{val}'_i, r_i, s_i, v_i)$ for the relation that checks

- Assert $\mathsf{coin}_i = \mathsf{Com}(\$\mathsf{val}_i; r_i)$
- Assert $\mathsf{coin}'_i = \mathsf{Com}(\$\mathsf{val}'_i; s_i)$
- Assert $u_i = g^{v_i}$
- Assert $z_i = t_i^{v_i} \cdot g^{\$\mathsf{val}'_i - \$\mathsf{val}_i}$

**Output:**

$\mathcal{P}_i$ sends the message $(\mathsf{finalize}, \mathsf{sid}, \mathsf{pid}, (\mathsf{coin}'_1, \ldots, \mathsf{coin}'_n), \sigma, \mathsf{coin}'_i, z_i, \pi'_i, \mathsf{CRS})$ to the Blockchain

**Fig. 9.** The User function for the finalization phase in V-zkHawk

parties. Additionally, we let $\bar{I} = [n] \setminus I = \{\bar{i}_1, \ldots, \bar{i}_{n-t}\}$ be the set of indices of the honest parties.

As explained in Section 2.1, in UC framework, a set of ITMs interact with each other in the real and ideal worlds. As in [14, 41] we elaborate on the real and ideal world executions:

**Real World Execution**: The protocol $\Pi$ is realized by a set of ITMs $\mathcal{M}_1, \ldots, \mathcal{M}_n$ that are executed by the $n$-parties $\{\mathcal{P}_j\}_{j\in[n]} \in \mathcal{P}$. Since, we are working with static adversaries, at the beginning of the protocol execution $\mathcal{A}$ corrupts certain parties among $\mathcal{P}_1, \ldots, \mathcal{P}_n$. Then, $\mathcal{A}$ and $\mathcal{Z}$ run in parallel with $\mathcal{M}_{\bar{i}}$ for every honest party $\mathcal{P}_{\bar{i}}$. The honest parties $\mathcal{P}_{\bar{i}}$ relay the queries and answers between $\mathcal{Z}$ and $\mathcal{M}_{\bar{i}}$. A party $\mathcal{P}_i$ corrupted by $\mathcal{A}$ ignores any further input and instead all the messages are sent to $\mathcal{A}$. Furthermore, $\mathcal{A}$ can pretend to run any $\mathcal{M}_j$ (Turing machines operated by a corrupted party) via any corrupted party. At some point, $\mathcal{Z}$ stops the execution and outputs a bit.

**Ideal World Execution**: Owing to the assumption of static adversaries, $\mathcal{S}$ corrupts certain parties in $\{\mathcal{P}_j\}_{j\in[n]} \in \mathcal{P}$ by notifying them and $\mathcal{F}$. $\mathcal{Z}, \mathcal{S}$ and $\mathcal{Z}$ then execute in parallel. During the execution there is no direct communication between $\mathcal{F}$ and $\mathcal{Z}$. The parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ instead interact with $\mathcal{Z}$ and responds

$B.Finalize()$ **- Blockchain Function**

**Input:**

Blockchain receives $(\mathsf{finalize}, \mathsf{sid}, \mathsf{pid}, (\mathsf{coin}'_1, \ldots, \mathsf{coin}'_n), \sigma, \mathsf{coin}'_i, z_i, \pi'_i, \mathsf{CRS})$ from party $\mathcal{P}_i$

**Compute:**

Assert that $\mathsf{id} \in \mathsf{Contracts}$
Let $(\mathsf{sid}, P, \mathsf{pid}, u_i, \mathsf{coin}_i)$ for $\mathsf{pid} \in P$ be the stored tuples in the freeze phase.
Assert $\mathsf{NIZK.Verify}(\mathsf{CRS}, x, \pi'_i)$ where the statement to be verified $x$ is $(\mathsf{coin}_i, \mathsf{coin}'_i, u_i, (u_j)_{j\in[n]}, z_i)$
Assert $\sum_{\mathsf{sig}}.\mathsf{Verify}(\prod_{j\in[n]} u_j, \sigma, (\mathsf{coin}'_1 \parallel \cdots \parallel \mathsf{coin}'_n))$
If coin tuple stored:
  Assert $(\mathsf{coin'}_1, \ldots, \mathsf{coin'}_n)$ same as coin tuple stored
Else:
  Store $(\mathsf{coin'}_1, \ldots, \mathsf{coin'}_n)$

**Output:**

Store the tuple $(\mathsf{sid}, \mathsf{pid}, z_i, \mathsf{coin}'_i)$
If a tuple (above) has been stored for every $\mathsf{pid} \in P$
  Assert $\prod_{j\in[n]} z_j = 1$
  For all $j \in [n]$, add $\mathsf{coin}'_j$ to $\mathsf{Coins}$

**Fig. 10.** The Blockchain function for the finalization phase in V-zkHawk

to the queries inquired by $\mathcal{Z}$. An honest party $\mathcal{P}_{\bar{i}}$ forwards the queries to $\mathcal{F}$ and returns the answer it gets from $\mathcal{F}$ to $\mathcal{Z}$. A corrupted party $\mathcal{P}_i$ has no part to play here again as $\mathcal{S}$ freely communicates with $\mathcal{F}$ and $\mathcal{Z}$. Again, at some point, $\mathcal{Z}$ stops the execution and outputs a bit.

The output bit that we get from $\mathcal{Z}$ is the guess of $\mathcal{Z}$ that whether it interacted with the real world protocol or the ideal world functionality.

We start by defining the the broadcast functionality from [19, 30] $\mathcal{F}_{ASBC}$ in Figure 11. From [18, 20] we borrow the idea of a non-interactive commitment schemes and define the commitment functionality $\mathcal{F}_{Com}$ (Figure 12). The Blockchain $\mathcal{B}$ acts as the receiver who receives the committed value from the functionality for each party $\mathcal{P}_i$ and later verifies it in the reveal phase. Next, we define the ideal NIZK functionality $\mathcal{F}_{NIZK}$ (Figure 14) and the ideal CRS functionality $\mathcal{F}_{CRS}$ (Figure 13). We leverage these functionalities as shown in [33]. The UC-NIZK arguments defined in the previous section for different phases of the V-zkHawk realizes the $\mathcal{F}_{NIZK}$ functionality in the $\mathcal{F}_{CRS}$-hybrid model. The commitment functionality $\mathcal{F}_{Com}$ is also realized in the $\mathcal{F}_{CRS}$-hybrid model.

Finally, we design the ideal MPC functionality $\mathcal{F}_{MPC}$ which computes the MPC function to execute the smart contract in the ideal world process. This is designed as a black box functionality which takes in input from all the parties

**Ideal Functionality $\mathcal{F}_{ASBC}$**

Upon receiving the message $(\mathsf{broadcast}, P, \mathsf{sid}, u_i)$ from $\mathcal{P}_i$, send $(\mathsf{broadcast}, \mathcal{P}_i, \mathsf{sid}, u_i)$
to all the parties in $P$ and to the simulator $\mathcal{S}$.
Halt.

**Fig. 11.** The ideal functionality for broadcasting among the parties

**$\mathcal{F}_{Com}$ - Commitment Functionality**

**Commit Phase:**
On receiving the message $(\mathsf{Commit}, \mathsf{sid}, \mathsf{pid}, \mathcal{B}, b)$ from $\mathcal{P}_i$, where $b \in \{0, 1\}$, record the tuple $(\mathsf{sid}, \mathsf{pid}, \mathcal{B}, b)$ and send message $(\mathsf{Received}, \mathsf{sid}, \mathsf{pid}, \mathcal{B})$ to $\mathcal{S}$ and $\mathcal{B}$. Ignore any further $\mathsf{Commit}$ messages from $\mathcal{P}_i$ to $\mathcal{B}$ with the same $\mathsf{sid}$.

**Reveal Phase:**
On receiving the message $(\mathsf{Reveal}, sid)$ from $\mathcal{P}_i$:
If tuple $(\mathsf{sid}, \mathsf{pid}, \mathcal{B}, b)$ is previously recorded
Send message $(\mathsf{Reveal}, \mathsf{sid}, \mathsf{pid}, \mathcal{B}, b)$ to $\mathcal{B}$ and $\mathcal{S}$.
Else
Ignore the message

**Fig. 12.** The ideal commitment functionality

**$\mathcal{F}_{CRS}$ - Ideal CRS Functionality**

**Input:**
On input $(\mathsf{Input}, \mathsf{sid})$, run $\mathsf{CRS} \leftarrow \Pi_{CRS}(1^\lambda)$.

**Output:**
Send $(crs, \mathsf{sid}, CRS)$ to all parties.
Halt.

**Fig. 13.** The ideal CRS functionality

via a secure channel, executes the smart contract and send the output values after the smart contract execution back to the respective parties (Figure 15).

$\mathcal{F}_{NIZK}$ **- Ideal NIZK Functionality**

**Prove:**
Party $\mathcal{P}_i$ waits to receive $(crs, \mathsf{sid}, CRS)$ from $\mathcal{F}_{CRS}$
On input $(\mathsf{Prove}, \mathsf{sid}, x, w)$ such that $x(w) = 1$,
run $\Pi_{NIZK} \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)$
Output $(\mathsf{Proved}, sid, \Pi_{NIZK})$.

**Verify:**
Blockchain $\mathcal{B}$ waits to receive $(crs, \mathsf{sid}, CRS)$ from $\mathcal{F}_{CRS}$
On input $(\mathsf{Verify}, \mathsf{sid}, x, w)$ run $b \leftarrow \mathsf{NIZK.Verify}(\mathsf{CRS}, x, \Pi_{NIZK})$
Output $(\mathsf{Verified}, sid, b)$, where $b \in \{0, 1\}$

**Fig. 14.** The ideal NIZK functionality

$\mathcal{F}_{MPC}$ **- Ideal Functionality**

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

**Input:** Upon receiving $(\mathsf{input}, f, \mathsf{pid}, P, x_i)$ from $\mathcal{P}_i$:
Assert $\mathsf{pid} \in P$
Remove $(f, P, \mathsf{pid}, \cdot)$ from $\mathsf{Inputs}$ if it exists
$\mathsf{Inputs} \leftarrow \mathsf{Inputs} \cup (f, P, \mathsf{pid}, x_i)$
If $(f, P, \mathsf{pid}, x_i) \in \mathsf{Inputs}\ \forall\ \mathsf{pid} \in P$:
$y \leftarrow f(x_1, \ldots, x_{n'})$ where $n' = |P|$
Send $(\mathsf{output}, f, P, y)$ to $\mathcal{P}_i\ \forall\ \mathsf{pid} \in P$

**Fig. 15.** Idealized MPC functionality $\mathcal{F}_{MPC}$ that executes the smart contract function on inputs supplied by the set of parties.

**Theorem 1.** *Let $t$ be the number of static corrupted parties. Assuming a UC-secure* MPC *protocol UC-realizes an ideal $\mathcal{F}_{\mathsf{MPC}}$ functionality, a UC-secure* Com *protocol and a UC-secure* NIZK *protocol UC-securely realizes an ideal $\mathcal{F}_{\mathsf{Com}}$ functionality and an ideal $\mathcal{F}_{\mathsf{NIZK}}$ functionality respectively in the $(\mathcal{F}_{CRS}, \mathcal{F}_{ASBC})$-hybrid model, an $n$-party V-zkHawk protocol is UC-secure $\forall\ t < n$.*

*Proof.* We prove the theorem via a hybrid argument. Our goal is to move to a hybrid that shows that running the entire real world process with static adversary $\mathcal{A}$ is indistinguishable from an ideal world process with simulator $\mathcal{S}$ to a PPT environment $\mathcal{Z}$. This signifies that for the desired hybrid 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 we

assumed the Blockchain to be an honest verifier $\mathcal{B}$ which cannot be corrupted by an adversary $\mathcal{A}$ (or $\mathcal{S}$) but $\mathcal{A}$ (or $\mathcal{S}$) can read its input/output tape.

**Simulating communication with the environment $\mathcal{Z}$:** Every input value $\mathcal{A}$ receives from $\mathcal{Z}$ is written on the input tape of $\mathcal{S}$ as if $\mathcal{Z}$ is interacting with $\mathcal{S}$. Similarly, every output value written by $\mathcal{A}$ on its own output tape is copied to $\mathcal{S}$'s output tape to be read by the environment $\mathcal{Z}$.

**Hybrid 0**: This is the real system with static adversary $\mathcal{A}$ and the parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$

**Hybrid 1**: In this hybrid, $\mathcal{S}$ simulates the ideal functionality broadcast $\mathcal{F}_{ASBC}$ instead of the broadcast protocol being executed via asynchronous authenticated broadcast channels.

The hybrids Hybrid 0 and Hybrid 1 are indistinguishable because of asynchronous broadcast UC functionality. Hence, it is hard to identify how a party $\mathcal{P}_i$ receives the broadcast values from remaining the $(n-1)$ parties.

**Hybrid 2**: In this hybrid, $\mathcal{S}$ simulates the ideal MPC functionality $\mathcal{F}_{MPC}$ instead of the MPC protocol. Here, the honest parties send their inputs to $\mathcal{F}_{MPC}$ and receive their outputs via a secure channel. Additionally, the input of the corrupt parties controlled by the adversary is also sent to the functionality and they receive their outputs as well. This is done by triggering the $(\mathsf{input}, \cdot)$ message to $\mathcal{F}_{MPC}$ and the parties receive $(\mathsf{output}, \cdot)$ as their output which also consists of $y$ as their output value for the smart contract as in the real MPC function $\hat{f}$.

The hybrids Hybrid 1 and Hybrid 2 are indistinguishable from each other because an environment cannot determine whether the MPC protocol or the ideal functionality was run to compute the smart contract function and receive outputs.

**Hybrid 3**: In this hybrid, $\mathcal{S}$ simulates the ideal commitment functionality instead of $\mathcal{F}_{Com}$ instead of the commitment protocol using $\mathcal{F}_{\mathsf{CRS}}$. We observe that as in real protocol, the functionality notifies the blockchain and ideal world adversary of the committed value. Then, on receiving reveal message they open the values to blockchain and the adversary for verification of the commitment.

The hybrids Hybrid 2 and Hybrid 3 are indistinguishable from each other because of the blinding property of Pederson commitments and any $\mathsf{PPT}$ environment interacting with $\mathcal{S}$ or $\mathcal{A}$ cannot determine whether the committed value or the opening value was generated from a commitment scheme or ideal commitment functionality.

**Hybrid 4** We now modify the NIZK proof $\pi$ for verifying serial numbers for coin commitments with the ideal NIZK functionality using ideal CRS functionality.

This hybrid is indistinguishable from Hybrid 3 because of the perfect zero-knowledge property of the NIZK proofs.

**Hybrid 5** In this hybrid, we modify the NIZK proof $\pi'$ for proving the zero-sum constraint with the ideal NIZK functionality using the ideal CRS functionality.

This hybrid is indistinguishable from Hybrid 4 because of the perfect zero-knowledge property of the NIZK proofs.

**SIM** This is the ideal world process running with $\mathcal{F}_{NIZK}$, $\mathcal{F}_{MPC}$, $\mathcal{F}_{Com}$, $\mathcal{F}_{ASBC}$ and the simulator $\mathcal{S}$.

Hybrid 5 is very similar to the ideal process. Honest Provers in Hybrid 5 run the proof $\pi'$ in the same way that $\mathcal{S}$ does without the knowledge of the witness. Similarly for $\pi$ in Hybrid 4, $\mathcal{S}$ runs the ideal functionality NIZK in the same way as honest provers without the knowledge of the secret key of the $\mathsf{PRF}$ function. Both these hybrids are indistinguishable from the real worlds as well as SIM. From Hybrid 3, we observe that $\mathcal{S}$ simulating the ideal commitment functionality without the knowledge of the value is similar to an honest party committing its value. Hence, Hybrid 3 is indistinguishable from SIM. This forms a chain of indistinguishable hybrids which follows that Hybrid 2 is indistinguishable from Hybrid 3 and Hybrid 2 is indistinguishable from real world and finally Hybrid 1 is indistinguishable from the real world. This proves that running $\pi$,$\pi'$,$MPC$, $\Pi_{ASBC}$ and $Com$ with $\mathcal{A}$ in the real world Hybrid 0 is indistinguishable from the running the ideal process as defined in SIM.

We do not provide a separate hybrid setup for replacing $\mathcal{F}_{Sig}$ with $\sum_{\mathsf{Sig}}$ as the signature schemes are proved and verified within the MPC function. Hence, if UC security holds for the MPC functionality, we assume that UC security holds for the signature functionality as well. ■

## 5 Conclusion and Future Directions

The V-zkHawk protocol draws inspiration from Hawk and zkHawk and constructs a private smart contract protocol based on MPC evaluation. We have constructed the protocol in the UC security framework assuming $\mathsf{CRS}$-hybrid functionality, presence of *authenticated*, *asynchronous* broadcast channels among parties, secure side channels with the blockchain. The adversary setting is *malicious* and *static*. The corruption is focused in a dishonest majority setting where any $t < n$ parties can be corrupted. The worst case situation is of course $n - 1$ parties being corrupted but the protocol works for that situation as well since one honest party is enough to execute the entire protocol. The paper by Damgård et. al [26] described an interesting approach to work on 2-round broadcast MPC with minimal setup. It would be interesting to observe how we can modify the designed V-zkHawk broadcast 2-round MPC method with said method [26] as well as other 2-round broadcast optimal methods [21,22]. We can further improve the security of our protocol by extending it to work for *adaptive* adversaries [29]. The Global setup UC model (GUC) [2,17] solves the deniability and malleability problem of running multiple protocols using the same CRS setup by achieving a global setup that can be used by all malicious protocols running concurrently with given protocol $\pi$. Our protocol relies on the $\mathcal{F}_{CRS}, \mathcal{F}_{ASBC}$-hybrid model which assumes the presence of secure channels among parties and that each CRS is generated during and destroyed after protocol execution. We aim to optimize this using GUC leveraging a Global Setup.

**Acknowledgements.** I would like to thank my supervisor Dr. Hitesh Tewari for his constant guidance and support. I would also like to thank Dr. Michael Clear for the discussions on MPC functions and the UC model.

## References

1. Archer, D.W., Bogdanov, D., Lindell, Y., Kamm, L., Nielsen, K., Pagter, J.I., Smart, N.P., Wright, R.N.: From keys to databases—real-world applications of secure multi-party computation. The Computer Journal **61**(12), 1749–1771 (2018)
2. Badertscher, C., Canetti, R., Hesse, J., Tackmann, B., Zikas, V.: Universal composition with global subroutines: Capturing global setup within plain uc. In: Theory of Cryptography Conference. pp. 1–30. Springer (2020)
3. Banerjee, A., Clear, M., Tewari, H.: Demystifying the role of zk-snarks in zcash. In: 2020 IEEE Conference on Application, Information and Network Security (AINS). pp. 12–19. IEEE (2020)
4. Banerjee, A., Clear, M., Tewari, H.: zkhawk: Practical private smart contracts from mpc-based hawk. arXiv preprint arXiv:2104.09180 (2021)
5. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, pp. 329–349. ACM (2019)
6. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., et al.: Secure multiparty computation goes live. In: International Conference on Financial Cryptography and Data Security. pp. 325–343. Springer (2009)
7. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 435–464. Springer (2018)
8. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: International conference on the theory and applications of cryptographic techniques. pp. 416–432. Springer (2003)
9. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: International conference on the theory and application of cryptology and information security. pp. 514–532. Springer (2001)
10. Botta, V., Friolo, D., Venturi, D., Visconti, I.: Shielded computations in smart contracts overcoming forks. In: International Conference on Financial Cryptography and Data Security. pp. 73–92. Springer (2021)
11. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 337–367. Springer (2015)
12. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: International Conference on Financial Cryptography and Data Security. pp. 423–443. Springer (2020)
13. Buterin, V.: A next generation smart contract & decentralized application platform (2013) whitepaper. Ethereum Foundation (2013)
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001)
15. Canetti, R.: Universally composable signature, certification, and authentication. In: Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004. pp. 219–233. IEEE (2004)

16. Canetti, R.: Security and composition of cryptographic protocols: A tutorial. Cryptology ePrint Archive, Report 2006/465 (2006), https://ia.cr/2006/465
17. Canetti, R., Dodis, Y., Pass, R., Walfish, S.: Universally composable security with global setup. In: Theory of Cryptography Conference. pp. 61–85. Springer (2007)
18. Canetti, R., Fischlin, M.: Universally composable commitments. In: Annual International Cryptology Conference. pp. 19–40. Springer (2001)
19. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. pp. 494–503 (2002)
20. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. Cryptology ePrint Archive, Report 2002/140 (2002), https://ia.cr/2002/140
21. Ciampi, M., Ostrovsky, R., Waldner, H., Zikas, V.: Round-optimal and communication-efficient multiparty computation. Cryptology ePrint Archive (2020)
22. Cohen, R., Garay, J., Zikas, V.: Broadcast-optimal two-round mpc. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 828–858. Springer (2020)
23. Costan, V., Devadas, S.: Intel sgx explained. IACR Cryptol. ePrint Arch. **2016**(86), 1–118 (2016)
24. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: International workshop on public key cryptography. pp. 160–179. Springer (2009)
25. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits. In: European Symposium on Research in Computer Security. pp. 1–18. Springer (2013)
26. Damgård, I., Ravi, D., Siniscalchi, L., Yakoubov, S.: Minimizing setup in broadcast-optimal two round mpc. Cryptology ePrint Archive, Report 2022/293 (2022), https://ia.cr/2022/293
27. Derler, D., Slamanig, D.: Key-homomorphic signatures: definitions and applications to multiparty signatures and non-interactive zero-knowledge. Designs, Codes and Cryptography **87**(6), 1373–1413 (2019)
28. Feige, U., Lapidot, D., Shamir, A.: Multiple noninteractive zero knowledge proofs under general assumptions. SIAM Journal on computing **29**(1), 1–28 (1999)
29. Garg, S., Polychroniadou, A.: Two-round adaptively secure mpc from indistinguishability obfuscation. In: Theory of Cryptography Conference. pp. 614–637. Springer (2015)
30. Goldwasser, S., Lindell, Y.: Secure multi-party computation without agreement. Journal of Cryptology **18**(3), 247–287 (2005)
31. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 321–340. Springer (2010)
32. Groth, J.: On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 305–326. Springer (2016)
33. Groth, J., Ostrovsky, R., Sahai, A.: New techniques for noninteractive zero-knowledge. Journal of the ACM (JACM) **59**(3), 1–35 (2012)
34. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). International journal of information security **1**(1), 36–63 (2001)

35. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 1353–1370 (2018)
36. Keller, M.: Mp-spdz: A versatile framework for multi-party computation. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1575–1590 (2020)
37. Kerber, T., Kiayias, A., Kohlweiss, M.: Kachina–foundations of private smart contracts. In: 2021 IEEE 34th Computer Security Foundations Symposium (CSF). pp. 1–16. IEEE (2021)
38. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE symposium on security and privacy (SP). pp. 839–858. IEEE (2016)
39. Krawczyk, H.: Secret sharing made short. In: Annual international cryptology conference. pp. 136–146. Springer (1993)
40. Kursawe, K., Danezis, G., Kohlweiss, M.: Privacy-friendly aggregation for the smart-grid. In: Privacy Enhancing Technologies. pp. 175–191. Springer (2011)
41. Laud, P., Kamm, L.: Applications of secure multiparty computation, vol. 13. Ios Press (2015)
42. Li, Q., Cascudo, I., Christensen, M.G.: Privacy-preserving distributed average consensus based on additive secret sharing. In: 2019 27th European Signal Processing Conference (EUSIPCO). pp. 1–5. IEEE (2019)
43. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures and multisignatures without random oracles. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 465–485. Springer (2006)
44. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple schnorr multi-signatures with applications to bitcoin. Designs, Codes and Cryptography **87**(9), 2139–2164 (2019)
45. Micali, S., Goldreich, O., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC. pp. 218–229. ACM (1987)
46. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140. Springer (1991)
47. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE (2014)
48. Savelyev, A.: Contract law 2.0:'smart'contracts as the beginning of the end of classic contract law. Information & communications technology law **26**(2), 116–134 (2017)
49. Steffen, S., Bichsel, B., Gersbach, M., Melchior, N., Tsankov, P., Vechev, M.: zkay: Specifying and enforcing data privacy in smart contracts. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1759–1776 (2019)
50. Yao, A.C.: Protocols for secure computations. In: 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982). pp. 160–164 (1982). https://doi.org/10.1109/SFCS.1982.38
51. Yuan, R., Xia, Y.B., Chen, H.B., Zang, B.Y., Xie, J.: Shadoweth: Private smart contract on public blockchain. Journal of Computer Science and Technology **33**(3), 542–556 (2018)

# A Definitions

## A.1 Pseudorandom functions

**Definition 6 (PRF).** *Let $f : A \times B \to X$ be a family of functions and let $Y$ be the set of all functions $B \to X$. $f$ is a pseudorandom function (PRF) family if it is efficiently executable and $\forall$ PPT distinguisher $\mathcal{D}$ it holds that:*

$$\left|\mathsf{Pr}\left[a \xleftarrow{\$} A, \mathcal{D}_{f_a(.)}(1^\lambda)\right] - \mathsf{Pr}\left[y \xleftarrow{\$} Y, \mathcal{D}_{y(.)}(1^\lambda)\right]\right| \leq \mathsf{negl}(\lambda)$$

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

## A.2 EUF-CMA One-Time signatures

**Definition 7.** *A one-time aggregate multi-signature scheme $\sum_{sig}$ is strong EUF-CMA, if $\forall$ PPT adversaries $\mathcal{A}$*

$$\mathsf{Pr}\Big[(s_k, p_k) \leftarrow \mathsf{KeyGen}(1^\lambda), (m^*, \sigma^*) \leftarrow \mathcal{A}^{Sign(s_k,\cdot)}(p_k) : \\ \mathsf{Verify}(p_k, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin \mathcal{Q}^{Sign}\Big] \leq \mathsf{negl}(\lambda)$$

*where $\mathcal{Z}$ keeps track of the queries to the signing oracle via $\mathcal{Q}^{Sign}$ and the $Sign(s_k, \cdot)$ query can be called once.*

# B UC Framework

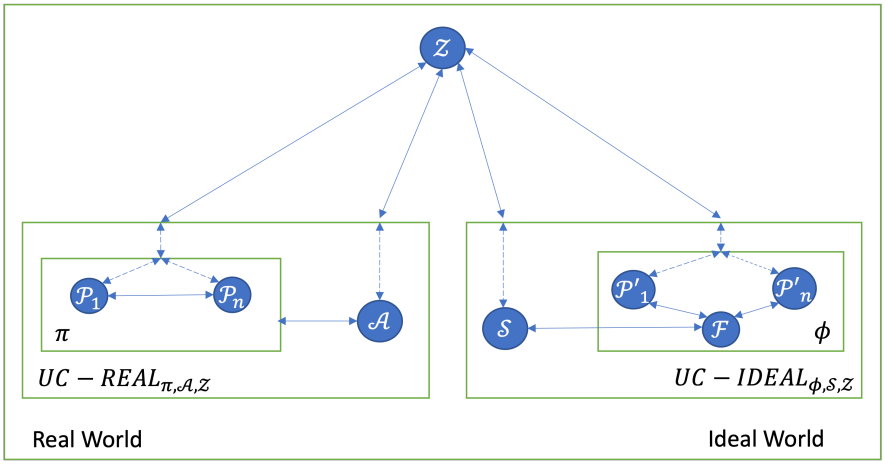


**Fig. 16.** Overview of the Real/Ideal World Execution in the UC Framework. The dotted lines symbolizes the interaction with the environment $\mathcal{Z}$. The solid lines indicates interaction among the parties (honest/corrupted)