cryptography

MDPI

Article

Multiverse of HawkNess: A Universally-Composable MPC-Based Hawk Variant

Aritra Banerjee [1,*] and Hitesh Tewari [1,2]

[1] ADAPT Centre, Trinity College Dublin, D02 PN40 Dublin, Ireland
[2] School of Computer Science and Statistics, Trinity College Dublin, D02 R123 Dublin, Ireland
* Correspondence: abanerje@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

Citation: Banerjee, A.; Tewari, H. Multiverse of HawkNess: A Universally-Composable MPC-Based Hawk Variant. *Cryptography* **2022**, *6*, 39. https://doi.org/10.3390/cryptography6030039

Academic Editor: Kentaroh Toyoda

Received: 24 June 2022
Accepted: 2 August 2022
Published: 4 August 2022

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 [1]. The implementation of early vending machines dates back to the 1st 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 [2] 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 [3] 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 [4–6] (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge).

The idea of Hawk [7] 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. [7] 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 applying MPC to the design of Hawk as it is incurs the prohibitively expensive overhead of executing a zk-SNARK proof [4,5] within an MPC program (a circuit in practice). Hence, we propose to remove the zk-SNARK proof from the MPC program [8]. 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) [9] two-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 [10] have proven to be quite efficient while proving security against malicious security models. Starting from Yao's protocols in the early 1980s [11] to the GMW protocol in the late 80s [12] to more recent protocols such as Danish sugar beet auction [13] or SPDZ [14] or MACE [15] 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.

1.1. A Private Smart Contract Execution

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 secret. Let the required loan amount be \$x and \$bo $\leftarrow 0$.

Smart Contract Function $\hat{f}(\$x, (\$len_1), \ldots, (\$len_n), \$bo)$

- $\$len_x \leftarrow \sum_{i \in [n]} \$len_i$
- If $\$len_x > \$x$
  - $\$bo \leftarrow \$len_x$
  - For $i \in \{1, \ldots, n\}$
    - $\$len_i \leftarrow 0$

Contract Closure:

- Return $\$len_i$ to len_i
- Send \$bo to bo

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

1.2. Our Contributions

In this paper, we present V-zkHawk, which is a variant of the zkHawk [8] protocol such that the smart contracts are evaluated off-chain without the trust assumption of a *manager*. This can be achieved using an MPC [13,14,16] 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 (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. On the contrary, 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). We provide formal proofs for the construction of the V-zkHawk protocol. The proofs and theorems in this paper are modeled in the UC [17] framework. We define the ideal functionalities for our real world protocols that also interact 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 are impossible to prove UC secure in the standard model [18]. Hence, we will be employing the CRS model [17,19] with a secure broadcast protocol for the setup/preprocessing stage. The NIZK protocol leverages the CRS functionality to realize the NIZK functionality in the ideal world. We define security against malicious (active) adversaries in the dishonest majority setting.

How V-zkHawk Is Different from zkHawk

Primarily, V-zkHawk achieves a higher level of security when compared to zkHawk. V-zkHawk attains a UC [17] security as compared to an isolated security model in zkHawk where an environment cannot interact with the protocol. In terms of contract closure computations, zkHawk's design is slightly faster as it provides immediate closure by requiring only one party to send a contract closure message to the blockchain. Both V-zkHawk and zkHawk achieve Strong Input/Output privacy. However, the zero-sum constraint check in V-zkHawk is much faster owing to the KDK [9] algorithm rather than the Schnorr proof leveraged in the zkHawk algorithm. Furthermore, V-zkHawk utilizes strong one-time EUF-CMA signatures to provide user authenticity during the computation phase. Additionally, the authenticity of the user is also checked during the freeze phase by verifying NIZK proofs for the serial number of frozen coins. Both these steps are missing from zkHawk.

Use cases: zkHawk is a useful primitive with multiple use cases including e-Voting, Crowdfunding, Rock-Paper-Scissors, Sealed bid Auctions, etc. V-zkHawk would have nearly the same use cases as zkHawk but with faster contract execution time and a higher level of security in form of UC secure protocols. The speciality of V-zkHawk and by extension zkHawk is that it can be leveraged for non-currency-based smart contract execution protocols. We refer the readers to the paper [20] for a use case of zkHawk on E-voting which by extension can also by performed by the V-zkHawk protocol. In this paper, however, we have explained V-zkHawk using a currency based smart contract. Due to the effectiveness of using EUF-CMA signatures, the V-zkHawk protocol can also be utilized for general purpose authentication in addition to executing smart contracts. The element of user authentication in the computation phase gives a unique edge as compared to zkHawk. However, the end goal (finalization phase) is a private smart contract evaluation.

1.3. 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 [7].

- **zkay**: zkay [21] 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 are correctly encrypted and that the smart contract executions are correct.
- **Arbitrum**: Arbitrum [22] 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 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 behavior of the VMs. It also relies on an honest majority setting for privacy guarantees.

- **Kachina**: Kachina [23] is 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 [24], being a retro-fitted privacy-preserving smart contract protocol for currency, can be utilized only in places such as sealed-bid auctions or crowdfunding. This protocol cannot be utilized in non-monetary smart contract applications such as e-Voting, Rock-Paper-Scissors, etc.
- **Shielded Computations in Smart Contracts**: Recent work by V. Botta et al. [25] leverages on-chain MPC protocols for executing smart contracts by forking blockchains such as Ethereum. It works for both honest and dishonest majority setting.
- **ShadowEth**: ShadowEth [26] utilizes the Trusted Execution Environment (TEE) to generate private smart contract evaluations for public blockchains such as Ethereum. It utilizes the Intel SGX [27] hardware enclave to implement the protocol that creates an isolated secure environment running parallel to the OS.

1.4. 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 definitions of the 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 their 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 [17] against *malicious*, *static* adversaries for a dishonest majority and how we can further refine this protocol in the Global UC (GUC) [28,29] model along with considering further optimizations 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. 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 PPT observer (here *environment*).

Definition 1 (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 λ

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

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

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

$\mathsf{Verify}(\mathsf{CRS}, x, \phi)$: *This algorithm generates an output* $b \in \{0,1\}$ *leveraging the proof* ϕ, 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's sake we will not go too much into details of these properties in this paper and refer the reader for further reading on NIZKs to [30,31]. 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 $(\mathbb{Y}, \cdot)$.

Definition 3. *A one-time aggregate multi-signature scheme* $\sum_{sig}$ *is strong EUF-CMA, if* $\forall$ PPT *adversaries* $\mathcal{A}$

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

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.*

We require the signature scheme leveraged in V-zkHawk (Schnorr [32,33] or BLS [34,35] or ECDSA [36]) to be secret-key-to-public key homomorphic. This concept was introduced in [37].

Definition 4 (Secret-to-Public-Key Homomorphic)**.** *We can say that a signature scheme* Σ_{sig} *provides secret-to-public-key homomorphism, if* $\exists$ *a map* δ: $\mathbb{X} \to \mathbb{Y}$ *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* $p_k = \delta(s_k)$

Definition 5 (Decisional Diffie–Hellman (DDH))**.** *The DDH assumption for a generator* $\mathbb{G}$ *states that, given the tuple* $(g, A = g^a, B = g^b)$, *where* $g \xleftarrow{\$} \mathbb{G}$ *and* $a, b \xleftarrow{\$} \mathbb{Z}_p$, *any* PPT *adversary* $\mathcal{A}$ *has negligible advantage in distinguishing* (g, A, B, C) *from* (g, A, B, u), *where* $u \xleftarrow{\$} \mathbb{G}$ *from* $C = g^{(ab)}$.

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 PPT environment which interacts with the ideal world protocol ϕ and real world protocol π 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 π or $\mathcal{F}$. We define a $\mathsf{UC-secure}$ protocol as follows:

Definition 6. *An n-party (where* $n \in \mathbb{N}$*) protocol* π *is* $\mathsf{UC-secure}$ $\forall$ $\mathcal{Z}$ *if for any* $\mathcal{A}$ $\exists$ $\mathcal{S}$*:*

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

Informally, we say that a protocol π securely emulates a protocol ϕ if for any given input the probability that $\mathcal{Z}$ will output 1 after interacting with π and $\mathcal{A}$ will differ negligibly to the probability that $\mathcal{Z}$ will output 1 after interacting with ϕ and $\mathcal{S}$. We also define that π $\mathsf{UC-realizes}$ $\mathcal{F}$ 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 π or $\mathcal{S}$ and $\mathcal{F}$ (Refer Appendix A: Figure A1 for an overview of Real/Ideal World Execution in the UC Framework).

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

From [17,38], 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 (sid) and a Party ID (pid). We refer the reader to [17,38,39] 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 \$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 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 $\mathsf{pk}_{\mathsf{PRF}} := \mathsf{PRF}_{\mathsf{sk}_{\mathsf{PRF}}}(0)$.

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. Coins denote the set of input and output coin commitments. Computations denote which parties (PIDs) have executed the given session (sid) of the MPC function evaluating the smart contract function. SpentCoins consist of a set of serial number of coins that has been spent by the blockchain. 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 $\mathcal{P} := \{\mathcal{P}_i\}_{i\in[n]}$ (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) 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 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 [8], 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 ℓ.

Definition 7. *An m-party smart contract is an m-ary function* $f : (\mathbb{V} \times \{0,1\}^*)^m \to (\mathbb{V}^m \times \{0,1\}^*) \cup \{\perp\}$ *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. $z = \perp$
  2. $z = (\$\mathsf{val}'_1, \ldots, \$\mathsf{val}'_n, \mathsf{out}) \wedge$

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

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 [8] 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 [9,13,14,40] function and its security in the UC framework [17] 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 PPT environment $\mathcal{Z}$.

The MPC function defined in V-zkHawk protocol uses the classical concept of secret sharing [40–42] to share the secret inputs among the parties and a two-round broadcast [9] for summation, thus checking the 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 imply secret sharing [13,14]. We also observe that we can use Function Secret Sharing (FSS) scheme [16] to share the MPC function among the different participants of the smart contract.

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. Figure 2 defined in the next page gives a brief overview of the V-zkHawk protocol.

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

Definition 8. *The V-zkHawk protocol* π *is a tuple* (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 3, to produce a blockchain functionality. The program U is passed to the user program wrapper* $\mathcal{H}$*, defined in Figure 4 to specify user behavior during execution of the protocol* π.

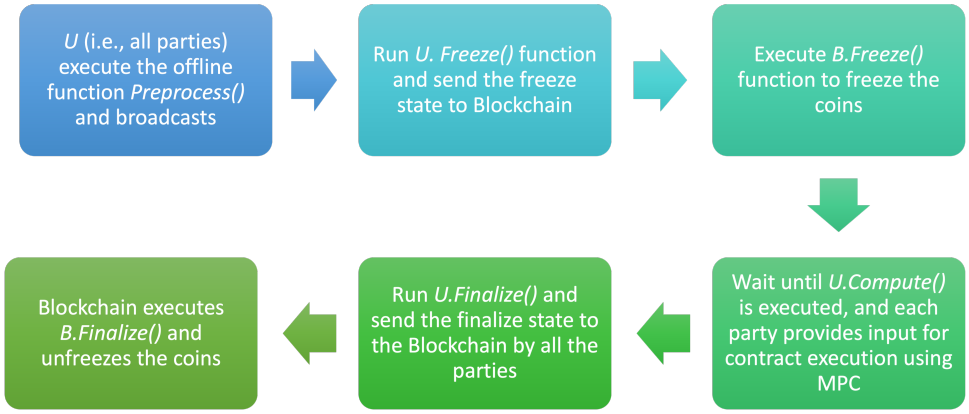

Figure 2. V-zkHawk pipeline: The step-by-step protocol overview.

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

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

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

Figure 3. Blockchain program wrapper $\mathcal{G}(B)$ for V-zkHawk evaluation.

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

Preprocess:
Computations $\leftarrow \{\}$
Coins $\leftarrow \{\}$
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 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 msg to blockchain
Remove (f', P, sid) from Computations
Remove $(\mathsf{sid}, \mathsf{coin})$ from 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 $\mathcal{Z}$

Figure 4. User program wrapper $\mathcal{H}(U)$ 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) [9]. 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 [14]), 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 act a public key and private key for the signature scheme to be used in V-zkHawk. We also observe that in this signature scheme setting the secret-to-public-key homomorphic (as explained in Definition 4) 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 5) and send the CRS to each other via a UC broadcast protocol [43]. We assume the presence of *authenticated, asynchronous* broadcast channels among the parties and the blockchain. The broadcast protocol Π_{ASBC} is defined in Figure 6.

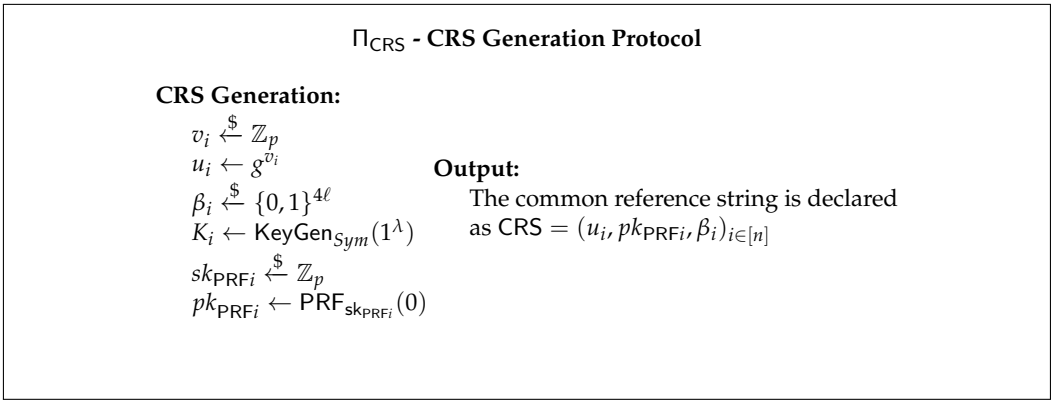

Figure 5. UC CRS generation protocol.

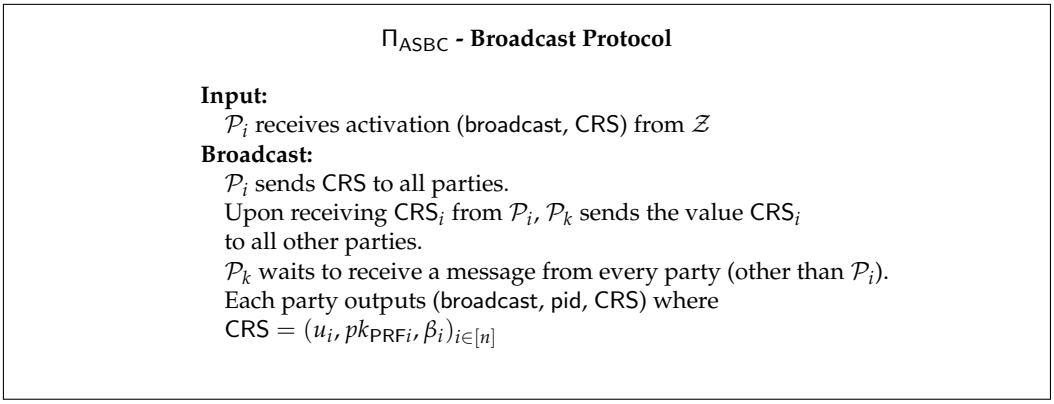

Figure 6. UC Broadcast protocol for the CRS.

3.2. Freeze Phase

The first phase of the protocol is the 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 7 message which generates a freeze to be sent to the blockchain.

As in [8], we generate the coin commitments based on the Pederson commitment scheme [44]. 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 x and the blinding factor (or randomness here) be r. Then a Pedersen commitment is defined as

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

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. For $b = 0$, the commitment operation works as shown above. For $b = 1$, the term β is included as a random string which returns $\mathsf{Com}(x, r) \oplus \beta$ as the output.

The blockchain, on receiving this message, instantiates the blockchain freeze protocol (as in Figure 8) 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.

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

Input:

$\mathcal{P}_i$ receives CRS from broadcast protocol Π_{ASBC}

Evaluate:

$\mathcal{P}_i$ 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_{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 $(\mathsf{freeze}, \mathsf{sid}, \mathsf{pid}, u_i, \mathsf{coin}_i, \mathsf{sn}_i, \pi_i, \mathsf{CRS})$

Figure 7. 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 $\mathcal{P}_i$

Evaluate:

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 sn_i to 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 sid to Contracts.

Figure 8. The Blockchain function for the freeze phase in V-zkHawk.

3.3. Computation Phase

The computation phase is where the smart contract function is 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 [7]. Consider a smart contract function $f : (\mathbb{Z} \times \{0,1\}^*)^n \to \mathbb{Z}^n \times \{0,1\}^*$. Let $\$\mathsf{val}_i$ and in_i be the input currency value and input string of party $\mathcal{P}_i$ respectively. 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 coin'_i after it learns its output value $\$\mathsf{val}'_i$. There are two underlying key ideas for this protocol. The 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 [32], 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 4) 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 [45]. We refer the reader to Figure 1 of [45] for the ideal $\mathcal{F}_{Sig}$ functionality that holds in the UC model. We can also optimize using elliptic pairing signature schemes such as aggregate multi-signature BLS [35] or compact BLS [46]. The compute function is defined as in Figure 9.

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

Evaluate:

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

Figure 9. 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 10). 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 11).

$U.Finalize()$ **- 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

Evaluate:

$\mathcal{P}_i$ 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

Figure 10. The User function for the finalization phase in V-zkHawk.

$B.Finalize()$ **- 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 $\mathcal{P}_i$

Evaluate:

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 coin'_j to Coins

Figure 11. The Blockchain function for the finalization phase in V-zkHawk.

3.5. 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 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 plan to further introduce the precondition of *deposits* in our future work. 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 by any protocol Π that is 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 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 [17,47] we elaborate on the real and ideal world executions:

Real World Execution: The protocol Π 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 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 [39,43] $\mathcal{F}_{ASBC}$ in Figure 12.

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.

Figure 12. The ideal functionality for broadcasting among the parties.

From [18,39] we borrow the idea of a non-interactive commitment schemes and define the commitment functionality $\mathcal{F}_{Com}$ (Figure 13). 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.

$\mathcal{F}_{Com}$ - Ideal 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 Commit messages from $\mathcal{P}_i$ to $\mathcal{B}$ with the same sid.

Reveal Phase:
On receiving the message $(\mathsf{Reveal}, \mathsf{sid})$ 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

Figure 13. The ideal commitment functionality.

Next, we define the ideal CRS functionality $\mathcal{F}_{CRS}$ (Figure 14) and the ideal NIZK functionality $\mathcal{F}_{NIZK}$ (Figure 15). We leverage these functionalities as shown in [31]. 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.

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

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

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

Figure 14. The ideal CRS functionality.

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

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

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

Figure 15. The ideal NIZK functionality.

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 via a secure channel, executes the smart contract, and sends the output values after the smart contract execution back to the respective parties (Figure 16).

$\mathcal{F}_{MPC}$ - Ideal MPC 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 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 $\mathcal{P}_i\ \forall\ \mathsf{pid} \in P$

Figure 16. Idealized MPC functionality $\mathcal{F}_{MPC}$ that executes the smart contract function on inputs supplied by the set of parties.

4.1. Threat Model

The adversary $\mathcal{A}$ (or $\mathcal{S}$ in an ideal world) can corrupt parties at the beginning of the protocol hence making it static. During the execution of the protocol, the adversary cannot randomly choose new parties to corrupt other than the ones already corrupted at the beginning. When the environment ($\mathcal{Z}$) interacts with an honest party, it can only read the input/output tape. On the contrary, when $\mathcal{Z}$ interacts with a dishonest party (by extension the adversary) it can read or write the contents of its input/output tape. The adversary can also communicate with the protocol (or ideal functionality) freely but can only run the Turing machines for corrupted parties.

4.2. Security Proof

In the security proof for Theorem 1, we show that indistinguishability between two hybrids (say Hybrid i and Hybrid $i+1$) based on the hardness of some problem X involves giving a reduction to solving an instance of X. In other words, we show that, if the difference between the success of the adversary in Hybrid i and the success of the adversary in Hybrid $i+1$ is non-negligible, then we can use this to solve an instance of problem X with non-negligible probability. Hybrid i and $i+1$ should be the same except for one single change which relates to problem X. Our security game involves the adversary $\mathcal{A}$ (by extension the environment $\mathcal{Z}$) guessing whether you are in the real world or some ideal world. We also assume the problem X is DDH i.e., to distinguish between a tuple $(g, g^a, g^b, g^{(ab)})$ generated by choosing uniform g, a and b, and a tuple (g, g^a, g^b, u) generated by choosing uniform g, a, b and u (see Definition 5). In the reduction, we are the adversary playing the DDH game (i.e., problem X) and we are given a tuple (g, A, B, C) as the challenge. We have to decide whether C is $g^{(ab)}$ or some uniform element u (note in this case $A = g^a$ and $B = g^b$ for some a, b). Then, simulate the security game for the Hybrids $i/i+1$ i.e., we simulate the view of the adversary who is interacting with these hybrids. Our goal is to embed the challenge from the DDH game (we for example use g, A and B for some elements) and say will use C for the one single change there is between the two hybrids. So if $C = g^(ab)$ then we successfully simulate Hybrid 1 whereas if $C = u$ (i.e., uniformly random) then we successfully simulate Hybrid 2. As a result, we use an adversary with a non-negligible difference in probability between its success in Hybrid 1 and its success in Hybrid 2 to distinguish the distributions in DDH. Since we assume DDH is hard, we can argue it is hard to distinguish the hybrids.

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 $\mathcal{F}_{Com}$ instead of the commitment protocol using $\mathcal{F}_{\mathsf{CRS}}$. We observe that, as in the 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 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 π 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 π' 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 π' in the same way that $\mathcal{S}$ does without the knowledge of the witness. Similarly for π 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 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 from which it follows that Hybrid 2 is indistinguishable from Hybrid 3 and Hybrid 2 is indistinguishable from the real world and finally Hybrid 1 is indistinguishable from the real world. This proves that running π, π', MPC, Π_{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. Conclusions

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 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. [48] described an interesting approach to work on two-round broadcast MPC with minimal setup. It would be interesting to observe how we can modify the designed V-zkHawk broadcast two-round MPC method with said method [48] as well as other two-round broadcast optimal methods [49,50]. We can further improve the security of our protocol by extending it to work for *adaptive* adversaries [51]. The Global setup UC model (GUC) [28,29] 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 π. 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 leveraging the Global Setup construction of GUC. Furthermore, we plan to provide an implementation of the above V-zkHawk in a follow-up work outlining the feasibility and complexity of running said protocol as compared to other existing private smart contract protocols.

Author Contributions: Conceptualization, A.B. and H.T.; Formal analysis, A.B.; Funding acquisition, H.T.; Investigation, A.B. and H.T.; Methodology, A.B.; Supervision, H.T.; Validation, A.B.; Writing—original draft, A.B.; Writing—review & editing, A.B. and H.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Science Foundation Ireland under grants 13/RC/2106 (ADAPT) and 17/SP/5447 (FinTech Fusion).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: I would like to thank Michael Clear for the discussions on MPC functions and the UC model.

Conflicts of Interest: The authors declare no conflict of interest.

Notations/Symbols

The following abbreviations are used in this manuscript:

$a \xleftarrow{\$} B$	This notation denotes that a is sampled according to the distribution B
$[n]$	The set of elements $\{1, ..., n\}$
$C \leftarrow \mathsf{Com}(x, r)$	C is a commitment for the value x and randomness r
$A_1 \overset{\approx}{\underset{C}{}} A_2$	The distributions A_1 and A_2 are computationally indistinguishable
$\mathcal{Z}$	Environment
$\mathcal{A}$	Real World Adversary
$\mathcal{S}$	Ideal World Simulator

$	Currency values
pid	Party identifiers
sid	Session identifiers
$\mathcal{F}_x$	x is an ideal functionality
$U.x$ or $B.x$	x is a V-zkHawk function
P	Set of pid_i
K_i	Symmetric Key for Encryption
$\sum_{\text{Sig}}$	EUF-CMA One time Key homomorphic signature scheme

Appendix A. UC Framework

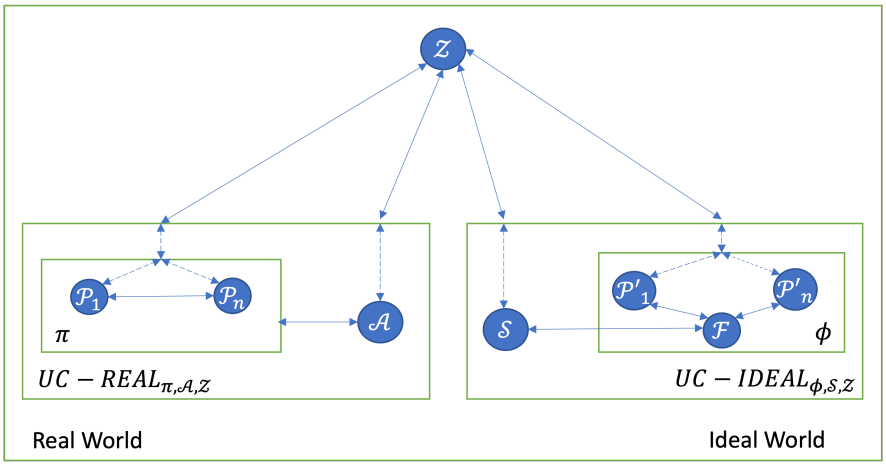

Figure A1. Overview of the Real/Ideal World Execution in the UC Framework. The dotted lines symbolizes the interaction with the environment $\mathcal{Z}$. The solid lines indicates interaction among the parties (honest/corrupted).

References

1. Savelyev, A. Contract law 2.0:'Smart'contracts as the beginning of the end of classic contract law. *Inf. Commun. Technol. Law* **2017**, *26*, 116–134. [CrossRef]
2. Buterin, V. A Next Generation Smart Contract & Decentralized Application Platform (2013) Whitepaper. Ethereum Foundation. 2013. Available online: https://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf (accessed on 23 June 2022).
3. Sasson, E.B.; Chiesa, A.; Garman, C.; Green, M.; Miers, I.; Tromer, E.; Virza, M. Zerocash: Decentralized anonymous payments from bitcoin. In Proceedings of the 2014 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 18–21 May 2014; pp. 459–474.
4. Groth, J. On the size of pairing-based non-interactive arguments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Vienna, Austria, 2016; pp. 305–326.
5. Groth, J. Short pairing-based non-interactive zero-knowledge arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*; Springer: Singapore, 2010; pp. 321–340.
6. Banerjee, A.; Clear, M.; Tewari, H. Demystifying the Role of zk-SNARKs in Zcash. In Proceedings of the 2020 IEEE Conference on Application, Information and Network Security (AINS), Kota Kinabalu, Malaysia, 17–19 November 2020; pp. 12–19.
7. Kosba, A.; Miller, A.; Shi, E.; Wen, Z.; Papamanthou, C. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In Proceedings of the 2016 IEEE symposium on security and privacy (SP), San Jose, CA, USA, 22–26 May 2016; pp. 839–858.
8. Banerjee, A.; Clear, M.; Tewari, H. zkHawk: Practical Private Smart Contracts from MPC-based Hawk. *arXiv* **2021**, arXiv:2104.09180.
9. Kursawe, K.; Danezis, G.; Kohlweiss, M. Privacy-friendly aggregation for the smart-grid. In *Privacy Enhancing Technologies*; Springer: New York, NY, USA, 2011; pp. 175–191.

10. Keller, M. MP-SPDZ: A versatile framework for multi-party computation. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual, 9–13 November 2020; pp. 1575–1590.
11. Yao, A.C. Protocols for secure computations. In Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), Chicago, IL, USA, 3–5 November 1982; pp. 160–164. [CrossRef]
12. Micali, S.; Goldreich, O.; Wigderson, A. How to play any mental game. In Proceedings of the Nineteenth ACM Symposium on Theory of Computing, STOC, ACM, New York, NY, USA, 1 January 1987; pp. 218–229.
13. 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*; Springer: Accra Beach, Barbados, 2009; pp. 325–343.
14. 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*; Springer: Egham, UK, 2013; pp. 1–18.
15. 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. *Comput. J.* **2018**, *61*, 1749–1771. [CrossRef]
16. Boyle, E.; Gilboa, N.; Ishai, Y. Function secret sharing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Sofia, Bulgaria, 2015; pp. 337–367.
17. Canetti, R. Universally composable security: A new paradigm for cryptographic protocols. In Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, Newport Beach, CA, USA, 8–11 October 2001; pp. 136–145.
18. Canetti, R.; Fischlin, M. Universally composable commitments. In *Annual International Cryptology Conference*; Springer: Santa Barbara, CA, USA, 2001; pp. 19–40.
19. 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*; ACM: New York, NY, USA, 2019; pp. 329–349.
20. Banerjee, A. A Fully Anonymous e-Voting Protocol Employing Universal Zk-SNARKs and Smart Contracts. In *International Congress on Blockchain and Applications*; Springer: Salamanca, Spain, 2021; pp. 349–354.
21. 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, London, UK, 11–15 November 2019; pp. 1759–1776.
22. Kalodner, H.; Goldfeder, S.; Chen, X.; Weinberg, S.M.; Felten, E.W. Arbitrum: Scalable, private smart contracts. In Proceedings of the 27th {USENIX} Security Symposium ({USENIX} Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1353–1370.
23. Kerber, T.; Kiayias, A.; Kohlweiss, M. KACHINA–Foundations of Private Smart Contracts. In Proceedings of the 2021 IEEE 34th Computer Security Foundations Symposium (CSF), IEEE, Dubrovnik, Croatia, 21–24 June 2021; pp. 1–16.
24. 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*; Springer: Kota Kinabalu, Malaysia, 10–14 February 2020; pp. 423–443.
25. Botta, V.; Friolo, D.; Venturi, D.; Visconti, I. Shielded computations in smart contracts overcoming forks. In Proceedings of the International Conference on Financial Cryptography and Data Security, Virtual Event, 1–5 March 2021; pp. 73–92.
26. Yuan, R.; Xia, Y.B.; Chen, H.B.; Zang, B.Y.; Xie, J. Shadoweth: Private smart contract on public blockchain. *J. Comput. Sci. Technol.* **2018**, *33*, 542–556. [CrossRef]
27. Costan, V.; Devadas, S. Intel sgx explained. *IACR Cryptol. ePrint Arch.* **2016**, *2016*, 1–118.
28. Canetti, R.; Dodis, Y.; Pass, R.; Walfish, S. Universally composable security with global setup. In *Theory of Cryptography Conference*; Springer: Amsterdam, The Netherlands, 2007; pp. 61–85.
29. 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*; Springer: Durham, NC, USA, 2020; pp. 1–30.
30. Feige, U.; Lapidot, D.; Shamir, A. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.* **1999**, *29*, 1–28. [CrossRef]
31. Groth, J.; Ostrovsky, R.; Sahai, A. New techniques for noninteractive zero-knowledge. *J. ACM (JACM)* **2012**, *59*, 1–35. [CrossRef]
32. 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*; Springer: St. Petersburg, Russia, 2006; pp. 465–485.
33. Maxwell, G.; Poelstra, A.; Seurin, Y.; Wuille, P. Simple Schnorr multi-signatures with applications to Bitcoin. *Des. Codes Cryptogr.* **2019**, *87*, 2139–2164. [CrossRef]
34. 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*; Springer: Gold Coast, Australia, 2001; pp. 514–532.
35. 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*; Springer: Warsaw, Poland, 2003; pp. 416–432.
36. Johnson, D.; Menezes, A.; Vanstone, S. The elliptic curve digital signature algorithm (ECDSA). *Int. J. Inf. Secur.* **2001**, *1*, 36–63. [CrossRef]
37. Derler, D.; Slamanig, D. Key-homomorphic signatures: Definitions and applications to multiparty signatures and non-interactive zero-knowledge. *Des. Codes Cryptogr.* **2019**, *87*, 1373–1413. [CrossRef]
38. Canetti, R. Security and Composition of Cryptographic Protocols: A Tutorial. Cryptology ePrint Archive, Report 2006/465. 2006. Available online: https://ia.cr/2006/465 (accessed on 23 June 2022).

39. Canetti, R.; Lindell, Y.; Ostrovsky, R.; Sahai, A. Universally composable two-party and multi-party secure computation. In Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing, Montreal, Quebec, QC, Canada, 19–21 May 2002; pp. 494–503.
40. Li, Q.; Cascudo, I.; Christensen, M.G. Privacy-preserving distributed average consensus based on additive secret sharing. In Proceedings of the 2019 27th European Signal Processing Conference (EUSIPCO), A Coruna, Spain, 2–6 September 2019; pp. 1–5.
41. Krawczyk, H. Secret sharing made short. In *Annual International Cryptology Conference*; Springer: Santa Barbara, CA, USA, 1993; pp. 136–146.
42. Damgård, I.; Geisler, M.; Krøigaard, M.; Nielsen, J.B. Asynchronous multiparty computation: Theory and implementation. In *International Workshop on Public Key Cryptography*; Springer: Irvine, CA, USA, 2009; pp. 160–179.
43. Goldwasser, S.; Lindell, Y. Secure multi-party computation without agreement. *J. Cryptol.* **2005**, *18*, 247–287. [CrossRef]
44. Pedersen, T.P. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*; Springer: Santa Barbara, CA, USA, 1991; pp. 129–140.
45. Canetti, R. Universally composable signature, certification, and authentication. In Proceedings of the 17th IEEE Computer Security Foundations Workshop, Pacific Grove, CA, USA, 30 June 2004; pp. 219–233.
46. 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*; Springer: Brisbane, QLD, Australia, 2018; pp. 435–464.
47. Laud, P.; Kamm, L. *Applications of Secure Multiparty Computation*; IOS Press: Amsterdam, The Netherlands, 2015; Volume 13.
48. Damgård, I.; Ravi, D.; Siniscalchi, L.; Yakoubov, S. Minimizing Setup in Broadcast-Optimal Two Round MPC. Cryptology ePrint Archive, Report 2022/293. 2022. Available online: https://ia.cr/2022/293 (accessed on 23 June 2022).
49. Cohen, R.; Garay, J.; Zikas, V. Broadcast-optimal two-round MPC. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Zagreb, Croatia, 2020; pp. 828–858.
50. Ciampi, M.; Ostrovsky, R.; Waldner, H.; Zikas, V. Round-Optimal and Communication-Efficient Multiparty Computation. Cryptology ePrint Archive, Report 2020/1437. 2020. Available online: https://ia.cr/2020/1437 (accessed on 23 June 2022).
51. Garg, S.; Polychroniadou, A. Two-round adaptively secure MPC from indistinguishability obfuscation. In *Theory of Cryptography Conference*; Springer: Warsaw, Poland, 2015; pp. 614–637.