On the Efficiency of Privacy-Preserving Smart Contract Systems

Karim Baghery

University of Tartu, Estonia

Abstract. Along with blockchain technology, smart contracts have found intense interest in lots of practical applications. A smart contract is a mechanism involving digital assets and some parties, where the parties deposit assets into the contract and the contract redistributes the assets among the parties based on provisions of the smart contract and inputs of the parties. Recently, several smart contract systems are constructed that use zk-SNARKs to provide privacy-preserving payments and interconnections in the contracts (e.g. Hawk [KMS+16] and Gyges [JKS16]). Efficiency of such systems severely are dominated by efficiency of the underlying UC-secure zk-SNARK that is achieved using C∅C∅ framework [KZM+15] applied on a non-UC-secure zk-SNARK. In this paper, we show that recent progresses on zk-SNARKs, allow one to simplify the structure and also improve the efficiency of both systems with a UC-secure zk-SNARK that has simpler construction and better efficiency in comparison with the currently used ones. More precisely, with minimal changes, we present a variation of Groth and Maller's zk-SNARK from Crypto 2017, and show that it achieves UC-security and has better efficiency than the ones that currently are used in Hawk and Gyges. We believe, new variation can be of independent interest.

Keywords: privacy-preserving smart contracts, zk-SNARKs, UC-security, CRS model

1 Introduction

Eliminating the need for a trusted third party in monetary transactions, consequently enabling direct transactions between individuals is one of the main achievements in the cryptocurrencies such as Bitcoin. Importantly, it is shown that the technology behind cryptocurrencies has more potential than what only is used in direct transactions. Different blockchain-based systems such as smart contracts [KMS+16,JKS16], distributed cloud storages [WLB14], digital coins such as Ethereum [Woo14] are some evidence that why blockchain technology offers much more functionalities than what we can see in Bitcoin. Smart contracts are one of popular applications that along with blockchain technology, have found intense interest recently. A smart contract is a generic term denoting programs written in Turing-complete cryptocurrency scripting languages, that involves digital assets and some parties. The parties deposit assets into the contract and the contract redistributes the assets among the parties based on provisions of the smart contract and inputs of the parties.

Different research have shown that even if payments (e.g. in Bitcoin) or interconnections (e.g. in smart contracts) are conducted between pseudorandom addresses, but still they lack privacy of end-users. Indeed, this mostly arises from the nature of technology that a decentralized publicly shared ledger records list of transactions along with related information (e.g. addresses of parties, transferred values, etc), and long-time monitoring and some data analysis (e.g. transaction graph analysis) on this ledger usually reveals some information about the identity of end-users. To address these concerns and provide strong privacy for end-users, several alternatives to Bitcoin protocol and smart contract systems have been proposed; e.g. confidential assets [PBF$^+$18], privacy-preserving auditing [NVV18], privacy-preserving cryptocurrencies such as Zerocash [BCG$^+$14] and Monero [Noe15], privacy-preserving smart contract systems such as Hawk [KMS$^+$16] and Gyges [JKS16].

Zerocash and Monero are two known anonymous cryptocurrencies that provide strong privacy for end-users. Each of them uses different cryptographic tools to guarantee strong privacy. Monero uses ring signatures that allow for an individual from a group to provide a signature such that it is impossible to identify which member of that group made the signature. On the other side, Zerocash uses zero-knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARKs [Gro10,Lip12,PHGR13,BCTV13,Gro16,GM17]) to prove the correctness of all computations inside a direct transaction, without revealing the source, destination and values of the transferred coins. In a similar technique, privacy-preserving smart contract system Hawk [KMS$^+$16] and criminal smart contract system Gyges [JKS16] use universally composable zk-SNARKs to provide anonymous interconnection and payment in a smart contract.

zk-SNARKs. Among various Non-Interactive Zero-Knowledge (NIZK) arguments, zk-SNARKs are one of the most popular ones in practical systems. This is happened because of their succinct proofs, and consequently very efficient verification. A zk-SNARK proof allows one to efficiently verify the veracity of statements without learning extra information about the prover. The proofs can be verified offline very quickly (in few milliseconds) by possibly many independent verifiers. This can be very effective in efficiency of large-scale distributed systems. Efficiency of zk-SNARKs mainly comes from the fact that their construction relies on non-falsifiable assumptions (e.g. knowledge assumptions [Dam91]) that allow succinct proofs and non-black-box extraction in security proofs. On the other hand, a zk-SNARK with non-black-box extraction cannot achieve Universally Composable Security (UC-security) which is imperative and necessary in constructing larger cryptographic systems [Can01]. Du to this fact, zk-SNAKRs cannot be directly adopted in larger systems that should guarantee UC-security.

Privacy-preserving smart contract systems. Recently, some elegant UC-based frameworks are presented that allow to construct privacy-preserving smart contracts, including Hawk [KMS$^+$16] and Gyges [JKS16] for criminal smart contracts. These systems record zk-SNARK proofs on ledger, instead of public transactions between pseudonyms, which brings stronger transactional privacy. Strictly speaking, Hawk is a system that gets a program and compiles it to a cryptographic protocol between the contract correspondents (including users and a manager) and the blockchain. It consists of two main blocks, where one is responsible for *private money transfers* and uses a variation of Zerocash [BCG$^+$14], while the second part handles other contract-defined operations of the system. Similar to Zerocash, operations such as *Mint*, that is required in minting a new coin, and *Pour*, that enables anonymous transactions, are located in the first block. On the other side, contract-related operations such as *Freeze*, *Compute* and *Finalize*, that are three necessary operations defined by Hawk for each smart contract, are addressed in the second block. More details regard to the mentioned operations can be found in [KMS$^+$16] [1]. To achieve anonymity in the mentioned operations and payments, Hawk widely uses zk-SNARKs to prove different statements. As the whole system intended to achieve UC-security, so they needed to use a UC-secure zk-SNARK in the system. Additionally, since Zerocash also uses a non-UC-secure zk-SNARK and it is not proved to satisfy UC-security, so to make it useable in Hawk, they needed a variation of Zerocash that uses a UC-secure zk-SNARK and also guarantees UC-security. To this aim, designers of Hawk have used C∅C∅ framework [KZM$^+$15] (a framework to lift a non-UC-secure sound NIZK to a UC-secure one; C∅C∅ stands for *Composable 0-knowledge, Compact 0-knowledge*) to lift the non-UC-secure zk-SNARK used in Zerocash [BCTV13], to a UC-secure zk-SNARK, such that the lifted scheme can be securely used in composition with the rest of system [Can01]. Then, due to using a UC-secure zk-SNARK in Zerocash, designer of Hawk modified the structure of original Zerocash and used the customized version in their system, which also guarantees UC-security. The lifted UC-secure zk-SNARK frequently is used in the system and plays an essential role in the efficiency of entire system.

Problem statement. In the performance evaluation of Hawk [KMS$^+$16] authors show that the efficiency of their system severely depend on efficiency of the lifted UC-secure zk-SNARK (which is the case in Gyges [JKS16] as well). In fact, computational complexity of both systems are dominated with complexity of the underlying UC-secure zk-SNARK. Particularly, Kosba et al. [KMS$^+$16] emphasize that practical efficiency is a permanent goal of Hawk's design, so to get the best, they also propose various optimizations. By considering this, one may ask, can we improve efficiency of the underlying UC-secure zk-SNARKs such that the efficiency of complete systems will be improved?

Our Contribution. As the main contribution, we show that one can improve efficiency of Hawk (and similarly Gyges) smart contract system by improving the efficiency of underlying UC-secure zk-SNARK. We will see that one can use a similar approach used by Kosba et al. (in Hawk [KMS$^+$16]) and Juels et al. (in Gyges [JKS16]) and construct a UC-secure version of Groth and Maller's zk-SNARK [GM17] (refereed as GM zk-SNARK in the rest), that has simpler

[1] A tutorial about the system can be found in `http://cryptowiki.net/index.php?title=Privacy_preserving_smart_contracts:_Hawk_project`

construction and better efficiency than the ones that currently are used in the systems. To do so, we slightly modify the construction of GM zk-SNARK by enforcing the prover to send encryption of witnesses along with the proof, and then show that it achieves black-box simulation extractability, equivalently UC-security, which allows to deploy in both systems to improve their efficiency.

Both Hawk and Gyges have used C∅C∅ framework to lift a variation of Pinocchio zk-SNARK [PHGR13] which is deployed in Zerocash (proposed by Ben Sasson et al. [BCTV13]). Later it details we show that, as GM zk-SNARK [GM17] has better efficiency than the mentioned variation of Pinocchio zk-SNARK, and as our changes are lighter than the changes that are applied on Ben Sasson et al.'s zk-SNARK in Hawk [KMS+16] and Gyges [JKS16], so we get a UC-secure zk-SNARK that has simpler construction and better efficiency than the ones that currently are deployed in the systems. Indeed, we will see that our changes are a small part of their changes, which leads to have less overload.

In the modified construction, we do the changes in CRS circuit level and try to keep the prover and verifier procedure as original one that both are considerably optimized in the original construction [GM17]. We believe, new constructed UC-secure zk-SNARK can be of independent interest and it can be deployed in any large cryptographic system that aims to guarantee UC-security and needs to use zk-SNARKs. From a different perspective, new construction also can be used as a commit-and-proof system, as prover can send encryption (sort of commitment) of witnesses earlier than the proof elements. In such cases, one can consider linear commitment size and succinct proof size (proof would be 2 elements in $\mathbb{G}_1$ and 1 element in $\mathbb{G}_2$). We note that in UC-secure zk-SNARKs, the proofs are linear in witness size but still independent of size of the circuit that encodes language.

Discussion of UC-secure NIZKs. Most of efficient zk-SNARKs only guarantee *knowledge soundness*, meaning that if an adversary can come up with a valid proof, there exists an extractor that can extract the witness from the adversary. But in some cases, e.g. in signatures of knowledge SoKs [CL06], *knowledge soundness* is not enough, and one needs more security guarantee. More accurately, most of zk-SNARKs are vulnerable to the malleability attack which allows an adversary to modify an old proof to a new valid one, that is not desired in some cases. To address this, the notion of *simulation exractability* is defined which ensures that an adversary cannot come up with a new acceptable proof (or an argument), even if he already has seen arbitrary simulated proofs, unless he knows the witness. In other words, simulation extractability implies that if an adversary, who has obtained arbitrary number of simulated proofs, can generate an acceptable new proof for a statement, there exists an extractor that can extract the witness. Based on extraction procedure which is categorized as Black-Box (BB) or non-Black-Box (nBB), there are various notions of simulation extractibility [Gro06,KZM+15,GM17]. In BB extraction, there exists a black-box (universal) extractor which can extract the witness from all adversaries, however in the nBB extraction, for each adversary there exists a particular extractor that can extract only if it has access to the adversary's source code and random coins. It is already observed and proven that a NIZK system that achieves simulation extractibility with BB extraction, can guarantee the UC-security [CLOS02,Gro06,GOS06].Therefore, constructing a simulation-extractable zk-SNARK with BB extraction is equivalent to constructing a UC-secure zk-SNARK (which the proof will be only circuit succinct). Strictly speaking, in a UC-secure NIZK the simulator of *ideal-world* should be able to extract witnesses without getting access to the source code of environment's algorithm, which this is guaranteed by BB extraction.

A known technique to achieve a simulation-extractable NIZK with BB extraction is to enforce the prover to send the encryption of witnesses (with a public key given in the CRS) along with proof, so that in security proofs the extractor can use the pair secret key for extraction [Gro06]. Using this technique, the proof (communication) size will not be succinct anymore, as impossibility result in [GW11] confirms, but the verification will be efficient yet and the extraction issue that zk-SNARKs have in the UC framework [Can01] will be solved.

2 Preliminaries

Let PPT denote probabilistic polynomial-time, and NUPPT denote non-uniform PPT. Let $\lambda \in \mathbb{N}$ be the security parameter, say $\lambda = 128$. All adversaries will be stateful. For an algorithm $\mathcal{A}$, let $\mathrm{im}(\mathcal{A})$ be the image of $\mathcal{A}$, i.e., the set of valid outputs of $\mathcal{A}$, let $\mathsf{RND}(\mathcal{A})$ denote the random tape of $\mathcal{A}$, and let $r \leftarrow_{\$} \mathsf{RND}(\mathcal{A})$ denote sampling of a randomizer r of sufficient length for $\mathcal{A}$'s needs. By $y \leftarrow \mathcal{A}(x; r)$ we mean given an input x and a randomizer r, $\mathcal{A}$ outputs y. For algorithms $\mathcal{A}$ and $\mathsf{Ext}_{\mathcal{A}}$, we write $(y \,\|\, y') \leftarrow (\mathcal{A} \,\|\, \mathsf{Ext}_{\mathcal{A}})(x; r)$ as a shorthand for "$y \leftarrow \mathcal{A}(x; r)$, $y' \leftarrow \mathsf{Ext}_{\mathcal{A}}(x; r)$". An arbitrary negligible function is shown with $\mathsf{negl}(\lambda)$. Two computationally indistinguishable distributions A and B are shown with $A \approx_c B$.

In pairing-based groups, we use additive notation together with the bracket notation, i.e., in group $\mathbb{G}_\mu$, $[a]_\mu = a\,[1]_\mu$, where $[1]_\mu$ is a fixed generator of $\mathbb{G}_\mu$. A *bilinear group generator* $\mathsf{BGgen}(1^\lambda)$ returns $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, [1]_1, [1]_2)$, where p (a large prime) is the order of cyclic abelian groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$. Finally, $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficient non-degenerate bilinear pairing, s.t. $\hat{e}([a]_1, [b]_2) = [ab]_T$. Denote $[a]_1 \bullet [b]_2 = \hat{e}([a]_1, [b]_2)$.

We bellow review Square Arithmetic Programs (SAPs) that defines NP-complete language specified by a quadratic equation over polynomials [GM17].

Square Arithmetic Program: Any quadratic arithmetic circuit with fan-in 2 gates over a finite field $\mathbb{Z}_p$ can be lifted to a SAP instance over the same finite field (e.g. by considering $ab = ((a+b)^2 - (a-b)^2)/4$) [GM17]. A SAP instance contains $\mathcal{S}_p = (\mathbb{Z}_p, m_0, \{u_j, w_j\}_{j=0}^m)$. This instance defines the following relation:

$$\mathbf{R}_{\mathcal{S}_p} = \left\{ \begin{array}{l} (\mathsf{x}, \mathsf{w}) \colon \mathsf{x} = (A_1, \ldots, A_{m_0})^\top \wedge \mathsf{w} = (A_{m_0+1}, \ldots, A_m)^\top \wedge \\ \left(\sum_{j=0}^m A_j u_j(X)\right)^2 \equiv \sum_{j=0}^m A_j w_j(X) \pmod{\ell(X)} \end{array} \right\}$$

where $\ell(X) := \prod_{i=1}^n (X - \omega^{i-1}) = X^n - 1$ is the unique degree n monic polynomial such that $\ell(\omega^{i-1}) = 0$ for all $i \in [1 \mathinner{..} n]$. Alternatively, $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}_{\mathcal{S}_p}$ if there exists a (degree $\leq n-2$) polynomial $h(X)$, s.t. $\left(\sum_{j=0}^{m} A_j u_j(X)\right)^2 - \sum_{j=0}^{m} A_j w_j(X) = h(X)\ell(X)$.

2.1 Definitions

We use the definitions of NIZK arguments from [Gro06,Gro16,GM17,KZM+15]. Let $\mathcal{R}$ be a relation generator, such that $\mathcal{R}(1^\lambda)$ returns a polynomial-time decidable binary relation $\mathbf{R} = \{(\mathsf{x}, \mathsf{w})\}$. Here, x is the statement and w is the witness. We assume one can deduce λ from the description of $\mathbf{R}$. The relation generator also outputs auxiliary information $\xi_{\mathbf{R}}$ that will be given to the honest parties and the adversary. As in [Gro16,ABLZ17], $\xi_{\mathbf{R}}$ is the value returned by $\mathsf{BGgen}(1^\lambda)$. Due to this, we also give $\xi_{\mathbf{R}}$ as an input to the honest parties; if needed, one can include an additional auxiliary input to the adversary. Let $\mathbf{L_R} = \{\mathsf{x} : \exists \mathsf{w}, (\mathsf{x}, \mathsf{w}) \in \mathbf{R}\}$ be an **NP**-language.

A *NIZK argument system Ψ* for $\mathcal{R}$ consists of tuple of PPT algorithms, s.t.:

CRS generator: K is a PPT algorithm that given $(\mathbf{R}, \xi_{\mathbf{R}})$, where $(\mathbf{R}, \xi_{\mathbf{R}}) \in \operatorname{im}(\mathcal{R}(1^\lambda))$ outputs $\mathbf{crs} = (\mathbf{crs}_{\mathsf{P}}, \mathbf{crs}_{\mathsf{V}})$ and stores trapdoors of $\mathbf{crs}$ as $\mathbf{ts}$. We distinguish $\mathbf{crs}_{\mathsf{P}}$ (needed by the prover) from $\mathbf{crs}_{\mathsf{V}}$ (needed by the verifier).

Prover: P is a PPT algorithm that, given $(\mathbf{R}, \xi_{\mathbf{R}}, \mathbf{crs}_{\mathsf{P}}, \mathsf{x}, \mathsf{w})$, where $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$, outputs an argument π. Otherwise, it outputs $\perp$.

Verifier: V is a PPT algorithm that, given $(\mathbf{R}, \xi_{\mathbf{R}}, \mathbf{crs}_{\mathsf{V}}, \mathsf{x}, \pi)$, returns either 0 (reject) or 1 (accept).

Simulator: Sim is a PPT algorithm that, given $(\mathbf{R}, \xi_{\mathbf{R}}, \mathbf{crs}, \mathbf{ts}, \mathsf{x})$, outputs an argument π.

Extractor: Ext is a PPT algorithm that, given $(\mathbf{R_L}, \xi_{\mathbf{R_L}}, \mathbf{crs}, \mathsf{x}, \pi, \mathbf{te})$ extracts the w; where $\mathbf{te}$ is extraction trapdoor (e.g. a secret key).

We require an argument system Ψ to be complete, computationally knowledge-sound and statistically ZK, as in the following definitions.

Definition 1 (Perfect Completeness [Gro16]). *A non-interactive argument Ψ is* perfectly complete for $\mathcal{R}$, *if for all λ, all $(\mathbf{R}, \xi_{\mathbf{R}}) \in \operatorname{im}(\mathcal{R}(1^\lambda))$, and $(\mathsf{x}, \mathsf{w}) \in \mathbf{R}$,*

$$\Pr\left[\mathbf{crs} \leftarrow \mathsf{K}(\mathbf{R}, \xi_{\mathbf{R}}) : \mathsf{V}(\mathbf{R}, \xi_{\mathbf{R}}, \mathbf{crs}_{\mathsf{V}}, \mathsf{x}, \mathsf{P}(\mathbf{R}, \xi_{\mathbf{R}}, \mathbf{crs}_{\mathsf{P}}, \mathsf{x}, \mathsf{w})) = 1\right] = 1 \ .$$

Definition 2 (Computational Knowledge-Soundness [Gro16]). *A non-interactive argument Ψ is computationally (adaptively)* knowledge-sound for $\mathcal{R}$, *if for every NUPPT $\mathcal{A}$, there exists a NUPPT extractor $\mathsf{Ext}_{\mathcal{A}}$, s.t. for all λ,*

$$\Pr\left[\begin{array}{l} (\mathbf{R}, \xi_{\mathbf{R}}) \leftarrow \mathcal{R}(1^\lambda), (\mathbf{crs} \,\|\, \mathbf{ts}) \leftarrow \mathsf{K}(\mathbf{R}, \xi_{\mathbf{R}}), \\ r \leftarrow_r \mathsf{RND}(\mathcal{A}), ((\mathsf{x}, \pi) \,\|\, \mathsf{w}) \leftarrow (\mathcal{A} \,\|\, \mathsf{Ext}_{\mathcal{A}})(\mathbf{R}, \xi_{\mathbf{R}}, \mathbf{crs}; r) : \\ (\mathsf{x}, \mathsf{w}) \not\in \mathbf{R} \wedge \mathsf{V}(\mathbf{R}, \xi_{\mathbf{R}}, \mathbf{crs}_{\mathsf{V}}, \mathsf{x}, \pi) = 1 \end{array}\right] \approx_\lambda 0 \ .$$

Here, $\xi_{\mathbf{R}}$ can be seen as a common auxiliary input to $\mathcal{A}$ and $\mathsf{Ext}_{\mathcal{A}}$ that is generated by using a benign [BCPR14] relation generator; A knowledge-sound argument system is called an *argument of knowledge*.

Definition 3 (Statistically Zero-Knowledge [Gro16]). *A non-interactive argument Ψ is* statistically ZK for $\mathcal{R}$*, if for all* λ*, all* $(\mathbf{R}, \xi_{\mathbf{R}}) \in \operatorname{im}(\mathcal{R}(1^\lambda))$*, and for all NUPPT* $\mathcal{A}$*,* $\varepsilon_0^{unb} \approx_\lambda \varepsilon_1^{unb}$*, where*

$$\varepsilon_b = \Pr[(\mathsf{crs} \,\|\, \mathsf{ts}) \leftarrow \mathsf{K}(\mathbf{R}, \xi_{\mathbf{R}}) : \mathcal{A}^{\mathsf{O}_b(\cdot,\cdot)}(\mathbf{R}, \xi_{\mathbf{R}}, \mathsf{crs}) = 1] \ .$$

Here, the oracle $\mathsf{O}_0(\mathsf{x}, \mathsf{w})$ *returns* $\perp$ *(reject) if* $(\mathsf{x}, \mathsf{w}) \not\in \mathbf{R}$*, and otherwise it returns* $\mathsf{P}(\mathbf{R}, \xi_{\mathbf{R}}, \mathsf{crs}_{\mathsf{P}}, \mathsf{x}, \mathsf{w})$*. Similarly,* $\mathsf{O}_1(\mathsf{x}, \mathsf{w})$ *returns* $\perp$ *(reject) if* $(\mathsf{x}, \mathsf{w}) \not\in \mathbf{R}$*, and otherwise it returns* $\mathsf{Sim}(\mathbf{R}, \xi_{\mathbf{R}}, \mathsf{crs}, \mathsf{x}, \mathsf{ts})$*.* Ψ *is* perfect ZK for $\mathcal{R}$ *if one requires that* $\varepsilon_0 = \varepsilon_1$*.*

Intuitively, a non-interactive argument Ψ is zero-knowledge if it does not leak extra information besides the truth of the statement. Beside the mentioned properties defined in Def. 1-3, a zk-SNARK has *succinctness* property, meaning that the proof size is $\mathsf{poly}(\lambda)$ and the verifier's computation is $\mathsf{poly}(\lambda)$ and the size of instance. In the rest, we recall the definitions of simulation soundness and simulation extractability that are used in construction of UC-secure zk-SNARKs.

Definition 4 (Simulation Soundness [Gro06]). *A non-interactive argument Ψ is* simulation sound for $\mathcal{R}$ *if for all NUPPT* $\mathcal{A}$*, and all* λ*,*

$$\Pr\left[\begin{array}{l}(\mathbf{R}, \xi_{\mathbf{R}}) \leftarrow \mathcal{R}(1^\lambda), (\mathsf{crs} \,\|\, \mathsf{ts}) \leftarrow \mathsf{K}(\mathbf{R}, \xi_{\mathbf{R}}), (\mathsf{x}, \pi) \leftarrow \mathcal{A}^{\mathsf{O}(\cdot)}(\mathbf{R}, \xi_{\mathbf{R}}, \mathsf{crs}) : \\ (\mathsf{x}, \pi) \not\in Q \wedge \mathsf{x} \not\in \mathbf{L} \wedge \mathsf{V}(\mathbf{R}, \xi_{\mathbf{R}}, \mathsf{crs}_{\mathsf{V}}, \mathsf{x}, \pi) = 1\end{array}\right] \approx_\lambda 0 \ .$$

Here, Q is the set of simulated statement-proof pairs generated by adversary's queries to O, that returns simulated proofs.

Definition 5 (Non-Black-Box Simulation Extractability [GM17]). *A non-interactive argument Ψ is* non-black-box simulation-extractable for $\mathcal{R}$*, if for any NUPPT* $\mathcal{A}$*, there exists a NUPPT extractor* $\mathsf{Ext}_{\mathcal{A}}$ *s.t. for all* λ*,*

$$\Pr\left[\begin{array}{l}(\mathbf{R}, \xi_{\mathbf{R}}) \leftarrow \mathcal{R}(1^\lambda), (\mathsf{crs} \,\|\, \mathsf{ts}) \leftarrow \mathsf{K}(\mathbf{R}, \xi_{\mathbf{R}}), \\ r \leftarrow_r \mathsf{RND}(\mathcal{A}), ((\mathsf{x}, \pi) \,\|\, \mathsf{w}) \leftarrow (\mathcal{A}^{\mathsf{O}(\cdot)} \,\|\, \mathsf{Ext}_{\mathcal{A}})(\mathbf{R}, \xi_{\mathbf{R}}, \mathsf{crs}; r) : \\ (\mathsf{x}, \pi) \not\in Q \wedge (\mathsf{x}, \mathsf{w}) \not\in \mathbf{R} \wedge \mathsf{V}(\mathbf{R}, \xi_{\mathbf{R}}, \mathsf{crs}_{\mathsf{V}}, \mathsf{x}, \pi) = 1\end{array}\right] \approx_\lambda 0 \ .$$

Here, Q is the set of simulated statement-proof pairs generated by adversary's queries to O that returns simulated proofs. It is worth to mention that *non-black-box simulation extractability* implies *knowledge soundness* (given in Def. 2), as the earlier is a strong notion of the later which additionally the adversary is allowed to send query to the proof simulation oracle. Similarly, one can observe that *non-black-box simulation extractability* implies *simulation soundness* (given in Def. 4) that is discussed in [Gro06] with more details.

Definition 6 (Black-Box Simulation Extractability [KZM$^+$15]). *A non-interactive argument Ψ is* black-box simulation-extractable for $\mathcal{R}$ *if there exists a black-box extractor* Ext *that for all NUPPT* $\mathcal{A}$*, and all* λ*,*

$$\Pr\left[\begin{array}{l}(\mathbf{R}, \xi_{\mathbf{R}}) \leftarrow \mathcal{R}(1^\lambda), (\mathsf{crs} \,\|\, \mathsf{ts} \,\|\, \mathsf{te}) \leftarrow \mathsf{K}(\mathbf{R}, \xi_{\mathbf{R}}), \\ (\mathsf{x}, \pi) \leftarrow \mathcal{A}^{\mathsf{O}(\cdot)}(\mathbf{R}, \xi_{\mathbf{R}}, \mathsf{crs}), \mathsf{w} \leftarrow \mathsf{Ext}(\mathbf{R}, \xi_{\mathbf{R}}, \mathsf{crs}, \mathsf{te}, \mathsf{x}, \pi) : \\ (\mathsf{x}, \pi) \not\in Q \wedge (\mathsf{x}, \mathsf{w}) \not\in \mathbf{R} \wedge \mathsf{V}(\mathbf{R}, \xi_{\mathbf{R}}, \mathsf{crs}_{\mathsf{V}}, \mathsf{x}, \pi) = 1\end{array}\right] \approx_\lambda 0 \ .$$

Similarly, Q is the set of simulated statement-proof pairs, and te is the extraction trapdoor. A key note about Def. 6 is that the extraction procedure is black-box and unlike the non-black-box case, the extractor Ext works for all adversaries.

2.2 C∅C∅ : a Framework for Constructing UC-secure zk-SNARKs

Kosba et al. [KZM+15] have constructed a framework with several converters which the most powerful one gets a sound NIZK and lifts to a NIZK that achieves *black-box simulation extractability* (defined in Def. 6), or equivalently UC-security [Gro06]. Here we review construction of the most powerful converter that is used by both Hawk and Gyges to construct a UC-secure zk-SNARK.

Construction. Given a sound NIZK, to achieve a UC-secure NIZK, C∅C∅ framework applies several changes in all setup, proof generation and verification procedures of the input NIZK. Initially the framework defines a new language $\mathbf{L}'$ based on the language $\mathbf{L}$ in underlying NIZK and some new primitives that are needed for the transformation. Let $(\mathsf{KGen}_e, \mathsf{Enc}_e, \mathsf{Dec}_e)$ be a set of algorithms for a semantically secure encryption scheme, $(\mathsf{KGen}_s, \mathsf{Sig}_s, \mathsf{Vfy}_s)$ be a one-time signature scheme and $(\mathsf{Com}_c, \mathsf{Vfy}_c)$ be a perfectly binding commitment scheme. Given a language $\mathbf{L}$ with the corresponding $\mathbf{NP}$ relation $\mathbf{R_L}$, define a new language $\mathbf{L}'$ such that $((\mathsf{x}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r, r_0, \mathsf{w}, s_0)) \in \mathbf{R_{L'}}$ iff:

$$(c = \mathsf{Enc}_e(\mathsf{pk}_e, \mathsf{w}; r)) \wedge ((\mathsf{x}, \mathsf{w}) \in \mathbf{R_L} \vee (\mu = f_{s_0}(\mathsf{pk}_s) \wedge \rho = \mathsf{Com}_c(s_0; r_0))) ,$$

where $\{f_s : \{0,1\}^* \to \{0,1\}^\lambda\}_{s \in \{0,1\}^\lambda}$ is a pseudo-random function family. Now, a sound NIZK argument system Ψ for $\mathcal{R}$ constructed from PPT algorithms $(\mathsf{K}, \mathsf{P}, \mathsf{V}, \mathsf{Sim}, \mathsf{Ext})$ can be lifted to a UC-secure NIZK Ψ' with PPT algorithms $(\mathsf{K}', \mathsf{P}', \mathsf{V}', \mathsf{Sim}', \mathsf{Ext}')$ as follows.

CRS and trapdoor generation $\mathsf{K}'(\mathbf{R_L}, \xi_{\mathbf{R_L}})$**:** Sample $(\mathsf{crs} \parallel \mathsf{ts}) \leftarrow \mathsf{K}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}})$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KGen}_e(1^\lambda)$; $s_0, r_0 \leftarrow_\$ \{0,1\}^\lambda$; $\rho := \mathsf{Com}_c(s_0; r_0)$; and output $(\mathsf{crs}' \parallel \mathsf{ts}' \parallel \mathsf{te}') := ((\mathsf{crs}, \mathsf{pk}_e, \rho) \parallel (s_0, r_0) \parallel \mathsf{sk}_e)$.

Prover $\mathsf{P}'(\mathbf{R_L}, \xi_{\mathbf{R_L}}, \mathsf{crs}, \mathsf{x}, \mathsf{w})$**:** Parse $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pk}_e, \rho)$; Abort if $(\mathsf{x}, \mathsf{w}) \notin \mathbf{R_L}$; $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KGen}_s(1^\lambda)$; sample $z_0, z_1, z_2, r_1 \leftarrow_\$ \{0,1\}^\lambda$; compute $c = \mathsf{Enc}_e(\mathsf{pk}_e, \mathsf{w}; r_1)$; generate $\pi \leftarrow \mathsf{P}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}, \mathsf{crs}, (\mathsf{x}, c, z_0, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r_1, z_1, w, z_2))$; sign $\sigma \leftarrow \mathsf{Sig}_s(\mathsf{sk}_s, (\mathsf{x}, c, z_0, \pi))$; and output $\pi' := (c, z_0, \pi, \mathsf{pk}_s, \sigma)$.

Verifier $\mathsf{V}'(\mathbf{R_L}, \xi_{\mathbf{R_L}}, \mathsf{crs}', \mathsf{x}, \pi')$**:** Parse $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pk}_e, \rho)$ and $\pi' := (c, \mu, \pi, \mathsf{pk}_s, \sigma)$; Abort if $\mathsf{Vfy}_s(\mathsf{pk}_s, (\mathsf{x}, c, \mu, \pi), \sigma) = 0$; call $\mathsf{V}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}, \mathsf{crs}, (\mathsf{x}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), \pi)$ and abort if it outputs 0.

Simulator $\mathsf{Sim}'(\mathbf{R_L}, \xi_{\mathbf{R_L}}, \mathsf{crs}', \mathsf{ts}', \mathsf{x})$**:** Parse $\mathsf{crs}' := (\mathsf{crs}, \mathsf{pk}_e, \rho)$ and $\mathsf{ts}' := (s_0, r_0)$; $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KGen}_s(1^\lambda)$; set $\mu = f_{s_0}(\mathsf{pk}_s)$; sample $z_3, r_1 \leftarrow_\$ \{0,1\}^\lambda$; compute $c = \mathsf{Enc}_e(\mathsf{pk}_e, z_3; r_1)$; generate $\pi \leftarrow \mathsf{P}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}, \mathsf{crs}, (\mathsf{x}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r_1, r_0, z_3, s_0))$; sign $\sigma \leftarrow \mathsf{Sig}_s(\mathsf{sk}_s, (\mathsf{x}, c, \mu, \pi))$; and output $\pi' := (c, \mu, \pi, \mathsf{pk}_s, \sigma)$.

Extractor $\mathsf{Ext}'(\mathbf{R_L}, \xi_{\mathbf{R_L}}, \mathsf{crs}', \mathsf{te}', \mathsf{x}, \pi')$**:** Parse $\pi' := (c, \mu, \pi, \mathsf{pk}_s, \sigma)$, $\mathsf{te}' := \mathsf{sk}_e$; extract $\mathsf{w} \leftarrow \mathsf{Dec}_e(\mathsf{sk}_e, c)$; output w.

On input a SAP instance $\mathcal{S}_p = (\mathbb{Z}_p, m_0, \{u_j, w_j\}_{j=0}^{m}, \ell)$.

$\mathsf{K}(\mathbf{R}_{\mathcal{S}_p}, \xi_{\mathbf{R}})$: Pick $\mathfrak{g}_1 \leftarrow_r \mathbb{G}_1^*, \mathfrak{g}_2 \leftarrow_r \mathbb{G}_2^*$, $(\alpha, \beta, \gamma, \chi) \leftarrow_r (\mathbb{Z}_p^*)^4$ (such that $\ell(\chi) \neq 0$), generate $\mathsf{crs} \leftarrow (\mathsf{crs}_\mathsf{P}, \mathsf{crs}_\mathsf{V})$ and return $(\mathsf{crs}, \mathsf{ts})$; where $\mathsf{ts} = (\alpha, \beta, \gamma, \chi)$ and

$$\mathsf{crs}_\mathsf{P} \leftarrow \begin{pmatrix} \mathbf{R}_{\mathcal{S}_p}, \left[\alpha, \gamma\ell(\chi), \gamma^2\ell(\chi)^2, (\alpha+\beta)\gamma\ell(\chi), (\gamma\chi^i, \gamma^2\ell(\chi)\chi^i)_{i=0}^{n-1}\right]_1, \\ \left[(\gamma^2 w_i(\chi) + (\alpha+\beta)\gamma u_i(\chi))_{i=m_0+1}^{m}\right]_1, \left[\gamma\ell(\chi), (\gamma\chi^i)_{i=0}^{n-1}\right]_2 \end{pmatrix},$$

$$\mathsf{crs}_\mathsf{V} \leftarrow \left([\alpha, \gamma, (\gamma w_i(\chi) + (\alpha+\beta)u_i(\chi))_{i=0}^{m_0}]_1, [1, \beta, \gamma]_2 \right).$$

$\mathsf{P}(\mathbf{R}_{\mathcal{S}_p}, \xi_{\mathbf{R}}, \mathsf{crs}_\mathsf{P}, \mathsf{x} = (A_1, \dots, A_{m_0}), \mathsf{w} = (A_{m_0+1}, \dots, A_m))$:

```
1. Let $a^\dagger(X) \leftarrow \sum_{j=0}^{m} A_j u_j(X)$,
2. Let $c^\dagger(X) \leftarrow \sum_{j=0}^{m} A_j w_j(X)$,
3. Set $h(X) = \sum_{i=0}^{n-2} h_i X^i \leftarrow (a^\dagger(X)^2 - c^\dagger(X))/\ell(X)$,
4. Set $\left[\gamma^2 h(\chi)\ell(\chi)\right]_1 \leftarrow \sum_{i=0}^{n-2} h_i \left[\gamma^2\chi^i\ell(\chi)\right]_1$,
5. Pick $r \leftarrow_r \mathbb{Z}_p$; Set
   – $\mathfrak{a} \leftarrow \left(\sum_{j=0}^{m} A_j [\gamma u_j(\chi)]_1 + r[\gamma\ell(\chi)]_1\right)$
   – $\mathfrak{b} \leftarrow \left(\sum_{j=0}^{m} A_j [\gamma u_j(\chi)]_2 + r[\gamma\ell(\chi)]_2\right)$
   – $\mathfrak{c} \leftarrow \sum_{j=m_0+1}^{m} A_j \left[(\gamma^2 w_j(\chi) + (\alpha+\beta)\gamma u_j(\chi)\right]_1 + r^2\left[\gamma^2\ell(\chi)^2\right]_1 + r[(\alpha+\beta)\gamma\ell(\chi)]_1 + \left[\gamma^2\ell(\chi)\left(h(\chi) + 2r\sum_{j=0}^{m} A_j u_j(\chi)\right)\right]_1$
6. Return $\pi \leftarrow (\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$.
```

$\mathsf{V}(\mathbf{R}_{\mathcal{S}_p}, \xi_{\mathbf{R}}, \mathsf{crs}_\mathsf{V}, \mathsf{x} = (A_1, \dots, A_{m_0}), \pi = (\mathfrak{a}, \mathfrak{b}, \mathfrak{c}))$: assuming $A_0 = 1$, check

$$\mathfrak{a} \bullet [\gamma]_1 = [\gamma]_2 \bullet \mathfrak{b} ,$$

$$(\mathfrak{a} + [\alpha]_1) \bullet (\mathfrak{b} + [\beta]_2) = [\alpha]_1 \bullet [\beta]_2 + \\ + \left(\sum_{j=0}^{m_0} A_j [(\gamma w_j(\chi) + (\alpha+\beta)u_j(\chi)]_1\right) \bullet [\gamma]_2 + \mathfrak{c} \bullet [1]_2 .$$

$\mathsf{Sim}(\mathbf{R}_{\mathcal{S}_p}, \xi_{\mathbf{R}}, \mathsf{crs}, \mathsf{x} = (A_1, \dots, A_{m_0}), \mathsf{ts})$: Pick $\mu \leftarrow \mathbb{Z}_p^*$, and compute $\pi = (\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ such that

$$\mathfrak{a} \leftarrow [\mu]_1, \; \mathfrak{b} \leftarrow [\mu]_2, \; \mathfrak{c} \leftarrow \left[\mu^2 + (\alpha+\beta)\mu - \gamma\sum_{j=0}^{m_0} A_j(\gamma w_j(\chi) + (\alpha+\beta)u_j(\chi))\right]_1$$

Fig. 1: Structure of GM zk-SNARK [GM17]

2.3 Groth and Maller's zk-SNARK

This section presents the construction of GM zk-SNARK that is presented by Groth and Maller in [GM17] [2]. It is the first SAP-based zk-SNARK that achieves non-black-box simulation extractability, which makes the scheme secure against the malleability attacks. The structure of GM zk-SNARK is shown in Fig. 1.

[2] We use original construction of GM zk-SNARK that is published in Crypto 2017 [GM17] and implemented in `Libsnark` library `https://github.com/scipr-lab/libsnark`. But one also can use the variation of GM zk-SNARK that recently is provided in full version of paper.

3 An Efficient UC-secure zk-SNARK

We present a variation of GM zk-SNARK [GM17] and show that it achieves black-box simulation extractability, and equivalently UC-security. GM zk-SNARK is the only scheme that guarantees non-black-box simulation extractablity which is stronger than knowledge-soundness that is usually achieved in most of pairing-based zk-SNARKs. We show that due to this strong security, with minimal changes we can achieve a UC-secure version of GM zk-SNARK.

Intuition. The goal is to present a UC-secure version of GM zk-SNARK but efficient than UC-secure zk-SNARKs that are lifted by C∅C∅ framework; especially more efficient than the ones that are deployed in [KMS+16,JKS16]. To do so, we slightly modify GM zk-SNARK and enforce prover P to encrypt its witnesses with a public key given in the CRS and send the ciphertext along with the proof. In this scenario, in security proof, the secret key of encryption scheme is given to the Ext which allows to extract witnesses in black-box manner, that is more realistic indeed. Actually this is an already known technique to achieve black-box extraction that also is used in C∅C∅ framework. It is undeniable that sending encryption of witnesses leads to have non-succinct proofs in witness size but still they are succinct in the size of circuit that encodes the language and it is simpler and more efficient than the ones that are lifted by C∅C∅ .

3.1 Construction

While modifying we keep internal computation of both prover and verifier as original one, that considerably are optimized for a SAP relation. Instead we define a new language $\mathbf{L}'$ based on the language $\mathbf{L}$ in GM zk-SNARK that is embedded with encryption of witness. Strictly speaking, given a language $\mathbf{L}$ with the corresponding $\mathbf{NP}$ relation $\mathbf{R_L}$, we define the following new language $\mathbf{L}'$ such that $((\mathsf{x}, c, \mathsf{pk}_e), (\mathsf{w}, r)) \in \mathbf{R_{L'}}$ iff:

$$(c = \mathsf{Enc}_e(\mathsf{pk}_e, \mathsf{w}; r)) \wedge ((\mathsf{x}, \mathsf{w}) \in \mathbf{R_L}),$$

where $(\mathsf{KGen}_e, \mathsf{Enc}_e, \mathsf{Dec}_e)$ is a set of algorithms for a semantically secure encryption scheme with keys $(\mathsf{pk}_e, \mathsf{sk}_e)$. Accordingly, the modified version of GM zk-SNARK is given in Fig. 2. It is worth to mention that, due to the particular structure of new language $\mathbf{L}'$, all verifications will be done inside the circuit, and interestingly verifier and prover's internal computations are the same as before, just prover needs to send encryption of witnesses along with the proof. This is the key modification in removing nBB extraction (particularly knowledge-assumption based in zk-SNARKs) and achieving BB extraction.

3.2 Efficiency

In the modified scheme, as the original one, proof is 2 elements in $\mathbb{G}_1$ and 1 element in $\mathbb{G}_2$, but along with c that is encryption of witnesses. So, proof size is dominated with size of c that is linear in witness size.

CRS and trapdoor generation $\mathsf{K}'(\mathbf{R_L}, \xi_{\mathbf{R_L}})$**:** Generate key pair $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KGen}_e(1^\lambda)$; execute CRS generator of GM zk-SNARK and sample $(\mathbf{crs} \,\|\, \mathbf{ts}) \leftarrow \mathsf{K}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}})$; output $(\mathbf{crs}' \,\|\, \mathbf{ts}' \,\|\, \mathbf{te}') := ((\mathbf{crs}, \mathsf{pk}_e) \,\|\, \mathbf{ts} \,\|\, \mathsf{sk}_e)$; where $\mathbf{ts}'$ are simulation trapdoors and $\mathbf{te}'$ is the extraction key.

Prover $\mathsf{P}'(\mathbf{R_L}, \xi_{\mathbf{R_L}}, \mathbf{crs}', \mathsf{x}, \mathsf{w})$**:** Parse $\mathbf{crs}' := (\mathbf{crs}, \mathsf{pk}_e)$; Abort if $(\mathsf{x}, \mathsf{w}) \notin \mathbf{R_L}$; sample $r \leftarrow_\$ \{0,1\}^\lambda$; compute encryption of witnesses $c = \mathsf{Enc}_e(\mathsf{pk}_e, \mathsf{w}; r)$; execute prover P of GM zk-SNARK and generate $\pi \leftarrow \mathsf{P}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}, \mathbf{crs}, (\mathsf{x}, c, \mathsf{pk}_e), (\mathsf{w}, r))$; and output $\pi' := (c, \pi)$.

Verifier $\mathsf{V}'(\mathbf{R_L}, \xi_{\mathbf{R_L}}, \mathbf{crs}', \mathsf{x}, \pi')$**:** Parse $\mathbf{crs}' := (\mathbf{crs}, \mathsf{pk}_e)$ and $\pi' := (c, \pi)$; call verifier $\mathsf{V}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}, \mathbf{crs}, (\mathsf{x}, c, \mathsf{pk}_e), \pi)$ of GM zk-SNARK and abort if it rejects.

Simulator $\mathsf{Sim}'(\mathbf{R_L}, \xi_{\mathbf{R_L}}, \mathbf{crs}', \mathsf{x}, \mathbf{ts}')$**:** Parse $\mathbf{crs}' := (\mathbf{crs}, \mathsf{pk}_e)$ and $\mathbf{ts}' := \mathbf{ts}$; sample $z, r \leftarrow_\$ \{0,1\}^\lambda$; compute $c = \mathsf{Enc}_e(\mathsf{pk}_e, z; r)$; execute simulator of GM zk-SNARK and generate $\pi \leftarrow \mathsf{Sim}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}, \mathbf{crs}, (\mathsf{x}, c, \mathsf{pk}_e), \mathbf{ts})$; and output $\pi' := (c, \pi)$.

Extractor $\mathsf{Ext}'(\mathbf{R_L}, \xi_{\mathbf{R_L}}, \mathbf{crs}', \mathbf{te}', \mathsf{x}, \pi')$**:** Parse $\pi' := (c, \pi)$ and $\mathbf{te}' := \mathsf{sk}_e$; extract $\mathsf{w} \leftarrow \mathsf{Dec}_e(\mathsf{sk}_e, c)$; output w.

Fig. 2: GM zk-SNARK with black-box simulation extractability

As verifier is untouched, so similar to GM zk-SNARK, the verification procedure consists of checking that the proof contains 3 appropriate group elements and checking 2 pairing product equations which in total it needs a multi-exponentiation $\mathbb{G}_1$ to m_0 exponents and 5 pairings.

In the setup, in result of our change, the arithmetic circuit will be slightly extended, but due to minimal changes (a more detailed comparison is provided in Fig. 3), the extension is less than the case that one uses C∅C∅ framework.

3.3 Security Proof

Theorem 1 (Perfect Completeness). *The protocol constructed in Sec. 3, is a non-interactive argument of knowledge that guarantees perfect completeness.*

Proof. We emphasizes that in the modified version, the internal computations of P and V are the same as original one, just they need to perform the computation for new SAP instance that has slightly larger size (e.g. $n = n_{old} + n_{new}$, where n is number of squaring gates in the new circuit, and n_{new} is the number of squaring gates that are added in result of new changes) and prover needs to output some new elements that are encryption of witnesses and will be used inside the unchanged verification equations. So by considering this fact, one can see that the completeness of modified protocol follows the original protocol. □

Theorem 2 (Computationally Zero-Knowledge). *The protocol constructed in Sec. 3, is a non-interactive argument of knowledge that guarantees computational zero-knowledge.*

Proof. To prove the theorem, we write a series of hybrid experiments that start from an experiment that encrypts a random value and uses the simulator, and finally gets to an experiment that uses the procedure of real prover. We show that all experiments are indistinguishable two-by-two. Before going through the games, recall that GM zk-SNARK scheme guarantees zero-knowledge and its simulation procedure is given in Fig. 1. So, consider the following experiment,

```
$\mathsf{EXP}_1^{zk}$
– Setup: $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KGen}_e(1^\lambda); (\mathbf{crs} \parallel \mathbf{ts}) \leftarrow \mathsf{K}(\mathbf{R}_{\mathbf{L}'}, \xi_{\mathbf{R}_{\mathbf{L}'}}); \mathbf{crs}' = (\mathbf{crs}, \mathsf{pk}_e)$
– $\mathsf{O}(\mathsf{x},\mathsf{w})$ : Abort if $(\mathsf{x},\mathsf{w}) \notin \mathbf{R}_\mathbf{L}$; Sample $z, r \leftarrow \{0,1\}^\lambda; c = \mathsf{Enc}_e(\mathsf{pk}_e, z; r)$;
        $\pi \leftarrow \mathsf{Sim}(\mathbf{R}_{\mathbf{L}'}, \xi_{\mathbf{R}_{\mathbf{L}'}}, \mathbf{crs}, (\mathsf{x}, c, \mathsf{pk}_e), \mathbf{ts})$;
– $b \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x},\mathsf{w})}(\mathbf{crs}')$;
return $b$; fi
```

```
$\mathsf{EXP}_2^{zk}$
– Setup: $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KGen}_e(1^\lambda); (\mathbf{crs} \parallel \mathbf{ts}) \leftarrow \mathsf{K}(\mathbf{R}_{\mathbf{L}'}, \xi_{\mathbf{R}_{\mathbf{L}'}}); \mathbf{crs}' = (\mathbf{crs}, \mathsf{pk}_e)$
– $\mathsf{O}(\mathsf{x},\mathsf{w})$ : Abort if $(\mathsf{x},\mathsf{w}) \notin \mathbf{R}_\mathbf{L}$; Sample $r \leftarrow \{0,1\}^\lambda$; $c = \mathsf{Enc}_e(\mathsf{pk}_e, \mathsf{w}; r)$;
        $\pi \leftarrow \mathsf{Sim}(\mathbf{R}_{\mathbf{L}'}, \xi_{\mathbf{R}_{\mathbf{L}'}}, \mathbf{crs}, (\mathsf{x}, c, \mathsf{pk}_e), \mathbf{ts})$;
– $b \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x},\mathsf{w})}(\mathbf{crs}')$;
return $b$; fi
```

Lemma 1. *If the used cryptosystem in the above games is semantically secure, then for two experiments* EXP_2^{zk} *and* EXP_1^{zk}*, we have* $\Pr[\mathsf{EXP}_2^{zk}] \approx \Pr[\mathsf{EXP}_1^{zk}]$.

Proof. By considering the fact that the cryptosystem $\Pi_{enc} = (\mathsf{KGen}_e, \mathsf{Enc}_e, \mathsf{Dec}_e)$ is a semantically secure, so no polynomial-time algorithm can distinguish an oracle that encrypts randomly chosen value z and uses simulator Sim from the case that it encrypts witness w and again uses Sim. □

```
$\mathsf{EXP}_3^{zk}$
– Setup:$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KGen}_e(1^\lambda); (\mathbf{crs} \parallel \mathbf{ts}) \leftarrow \mathsf{K}(\mathbf{R}_{\mathbf{L}'}, \xi_{\mathbf{R}_{\mathbf{L}'}}); \mathbf{crs}' = (\mathbf{crs}, \mathsf{pk}_e)$
– $\mathsf{O}(\mathsf{x},\mathsf{w})$ : Abort if $(\mathsf{x},\mathsf{w}) \notin \mathbf{R}_\mathbf{L}$; Sample $r \leftarrow \{0,1\}^\lambda; c = \mathsf{Enc}_e(\mathsf{pk}_e, \mathsf{w}; r)$;
        $\pi \leftarrow \mathsf{P}(\mathbf{R}_{\mathbf{L}'}, \xi_{\mathbf{R}_{\mathbf{L}'}}, \mathbf{crs}, (\mathsf{x}, c, \mathsf{pk}_e), (\mathsf{w}, r))$;
– $b \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x},\mathsf{w})}(\mathbf{crs}')$;
return $b$; fi
```

Lemma 2. *For experiments* EXP_3^{zk} *and* EXP_2^{zk} *we have* $\Pr[\mathsf{EXP}_3^{zk}] \approx \Pr[\mathsf{EXP}_2^{zk}]$.

Proof. As GM zk-SNARK guarantees zero-knowledge, so one can conclude that the real proof (generated by prover) in experiment EXP_3^{zk} is indistinguishable from the the simulated proof (generated by simulator) in experiment EXP_2^{zk}. □

This completes the proof of theorem. □

Theorem 3 (Black-Box Simulation Extractability). *Assuming the encryption scheme is semantically secure and perfectly correct, the modified version of GM zk-SNARK in Sec. 3, satisfies black-box simulation extractability.*

Proof. Similarly, we go through a sequence of hybrid experiences. The proof uses a similar approach that is used in C∅C∅ framework and consequently in Hawk and Gyges, but with considerable simplifications. As the first experiment, consider the following experiment,

```
$\mathsf{EXP}_1^{SimExt}$
– Setup: $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KGen}_e(1^\lambda); (\mathbf{crs} \,\|\, \mathbf{ts}) \leftarrow \mathsf{K}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}); \mathbf{crs}' = (\mathbf{crs}, \mathsf{pk}_e)$
– $\mathsf{O}(\mathsf{x})$ : Sample $r, z \leftarrow \{0,1\}^\lambda; c = \mathsf{Enc}_e(\mathsf{pk}_e, z; r)$;
        $\pi \leftarrow \mathsf{Sim}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}, \mathbf{crs}, (\mathsf{x}, c, \mathsf{pk}_e), \mathbf{ts})$; output $\pi' := (c, \pi)$
– $(\mathsf{x}, \pi') \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x})}(\mathbf{crs}', \mathsf{sk}_e)$;
– Parse $\pi' := (c, \pi)$; extract witness $\mathsf{w} \leftarrow \mathsf{Dec}_e(c, \mathsf{sk}_e)$;
return 1 iff $((\mathsf{x}, \pi') \notin Q) \wedge (\mathsf{V}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}, \mathbf{crs}, (\mathsf{x}, c, \mathsf{pk}_e), \pi) = 1) \wedge ((\mathsf{x}, \mathsf{w}) \notin \mathbf{R_L})$;
where $Q$ shows the set of statment-proof pairs generated by $\mathsf{O}(\mathsf{x})$. fi
```

```
$\mathsf{EXP}_2^{SimExt}$
– Setup: $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KGen}_e(1^\lambda); (\mathbf{crs} \,\|\, \mathbf{ts}) \leftarrow \mathsf{K}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}); \mathbf{crs}' = (\mathbf{crs}, \mathsf{pk}_e)$
– $\mathsf{O}(\mathsf{x})$ : Sample $r \leftarrow \{0,1\}^\lambda$; $c = \mathsf{Enc}_e(\mathsf{pk}_e, \mathsf{w}; r)$;
        $\pi \leftarrow \mathsf{Sim}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}, \mathbf{crs}, (\mathsf{x}, c, \mathsf{pk}_e), \mathbf{ts})$; output $\pi' := (c, \pi)$
– $(\mathsf{x}, \pi') \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x})}(\mathbf{crs}', \mathsf{sk}_e)$;
– Parse $\pi' := (c, \pi)$; extract witness $\mathsf{w} \leftarrow \mathsf{Dec}_e(c, \mathsf{sk}_e)$;
return 1 iff $((\mathsf{x}, \pi') \notin Q) \wedge (\mathsf{V}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}, \mathbf{crs}, (\mathsf{x}, c, \mathsf{pk}_e), \pi) = 1) \wedge ((\mathsf{x}, \mathsf{w}) \notin \mathbf{R_L})$;
where $Q$ shows the set of statment-proof pairs generated by $\mathsf{O}(\mathsf{x})$. fi
```

Lemma 3. *If the used cryptosystem in the above games is semantically secure, then for two experiments* EXP_2^{SimExt} *and* EXP_1^{SimExt} *we have* $\Pr[\mathsf{EXP}_2^{SimExt}] \approx \Pr[\mathsf{EXP}_1^{SimExt}]$.

Proof. By the fact that the used cryptosystem is semantically secure, so no polynomial-time algorithm can distinguish an oracle that encrypts randomly chosen value z and uses simulator Sim' from the one that encrypts true witness w and again uses simulator Sim'. □

```
$\mathsf{EXP}_3^{SimExt}$
– Setup: $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KGen}_e(1^\lambda); (\mathbf{crs} \,\|\, \mathbf{ts}) \leftarrow \mathsf{K}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}); \mathbf{crs}' = (\mathbf{crs}, \mathsf{pk}_e)$
– $\mathsf{O}(\mathsf{x})$ : Sample $r \leftarrow \{0,1\}^\lambda; c = \mathsf{Enc}_e(\mathsf{pk}_e, \mathsf{w}; r)$;
        $\pi \leftarrow \mathsf{P}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}, \mathbf{crs}, (\mathsf{x}, c, \mathsf{pk}_e), (\mathsf{w}, r))$; output $\pi' := (c, \pi)$
– $(\mathsf{x}, \pi') \leftarrow \mathcal{A}^{\mathsf{O}(\mathsf{x})}(\mathbf{crs}', \mathsf{sk}_e)$;
– Parse $\pi' := (c, \pi)$; extract witness $\mathsf{w} \leftarrow \mathsf{Dec}_e(c, \mathsf{sk}_e)$;
return 1 iff $((\mathsf{x}, \pi') \notin Q) \wedge (\mathsf{V}(\mathbf{R_{L'}}, \xi_{\mathbf{R_{L'}}}, \mathbf{crs}, (\mathsf{x}, c, \mathsf{pk}_e), \pi) = 1) \wedge ((\mathsf{x}, \mathsf{w}) \notin \mathbf{R_L})$;
where $Q$ shows the set of statment-proof pairs generated by $\mathsf{O}(\mathsf{x})$. fi
```

Lemma 4. *If the underlying NIZK is simulation sound, then for two experiments* EXP_3^{SimExt} *and* EXP_2^{SimExt} *we have* $\Pr[\mathsf{EXP}_3^{SimExt}] \approx \Pr[\mathsf{EXP}_2^{SimExt}]$.

Proof. We note that if $(\mathsf{x}, \pi') \notin Q$, then the (x, c, π) (from (x, π')) is a valid message pair. By simulation soundness property of GM zk-SNARK, that prevents mallability attacks, we know that $(\mathsf{x}, \pi') \notin Q$.

On the other hand, since the decrypted w is unique for all valid witnesses, so due to the soundness of GM zk-SNARK (note that the definition of simulation soundness implies standard soundness) the probability that some witness is valid for $\mathbf{L}'$ and $(\mathsf{x}, \mathsf{w}) \notin \mathbf{R_L}$ is $\mathsf{negl}(\lambda)$. $\square$

We note that in all above experiments, extraction of witnesses is done universally, independent of adversarial prover's code, that is a critical issue in constructing the UC simulator that extracts witness form the proof sent by environment and the adversarial prover. So, this results that the modified scheme satisfies *black-box simulation extractability.* Consequently, following previous result (shown in [CLOS02,Gro06,GOS06]) that a NIZK argument system with black-box simulation extractability guarantees UC-security, we conclude that the modified construction of GM zk-SNARK in Fig. 2 achieves UC-security. $\square$

4 On the Efficiency of Smart Contract Systems

Hawk and Gyges [KMS+16,JKS16] frequently generate CRS and use a UC-secure zk-SNARK to prove different statements. In Hawk author discuss that their system is dominated by efficiency of the underlying UC-secure zk-SNARK that are achieved from a variation of Pinocchio zk-SNARK [PHGR13] lifted by C∅C∅ framework (the same is done in Gyges as well). In the rest, we discuss how UC-secure construction in Sec. 3 can improve efficiency of both smart contract systems. Our evaluation is focused precisely on Hawk, but as Gyges also have used C∅C∅ framework, so the same evaluation can be considered for Gyges.

Improving Efficiency of Hawk. We begin evaluation on Hawk by reviewing the changes that are applied on Ben Sasson et al.'s zk-SNARK (to get UC-security) before using it in Hawk. As discussed in Sec. 2.2, in order to lift any NIZK to a UC-secure NIZK, C∅C∅ applies several changes in setup, proof generation and proof verification of input NIZK. For instance, each time prover needs to generate a pair of signing/verifying keys for a one-time secure signature scheme, encrypt the witnesses using a given public-key, and sign the generated proof using the mentioned one-time signing key. On the other side, verifier needs to do extra verifications than the NIZK verification.

As we discussed in Sec. 3, to achieve a UC-secure version of GM zk-SNARK, we added a key generation procedure for a public-key cryptosystem in the setup phase, and prover only needed to encrypt the witnesses using the public-key in CRS and then generate a proof for new language as the original zk-SNARK. We did not add new checking to the verifier side and it is as the non-UC-secure version.

Left side of Fig. 3 summarizes the modifications applied (by using C∅C∅) on a variation of Pinocchio zk-SNARK before using in Hawk; and right side summarizes our changes on GM zk-SNARK to get BB simulation extractability and

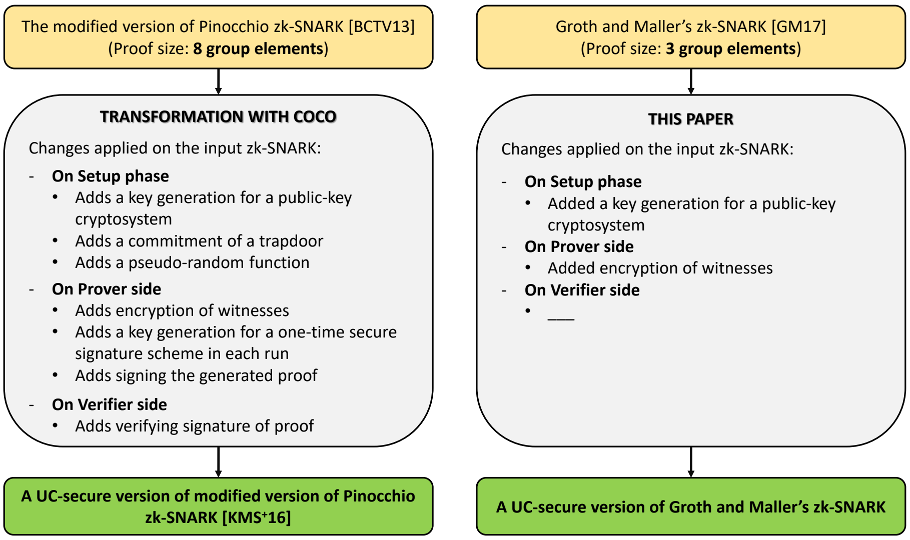
The modified version of Pinocchio zk-SNARK [BCTV13]
(Proof size: 8 group elements)
TRANSFORMATION WITH COCO
Changes applied on the input zk-SNARK:
- On Setup phase
• Adds a key generation for a public-key cryptosystem
• Adds a commitment of a trapdoor
• Adds a pseudo-random function
- On Prover side
• Adds encryption of witnesses
• Adds a key generation for a one-time secure signature scheme in each run
• Adds signing the generated proof
- On Verifier side
• Adds verifying signature of proof
A UC-secure version of modified version of Pinocchio zk-SNARK [KMS+16]
Groth and Maller's zk-SNARK [GM17]
(Proof size: 3 group elements)
THIS PAPER
Changes applied on the input zk-SNARK:
- On Setup phase
• Added a key generation for a public-key cryptosystem
- On Prover side
• Added encryption of witnesses
- On Verifier side
• ___
A UC-secure version of Groth and Maller's zk-SNARK

Fig. 3: The modifications applied by C∅C∅ transformation on the modified version of Pinocchio zk-SNARK [BCTV13] before using in Hawk system versus our changes on GM zk-SNARK (shown in Fig. 2) to get a UC-secure version.

	CRS Leg., Time	Proof Size	Prover Comp.	Verifier Comp.	Ver. Equ.
[BCTV13] &	$6m + n - m_0$ $\mathbb{G}_1$ m $\mathbb{G}_2$	7 $\mathbb{G}_1$ 1 $\mathbb{G}_2$	$6m + n - m_0$ E_1 m E_2	m_0 E_1 12 P	5
in libsnark	104.8 seconds	287 bytes	128.6 seconds	4.2 millisec.	—
[GM17] &	$m + 4n + 5$ $\mathbb{G}_1$ $2n + 3$ $\mathbb{G}_2$	2 $\mathbb{G}_1$ 1 $\mathbb{G}_2$	$m + 4n - m_0$ E_1 $2n$ E_2	m_0 E_1 5 P	2
in libsnark	100.4 seconds	127 bytes	116.4 seconds	2.3 millisec.	—

Table 1: Comparison of Ben Sasson et al.'s [BCTV13] and GM [GM17] zk-SNARKs for arithmetic circuit satisfiability with m_0 element instance, m wires, n multiplication gates. Since [GM17] uses squaring gates, so n multiplication gates translate to $2n$ squaring gates. Implementations are done on a PC with 3.40 GHz Intel Core i7-4770 CPU, in single-threaded mode, for an R1CS instance with $n = 10^6$ constraints and $m = 10^6$ variables, of which $m_0 = 10$ are input variables. $\mathbb{G}_1$ and $\mathbb{G}_2$: group elements, E: exponentiations and P: pairings.

equivalently UC-security. As both use encrypting of witnesses, it seems having linear proof size on witness size currently is an undeniable issue to get black-box extraction. So, except this unavoidable modification, we applied minimal changes in the structure of GM zk-SNARK to achieve a UC-secure version of it.

Additionally, Tab.1 compares efficiency and practical performance of Ben Sasson et al.'s [BCTV13] and GM [GM17] zk-SNARKs from various perspectives before applying any changes. Empirical performance reported in libsnark library for a particular instance[3]. The experiments are done on a machine equipped with

[3] Based on reported implementation on `https://github.com/scipr-lab/libsnark`

3.40 GHz Intel Core i7-4770 CPU, in single-threaded mode, using the BN128 curve. Following Pinocchio scheme, Ben Sasson et al.'s zk-SNARK [BCTV13] is constructed for the QAP relation, while Groth and Maller's scheme works for the SAP relation by default. As discussed in [Gro16,GM17], a SAP instance can be constructed based on a simplification of systems on arithmetic constraints, such that all multiplication gates are replaced with squaring gates, but with at most two times gates.

Tab. 1 shows that GM zk-SNARK outperforms Ben Sasson et al.'s zk-SNARK in all metrics. Beside faster running times in all algorithms, GM zk-SNARK has only 2 verification equations, instead of 5 in [BCTV13]. By considering efficiency report in Tab.1, and the fact that our modifications (summarized in Fig. 3) are lighter than what are applied on Ben Sasson et al.'s zk-SNARK before deploying in Hawk system, one can observe that new UC-secure zk-SNARK will simplify the system and would be more efficient than the one that currently is used in Hawk (similarly in Gyges). Indeed our changes are a small part of their already applied changes, so they will have less overload.

Hawk needs to generate CRS of zk-SNARK for each smart contract and as the UC-secure zk-SNARK is widely deployed in various operations of the system, so substituting current UC-secure zk-SNARK with the new one in Sec. 3, can simplify the system and improve the efficiency of whole system, specially in larger scales. Moreover, in the construction of Hawk system, authors applied various effective optimizations to maximize the efficiency of underlying UC-secure zk-SNARK (Sec. V in [KMS$^+$16]). The same techniques can work with new construction. For instance, it is shown that in the *Finalize* operation of a smart contract in Hawk, one may use non-UC-secure zk-SNARK, which similarly in new case one can use non-UC-secure version of GM zk-SNARK that is more efficient than the one that currently is used (compared in Tab. 1) and additionally it ensures non-block-box simulation extractability. In another noticeable optimization, Kosba et al. used some independently optimized primitives in the lifted UC-secure zk-SNARK, that had considerable effect in the practical efficiency of Hawk. Again, by reminding that our changes are a small part of the changes applied by C∅C∅ , so a part of their optimized primitives (for encryption scheme) can be used in this case as well, but the rest can be ignored. Based on their experiences, such optimizations lead to have a gain of more than 10$\times$ in the arithmetic circuit that is required for *Finalize* operation. We predict it should be even more with new scheme.

5 Open Discussions

In Hawk and Gyges [KMS$^+$16,JKS16], authors used the fact that Pinocchio zk-SNARK and its variation by Ben-Sasson et al. [BCTV13] satisfies *knowledge soundness* and consequently *soundness*, and then used C∅C∅ framework and lifted a variation of Pinocchio zk-SNARK to a UC-secure one. On the other hand, *knowledge soundness* of the mentioned zk-SNARKs are proven under some knowledge assumptions, that are not clear how to use such assumptions in the UC framework. We used a similar technique and corollary in our security proofs. We considered the fact that *simulation extracability* implies *simulation-soundness* [Gro06], because if we can extract a witness from the adversary's proof, then the statement must belong the language. So, an interesting future direction might be reproving the soundness of Pinocchio zk-SNARK [PHGR13] (or the variation by Ben-Sasson et al. [BCTV13]), or simulation-soundness of GM zk-SNARK [GM17] under some different non-falsifiable assumptions (different from knowledge assumptions).

Acknowledgement. The author were supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 780477 (project PRIViLEDGE), and by the Estonian Research Council grant (PRG49).

References

ABLZ17. Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2017.

BCG$^+$14. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

BCPR14. Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In David B. Shmoys, editor, *46th ACM STOC*, pages 505–514. ACM Press, May / June 2014.

BCTV13. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive arguments for a von neumann architecture. Cryptology ePrint Archive, Report 2013/879, 2013. `http://eprint.iacr.org/2013/879`.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CL06. Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Heidelberg, August 2006.

CLOS02. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

Dam91. Ivan Damgård. Towards Practical Public Key Systems Secure against Chosen Ciphertext Attacks. In Joan Feigenbaum, editor, *CRYPTO 1991*, volume 576 of *LNCS*, pages 445–456, Santa Barbara, California, USA, August 11–15, 1991. Springer, Heidelberg, 1992.

GM17. Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Heidelberg, August 2017.

GOS06. Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Heidelberg, May / June 2006.

Gro06. Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459. Springer, Heidelberg, December 2006.

Gro10. Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Heidelberg, May 2016.

GW11. Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 99–108. ACM Press, June 2011.

JKS16. Ari Juels, Ahmed E. Kosba, and Elaine Shi. The ring of Gyges: Investigating the future of criminal smart contracts. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16*, pages 283–295. ACM Press, October 2016.

KMS$^+$16. Ahmed E. Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy*, pages 839–858. IEEE Computer Society Press, May 2016.

KZM$^+$15. Ahmed E. Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, T.-H. Hubert Chan, Charalampos Papamanthou, Rafael Pass, Abhi Shelat, and Elaine Shi. C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs. Technical Report 2015/1093, IACR, November 10, 2015. `http://eprint.iacr.org/2015/1093`, last accessed version from 9 Apr 2017.

Lip12. Helger Lipmaa. Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189, Taormina, Italy, March 18–21, 2012. Springer, Heidelberg.

Noe15. Shen Noether. Ring signature confidential transactions for monero. *IACR Cryptology ePrint Archive*, 2015:1098, 2015.

NVV18. Neha Narula, Willy Vasquez, and Madars Virza. zkledger: Privacy-preserving auditing for distributed ledgers. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 65–80, 2018.

PBF$^+$18. Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In *International Conference on Financial Cryptography and Data Security*, pages 43–63. Springer, 2018.

PHGR13. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.

WLB14. Shawn Wilkinson, Jim Lowry, and Tome Boshevski. Metadisk a blockchain-based decentralized file storage application. *Storj Labs Inc., Technical Report, hal*, pages 1–11, 2014.

Woo14. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.