SoK: Lifting Transformations for Simulation Extractable Subversion and Updatable SNARKs*

Behzad Abdolmaleki\textsuperscript{1}, Sebastian Ramacher\textsuperscript{2}, and Daniel Slamanig\textsuperscript{2}

\textsuperscript{1} University of Tartu, Estonia
behzad.abdolmaleki@ut.ee

\textsuperscript{2} AIT Austrian Institute of Technology, Austria
{sebastian.ramacher, daniel.slamanig}@ait.ac.at

Abstract. Zero-knowledge proofs and in particular succinct non-interactive zero-knowledge proofs (so called zk-SNARKs) are getting increasingly used in real-world applications, with cryptocurrencies being the prime example. Simulation extractability (SE) is a strong security notion of zk-SNARKs which informally ensures non-malleability of proofs, which is considered highly important in practical applications. Another problematic issue for the practical use of zk-SNARKs is the requirement of a fully trusted setup, as especially for large-scale decentralized applications finding a trusted party that runs the setup is practically impossible. Quite recently, the study of approaches to relax or even remove the trust in the setup procedure, and in particular subversion as well as updatable zk-SNARKs (with latter being the most promising approach), has been initiated and received considerable attention since then. Unfortunately, so far SE-SNARKs with aforementioned properties are only constructed in an ad-hoc manner and no generic techniques are available.

In this SoK paper we present the state-of-the-art in generic techniques to obtain SE subversion and updatable SNARKs. In particular, we present a revisited version of the lifting technique due to Kosba et al. (called C$\emptyset$C$\emptyset$). This revisited version called OC$\emptyset$C$\emptyset$ explores the design space of many recently proposed SNARK- and STARK-friendly symmetric-key primitives. While C$\emptyset$C$\emptyset$ and OC$\emptyset$C$\emptyset$ are compatible with subversion SNARKs, they are not compatible with updatable SNARKs. Then, we present another lifting transformation called LAMASSU, which is build upon key-homomorphic signatures as well as so called updatable signatures. LAMASSU preserves the subversion and in particular updatable properties of the underlying zk-SNARK. Finally, we present an comprehensive comparison of these lifting transformations with ad-hoc techniques as well as a discussion of many aspects regarding the instantiation of the techniques.

1 Introduction

Zero-knowledge (ZK) proofs were introduced by Goldwasser, Micali and Rackoff [GMR85] more than 3 decades ago. They represent a cryptographic protocol between two parties called the prover and the verifier, with the goal that

* This paper is based on and contains a significant part of the results in [ARS20].
the prover convinces the verifier of the membership of a word \( x \) in any language in NP without revealing any information about the witness \( w \) certifying language membership of word \( x \). Besides this zero-knowledge property, such a system needs to provide soundness, i.e., it must be infeasible for the prover to provide proofs for words outside of the language. While ZK proofs in general may require many rounds of interaction, a variant highly relevant to practical applications are non-interactive zero-knowledge (NIZK) proofs [BFM88]. They require only a single round, i.e., the prover outputs a proof which can then be verified by anybody. (NI)ZK plays a central role in the theory of cryptography and meanwhile increasingly finds its way into practice.\(^3\) \(^4\) \(^5\) Important applications are electronic voting [SK95, DGS03, Gro10b], anonymous credentials [Cha86, CL01, CL03, CL04, BCC+09, CKL+16, FHS19], and group signatures [Cv91, ACJT00, BBS04, DP06, BCC+16, DS18], including widely deployed schemes such as direct anonymous attestation (DAA) [BCC04, CCD+17] used in the Trusted Platform Module (TPM) or Intel’s Enhanced Privacy ID (EPID) [BL09], as well as many other applications that require balancing privacy and integrity (cf. [FPS+18]). They are also a core building block of verifiable computation [GGP10, GGPR13, PHGR13, BCG+18] and in the increasingly popular domain of privacy-respecting cryptocurrencies [BCG+14, CGL+17], smart contracts [KMS+16] and self-sovereign identity systems [MGG18]. Latter arguably represent the most popular real-world applications of zero-knowledge nowadays, where it sees deployments in systems such as Zcash, Ethereum or sovrin.

A challenging issue, particularly important in context of blockchains, is that users need to download and verify the state of the chain. Thus, small proof sizes and fast verification are important criteria for the practical use of ZK proofs. These desired features are provided by zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs)\(^6\), which are NIZK proofs in which proofs as well as the computation of the verifier are succinct and ideally represent a small constant amount of space and computation respectively. Additionally, they satisfy a stronger notion of soundness called knowledge soundness, which guarantees that if an adversarial prover comes up with a proof that is accepted by the verifier, then there exists an efficient extractor which given some secret information can extract the witness. A combined effort of a large number of recent research works [Gro10a, Lip12, GGPR13, PHGR13, Lip13, DFGK14, Gro16] (to only mention a few) has made it possible to construct very efficient zk-SNARKs for both the Boolean and the Arithmetic CIRCUIT-SAT and thus for NP. The

\(^3\) ZKProof (https://zkproof.org/) being the most notable industry and academic initiative towards a common framework and standards in the field of zero-knowledge has been founded in 2018.


\(^5\) MIT technology review named zk-SNARKS as one of the “10 Breakthrough Technologies of 2018” cf. https://www.technologyreview.com/lists/technologies/2018/.

\(^6\) We note that we might drop the zk and simply write SNARK occasionally, though we are always talking about zk-SNARKs.
most efficient known approach for constructing zk-SNARKs for the Arithmetic Circuit-SAT is based on Quadratic Arithmetic Programs (QAPs) [GGPR13], where the prover builds a set of polynomial equations that are then checked by the verifier by using a small number of pairings. The current interest in zk-SNARKs is significant and recently first modular frameworks to efficiently compose zk-SNARKs [CFQ19] and also first important steps towards realizing zk-SNARKs from conjectured post-quantum secure assumptions have been made [GMNO18, BBC+.18]. We note that in this work we do not consider recent NIZK proofs that allow larger proof sizes, e.g., logarithmic in the witness size, such as Bulletproofs [BBB+.18] or STARKs [BBHR19] but do not require a trusted setup. The currently most efficient zk-SNARK for Arithmetic Circuit-SAT was proposed by Groth [Gro16], who proved it to be knowledge-sound in the generic bilinear group model. In Groth’s zk-SNARK, a proof consists of only 3 bilinear group elements and the verifier has to check a single pairing equation.

**Strong security for zk-SNARKs.** For practical applications of NIZKs even stronger security notions than soundness and knowledge soundness, called simulation soundness (SS) and simulation knowledge soundness (or simply simulation extractability or SE) [Sah99, Sah01], are required. Informally, these notions require soundness and knowledge soundness respectively to hold even if an adversary is allowed to see an arbitrary number of simulated proofs (which she can obtain adaptively on words of her choice). Firstly, these properties are important if NIZKs are used within larger cryptographic protocols and in particular if they are modeled and analyzed in the universal composability (UC) framework [Can01], as frequently used in blockchain-related protocols (e.g., [JKS16, CDD17, KKKZ19, FMMO19] to name a few). Secondly, NIZKs not satisfying this strong security may face severe threats when used in applications. Therefore, let us informally recall what this property does. It guarantees that proofs are non-malleable in a way that one can neither obtain another valid proof for the same word nor a new proof for a potentially related word from a given proof. Now, let us assume the typical blockchain setting where proofs are incorporated into the state of the blockchain via transactions (e.g., as in Zcash). This means that anyone could take a proof \( \pi \) and obtain from it another new proof \( \pi' \) for the same word and could advertise it as its own proof (as \( \pi' \neq \pi \)). This is what is often called man-in-the-middle attacks in context of NIZKs and SNARKs (cf. [GM17]). Even worse, it might be possible to obtain from a proof \( \pi \) another proof \( \pi' \) for another word \( x' \neq x \) (in the same language). For example, if \( \pi \) proves that 100$ are transferred from \( A \) to \( B \) with transaction \( ID = id \), \( \pi' \) might transfer 1000$ from \( A \) to \( B \) with new \( ID = id' \), which can be a devastating attack in systems deployed in the real-world. In fact, ECDSA signature malleability already enabled an attack on Bitcoin that is suspected to have caused a loss of over $ 30 million.\(^7\) Therefore, to avoid such attacks in zk-SNARKs based cryp-

\(^7\) https://www.coindesk.com/study-finds-mt-gox-lost-386-bitcoins-due-to-transaction-malleability
tocurrencies, non-malleability of the proofs is of utmost importance. All these
problems are mitigated by the use of simulation-extractable (SE) zk-SNARKs.

Simulation soundness and simulation extractability can be added generically
to any NIZK. Compilers for the former are usually inspired by [Sah01, Gro06]
and basically use the idea of extending the language to an OR language where
the trapdoor for the OR part can be used to simulate proofs. Extractability
can be obtained by additionally encrypting the witness under a public key
in the common reference string (CRS) and prove correct encryption [DP92].
The most prominent compiler that exactly follows the ideas outlined before is
the C∅C∅ framework [KZM+15] (e.g., used in [AB19, Bag19] and most promi-
nently in the celebrated Hawk paper [KMS+16]). In addition to generic com-
pilers, Groth and Maller in [GM17] initiated the study of ad-hoc constructions
of SE zk-SNARKs. This work continued in [BG18] by extending Groth’s zk-
SNARK [Gro16] in a non black-box way to obtain SE. There is also other recent
work in this direction proposing other ad-hoc zk-SNARKs with these properties
(cf. [KLO19, Lip19]). Beyond the C∅C∅ framework, which, given the progress in
the field of SNARKs (such as universal CRS) and SNARK-friendly primitives,
is already quite outdated, there is no work towards lifting zk-SNARKs to SE
zk-SNARKs generically.

Trust in CRS generation. Another important aspect for practical applica-
tions of zk-SNARKs is the question of the generation of the required common
reference string (CRS) [BFM88], a structured random string available to the
prover and the verifier. While the CRS model is widely accepted, one has to be
very careful to ensure that the CRS has been created honestly, meaning that
no one knows the associated trapdoor which allows to break zero-knowledge or
soundness. In theory, it is simply assumed that some trusted party will per-
form the CRS generation, but such a party is hard to find in the real-world.
After the Snowden revelations, there has been a recent surge of interest in con-
structing cryptographic primitives and protocols secure against active subversion
and the CRS generation is exactly one of those processes where subversion can
happen. In [BFS16], Bellare, Fuchsbauer and Scafuro tackled this problem for
NIZK proofs by studying how much security one can still achieve when the
CRS generator cannot be trusted. They proved several negative and positive
results. In particular, they showed that it is impossible to achieve subversion
soundness and (even non-subversion) zero knowledge simultaneously. However,
subversion zero-knowledge can be achieved. Later, this notion has also be con-
sidered for SNARKs [ABLZ17, Fuc18] and used within practical applications
in cryptocurrencies [CGGN17, Fuc19]. For deployed systems such as Zcash
and Ethereum, instead of building them on top of subversion-resistant zk-SNARKs,
they followed an alternative route to reduce the trust put in the CRS generation.
Following a generic method to implement the CRS generation within a secure
multi-party computation (MPC) protocol [BCG+15], the CRS for Pinocchio zk-
SNARKs [PHGR13] was generated in a large “ceremony” [BGG19]. Coinciden-
tially, they end up with a subversion-resistant zk-SNARK with a polynomial
error even in the case where all parties are corrupted, and subversion soundness
as long as at least one party is honest. While this is an important achievement, MPC protocols for such tasks in practice are complicated and expensive procedures, requiring much effort besides the technical realization. Thus, more practical solutions are desirable.

Quite recently, to overcome this problem Groth et al. [GKM+18] proposed the notion of a so-called updatable CRS, where everyone can update a CRS and there is a way to check correctness of an update. Here, zero-knowledge holds in the face of a malicious CRS generator and the verifier can trust the CRS (soundness holds) as long as one operation, either the creation of the CRS or one update, have been performed honestly. So the verifier could perform this update operation on its own and then send the CRS to the prover. This updatable setting thus seems much more practical than using MPC protocols, it is more promising than the subversion setting (as it overcomes the impossibility of subversion soundness), and thus has recently attracted lots of researchers studying approaches to realize updatable zk-SNARKs (cf. [MBKM19, GR19, KLO19, CHM+19]).

1.1 Outline

(Revisiting) \(\text{COCO}\). We first present the \(\text{COCO}\) lifting technique [KZM+15] to generically obtain SE-SNARKs from SNARKs. We discuss the concrete instantiation in [KZM+15] and point to efficiency problems and problems regarding provable security of this instantiation. Then, we extensively investigate alternative provably secure instantiations of their techniques by exploring the design space of many recently proposed SNARK- and STARK-friendly symmetric primitives including the most recent proposals Poseidon [GKK+19] as well as Vision and Rescue [AABS+19]. As these primitives are, however, all very recent and their cryptanalysis either still needs to start or has only recently started [ACG+19, LP19, Bon19, BBUV19], confidence in their proposed security is far from certain. Nevertheless, we provide concrete recommendations for the selection of primitives and provide lower bounds for their efficiency based on the currently available parameters. Additionally, we also propose a more conservative instantiation based on LowMC [ARS+15], which is the oldest of these proposals and has already received independent cryptanalysis [DEM16, BDD+15, DLMW15, RST18]. In comparison to the original \(\text{COCO}\) framework, the revisited \(\text{COCO}\) framework (dubbed \(\text{OCOCO}\)) yields an improvement by a factor 10.4x in the number of rank-1 constraints with a conservative choice of symmetric primitives, whereas the most aggressive choice yields an improvement by up to a factor 55.4x.

**Lamassu Framework.** We then discuss the Lamassu [ARS20] framework that is based on completely different cryptographic primitives. In particular, it is based on the ideas of Derler and Slamanig [DS19] using the notion of key-homomorphic signatures and thus only requires signature schemes. It allows instantiations based on well studied and widely used signature schemes such as ECDSA or EC-Schnorr. Also for Lamassu we discuss concrete choices for the primitives
and an extensive comparison with ad-hoc constructions. We show that Lamassu yields efficient instantiations that compared to OC∅C∅ only need 200 rank-1 constraints more than using the most aggressive parameter choice for the most efficient SNARK-friendly primitive Poseidon in OC∅C∅. For all other and more conservative choices of SNARK-friendly symmetric-key primitives and their parameters, Lamassu beats them in the number of constraints and outperforms OC∅C∅ by a factor of up to 4.2x. Considering that EC-Schnorr and ECDSA signatures are well established primitives, and that the confidence in their security is far bigger than all the recent SNARK/STARK-friendly primitives, this additional confidence comes at only a very small cost and makes Lamassu an attractive alternative to C∅C∅.

Subversion and updatable CRS. C∅C∅ as well as OC∅C∅ do not support lifting of subversion or updatable CRS zk-SNARKs to SE subversion or updatable SNARKs. While for the case of subversion zero-knowledge, Baghery in [Bag19] shows that using a part of the C∅C∅ framework (without the encryption of the witness) it is possible to preserve the subversion zero-knowledge property, the case of zk-SNARKS with updatable CRS is more problematic. In particular, the C∅C∅ and OC∅C∅ frameworks cannot be easily made updatable due to the missing algebraic structure in the used primitives, i.e., (hash) commitments. Lamassu does not suffer from this problem and when using Lamassu with updatable signatures instead of key-homomorphic signatures, the property of updatable signatures is preserved if the underlying zk-SNARK possesses this property, i.e., is updatable. Updatable signatures can be constructed from key-homomorphic signatures with some additional natural properties and can be constructed from widely used signatures such as EC-Schnorr signatures when instantiated in bilinear groups. Besides, Lamassu also preserves subversion of the underlying SNARK. Consequently, when starting from a subversion/updatable zk-SNARK, Lamassu yields SE subversion/updatable SNARKs. Consequently, Lamassu is the only known framework that allows to generically lift updatable zk-SNARKs to SE updatable SNARKs using well established cryptographic primitives.

2 Preliminaries

Let PPT denote probabilistic polynomial-time. Let $\lambda \in \mathbb{N}$ be the security parameter. All adversaries will be stateful. By $y \leftarrow \mathcal{A}(x; \omega)$ we denote the fact that $\mathcal{A}$, given an input $x$ and random coins $\omega$, outputs $y$. By $x \leftarrow \mathcal{D}$ we denote that $x$ is sampled according to distribution $\mathcal{D}$ or uniformly randomly if $\mathcal{D}$ is a set. Let $\text{RND}(\mathcal{A})$ denote the random tape of $\mathcal{A}$, and let $\omega \leftarrow \text{RND}(\mathcal{A})$ denote the random choice of the random coins $\omega$ from $\text{RND}(\mathcal{A})$. We denote by $\text{negl}(\lambda)$ an arbitrary negligible function. We write $a \approx_{\lambda} b$ if $|a - b| \leq \text{negl}(\lambda)$. A bilinear

---

8 Even using the C∅C∅ framework with commitments that have enough algebraic structure, i.e., use of exponential ElGamal or Pedersen commitments, does not seem to yield updatability. And even if it would work, it would be less efficient than Lamassu.
group generator $P_{\text{gen}}(1^\lambda)$ returns $BG = (p, G_1, G_2, G_T, e)$, where $G_1$, $G_2$, and $G_T$ are three additive cyclic groups of prime order $p$, and $e : G_1 \times G_2 \to G_T$ is a non-degenerate efficiently computable bilinear map (pairing).

2.1 Pseudorandom Functions

We recall the standard notion of pseudorandom functions.

**Definition 1 (PRF).** Let $f : S \times D \to R$ be a family of functions and let $\Gamma$ be the set of all functions $D \to R$. $f$ is a pseudorandom function (PRF) (family) if it is efficiently computable and for all PPT distinguishers $D$ such that
$$\left| \Pr\left[ s \leftarrow S, D^{f_s}(1^\kappa) \right] - \Pr\left[ g \leftarrow \Gamma, D^{g(\cdot)}(1^\kappa) \right] \right| \approx \lambda 0.$$

2.2 X-SNARK

In the following we provide a formal definition of SNARKs (cf. Appendix A.1 for the basic definition of NIZK proofs).

**Definition 2 (SNARK).** A non-interactive system $\Pi$ is a succinct non-interactive argument of knowledge (SNARK) for relation generator $R_{\text{Gen}}$ if it is complete and knowledge sound, and moreover succinct, meaning that for all $\lambda$, all $(R, aux_R) \in \text{image}(R_{\text{Gen}}(1^\lambda))$, all $\text{crs} \leftarrow K_{\text{Gen}}(R, aux_R)$, all $(x, w) \in R$ and all proofs $\pi \leftarrow P(R, aux_R, \text{crs}, x, w)$ we have $|\pi| = \text{poly}(\lambda)$ and $V(R, aux_R, \text{crs}, x, \pi)$ runs in time polynomial in $\lambda + |x|$. $\Pi$ is a zk-SNARK if it additionally satisfies zero-knowledge and an SE (zk-)SNARK if instead of knowledge soundness it provides strong simulation extractability.

We adopt the (SE) X-SNARK definitions from [ABLZ17, Fuc18, GKM+18] where $X \in \{\text{trusted}, \text{subverted}, \text{updatable}\}$. In other words, besides considering the standard setting with a trusted CRS generation, we also capture the subversion and updatable CRS setting. Trusted means generated by a trusted third party, or even a more general MPC protocol, subverted means that the setup generator gets the CRS from the adversary and uses it after checking that it is well formed, and, updatable means that an adversary can adaptively generate sequences of CRSs and arbitrarily interleave its own malicious updates into them. The only constraints on the final CRS are that it is well formed and that at least one honest participant has contributed to it by providing an update.

A X-SNARK $\Pi = (K_{\text{Gen}}, U_{\text{crs}}, V_{\text{crs}}, P, V)$ for $R_{\text{Gen}}$ consists of the following PPT algorithms (it contains $V_{\text{crs}}$ when $X = \text{subverted}$ and contains $U_{\text{crs}}$ and $V_{\text{crs}}$ when $X = \text{update}$):

$K_{\text{Gen}}_{\text{crs}}(R, aux_R)$: On input $(R, aux_R) \in \text{image}(R_{\text{Gen}}(1^\lambda))$, outputs CRS $\text{crs}$ and trapdoor $\text{tc}$.

$U_{\text{crs}}(R, \text{crs})$: On input $(R, \text{crs})$ outputs $(\text{crs}_{\text{up}}, \zeta_{\text{up}})$ where $\text{crs}_{\text{up}}$ is the updated CRS and $\zeta_{\text{up}}$ is a proof for the correctness of the updating procedure.
Vcrs(R, auxR, crs, ζ): On input (R, auxR, crs, ζ), returns either 0 (the CRS is ill-formed) or 1 (the CRS is well-formed).

P(R, auxR, crs, x, w): On input (R, auxR, crs, x, w), where (x, w) ∈ R, output a proof π.

V(R, auxR, crs, x, π): On input (R, auxR, crs, x, π), returns either 0 (reject) or 1 (accept).

Sim(R, auxR, crs, x, tc): In input (R, auxR, crs, x, tc), outputs a simulated proof π.

Definition 3. Let Π = (KGen crs, Ucrs, Vcrs, P, V) be a non-interactive argument for the relation R. Then the argument Π is X-secure for X ∈ {trusted, subverted, updatable}, if it satisfies the following properties:

X-Completeness. Π is complete for RGen, if for all λ, (x, w) ∈ R, and PPT algorithms A,

\[
\Pr \left[ \begin{array}{c}
(R, auxR) \leftarrow RGen(1^\lambda), \\
(crs, tc, \zeta) \leftarrow A(R, auxR), \\
V(R, auxR, crs, x, P(R, auxR, crs, x, w)) = 1
\end{array} \right] =_{\lambda} 1.
\]

Where ζ is a proof for the correctness of the generating (or updating) the CRS. If X = trusted then A is KGen crs and ζ = ⊥ and A is adversary A otherwise.

X-Strong simulation extractability. For X ∈ {trusted, subverted}, Π is strong simulation extractable for RGen, if for every PPT A, there exists a PPT extractor Ext A, s.t. \forall λ,

\[
\Pr \left[ \begin{array}{c}
(R, auxR) \leftarrow RGen(1^\lambda), \\
(crs, tc) \leftarrow KGen crs(R, auxR), \omega_A \leftarrow RND(A), \\
(x, \pi) \leftarrow A^{O(\cdot)}(R, auxR, crs; \omega_A), \\
w \leftarrow Ext A(R, auxR, crs; \omega_A): \\
(x, \pi) \notin Q \land (x, w) \notin R \land \\
V(R, auxR, crs, x, \pi) = 1
\end{array} \right] \approx_{\lambda} 0.
\]

Here, O(x) returns π := Sim(R, auxR, crs, x, tc) and keeps track of all queries and the result, (x, π), via Q. For X = updatable, Π is strong simulation extractable for RGen, if for every PPT A and any subverter Z, there exists a
PPT extractor $\text{Ext}_A$, s.t. $\forall \lambda$, 

$$
\begin{align*}
& \left[ (R, \text{aux}_R) \leftarrow \text{RGen}(1^\lambda), \\
& (\text{crs}, tc) \leftarrow \text{KGen}_{\text{crs}}(R, \text{aux}_R) \\
& \omega_Z \leftarrow \text{RND}(Z), \\
& (\text{crs}_\text{up}, \zeta_{\text{up}}, \text{aux}_Z) \leftarrow Z(\text{crs}, \{\zeta_i\}_{i=1}^n, \omega_Z), \\
& \text{if } V_{\text{crs}}(\text{crs}, \{\zeta_i\}_{i=1}^n) = 0 \text{ then return } 0, \\
& \omega_A \leftarrow \text{RND}(A), \\
& (x, \pi) \leftarrow A^O(\cdot)(R, \text{aux}_R, \text{crs}_\text{up}, \text{aux}_Z; \omega_A), \\
& w \leftarrow \text{Ext}_A(R, \text{aux}_R, \text{crs}_\text{up}, \text{aux}_Z; \omega_A): \\
& (x, \pi) \notin Q \land (x, w) \notin R \land \\
& V(R, \text{aux}_R, \text{crs}_\text{up}, x, \pi) = 1
\right] \approx_\lambda 0.
\end{align*}
$$

Here $\text{RND}(Z) = \text{RND}(A)$ and $\zeta$ is a proof for the correctness of the updating procedure. $O(x)$ returns $\pi := \text{Sim}(R, \text{aux}_R, \text{crs}, x, tc)$ and keeps track of all queried $(x, \pi)$ via $Q$. Note that $Z$ can also first generate $\text{crs}$ and then an honest updater updates it and outputs $\text{crs}_\text{up}$.

**X-Zero-knowledge.** For $X = \text{trusted}$, $\Pi$ is statistically unbounded ZK for $\text{RGen}$ [Gro06], if for all $\lambda$, all $(R, \text{aux}_R) \in \text{im}(\text{RGen}(1^\lambda))$, and all computationally unbounded $A$, $\varepsilon_0^{\text{unb}} \approx_\lambda \varepsilon_1^{\text{unb}}$, where

$$
\varepsilon_b^{\text{unb}} = \Pr[(\text{crs}, tc) \leftarrow \text{KGen}_{\text{crs}}(R, \text{aux}_R): A^O(\cdot)(R, \text{aux}_R, \text{crs}) = 1].
$$

Here, the oracle $O_0(x, w)$ returns $\bot$ (reject) if $(x, w) \notin R$, and otherwise it returns $P(R, \text{aux}_R, \text{crs}, x, w)$. Similarly, $O_1(x, w)$ returns $\bot$ (reject) if $(x, w) \notin R$, and otherwise it returns $\text{Sim}(R, \text{aux}_R, \text{crs}, x, tc)$. $\Pi$ is perfectly unbounded ZK for $\text{RGen}$ if $\exists$ oracle $O$ such that $\varepsilon_0^{\text{unb}} = \varepsilon_1^{\text{unb}}$.

For $X \in \{\text{subverted, updatable}\}$, $\Pi$ is statistically unbounded X-ZK for $\text{RGen}$ [ABLZ17, Fuc18, GKM+18], if for any PPT $Z$ there exists a PPT $\text{Ext}_Z$, such that for all $\lambda$, $(R, \text{aux}_R) \in \text{im}(\text{RGen}(1^\lambda))$, and computationally unbounded $A$, $\varepsilon_0^{\text{unb}} \approx_\lambda \varepsilon_1^{\text{unb}}$, where

$$
\varepsilon_b^{\text{unb}} = \Pr[\omega_Z \leftarrow \text{RND}(Z), (\text{crs}, \zeta, \text{aux}_Z) \leftarrow Z(R, \text{aux}_R; \omega_Z), \\
V_{\text{crs}}(R, \text{aux}_R, \text{crs}, \zeta) = 1 \land A^O(\cdot)(R, \text{aux}_R, \text{crs}, tc, \text{aux}_Z) = 1].
$$

Here $\text{RND}(Z) = \text{RND}(A)$, the oracle $O_0(x, w)$ returns $\bot$ (reject) if $(x, w) \notin R$, and otherwise it returns $P(R, \text{aux}_R, \text{crs}, x, w)$. Similarly, $O_1(x, w)$ returns $\bot$ (reject) if $(x, w) \notin R$, and otherwise it returns $\text{Sim}(R, \text{aux}_R, \text{crs}, x, tc)$. $\Pi$ is perfectly unbounded X-ZK for $\text{RGen}$ if $\exists$ oracle $O$ such that $\varepsilon_0^{\text{unb}} = \varepsilon_1^{\text{unb}}$.

In all of the above properties, $\text{aux}_R$ can be seen as a common auxiliary input to algorithm $A$ that is generated by using a benign [BCPR14] relation generator; we recall that we just think of $\text{aux}_R$ as being the description of a bilinear group.
We note that what is called strong simulation-sound extractability in this work (in order to be consistent with [KZM+15]) is often simply called simulation-sound extractability (e.g., in [DS19] which will be the basis for the LAMASSU framework). For completeness, quadratic arithmetic programs and rank 1 constraint systems are discussed in Appendix A.2

2.3 Signature Schemes

A signature scheme \( \Sigma = (KGen, \text{Sign}, \text{Verify}) \) consists of the following PPT algorithms:

\( KGen(1^\kappa) : \) On input security parameter \( \kappa \) it outputs a signing key \( sk \) and a verification key \( pk \) with associated message space \( M \).

\( \text{Sign}(sk, m) : \) On input a secret key \( sk \) and a message \( m \in M \) it outputs a signature \( \sigma \).

\( \text{Verify}(pk, m, \sigma) : \) On input a public key \( pk \), a message \( m \in M \) and a signature \( \sigma \) it outputs a bit \( b \in \{0, 1\} \).

We note that for a signature scheme many independently generated public keys may be with respect to the same parameters \( \mathbb{P} \), e.g., some elliptic curve group parameters. In such a case we use an additional algorithm \( \mathbb{P}Gen \) and \( \mathbb{P} \leftarrow \mathbb{P}Gen(1^\kappa) \) is then given to \( KGen \). We assume that a signature scheme satisfies the usual (perfect) correctness notion. Below, we present the standard existential unforgeability under adaptively chosen message attacks (EUF-CMA security) notion.

**Definition 4 (EUF-CMA).** A signature scheme \( \Sigma \) is EUF-CMA secure, if for all PPT adversaries \( \mathcal{A} \)

\[
\Pr \left[ (sk, pk) \leftarrow KGen(1^\kappa), (m^*, \sigma^*) \leftarrow A^{\text{Sign}}(sk, \cdot)(pk) : \right. \left. \text{Verify}(pk, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin Q^{\text{Sign}} \right] \approx_{\lambda} 0,
\]

where the environment keeps track of the queries to the signing oracle via \( Q^{\text{Sign}} \).

The compiler from [ARS20] also requires one-time signature schemes that are sEUF-CMA secure (also called sOTS schemes).

**Definition 5 (Strong One-Time Signature Scheme).** A strong one-time signature scheme \( \Sigma_{\text{OT}} \) is a signature scheme \( \Sigma \) which satisfies the following unforgeability notion: For all PPT adversaries \( \mathcal{A} \)

\[
\Pr \left[ (sk, pk) \leftarrow KGen(1^\kappa), (m^*, \sigma^*) \leftarrow A^{\text{Sign}}(sk, \cdot)(pk) : \right. \left. \text{Verify}(pk, m^*, \sigma^*) = 1 \wedge (m^*, \sigma^*) \notin Q^{\text{Sign}} \right] \approx_{\lambda} 0,
\]

where the oracle \( \text{Sign}(sk, m) := \Sigma.\text{Sign}(sk, m) \) can only be called once.
2.4 Key-Homomorphic Signatures

We recall relevant parts of the definitional framework of key-homomorphic signatures as introduced in [DS19]. Let $\Sigma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme and the secret and public key elements live in groups $(\mathbb{H}, +)$ and $(\mathbb{E}, \cdot)$, respectively. For these two groups is is required that group operations, inversions, membership testing as well as sampling from the uniform distribution are efficient.

**Definition 6 (Secret Key to Public Key Homomorphism).** A signature scheme $\Sigma$ provides a secret key to public key homomorphism, if there exists an efficiently computable map $\mu : \mathbb{H} \to \mathbb{E}$ such that for all $sk, sk' \in \mathbb{H}$ it holds that $\mu(sk + sk') = \mu(sk) \cdot \mu(sk')$, and for all $(sk, pk) \leftarrow \mathsf{KGen}$, it holds that $pk = \mu(sk)$.

In the discrete logarithm setting, it is usually the case $sk \leftarrow \mathbb{Z}_p$ and $pk = g^{sk}$ with $g$ being the generator of some group $G$ of prime order $p$, e.g., for EdDSA/ECDSA or EC-Schnorr signatures (cf. [DS19] for a detailed exposition).

**Definition 7 (Key-Homomorphic Signatures).** A signature scheme is called key-homomorphic, if it provides a secret key to public key homomorphism and an additional PPT algorithm $\mathsf{Adapt}$, defined as:

$$\mathsf{Adapt}(pk, m, \sigma, \Delta): \quad \text{Given a public key } pk, \text{ a message } m, \text{ a signature } \sigma, \text{ and a shift amount } \Delta \text{ outputs a public key } pk' \text{ and a signature } \sigma',$$

such that for all $\Delta \in \mathbb{H}$ and all $(pk, sk) \leftarrow \mathsf{KGen}(1^\kappa)$, all messages $m \in M$ and all $\sigma \leftarrow \mathsf{Sign}(sk, m)$ and $(pk', \sigma') \leftarrow \mathsf{Adapt}(pk, m, \sigma, \Delta)$ it holds that

$$\Pr[\mathsf{Verify}(pk', m, \sigma') = 1] = 1 \land pk' = \mu(\Delta) \cdot pk.$$

The following notion covers whether adapted signatures look like freshly generated signatures, where we do not need the strongest notion in [DS19], which requires this to hold even if the initial signature used in $\mathsf{Adapt}$ is known.

**Definition 8 (Adaptability of Signatures).** A key-homomorphic signature scheme provides adaptability of signatures, if for every $\kappa \in \mathbb{N}$ and every message $m \in M$, it holds that

$$[(sk, pk), \mathsf{Adapt}(pk, m, \mathsf{Sign}(sk, m), \Delta)],$$

where $(sk, pk) \leftarrow \mathsf{KGen}(1^\kappa), \Delta \leftarrow \mathbb{H}$, and

$$[(sk, \mu(sk)), (\mu(sk) \cdot \mu(\Delta), \mathsf{Sign}(sk + \Delta, m))],$$

where $sk \leftarrow \mathbb{H}, \Delta \leftarrow \mathbb{H}$, are identically distributed.

For illustration purposes we will use the Schnorr signature scheme [Sch90], which is very popular in the blockchain and distributed ledger domain, and whose adaption notion we discuss in Appendix A.4.
2.5 Updatable Signature Schemes

Updatable signatures are signatures that allow updates on the key and unforgeability guarantees need to hold as long as either the initial key generation or at least one of the updates was performed honestly. However, signing is performed honestly. We note that like in Groth et al. [GKM+18] for updatable CRS (using Lemma 6), one models only a single update as a single adversarial update implies updatable signatures with arbitrary many updates.

**Definition 9 (Updatable signature schemes).** An updatable signature scheme \( \Sigma = (\text{KGen}, \text{Ucrs}, \text{Vpk}, \text{Sign}, \text{Verify}) \) consists of the following PPT algorithms:

- \( \text{KGen}(\lambda) \): Given a security parameter \( \lambda \) it outputs a signing key \( \text{sk} \) and a verification key \( \text{pk} \) with associated message space \( \mathcal{M} \).
- \( \text{Upk}(\text{pk}) \): Given a verification key \( \text{pk} \) it outputs an updated verification key \( \text{pk}_{\text{up}} \).
- \( \text{Vpk}(\text{pk}, \text{pk}_{\text{up}}, \zeta) \): Given a verification key \( \text{pk} \), a potentially updated verification key \( \text{pk}_{\text{up}} \), and the proof \( \zeta \) it checks if \( \text{pk}_{\text{up}} \) has been updated correctly.
- \( \text{Sign}(\text{sk}, m) \): Given potentially updated secret key \( \text{sk} \) (in case of \( \text{sk}_{\text{up}} \) it contains \( \text{sk} \) and \( \text{up}_{\text{sk}} \)) and a message \( m \in \mathcal{M} \) it outputs a signature \( \sigma \).
- \( \text{Verify}(\text{pk}, m, \sigma) \): Given potentially updated public key \( \text{pk} \), a message \( m \in \mathcal{M} \) and a signature \( \sigma \) it outputs a bit \( b \in \{0,1\} \).

**Definition 10 (Updatable correctness).** A signature scheme \( \Sigma \) is updatable correct, if

\[
\Pr \left[ \begin{array}{l}
(sk, pk, \zeta) \leftarrow \text{KGen}(\lambda), (up_{sk}, pk_{up}, \zeta_{up}) \leftarrow \text{Upk}(pk), \\
Vpk(pk, pk_{up}, \zeta_{up}) = 1 \land Vpk(pk, pk, \zeta) = 1: \\
\text{Verify}(pk, m, \text{Sign}(sk, m)) = 1 \land \\
\text{Verify}(pk_{up}, m, \text{Sign}(sk_{up}, m)) = 1
\end{array} \right] = 1,
\]

where the probability is taken over the randomness of the signing algorithm.

**Definition 11 (Updatable strong key hiding).** We have that for \( (sk, pk) \leftarrow \text{KGen}(\lambda) \) and \( (up_{sk}, pk_{up}, \zeta_{up}) \leftarrow \text{Upk}(pk) \) it holds that \( (sk, pk) \approx (sk_{up}, pk_{up}) \) if one of the following setting holds,

- the original \( pk \) was honestly generated and the key-update verifies: \( (sk, pk) \leftarrow \text{KGen}(\lambda) \) and \( Vpk(pk, pk, \zeta) = 1 \).
- the original \( pk \) verifies and the key-update was honest: \( Vpk(pk, pk, \zeta) = 1 \), and \( (up_{sk}, pk_{up}, \zeta_{up}) \leftarrow \text{Upk}(pk) \).

Now, we present the updatable EUF-CMA security notion.

**Definition 12 (Updatable EUF-CMA).** A signature scheme \( \Sigma \) is updatable EUF-CMA secure, if for all PPT subverter \( Z \), there exists a PPT extractor \( \text{Ext}_Z \), s.t. for all \( \lambda \), and all PPT adversaries \( A \)

\[
\Pr \left[ \begin{array}{l}
(sk, pk, \zeta) \leftarrow \text{KGen}(1^\lambda), \omega_Z \leftarrow \text{RND}(Z), (pk_{up}, \zeta_{up}, aux_Z) \leftarrow A^{\text{Sign}(sk_{up}, \cdot)}(pk_{up}, aux_Z): \\
up_{sk} \leftarrow \text{Ext}_Z(pk, \omega_Z), (m^*, \sigma^*) \leftarrow A^{\text{Sign}(sk_{up}, \cdot)}(pk_{up}, aux_Z): \\
Vpk(pk, pk_{up}, \zeta_{up}) = 1 \land \text{Verify}(pk_{up}, m^*, \sigma^*) = 1 \land m^* \notin Q^{\text{Sign}}
\end{array} \right] \approx 0,
\]

12
where the environment keeps track of the queries to the oracle via $Q^\text{Sign}$. Note that $Z$ can also generate the initial $pk$ and the an honest updater $Upk$ updates it and outputs $pk_{up}$, $sk_{up}$, and the proof $\zeta$ (then we require that $V_{pk}(pk, pk, \zeta) = 1$).

In [ARS20] the following is shown.

**Theorem 1.** Every correct and EUF-CMA secure key-homomorphic signature scheme $\Sigma$ that is adaptable according to Definition 8 and provides an efficient extractor $\text{Ext}_Z$ satisfies updatable correctness, updatable strong key hiding and updatable EUF-CMA security.

### 3 Lifting Transformations for SE Subversion/Updatable SNARKs

In this section we briefly present the $C^\emptyset C^\emptyset$ framework, then a revisited version of the $C^\emptyset C^\emptyset$ framework (called $OC^\emptyset C^\emptyset$) and finally the LAMASSU framework.

#### 3.1 The $C^\emptyset C^\emptyset$ Framework

Kosba et al. [KZM+15] proposed lifting transformations for SNARKs in three different versions basic, improved lifting, and the strengthening lifting. We only consider the strongest version which lifts a SNARK to a strongly simulation extractable (SE) SNARK. In particular, their construction, which we recall in Fig. 1, transforms any NIZK $\Pi$ to one that satisfies SE. Given a language $L$ with NP relation $R_L$, let $L'$ be s.t.

$$\{(x, c, \mu, pk_{OT}, pk_e, \rho), (w, r_1, r_0, s_0)\} \in R_L \rightarrow (1)$$

where $pk_e$ is the public key of a public key encryption scheme $\Omega$ (cf. Appendix A.3), $f$ is a pseudorandom function, and $pk_{OT}$ is the verification key of a strong one-time signature (OTS) scheme $\Sigma_{OT}$ (cf. Definition 5). Note that extraction is defined here with respect to a black-box extractor (i.e., decrypting to obtain the witness), which Kosba et al. [KZM+15] do to support UC-security. If this is not required, then one can use the non black-box extractor of the underlying SNARK and simplify the language $L'$ by removing the part in the gray box, which we will do subsequently (cf. [Bag19] for a formal treatment). In this case, $C^\emptyset C^\emptyset$ retains subversion resistance of the underlying SNARK.

#### 3.2 The Revisited $C^\emptyset C^\emptyset$ Framework $OC^\emptyset C^\emptyset$

The most efficient version of the $C^\emptyset C^\emptyset$ framework is based on a commitment and PRF evaluation (Equation (1) without the gray box). Kosba et al. [KZM+15] proposed to instantiate the commitment and the PRF using hash functions, and in
the PRF is built from symmetric-key primitives. The standard notion of PRF natures consists of a simulation extractable NIZK proof of a PRF key, where Goldwasser paradigm [BG90] also needs to “commit” to a PRF key. There, the specific commitment used in C∅∅ is also problematic in a different sense, because the choice of the commitment is non-optimal from an efficiency point of view. Moreover, friendly primitives soon after the introduction of the C∅∅ framework, we observe that this choice is non-optimal from an efficiency point of view. Moreover, the choice of the commitment is also problematic in a different sense, because the specific commitment used in C∅∅ is secure in the random oracle model (ROM). Since this implies that statements need to be proven with respect to the preimage of a random oracle, instantiating the framework in a provable secure way is not possible. Consequently, we discuss an alternative approach to commit to the PRF key. It can be instantiated in a provably secure way and, on top of that, is also more efficient while still relying on symmetric-key primitives only.

The problem in the symmetric setting is to find efficient binding commitments. The signature scheme construction in [DOR+16] based on the Bellare-Goldwasser paradigm [BG90] also needs to “commit” to a PRF key. There, signatures consists of a simulation extractable NIZK proof of a PRF key, where the PRF is built from symmetric-key primitives. The standard notion of PRF

Fig. 1. The strong version of the C∅∅ transformation.

<table>
<thead>
<tr>
<th>KGen_{crs}(1^λ, L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Π.crs ← Π.KGen;(pk_c, sk_c) ← Ω.KGen(1^λ);</td>
</tr>
<tr>
<td>- τc ← (s_0, r_0) ←{0, 1}^λ; ρ ← Commit(s_0, r_0);</td>
</tr>
<tr>
<td>- return (crs := (Π.crs, pk_c, ρ), τ_{ext} := sk_c)</td>
</tr>
<tr>
<td>P(crs, x, w)</td>
</tr>
<tr>
<td>- (pk_{OT}, sk_{OT}) ← Σ_{OT}.KGen(1^λ); r_1, z_0, z_1, z_2 ← {0, 1}^λ;</td>
</tr>
<tr>
<td>- c = Ω.Enc(pk_c, w; r_1); μ ← z_0;</td>
</tr>
<tr>
<td>- π_Π ← Π.P(Π.crs, (x, c, pk_c, pk_{OT}, μ, ρ), (w, r_1, z_1, z_2));</td>
</tr>
<tr>
<td>- σ_{OT} ← Σ_{OT}.Sign(sk_{OT}, (x, c, μ, π_Π));</td>
</tr>
<tr>
<td>- return π := (c, μ, π_Π, pk_{OT}, σ_{OT});</td>
</tr>
<tr>
<td>V(crs, x, π)</td>
</tr>
<tr>
<td>- if Σ_{OT}.Verify(pk_{OT}, (x, c, μ, π_Π, σ_{OT})) = 0</td>
</tr>
<tr>
<td>- ∨ Π.V(Π.crs, x, c, μ, pk_c, pk_{OT}, ρ, π_Π) = 0</td>
</tr>
<tr>
<td>- then return 0 else return 1;</td>
</tr>
<tr>
<td>Sim(crs, x, tc)</td>
</tr>
<tr>
<td>- (pk_{OT}, sk_{OT}) ← Σ_{OT}.KGen(1^λ); μ = f_{s_0}(pk_{OT});</td>
</tr>
<tr>
<td>- r_1, z_3 ← {0, 1}^λ; c ← Ω.Enc(pk_c, z_3; r_1); w ← z_3;</td>
</tr>
<tr>
<td>- π_Π ← Π.P(Π.crs, (x, c, pk_c, pk_{OT}, μ, ρ), (w, r_1, r_0, s_0));</td>
</tr>
<tr>
<td>- σ_{OT} ← Σ_{OT}.Sign(sk_{OT}, (x, c, μ, π_Π));</td>
</tr>
<tr>
<td>- return π = (c, μ, π_Π, pk_{OT}, σ_{OT});</td>
</tr>
<tr>
<td>Ext(crs, x, π, τ_{ext})</td>
</tr>
<tr>
<td>- return w ← Ω.Dec(τ_{ext}, c);</td>
</tr>
</tbody>
</table>

particular SHA-256. Similarly, the commitment is instantiated as hash commitment using the same hash function. With the development of SNARK/STARK-friendly primitives soon after the introduction of the C∅∅ framework, we observe that this choice is non-optimal from an efficiency point of view. Moreover, the choice of the commitment is also problematic in a different sense, because the specific commitment used in C∅∅ is secure in the random oracle model (ROM). Since this implies that statements need to be proven with respect to the preimage of a random oracle, instantiating the framework in a provable secure way is not possible. Consequently, we discuss an alternative approach to commit to the PRF key. It can be instantiated in a provably secure way and, on top of that, is also more efficient while still relying on symmetric-key primitives only.
security, however, does not immediately imply any binding property on the key. Therefore, the construction relies on a computational fixed-valued-key-binding PRF [CMR98, Fis99], i.e., a PRF $f$ with the additional property that there exists a $\beta$ such that for a PRF key $s$ and given $y = f_s(\beta)$ it is hard to provide a second PRF key $s', s \neq s'$, satisfying $y = f_{s'}(\beta)$:

**Definition 13 (Computational Fixed-Value-Key-Binding PRF).** A PRF family $f : S \times D \rightarrow R$ is computationally key-binding if there exists a special value $\beta \in D$ so that it holds for all adversaries $A$ that:

$$\Pr\left[ s \leftarrow S, \ s' \leftarrow A^{f_s(\cdot)}(f_s(\beta), \beta) : f_{s'}(\beta) = f_s(\beta) \right] \approx \lambda 0.$$ 

Extending the public key with the PRF evaluation at $\beta$ and proving its well-formedness is then sufficient to “commit” to the PRF key.\(^9\)

For C∅C∅, we can apply the same idea: we replace the commitment to the PRF key with the evaluation of the PRF at $\beta$ and adapt the language accordingly. That is, for the construction depicted in Fig. 1\(^{10}\), let the language $L'$ be such that $\{(x, \mu, p_k_{\text{OT}}, \rho, \beta), (w, s_0)\} \in R_{L'}$ if and only if:

$$\{(x, w) \in R_L \lor (\mu = f_{s_0}(p_k_{\text{OT}}) \land \rho = f_{s_0}(\beta))\}.$$

We denote the C∅C∅ framework using the language $L'$ as Optimized C∅C∅, or OC∅C∅ for short. For the security proofs (Theorem 2 and Theorem 3 in [Bag19]), we note for each game change based on computational hiding of the commitment, we now use the PRF property and replace them with the evaluation of a random function (Lemma 4). For the step relying on the commitment scheme’s binding property (Lemma 2), one can now argue the uniqueness of the PRF key using the fixed-value-key-binding property of the PRF. One obtains the following:

**Corollary 1.** If the underlying NIZK scheme satisfies perfect completeness, knowledge soundness, subversion zero-knowledge, the PRF is secure and computationally fixed-value-key-binding, and the one-time signature is sEUF-CMA secure, then OC∅C∅ is a zero-knowledge proof system satisfying perfect completeness, subversion zero-knowledge, and strong simulation extractability.

**Instantiating the OC∅C∅ Framework.** When instantiating the original C∅C∅ framework or OC∅C∅, SHA-256 as well as any other variant of the SHA2 family or the SHA3 family are non-optimal choices from a efficiency point of view. Indeed, representing the SHA-256 compression function as R1CS requires 22,272 constraints [CGGN17]. The permutation used in SHA3 is even more expensive with 38,400 constraints [AGR+16]. Recent lines of work specifically designed block ciphers and hash functions that work especially well in the context of SNARKs. These include MiMC [AGR+16], GMiMC [AGP+19], Poseidon [GKK+19], Friday [AD18], Vision and Rescue [AABS+19], which all were

---

\(^9\) Similarly, [DRS18] employs the same idea to commit to a PRF key.

\(^{10}\) Now, one will use the non black-box extractor of the underlying NIZK instead of the black-box extractor $\text{Ext}$ from Fig. 1.
specifically designed with SNARK/STARK-based applications in mind. We how-
ever want to note that these designs are all relatively young and were not avail-
able at the time C∅C∅ was proposed.

Since those designs are all very recent, their cryptanalysis is still ongoing. Friday suffers from a Gröbner-basis attack [ACG+19], the key schedule of some variants of MiMC can be attacked using an interpolation attack [LP19] and they also suffer from a collision attack [Bon19], which can also be applied to some variants of GMiMC. Notably, the designs also received some interest as part of a hash collision challenge for STARK-friendly designs,11 where collisions have been found for low-security instances already. Therefore, we will only include instances in our evaluation that – to the best of our knowledge – have not been broken so far.

Even though these symmetric primitives are designed for SNARKs, they often run into practical problems. For instance, one of the popular choices for instantiating SNARKs is the pairing-friendly BLS12-38112 curve. However, its group order \( q \) does not match MiMC’s and GMiMC’s requirement coming from the choice of \( x \mapsto x^3 \) as Sbox that \( \gcd(q-1, 3) = 1 \). Additionally, MiMC operates in large prime fields, requiring one to emulate the required fields on top of \( \mathbb{F}_q \). The latter issue is solved by GMiMC working over smaller fields, but the order requirement is still an issue. Poseidon, which allows one to choose \( x \mapsto x^5 \) as Sbox meaning that \( \gcd(q-1, 5) = 1 \) needs to be satisfied, fixes both problems and can be directly implemented in \( \mathbb{F}_q \) arithmetic. Similarly, Rescue faces similar issues as the Sboxes used there are \( x \mapsto x^\alpha \) and \( x \mapsto x^{1/\alpha} \). Hence, for the specific choice of BLS12-381 this would imply \( \alpha = 5 \). Vision, on the other hand, is specified over a binary field and can thus also not be directly implemented in \( \mathbb{F}_q \) arithmetic.

Additionally, the signature scheme PICNIC [CDG+17] demonstrated that LowMC [ARS+15], initially designed for the application in secure multiparty computation and fully homomorphic encryption, performs well enough in the context of NIZKs. We consider LowMC in our evaluation as the conservative choice of SNARK-friendly primitives, since it has seen some rounds of cryptanal-

ysis [DEM16, DLMW15] and corresponding updates to the round formula [RST18], and additionally gained attention in terms of efficient implementations [DKP+19].

Evaluation. In Table 1 we evaluate a variety of SNARK-friendly primitives to-
gether with the SHA2 and SHA3 families of hash functions. Our evaluation focuses on the provable secure version using fixed-value-key-binding PRFs as discussed above with a PRF having 256 bit keys, inputs and outputs. The number of constraints are computed according to the formulas given in the respec-
tive works. We consider MiMC-(\( N, R \)), GMiMC-(\( N, t, R \)) with the expanding round function (ERF) construction, Poseidon-(\( N, t, R_f, R_p \)) with \( x \mapsto x^5 \) as SBox, Rescue-(\( N, t, R \)) with \( x \mapsto x^5 \) and \( x \mapsto x^{1/5} \), Vision-(\( N, t, R \)), and LowMC-(\( N, k, m, R \)), where \( N \) denotes the block size, \( t \) the number of branches, \( R \) the number of rounds, \( R_f \) and \( R_p \) the number of full and partial rounds, \( k \) the key size and \( m \) the number of Sboxes.

11 https://starkware.co/hash-challenge/
12 https://electriccoin.co/blog/new-snark-curve/


<table>
<thead>
<tr>
<th>Framework</th>
<th>Symmetric primitive</th>
<th>PRF / Commitment</th>
<th>Provably secure</th>
<th># of constraints PRF / Commitment</th>
</tr>
</thead>
<tbody>
<tr>
<td>C∅C∅</td>
<td>SHA256</td>
<td>HMAC PRF + hash com.</td>
<td>✗</td>
<td>111360 + 44544 244992</td>
</tr>
<tr>
<td></td>
<td>SHA256</td>
<td>HMAC PRF</td>
<td>✓</td>
<td>111360 222720</td>
</tr>
<tr>
<td></td>
<td>SHAKE256</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>38400 76800</td>
</tr>
<tr>
<td></td>
<td>MiMC-(1025, 646)</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>646 1292</td>
</tr>
<tr>
<td></td>
<td>GMiMC-(1024, 4, 332)</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>999 1998</td>
</tr>
<tr>
<td></td>
<td>Poseidon-(1536, 2, 10, 114)</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>402 804</td>
</tr>
<tr>
<td></td>
<td>Vision-(1778, 14, 10)</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>1400 2800</td>
</tr>
<tr>
<td></td>
<td>Rescue-(1750, 14, 10)</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>840 1680</td>
</tr>
<tr>
<td></td>
<td>LowMC-(256, 256, 1, 537)</td>
<td>feed-forward PRF</td>
<td>✓</td>
<td>1074 2148</td>
</tr>
<tr>
<td></td>
<td>LowMC-(1024, 256, 1, 1027)</td>
<td>Sponge PRF</td>
<td>✓</td>
<td>2144 4288</td>
</tr>
</tbody>
</table>

Table 1. Number of constraints required for C∅C∅ and OC∅C∅.
Where possible, we selected instances compatible with the field induced by BLS12-381, i.e., for Poseidon and Rescue. The table also provides various different PRF constructions. Where possible, we use a Sponge-based approach [BDPV08] akin to SHAKE256. For LowMC, we also consider a feed-forward PRF built as \( f_s(x) = E(s, x) \oplus x \) where \( E \) denotes the encryption of a block. In the case of SHA256, we consider three variants that can partly also be observed in practice – directly using the HMAC output as PRF and the one from TLS 1.2 [DR08]. Regardless of the concrete choice, even the rather expensive SHAKE256 PRF is a better choice than any of the SHA256-based ones.

We stress that the numbers in Table 1 should be treated as lower bounds. One the one hand, as the security analysis of these primitives evolves, the rather aggressive choice of round numbers may need to be increased. Considering that the STARK-friendly hash challenge was almost immediately solved for the low security instances of MiMC, GMiMC and Poseidon, we expect those numbers to grow. On the other hand, for some of the instantiations it might not be immediately clear if they actually provide the fixed-value-key-binding property. For a very conservative instantiation, one could fallback to the tree-based approach by Fischlin [Fis99], which would be even more expensive, since then every PRF evaluation would internally call the PRF multiple times.

Other Important Remarks. Furthermore, beside more efficient instantiations than within the original C∅C∅ framework, the approach based on fixed-value-key-binding PRFs also circumvents another issue in concrete instantiations. Hash commitments can only be proven secure in the ROM, which would require to prove preimages of a random oracle. Hence, the construction is impossible to properly instantiate with provable security guarantees. In any case, the choice of a commitment based on symmetric primitive comes with other drawbacks as well. Since such a commitment lacks any useful algebraic structure, it is not obvious how to obtain SE updatable SNARKs.

Regarding the choice of strongly unforgeable one-time signature schemes, Groth’s sOTS (as discussed in [ARS20]) or Boneh-Boyen signatures [BB04] (as proposed in other instantiations of C∅C∅ [AB19, Bag19]) would be natural choices especially when considering the underlying SNARKs already rely on discrete logarithm assumptions (in bilinear groups). Alternatively, any strong EUF-CMA secure signature such as EC-Schnorr would fit as well. We note however, while this choice would avoid the need for a pairing evaluation for signature verification (in the case of Boneh-Boyen) and the proof overhead would be slightly smaller, EC-Schnorr provides the necessary security guarantees only in the ROM.\(^{13}\)

\(^{13}\) In private communication, A. Kosba confirmed that their implementation used a non-malleable variant of ECDSA for benchmarking. To the best of our knowledge, this variant is only suspected to be strongly unforgeable without proof so far. Thus we consider EC-Schnorr as candidate. The performance and overhead is expected to be the same.
Putting everything together, instantiating the C∅C∅ or OC∅C∅ framework with concrete symmetric primitives is non-trivial and comes with some limitations. Subsequently, we will propose an alternative framework LAMASSU, which comes with the same cost as the most aggressive choice of symmetric-key primitive and in contrast to C∅C∅ also provides SE updatable SNARKs.

3.3 The LAMASSU Framework

Now, we present the LAMASSU framework, which builds upon the recent compiler to obtain SE-NIZK proposed in [DS19]. However, we want to stress that we cannot directly use their compiler in order to construct SE updatable SNARKs and this requires non-trivial changes. The ingredients of their construction is to use a combination of an EUF-CMA secure adaptable key-homomorphic signature scheme Σ (EC-Schnorr or ECDSA are good candidate for pairing based SNARKs) and a strongly unforgeable one-time signature (sOTS) scheme ΣOT (Groth’s sOTS under the discrete logarithm assumption is a good candidate) to add the required non-malleability guarantees to the underlying knowledge sound NIZK proof system Π together with the folklore OR-trick to add simulation soundness. The distinguishing feature of this transformation is that in the proof computation one computes a signature to certify a public key of OTS using freshly sampled signing key sk of Σ in plain and thus does not need to encrypt a signature and prove that it verifies with a verification key in the CRS (e.g., as done in [Gro06]). Consequently, in the OR part of the proof one just needs to prove that one knows the shift csk (which is the trapdoor of the CRS) to adapt signatures from pk to ones valid under verification key cpk in the CRS. As it turns out, this feature lays the foundation for being able to support updatability. Now, given any language L with NP relation RL, the language obtained via the compiler is L’ s.t. \{(x, cpk), (w, csk - sk)\} ∈ RL’ iff:

\{(x, w) ∈ RL ∨ cpk = pk · μ(csk - sk)\}.

More precisely, in every proof computation one uses Σ to “certify” the public key of a newly generated key pair of ΣOT. The secret key of ΣOT is then used to sign the parts of the proof which must be non-malleable. Adaptability of Σ makes it possible to also use newly generated keys of Σ upon each proof computation. In particular, the relation associated to L’ is designed so that the additional clause introduced via the OR-trick is the “shift amount” required to shift such signatures to signatures under a key cpk of Σ in the CRS. A proof for x ∈ L is easy to compute when given w such that (x, w) ∈ Lπ. One does not need a satisfying assignment for the second clause in the OR statement, and can thus compute all signatures under newly generated keys. To simulate proofs, however, one can set up CRS in a way that we know csk corresponding to cpk, compute the “shift amount” and use it as a satisfying witness for the other clause in the OR statement. We recall the construction in Fig. 2 and for completeness recall
the Theorem given in [DS19] below. We note that for non black-box extraction as it is the case with SNARKs, the trapdoor $tc_{\text{ext}} = \perp$ and one simply uses the non black-box extractor of the underlying SNARK.

Theorem 2 ([DS19]). Let $\Pi$ be a complete, witness indistinguishable non-interactive argument of knowledge system for the language $\mathcal{L}$, let $\Sigma$ be an EUF-CMA secure signature scheme that adapts signatures, and let $\Sigma_{\text{OT}}$ be a strongly unforgeable one-time signature scheme, then the argument system $\Pi'$ is a complete and strong simulation extractable argument system for language $\mathcal{L}'$.

Note that the theorem clearly applies to any proof system that is zero-knowledge, as this implies the weaker notion of witness-indistinguishability.

Applying [DS19] to NIZKs without knowledge soundness. We now argue that, although we do not require it in context of SNARKs, analogous to the folklore compiler used in [KZM+15], we can also start form any NIZK that is only sound instead of knowledge sound. Then, using the compiler in [DS19] we still can obtain SE-NIZK when starting from any conventional NIZK. More precisely, the by now folklore compiler [DP92] to obtain knowledge soundness for any sound NIZK is to put a public key $pk_e$ of any perfectly correct IND-CPA secure public key encryption scheme into the CRS, where the extraction trapdoor $tc_{\text{ext}}$ is the corresponding secret key, and extend the language such that it contains an encryption of the witness of the original language. We will capture this in the following corollary, where starting from a NIZK for $\mathcal{L}$ with NP relation $\mathcal{R}_\mathcal{L}$, we obtain a knowledge sound NIZK by extending the language to $\mathcal{L}'$ such that $\{(x, c), (w, \omega)\} \in \mathcal{R}_{\mathcal{L}'}$ iff:

$$\{(x, w) \in \mathcal{R}_\mathcal{L} \land c = \text{Enc}(pk_e, w; \omega)\}.$$

Corollary 2. Let NIZK for language $\mathcal{L}$ be complete, sound and zero-knowledge, the public key encryption scheme be perfectly correct and IND-CPA secure, then the NIZK for language $\mathcal{L}'$ is complete, knowledge-sound and zero-knowledge.

The proof exactly follows the argumentation in [KZM+15] and is thus omitted. We stress that if one bases the compiler of [DS19] on a NIZK that is based on standard or falsifiable assumptions that is only sound, then one requires this additional encryption of the witness $w$. However, when one relies on knowledge assumptions, as it is the case within SNARKs, then one does not need the language extension in Corollary 2 and can simply use the non black-box extractor of the underlying SNARK.

---

14 We note that what is called simulation sound extractable in [DS19] is called strong simulation extractable in this paper in order to be aligned with the notation used in the C\&C\&C framework.
KGen_{crs}(1^{\lambda})

- \( (\text{crs}_\Pi, \text{tc}_\Pi, \text{tc}_\text{ext}) \leftarrow \Pi.\text{KGen}(1^{\lambda}); \)
- \( (\text{csk}, \text{cpk}) \leftarrow \Sigma.\text{KGen}(1^{\kappa}); \)
- \( \text{crs} := (\text{crs}_\Pi, \text{cpk}), \text{tc} := (\text{tc}_\Pi, \text{csk}); \) return crs.

P(\text{crs}, x, w)

- \( (\text{sk}, \text{pk}) \leftarrow \Sigma.\text{KGen}(1^{\kappa}); \)
- \( (\text{sk}_{\text{OT}}, \text{pk}_{\text{OT}}) \leftarrow \Sigma_{\text{OT}}.\text{KGen}(1^{\kappa}); \)
- \( \pi_\Pi \leftarrow \Pi.\text{P}(\text{crs}, x, (w, \bot)); \sigma \leftarrow \Sigma.\text{Sign}(\text{sk}, \text{pk}_{\text{OT}})); \)
- \( \sigma_{\text{OT}} \leftarrow \Sigma_{\text{OT}}.\text{Sign}(\text{sk}_{\text{OT}}, \pi_\Pi \| x \| \text{pk} \| \sigma); \)
return \( \pi := (\pi_\Pi, \text{pk}, \sigma, \text{pk}_{\text{OT}}, \sigma_{\text{OT}}). \)

V(\text{crs}, x, \pi)

- Parse \( \pi \) as \( (\pi_\Pi, \text{pk}, \sigma, \text{pk}_{\text{OT}}, \sigma_{\text{OT}}); \)
- if \( \Pi.V(\text{crs}_\Pi, x, \pi_\Pi) = 0 \) \( \lor \) \( \Sigma.\text{Verify}(\text{pk}, \text{pk}_{\text{OT}}, \sigma) = 0 \) \( \lor \) \( \Sigma_{\text{OT}}.\text{Verify}(\text{pk}_{\text{OT}}, \pi_\Pi \| x \| \text{pk} \| \sigma, \sigma_{\text{OT}}) = 0 \) then return 0; else return 1.

Sim(\text{crs}, x, tc)

- \( (\text{sk}, \text{pk}) \leftarrow \Sigma.\text{KGen}(1^{\kappa}); \) \( (\text{sk}_{\text{OT}}, \text{pk}_{\text{OT}}) \leftarrow \Sigma_{\text{OT}}.\text{KGen}(1^{\kappa}); \)
- \( \pi_\Pi \leftarrow \Pi.\text{P}(\text{crs}, x, (\bot, \text{csk} - \text{sk})); \sigma \leftarrow \Sigma.\text{Sign}(\text{sk}, \text{pk}_{\text{OT}}); \)
- \( \sigma_{\text{OT}} \leftarrow \Sigma_{\text{OT}}.\text{Sign}(\text{sk}_{\text{OT}}, \pi_\Pi \| x \| \text{pk} \| \sigma); \)
return \( \pi := (\pi_\Pi, \text{pk}, \sigma, \text{pk}_{\text{OT}}, \sigma_{\text{OT}}). \)

Ext(\text{crs}, x, \pi, \text{tc}_\text{ext})

- \( (w, \bot) \leftarrow \Pi.\text{Ext}(\text{crs}, x, \pi, \text{tc}_\text{ext}); \) return \( w. \)

Fig. 2. The generic SE-NIZK compiler from [DS19].

Updatable SNARKs from Lamassu. In the following we present the generic construction using the definitional framework in [GKM+18] for updatable SNARKs and refer to [ARS20] for subversion version of the compiler. Roughly speaking, in the updatable CRS definition, Groth et. al relaxed the CRS model by allowing the adversary to either fully generate the CRS itself, or at least contribute to its computation as one of the parties performing updates. In other words, we can think of this as having the adversary interact with the KGen_{crs} algorithm. An updatable SNARK has the following additional PPT algorithms on top of \((\text{KGen}_{crs}, \Pi, \text{V}, \text{Sim})\). After running \((\text{crs}, \text{tc}, \zeta) \leftarrow \text{KGen}_{crs}, \) where \( \zeta \) is a proof of correctness of crs.

Ucrs(1^{\lambda}, \text{crs}, \{\zeta_i\}_{i=1}^n). Takes as input the security parameter \( \lambda \), a CRS \text{crs}, and a list of update proofs for the CRS. It outputs an updated CRS \text{crs}_\text{up} and a proof \( \zeta_{\text{up}} \) of the correctness of the update.
Vcrs(1^\lambda, crs, \{\zeta_i\}^i_{i=1}). Given the security parameter \lambda, a CRS crs, and a list of proofs \zeta_i. It outputs a bit indicating accept (b = 1), or reject (b = 0).

The standard trusted setup can be considered as an updatable setup with \zeta = \epsilon as the update proof and where the verification algorithm accepts anything. For a subversion resistant setup (Sub-zk-SNARKs), the proof \zeta could be added as extra elements into the CRS solely to make the CRS verifiable.

We present the full construction of SE updatable SNARKs in Fig. 3. Notice that in the Fig. 3, the subverter Z could be either the algorithms (Π.KGen, \Sigma.KGen) or the updater Ucrs.

**Theorem 3 ([ARS20]).** Let the underlying updatable SNARK scheme satisfy perfect completeness, updatable zero-knowledge, and updatable knowledge soundness. Let \Sigma be an EUF-CMA secure adaptable and updatable signature scheme and \Sigma_{OT} is a strongly unforgeable one-time signature scheme. Then, the SE updatable SNARKs argument system from Fig. 3, is (i) perfectly complete, (ii) updatable zero-knowledge, and (iii) updatable strong simulation extractable.

**Instantiation.** As an example instantiation, by taking updatable Schnorr signatures as presented in Section 2.5, using the LAMASSU framework we can now obtain an SE updatable SNARK by lifting the updatable SNARK in [GKM+18]. This, for instance, results in an overhead of \(1G_1 + 1G_2\) elements in the CRS and \(2G_1 + 2G_2 + 2Z_q\) elements in the proofs (cf. Table 2 for a comparison of different instantiations and existing ad-hoc approaches).

**4 Evaluation**

For the evaluation of OC∅C∅ and LAMASSU, we focus on SNARKs built from the pairing-friendly elliptic curve BLS12-381. In this case, we can leverage the Jubjub curve [HBHW19] used by Zcash for fast elliptic-curve arithmetic in the circuit. The Jubjub curve is a twisted Edwards curve defined over \(\mathbb{F}_r\), with \(r\) being the prime order of BLS12-381. Twisted Edwards curves enjoy complete addition laws and they naturally fit the requirements of Schnorr signatures.
KGen_{crs}(1^\lambda)

- (crs_{\Pi, tc_{\Pi}}, \zeta_{\Pi}) \leftarrow \Pi.KGen(1^\lambda);
- (csk, cpk, \zeta_{cpk}) \leftarrow \Sigma.KGen(1^\sigma);
- crs := (crs_{\Pi}, cpk), tc := (tc_{\Pi}, csk); return crs.

Ucrs(1^\lambda, crs, \{\zeta_i\}_{i=1}^n)

- (crs_{\Pi, up}, \zeta_{\Pi, up}) \leftarrow \Pi.Ucrs(1^\lambda, crs_{\Pi}, \{\zeta_{i, crs}\}_{i=1}^n);
- (cpk_{up}, \zeta_{cpk, up}) \leftarrow \Sigma.Ucrs(cpk, \{\zeta_{cpk_{i, \ell}}\}_{i=1}^n);
- return (crs_{up} = (crs_{\Pi, up}, cpk_{up}), \zeta_{up} = (\zeta_{\Pi, up}, \zeta_{cpk, up})).

Vcrs(1^\lambda, crs, \{\zeta_i\}_{i=1}^n)

- Parse \zeta_i as (\zeta_{i, \Pi}, \zeta_{cpk_{i, \ell}});
- if Vcrs_{\Pi}(1^\lambda, crs, \{\zeta_{i, crs}\}_{i=1}^n) = 1 \wedge 
  \Sigma.Vpk(pk, cpk, \{\zeta_{cpk_{i, \ell}}\}_{i=1}^n) = 1
then return 1; else return 0.

P(crs_{up}, x, w)

- (sk, pk) \leftarrow \Sigma.KGen(1^\sigma);
- (sk_{\Pi}, pk_{\Pi}) \leftarrow \Sigma_{\Pi}.KGen(1^\sigma);
- \pi_{\Pi} \leftarrow \Pi.P(crs_{up}, x, (w, \bot), \bot); \sigma \leftarrow \Sigma.Sign(sk, pk_{\Pi});
- \sigma_{\Pi} \leftarrow \Sigma_{\Pi}.Sign(sk_{\Pi}, \pi_{\Pi} || pk || \sigma);
return \pi := (\pi_{\Pi}, pk, \sigma, pk_{\Pi}, \sigma_{\Pi}).

V(crs_{up}, x, \pi)

- Parse \pi as (\pi_{\Pi}, pk, \sigma, pk_{\Pi}, \sigma_{\Pi});
- if \Pi.V(crs_{\Pi, up}, x, \pi_{\Pi}) = 0 \lor \Sigma.Verify(pk, pk_{\Pi}, \sigma) = 0
\lor \Sigma_{\Pi}.Verify(pk_{\Pi}, \pi_{\Pi} || pk || \sigma, \sigma_{\Pi}) = 0 then return 0;
else return 1.

Sim(crs_{up}, x, tc)

- (sk, pk) \leftarrow \Sigma.KGen(1^\sigma); (sk_{\Pi}, pk_{\Pi}) \leftarrow \Sigma_{\Pi}.KGen(1^\sigma);
- \pi_{\Pi, Sim} \leftarrow \Pi.Sim(crs_{\Pi, up}, x, (\bot, tc_{\Pi}), \bot);
- \sigma \leftarrow \Sigma.Sign(sk, pk_{\Pi});
- \sigma_{\Pi} \leftarrow \Sigma_{\Pi}.Sign(sk_{\Pi}, \pi_{\Pi, Sim} || pk || \sigma);
return \pi := (\pi_{\Pi}, pk, \sigma, pk_{\Pi}, \sigma_{\Pi}).

Ext_{\Sigma}(1^\lambda, crs, crs_{up}, \omega_{\Sigma})

- tc \leftarrow \Pi.Ext(1^\lambda, crs, crs_{up}, \omega_{\Sigma}); return tc.

Fig. 3. The SE updatable SNARKs from Lamassu.
Table 2. Comparison of SE-SNARKs. The given sizes for the CRS and proofs as well as the number of operations are overheads compared to the underlying SNARKs. For ad-hoc constructions the overhead is relative to Groth’s SNARK. n denotes the number of multiplication gates.

<table>
<thead>
<tr>
<th>Features</th>
<th>generic subversion updatable</th>
<th>crs</th>
<th>Overhead</th>
<th>(\pi) bits</th>
<th>(V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C(\emptyset)C(\emptyset) [KZM(\dagger)15](\dagger)</td>
<td>✔</td>
<td>✔*</td>
<td>✔</td>
<td>1(\lambda)</td>
<td>256</td>
</tr>
<tr>
<td>OC(\emptyset)C(\emptyset)[S]</td>
<td>✔</td>
<td>✔*</td>
<td>✔</td>
<td>2(\lambda)</td>
<td>512</td>
</tr>
<tr>
<td>OC(\emptyset)C(\emptyset)[G]</td>
<td>✔</td>
<td>✔*</td>
<td>✔</td>
<td>2(\lambda)</td>
<td>512</td>
</tr>
<tr>
<td>LAMASSU[S,S]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>1(G)</td>
<td>256</td>
</tr>
<tr>
<td>LAMASSU[S,S]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>1(G), 1(G)</td>
<td>1145</td>
</tr>
<tr>
<td>LAMASSU[S,G]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>1(G)</td>
<td>256</td>
</tr>
<tr>
<td>LAMASSU[S,G]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>1(G), 1(G)</td>
<td>1145</td>
</tr>
<tr>
<td>LAMASSU[S,BB]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>1(G)</td>
<td>256</td>
</tr>
<tr>
<td>LAMASSU[S,BB]</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>1(G), 1(G)</td>
<td>1145</td>
</tr>
<tr>
<td>Groth-Maller [GM17]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>(2(n) + 5)(G_1), (n)(G_2) 1910 + 1527(n)</td>
<td>–</td>
</tr>
<tr>
<td>Bowe-Gabizon [BG18]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>–(\dagger)</td>
<td>0</td>
</tr>
<tr>
<td>Lipmaa ((S_{\text{sep}})) [Lip19]</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>(n)(G_1)</td>
<td>382(n)</td>
</tr>
<tr>
<td>Kim-Lee-Oh [KLO19]</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>(n)(G_1)</td>
<td>382(n)</td>
</tr>
<tr>
<td>Atapoor-Baghery [AB19]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>1(\lambda)</td>
<td>256</td>
</tr>
<tr>
<td>Baghery [Bag19]</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
<td>1(\lambda)</td>
<td>256</td>
</tr>
</tbody>
</table>

\(\dagger\) Proves statements with respect to the evaluation of a random oracle (cf. Section 3.2).

\(\dagger\) Achieves no crs overhead by additionally requiring random oracles.

* With the non-black box extractor, C\(\emptyset\)C\(\emptyset\) retains the subversion resistance of the underlying SNARK [Bag19].
The Sapling protocol uses the Jubjub curve to prove relations of the form \(rk = ak \cdot g^a\) and checking that \(\alpha\) is in the correct range for the witness \(\alpha\). The first part of the relation can be expressed with 756 constraints, whereas the latter can be expressed with 252 constraints, so a total of 1008 constraints [HBHW19, Section A.4]. For Lamassu, we extend the relation with a proof of the statement \(cpk = pk \cdot \mu(csk - sk)\) with the witness \(csk - sk\). For Schnorr signatures (cf. Appendix A.4), but also other DLOG-based signature schemes such as Schnorr, the public key is a group element of the form \(g^{sk}\) and similarly \(\mu\) simply maps scalars to the corresponding group element, i.e., \(\mu(x) = g^x\). Hence, the circuit for this relation also requires 1008 constraints. Compared to the OC∅∅C∅ framework instantiations (cf. Table 1), Lamassu needs only 200 constraints more than the most aggressive choice using Poseidon and beats all others in the number of constraints. Considering that Schnorr signatures are well established primitives, and that the confidence in their security is far bigger than all the recent SNARK/STARK-friendly primitives, this additional confidence and the updatability feature come at a very small cost for the prover.

In terms of bandwidth overhead, we only need to compare the overhead induced by \(cpk = pk \cdot \mu (csk - sk)\) together with the signature and one-time signature in Lamassu, and \(\mu = f_{s0}(pk_s) \land \rho = f_{s0}(\beta_0)\) and the one-time signature in the case of OC∅∅C∅. We start with Lamassu. The CRS is extended with a public key \(cpk\) of signature scheme \(\Sigma\), i.e., when using Schnorr (or ECDSA) a point on the Jubjub curve which requires 510 bits without or 256 bits with point compression. For each proof, new \(\Sigma\) and \(\Sigma_{OT}\) keys are sampled. The proof then includes a \(\Sigma\) public key and signature, as well as as \(\Sigma_{OT}\) public key and signature. The former amounts to 256 bits for the public key and 504 bits for the signature (2 integers modulo the group order), and the latter – when instantiated as Groth’s sOTS over Jubjub (or a curve of similar size) – amounts to 768 bits for the public key (3 group elements) and 504 bits for the signature (2 integers modulo the group order). In total, the size of the proof is increased by 2032 bits. The updatable version is similar, but Schnorr is performed in \(G_1\) with additional public key and update in \(G_2\).

For C∅∅C∅, the CRS is extended with a SHA256 commitment. The proofs are extended with a freshly generated \(\Sigma_{OT}\) public key and a signature together with the evaluation of a PRF also instantiated with SHA256. Hence, the CRS grows by 256 bits and each proof grows by 1016 bits assuming the use of Schnorr over Jubjub. For OC∅∅C∅, the CRS is extended with \(\rho\) and \(\beta\), both 256 bits each. Each proof additionally contains \(\mu\) as well as freshly generated \(\Sigma_{OT}\) public key and signature. Using Groth’s sOTS, the proof grows by 1528 bits in total.

In Table 2 we present a comparison of SE-SNARKs including OC∅∅C∅ using Groth’s OTS, OC∅∅C∅[G], and Schnorr, OC∅∅C∅[S], Lamassu using Schnorr, Lamassu[S,S], Groth’s OTS, Lamassu[S,G], and Boneh-Boyen signatures [BB04], Lamassu[S,BB], both as non-updatable and updatable variant. The overhead is relative to the underlying SNARK (for the generic constructions) or the SNARK they are based on, e.g., relative to [Gro16]. In the table, \(n\) denotes the number of multiplication gates in the circuit, \(G_1\) and \(G_2\) the two source groups of a bi-
linear group, $G$ a group with prime order $q$, and $\lambda$ the sizes of commitments and PRF evaluations. For concrete numbers, we followed the above choice of curves, namely Jubjub ($\mathcal{G}$) and BLS12-381 as bilinear group ($p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, \tilde{g}$), respectively. For commitments and PRF images, we assume that they are 256 bits. For the verifier overhead, we consider the most expensive operations. $E_G$ denotes an exponentiation in $\mathbb{G}$ and $P$ a pairing evaluation. Thereby, $P$ is a factor 10 slower than $E_G$.

Compared to the ad-hoc constructions, the generic frameworks $C\emptyset C\emptyset$, $OC\emptyset C\emptyset$, and LAMASSU offer a trade-off between the size of the CRS, proof sizes and verifier overhead. Especially when comparing to Kim-Lee-Oh [KLO19], which only extends the CRS, this trade-off becomes apparent. When comparing to the others, the verifier overhead is smaller than the ones observed for Groth-Maller [GM17], Bowe-Gabizon [BG18] and Lipmaa [Lip19] and is comparable to the constructions of Atappoor-Baghery [AB19] and Baghery [Bag19], yet LAMASSU offers more features.

Acknowledgements. We thank Ahmed Kosba for valuable feedback on the choice of the one-time signature scheme in the $C\emptyset C\emptyset$ framework. We also like to thank the anonymous reviewers. This work was supported by the European Union’s Horizon 2020 research and innovation programme under grant agreements n°830929 (CyberSec4Europe) and n°871473 (KRAKEN), by the Austrian Science Fund (FWF) and netidee SCIENCE under grant agreement P31621-N38 (PROFET) and the Estonian Research Council grant PRG49.

References


26


GGP10. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive ver-


A Omitted Primitives

A.1 Non-Interactive Zero-Knowledge

Let $RGen$ be a relation generator, such that $RGen(1^\lambda)$ returns a polynomial-time decidable binary relation $R = \{(x, w)\}$. Here, $x$ is the statement and $w$ is the witness. We assume that $\lambda$ is explicitly deducible from the description of $R$. The relation generator also outputs auxiliary information $aux_R$ that will be given to the honest parties and the adversary. Let $L_R = \{x : \exists w, (x, w) \in R\}$ be an NP-language. Non-interactive zero-knowledge (NIZK) proofs and arguments in the CRS model consist of algorithms $(KGen_{crs}, P, V, Sim)$, and satisfy the following properties: completeness (for all common reference strings $crs$ generated by $KGen_{crs}$ and $(x, w) \in R$, we have that $V(crs, x, P(crs, x, w)) = 1$), zero-knowledge (there exists a simulator $Sim$ that outputs a simulated proof such that an adversary cannot distinguish it from proofs computed by $P(crs, x, w)$), soundness (an adversary cannot output a proof $\pi$ and an instance $x \notin L_R$ such that $V(crs, x, \pi) = 1$). Moreover, knowledge soundness steps further and says that for any prover generating a valid proof there is an extractor $Ext$ that can extract a valid witness.

A.2 QAPs and R1CS

Quadratic Arithmetic Programs (QAPs) have been introduced by Gennaro et al. [GGPR13] as a language where for an input $x$ and witness $w$, $(x, w) \in R$ can be verified by using a parallel quadratic check, and that has an efficient reduction from a well-known language (either Boolean or Arithmetic) Circuit-SAT.

Definition 14 (QAP). A quadratic arithmetic program over a field $\mathbb{F}$ is a tuple of the form

$$(\mathbb{F}, n, \{A_i(X), B_i(X), C_i(X)\}_{i=0}^{i=m}; D(X))$$
where \( A_i(X), B_i(X), C_i(X), D(X) \in \mathbb{F}[X] \), define a language of statements 
\((s_1, \ldots, s_n) \in \mathbb{F}\) and witnesses \((s_{n+1}, \ldots, s_m) \in \mathbb{F}^{m-n}\) such that

\[
\left( \sum_{i=0}^{m} s_i A_i(X) \right) \cdot \left( \sum_{i=0}^{m} s_i B_i(X) \right) = \left( \sum_{i=0}^{m} s_i C_i(X) \right) + H(X) \cdot D(X) \tag{2}
\]

where \( s_0 = 1 \) and for some degree-\((d - 2)\) quotient polynomial \( H(X) \), where \( d \) is the degree of \( D(X) \). Let the degrees of all \( A_i(X), B_i(X) \) and \( C_i(X) \) are at most \( d - 1 \).

We note that all the considered SNARK constructions are for QAPs defined over a bilinear group. Thus we consider relation generators \( \text{RGen} \) of the following form:

**Definition 15 (QAP relation).** A QAP relation generator \( \text{RGen} \) is a PPT algorithm that on input \( \lambda \) returns a relation description \( R = (\text{pars}, n, (A, B, C) \in \mathbb{F}^{(d-1)}[X]^{m-1}, D \in \mathbb{F}^{(d)}[X]) \) where \( \text{pars} \) is a bilinear group whose order \( p \) defines \( \mathbb{F} := \mathbb{Z}_p \) and \( n \leq m \). Fix \( x \in \mathbb{F}^n \) and \( w \in \mathbb{F}^{m-n} \), we define \( R(x, w) = 1 \) if there exists \( H(X) \in \mathbb{F}[X] \) so that Eq. (2) holds for \( x = (s_1, \ldots, s_n) \) and \( w = (s_{n+1}, \ldots, s_m) \).

For reducing arithmetic circuits to QAP relations, circuits can first be transformed to a system of rank-1 quadratic equations (R1CS) which is latter transformed into a QAP \([\text{BCG} + 13]\). The R1CS relation over a field \( \mathbb{F} \) consists of instance-witness pairs \(((A, B, C, v), w)\) with matrices \( A, B, C \in \mathbb{F}^{n \times m} \) and vectors \( v, w \) such that \((Az) \circ (Bz) = Cz\) with \( z = (1, v, w) \in \mathbb{F}^m \) where \( \circ \) denotes the entry-wise product. For capturing arithmetic circuit satisfaction, \( A, B, C \) represent the gates, \( v \) the public inputs, and \( w \) the private inputs and wire values.

### A.3 Public-key Encryption

**Definition 16.** A public key encryption scheme \( \Omega = (\text{KGen}, \text{Enc}, \text{Dec}) \) consists of the following PPT algorithms:

- \( \text{KGen}(1^\lambda) \): Given a security parameter \( \lambda \) it outputs the secret key \( \text{sk} \) and public key \( \text{pk} \) with message space \( \mathcal{M} \).
- \( \text{Enc}(\text{pk}, m) \): Given a public key \( \text{pk} \) and a message \( m \in \mathcal{M} \) it outputs a ciphertext \( c \).
- \( \text{Dec}(\text{sk}, C) \): Given a secret key \( \text{sk} \) and a ciphertext \( c \) it outputs a message \( m \in \mathcal{M} \cup \{\bot\} \).

We say that an encryption scheme \( \Omega \) is perfectly correct if for all \( \kappa \in \mathbb{N} \), for all \((\text{sk}, \text{pk}) \leftarrow \text{KGen}(1^\lambda)\) and for all \( m \in \mathcal{M} \) it holds that \( \text{Dec}(\text{sk}, \text{Enc}(\text{pk}, m)) = m \). Below, we recall the standard notion of indistinguishability under chosen plaintext attacks (IND-CPA security).
Definition 17 (IND-CPA). A public key encryption scheme $\Omega$ is IND-CPA secure, if for all PPT adversaries $A$ it holds that

$$\Pr \left[ (sk, pk) \leftarrow KGen(1^\lambda), b \leftarrow \{0, 1\}, \\ (m_0, m_1, st) \leftarrow A(pk), b^* \leftarrow A(Enc(pk, m_b), st) : b = b^* \right] \approx \frac{1}{2}.$$ 

A.4 Schnorr Signatures

We recall the Schnorr signature scheme [Sch90] together with the required Adapt algorithm (cf. [DS19]) in Fig. 4. It can be shown to provide EUF-CMA security in the random oracle model (ROM) under the DLP in $G$ by using the now popular rewinding technique [PS96] (cf. also [KMP16] for a recent treatment on tightness and optimality of such reductions). In the following we present Schnorr signatures with respect to a common setup, i.e., $PP \leftarrow PGen(1^\lambda)$ are given to all instances of $KGen$ and let $GGen$ be a group generator that on input $1^\lambda$ outputs the description of a prime order group $G = (G, g, p)$ s.t. $\lambda = \log_2 p$ and generator $g$. Recall, that in addition Schnorr requires a collision resistant hash function $H : G \times M \rightarrow \mathbb{Z}_p$ (formally sampled uniformly at random from a family $\{H_k\}_{k \in K}$ of hash functions) and thus we have $PP := (G, H)$ (which we assume to be an implicit input to all algorithms). We recall a lemma from [DS19] showing that Schnorr signatures using the Adapt algorithm in Fig. 4 satisfies the signature adaption notion in Definition 8.

Lemma 1 ([DS19]). Schnorr signatures are adaptable according to Definition 8.

A.5 Groth’s Strong One-Time Signatures

In Fig. 5 we recall the strong one-time signature scheme from Groth [Gro06] and its security below:

Theorem 4 ([Gro06]). Assuming hardness of computing discrete logarithms and collision-resistance of the hash function, the scheme $(PGen_{ots}, KGen_{ots}, Sign_{ots}, Verify_{ots})$ described in Fig. 5 is a strong one-time signature scheme for signing messages $m \in \{0, 1\}^*$ with perfect correctness.

A.6 BDH Knowledge Assumption

Let $BGen$ be a PPT algorithm that, on input a security parameter $\lambda$, outputs $BG = (p, G_1, G_2, G_T, e, g, \hat{g})$ for generators $g$ and $\hat{g}$ of $G_1$ and $G_2$, respectively, and $\Theta(\lambda)$-bit prime $p$.

Assumption 1 (BDH-Knowledge Assumption [ABLZ17]) We say that $BGen$ is BDH-KE secure for $R$ if for any $\lambda$, $(R, aux_R) \in \text{im}(R(1^\lambda))$, and PPT adversary $A$ there exists a PPT extractor $\text{Ext}_{BDH}^A$, such that

$$\Pr \left[ r \leftarrow \text{RND}(A), \\ (V, \hat{V} || a) \leftarrow (A || \text{Ext}_{BDH}^A)(R, aux_R; \omega_A) : e(V, \hat{g}) = e(g, \hat{V}) \land g^a \neq V \right] \approx_{\lambda} 0.$$
PGen(1^λ)
- \( G \leftarrow \text{GGen}(1^λ) \); \( H \leftarrow \{H_k\}_{k \in \mathcal{K}} \);
- return \( PP := (G, H) \);

KGen(PP):
- Parse \( PP = ((G, g, p), H) \);
- \( x \leftarrow \mathbb{Z}_p \);
- return \( (sk, pk) := (x, g^x) \).

Sign(sk, m):
- Parse \( sk = x \);
- \( r \leftarrow \mathbb{Z}_p \); \( R := g^r \); \( c := H(R||m) \); \( y := r + x \cdot c \mod p \)
- return \( \sigma := (c, y) \).

Verify(pk, m, σ):
- Parse \( pk = g^x; \sigma = (c, y) \);
- if \( c = H((g^x)^c g^y, m) \) return 1 else return 0.

Adapt(pk, m, σ, Δ):
- Parse \( pk = g^x; \sigma = (c, y) \); \( Δ \in \mathbb{Z}_p \);
- \( pk' := g^{Δ}; \ y' := y + c \cdot Δ \mod p \);
- return \( \sigma' := (c, y') \).

Fig. 4. Schnorr signatures.

Note that the BDH assumption can be considered as a simple case of the PKE assumption of [DFGK14] (where \( \mathcal{A} \) is given as an input the tuple \( \{(g^{x_i}, \hat{g}^{x_i})\}_{i=0}^{n} \) for some \( n \geq 0 \), and assumed that if \( \mathcal{A} \) outputs \((V, \hat{V})\) then she knows \((a_0, a_1, \ldots, a_n)\), such that \( V = g^{\sum_{i=0}^{n} a_i x^i} \) as used in the case of asymmetric pairings in [DFGK14]. Thus, BDH can be seen as an asymmetric-pairing version of the original and by now well established KoE assumption due to Damgård [Dam92].
\begin{align*}
PGen_{\text{ots}}(1^\lambda) & : \\
& \begin{array}{l}
\text{- } \mathcal{G} \leftarrow \text{GGen}(1^\lambda); \text{ } H \leftarrow \{H_k\}_{k \in \mathcal{K}}; \\
\text{- } \text{return } \text{PP} := (\mathcal{G}, H);
\end{array} \\
KGen_{\text{ots}}(\text{PP}) & : \\
& \begin{array}{l}
\text{- } \text{Parse } \text{PP} = ((\mathcal{G}, g, p), H); \\
\text{- } x_s, y_s, r_s, s_s \leftarrow \mathbb{Z}_p; \\
\text{- } f_s := g^{x_s}; \text{ } h_s := g^{y_s}; \text{ } c_s := g^{x_s} \cdot h_s^{s_s}; \\
\text{- } \text{return } (\text{sk}, \text{pk}) := ((x_s, y_s, r_s, s_s), (f_s, h_s, c_s)).
\end{array} \\
\text{Sign}_{\text{ots}}(\text{sk}, m) & : \\
& \begin{array}{l}
\text{- } \text{Parse } \text{sk} = (x_s, y_s); \\
\text{- } r \leftarrow \mathbb{Z}_p; \text{ } z := x_s(r_s - r) + y_s \cdot s_s - H(m) \cdot y_s^{-1} \mod p \\
\text{- } \text{return } \sigma := (r, z).
\end{array} \\
\text{Verify}_{\text{ots}}(\text{pk}, m, \sigma) & : \\
& \begin{array}{l}
\text{- } \text{Parse } \text{pk} = (f_s, h_s, c_s); \sigma = (r, z); \\
\text{- } \text{if } c_s = g^{H(m)} \cdot f_s^r \cdot h_s^z \text{ return } 1 \text{ else return } 0.
\end{array}
\end{align*}

Fig. 5. Groth’s strong one-time signature scheme.