Circuit-Succinct Universally-Composable NIZKs with Updatable CRS

Behzad Abdolmaleki[1], Noemi Glaeser[2,3], Sebastian Ramacher[4], and Daniel Slamanig[4]

[1] University of Sheffield, Sheffield, UK
abdolmaleki.behzad.ir@gmail.com
[2] Max Planck Institute for Security and Privacy, Bochum, Germany
[3] University of Maryland, College Park, USA
nglaeser@umd.edu
[4] AIT Austrian Institute of Technology, Vienna, Austria
{sebastian.ramacher, daniel.slamanig}@ait.ac.at

Abstract. Non-interactive zero-knowledge proofs (NIZKs) and in particular succinct NIZK arguments of knowledge (zk-SNARKs) increasingly see real-world adoption in large and complex systems. Many zk-SNARKs require a trusted setup, i.e., a common reference string (CRS), and for practical use it is desirable to reduce the trust in the CRS generation. The latter can be achieved via the notions of *subversion* or *updatable* CRS. Another important property when deployed in large systems is the ability to securely compose them to obtain more complex protocols, e.g., via the Universal Composability (UC) framework. Relying on the UC framework allows arbitrary and secure composition of protocols in a modular way.

In this work, we investigate whether zk-SNARKs can provide updatability and composability simultaneously. This is a challenging task as the UC framework rules out several natural techniques for such a construction. As our main result, we show that it is indeed possible to achieve these properties in a generic and modular way if we relax the succinctness properties of zk-SNARKs slightly to those of a circuit-succinct NIZK which is not witness-succinct, i.e., by increasing the proof size of the underlying zk-SNARK by the size of the witness w. We argue that for various practical applications of zk-SNARKs this overhead is acceptable. Our starting point is the Lamassu framework (ACM CCS'20), which we extend in several directions. Our new generic compiler adds only minimal overhead, which we demonstrate by benchmarking its application to the Sonic proof system (ACM CCS'19).

1 Introduction

Non-Interactive Zero-Knowledge proofs (NIZKs) [GMR85, BFM88] are a powerful primitive which allows parties to prove the validity of an arbitrary NP statement in a single message (the proof) in a publicly verifiable way, without revealing anything beyond its validity. Especially NIZKs for certain classes of algebraic languages [FS87, GS08, JR13] are extensively used in the design of privacy-preserving systems (such as anonymous credentials and digital currencies) as well as multi-party computation protocols.

Due to their numerous applications in privacy-preserving cryptocurrencies and blockchains in general, tremendous research has been dedicated to designing short NIZKs with efficient verification (at the cost of tolerating a less efficient prover) [Gro10, Lip12, GGPR13, PHGR13, Lip13, DFGK14, Gro16, BCR$^+$19, MBKM19, GWC19, CHM$^+$20, GLS$^+$21]. These so-called zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs) [BCCT12] have also enabled proofs of statements that are not efficiently realizable with NIZKs for algebraic languages. Despite their well-known drawback of requiring non-falsifiable assumptions [GW11], zk-SNARKs are attractive in practice not only due to their (relative) practical efficiency, but more importantly due to their general-purpose nature. While customized NIZK proofs for application-specific statements can result in protocols highly optimized for the specific task at hand, the enormous cryptographic expertise and time required to develop new protocols for each application severely limits the adoption of modern cryptographic building blocks. In contrast, the plethora of zk-SNARK toolchains[5] enable non-cryptography experts to easily express a statement to be proven in a familiar programming language and the corresponding implementations are generated automatically.

Secure adoption of zk-SNARKs. Simplifying the use and adoption of zk-SNARKs in (complex) applications introduces the risk that the security of entire systems could be compromised due to the lack of some property by the utilized zk-SNARK. To address this issue, recent works have looked at composing zk-SNARKs in a modular way [CFQ19, CFF$^+$21] and designing generalized frameworks for their construction [RZ21]. One important but not readily available property is *non-malleability*. Non-malleability [Sah99, Sah01], modeled via *simulation extractability* (SE), guarantees that a proof cannot be "mauled" into another valid proof (either for the same statement or a related statement). SE has been intensively studied in zk-SNARK constructions [GM17, Lip19, BPR20, BKSV21, GOP$^+$22, GKK$^+$22]. A stronger property is *composability*, which states that a zk-SNARK can be arbitrarily composed with other cryptographic primitives and its security properties will still be guaranteed to hold. Composability is generally modeled using the universal composability (UC) framework [Can01], a very popular tool for modeling security in blockchain-based systems [KMS$^+$16, AME$^+$21, TMM21, TMM22, GMM$^+$22]. Ideally, it would be possible to generically add UC security to a large class of (if not all) zk-SNARKs, making minimal assumptions on the underlying SNARK.

Another issue for secure deployment is that a large class of zk-SNARKs require a trusted setup to generate a common reference string (CRS), whose trapdoor must be deleted after setup. One technique to reduce trust in the CRS generation is a distributed setup ceremony [KMSV21], but this is cumbersome in practice. An alternative approach is the notion of subversion NIZK [BFS16] or

[5] `https://github.com/ventali/awesome-zk#tools`

subversion (zk-)SNARKs [ABLZ17, Fuc18]. Unfortunately, this approach only provides guarantees for the prover. A viable middle ground is an updatable CRS [GKM+18]: anyone can update the CRS, and their update can be verified by anyone. Even in the presence of a malicious CRS generator, as long as one operation – the CRS creation or one of its updates – has been performed honestly, zero-knowledge is guaranteed for the prover and soundness for the verifier. This concept is becoming increasingly popular [MBKM19, GWC19, CHM+20, RZ21, CFF+21, Lip22, NRBB22]. Ideally, a generic transformation to add composability should be compatible with an updatable CRS.

Hurdles for generic composability. Most zk-SNARKs, both with transparent setup [BCR+19, Set20] and (updatable) CRS [MBKM19, GWC19, CHM+20], achieve non-interactivity via the Fiat-Shamir (FS) heuristic [FS87], which requires a rewinding extractor for knowledge soundness and is therefore not compatible with the UC framework. It is unclear whether FS protocols meet even the weaker definition of SE; with the exception of three-round public-coin interactive arguments [FKMV12], this is an ongoing area of research [GKK+22]. Straight-line extractable alternatives to the FS transform [Fis05, Unr15] to achieve knowledge soundness unfortunately incur a performance penalty and their application to multi-round protocols has also not been studied. We will later show that such a transform is useful for the proof of CRS updates, which does not affect the efficiency of the overall proof system.

Other zk-SNARKs which avoid the FS transform rely directly on knowledge assumptions [Gro10, Gro16, Lip22]. Unfortunately, the use of knowledge assumptions is also not fully compatible with the UC framework: although there is recent progress on knowledge assumptions [KKK21a] and algebraic adversaries [ABK+21] in UC, the results are still not generically applicable. When it comes to SE, the popular zk-SNARK due to Groth [Gro16] only satisfies a weak notion of SE [BKSV21] or requires specifically crafted designs [GM17] to achieve SE. In case of updatable CRS, SE is achieved only via custom designs or non-black-box modifications of existing designs [GKM+18, Lip19]: no general UC-compatible transformation, apart from a recent concurrent and independent work [GKO+23] which we discuss soon, is known.

Generic composability and non-malleability. The central problem underlying the above issues is the reliance on non-black-box extractors, i.e., either rewinding extractors for FS or the direct use of knowledge assumptions. When a CRS is available, a well-known technique to avoid these issues and provide straight-line extraction is to extend the CRS by a public key and include an encryption of the witness in the proof. This also requires extending the original statement to show that the correct witness was encrypted ($\mathcal{R}_{enc}$ in Fig. 1) [DP92]. This trick can be combined with the classical OR trick (i.e., an alternate clause $\mathcal{R}_{sig}$, to be used only by the simulator, which checks for a valid signature) to enable unbounded simulation of proofs [DDO+01], i.e., SE with a straight-line extractor. The C∅C∅ framework [KZM+15] uses these two ideas to generically obtain UC-secure SE NIZKs. Recent work [ARS20] revisited the

CØCØ framework and tailored it to updatable zk-SNARKs. In particular, the new framework LAMASSU uses the non-black-box extractor of the underlying zk-SNARK instead of an encryption of the witness, thus keeping the transformed zk-SNARK succinct (modulo some small constant overhead). To make unbounded proof simulation compatible with an updatable CRS, LAMASSU applies a variant of key-homomorphic signatures of [DS19], i.e., updatable signatures. The result is a generic framework for CRS-updatable SE-SNARKs.

Unfortunately, the non-black-box extractor of LAMASSU makes it incompatible with the UC framework. Switching back to a black-box extractor (i.e., an encryption of the witness) solves this issue at the cost of increasing the proof size, but reintroduces a problem with CRS updatability. In particular, since the CRS now additionally contains an encryption of the witness, the public-key encryption (PKE) scheme used needs to be compatible with updatability. Recent work [BS21] tries to overcome this issue by introducing a PKE with updatable keys. However, despite claiming to provide a black-box approach, the updatability of their PKE is based on an extractor which relies on a concrete knowledge assumption, making it non-black-box and therefore not UC-compatible.

In summary, although previous works have shown how to generically construct composable NIZKs and non-malleable updatable SNARKs, no generic transformation *simultaneously* achieves composability and compatibility with updatable CRS. We summarize these approaches in Table 1.

Table 1. Comparison with concurrent and previous work.

	UC		succinctness-preserving		
	SE	BBE	in $\lvert C\rvert$	in $\lvert \mathtt{w}\rvert$	upd. CRS
CØCØ [KZM+15]	✓	✓	✓	✗	✗
DS [DS19]	✓	✗	✓	✓	✗
LAMASSU [ARS20]	✓	✗	✓	✓	✓
This work	✓	✓	✓	✗	✓
Concurr. work [GKO+23]	✓	✓	✓	✓	✗

Black-box extractability (BBE) and the succinctness of zk-SNARKs. In CRS-based NIZKs, the proof size is linear in the size of the circuit C computing the NP relation, except for either a multiplicative [GOS12] or additive [KNYY19] overhead. Further reducing the proof size requires reliance on heavy machinery, e.g., indistinguishability obfuscation [SW14] or knowledge assumptions [Gro10, Lip12, GGPR13]. The latter is the approach taken by zk-SNARKs, which are *circuit-* and *witness-succinct*. Typically this means that the proof size is $\mathsf{poly}(\lambda, \log|\mathsf{C}|)$, where λ is the security parameter (in fact, this is even independent of the witness size). A weaker notion of succinctness is *circuit-succinctness* [KMS+16, KNYY20], where the proof size is $\mathsf{poly}(\lambda, \log|\mathsf{C}|) + |\mathtt{w}|$.

In other words, the size of the proof and verification time are (quasi-)linear in the witness size $|\mathtt{w}|$, but sublinear in size of the circuit that encodes the language.

As mentioned above, black-box simulation extractability (and thus UC) requires us to additionally include a ciphertext in the proof, so this paradigm can only give circuit-succinct proofs. Because of the resulting additive $|\mathtt{w}|$ overhead, this approach is not suitable for applications with huge witnesses such as scalability solutions in blockchains (e.g., zk-rollups). However, there are many practical applications which often deal with relatively small witnesses. One prominent example is the witness of the Sapling output or Spend circuit in Zcash [HBHW22]. These consist of group elements from the Jubjub elliptic curve, scalars, and paths in a Merkle tree. For the Spend circuit the size is bound by 1413 bytes. Moreover, there are many applications with small to moderate-sized witnesses such as SNARK-based authentication schemes in the context of self-sovereign identity [LCOK21] or anonymous credentials [RWGM22]. Blockchain applications include blockchain-based e-voting (e.g., the recently launched Vocdoni[6]) or proofs of assets or swaps in cryptocurrencies [EKKV22]. Furthermore, the Merkle membership proofs used by Filecoin for proofs of replication[7] start from small nodes which serve as witnesses and thus our techniques also appear applicable in this context.

1.1 Our Contributions

In this work, we give the first *fully black-box approach to generically build* circuit-succinct UC-secure NIZKs with updatable CRS from zk-SNARKs, thereby circumventing the above problems. Our contributions can be summarized as follows:

Framework for (circuit-succinct) UC NIZKs with updatable CRS. We present BB-Lamassu, a framework for black-box (BB) SE (i.e., universally-composable) circuit-succinct NIZKs with updatable CRS. Our framework can be seen as a hybrid of C∅C∅ and Lamassu, combining the BB extractability of the former with the updatable CRS of the latter (see Figure 1). However, this requires novel tools and we provide a more detailed intuition in Section 1.2.

Treatment of updatable NIZK in the UC framework. We provide an explicit treatment of BB-Lamassu in the UC framework. To the best of our knowledge, there is no treatment of SNARKs/NIZKs with updatable CRS in the UC framework so far. In an independent work, Kerber *et al.* [KKK21b] defined a functionality for updatable SRS to perform this secure generation in a distributed manner, but did not investigate the UC-security of the whole NIZK construction. Our analysis is carried out in the local ROM, which can be realized

[6] https://docs.vocdoni.io/architecture/protocol/anonymous-voting/zk-census-proof.html

[7] https://trapdoortech.medium.com/filecoin-how-storage-replication-is-proved-using-zk-snark-8a2a06b1c582

in practice by domain separation in the hash function. We note that the use of an RO arises from a building block (the proof of CRS update) and not from the construction of our compiler. While we currently consider only the local ROM, we expect that an analysis in the global ROM is possible when relying on Fischlin for the update proofs via the techniques in [LR22].

Implementation and evaluation. To demonstrate the applicability of BB-Lamassu, we provide a detailed analysis of the induced overheads. For concrete instantiations, we estimate overheads of 32 bytes for the CRS, 170 bytes for the CRS update, and 256 bytes plus the size of the witness for the proof. This is a reduction in both storage and runtime overheads compared to Lamassu [ARS20]. For witness sizes observed in practical applications such as Zcash, BB-Lamassu adds well below 10,000 additional constraints.

As a concrete example, we describe how BB-Lamassu can be applied to Sonic [MBKM19], a zk-SNARK with updatable CRS. Specifically, we discuss how Sonic's CRS update procedure can be modified to make update proofs UC-compatible. We also experimentally evaluate the overhead introduced by BB-Lamassu when applied to Sonic. For a SHA-256 preimage, which is interesting for Merkle-tree membership proofs, the prover and verifier overhead, respectively, is $\approx 1.2\times$ and $1.07\times$. Our evaluation shows that as the circuits become larger and more complex, proving and verifying the original circuit dominates the overall performance costs and the overhead added by BB-Lamassu converges to the size of the witness.

Concurrent and independent work. A recent concurrent and independent work [GKO+23] presents an approach that avoids the linear dependency on the size of the witness and thus obtains circuit- *and witness-succinct* UC SNARKs in the global random oracle model (GROM). The core idea is to replace the public-key encryption of the witness with a succinct commitment that is straight-line extractable. While their overall approach is generic, they show that the KZG polynomial commitment [KZG10, CHM+20] provides all the required properties and use it to encode the witness as the coefficients of the polynomial. Then they apply Fischlin's approach [Fis05] to obtain straight-line extractability.

Unfortunately, the authors only discuss the generic compiler and leave (custom) instantiations for future work. Consequently, it is hard to estimate the concrete overhead. But we can analyze lower bounds for the proof of the KZG evaluation algorithm and the witness encoding. For security parameter λ, their approach requires at least λ elements from $\mathbb{G}_1$ and $\mathbb{F}$ for the polynomial commitment evaluation and λ elements from $\mathbb{F}$ for the polynomial evaluation. Assuming a statistical security parameter of 80 bits and a pairing-friendly elliptic curve group $\mathbb{G}_1$ such as BLS-381, the constant overhead is at least 6.2 KB.

In contrast, our approach only needs a single call to a PKE and a symmetric encryption (a hybrid encryption to encrypt the witness), so the constant overhead is below 0.6 KB (cf. Section 5). For large witnesses, the approach in [GKO+23] (ignoring computational costs) will clearly be superior in terms of proof size due to being witness-succinct, but for witnesses up to ≈ 6 KB (such as in cases discussed earlier) our approach is competitive.

Finally, while in principle the CRS of the KZG polynomial commitment is amenable to updatability, the concurrent work [GKO+23] does not explicitly consider updatability. Obtaining an *updatable* UC SNARK from their approach does not appear to be straightforward as for KZG as well as the underlying SNARK one would require knowledge assumptions, preventing UC compatibility. We leave combining our techniques with those in [GKO+23] to obtain alternative updatable witness-succinct UC SNARKs as an interesting direction for future work.

1.2 Technical Overview

We now describe the idea of our new framework BB-Lamassu in more detail (cf. Fig. 1). Our starting point is the Lamassu framework [ARS20], which transforms any updatable SNARK into an SE updatable SNARK (yellow box). Lamassu adapts the simulation technique of DS [DS19] (brown box), which used the OR trick to combine the underlying SNARK's non-BB extractor with key-homomorphic signatures (adding the $\mathcal{R}_{sig}$ clause to the relation). To support an updatable CRS, Lamassu swaps the signature for an updatable signature (US). Extractability of the US updates in [ARS20] requires a non-BB extractor based on a knowledge assumption.

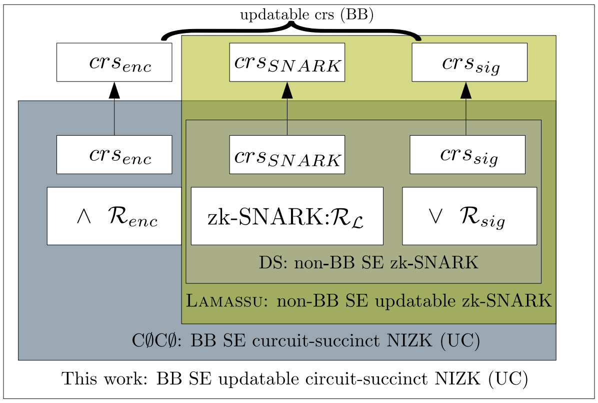
updatable crs (BB)
crs_{enc}
crs_{SNARK}
crs_{sig}
crs_{enc}
crs_{SNARK}
crs_{sig}
$\wedge\ \mathcal{R}_{enc}$
zk-SNARK:$\mathcal{R}_{\mathcal{L}}$
$\vee\ \mathcal{R}_{sig}$
DS: non-BB SE zk-SNARK
Lamassu: non-BB SE updatable zk-SNARK
C∅C∅: BB SE curcuit-succinct NIZK (UC)
This work: BB SE updatable circuit-succinct NIZK (UC)

Fig. 1. Overview of our approach including previous work from Table 1.

In BB-Lamassu, as in C∅C∅ [KZM+15], our proofs include an encryption of the witness ($\mathcal{R}_{enc}$) for BB-extractability. To be compatible with updatability, we instantiate this with a novel public-key encryption (PKE) primitive which we call *extractable key-updatable PKE (EKU-PKE)*, for which we show an efficient construction. (As mentioned earlier, a similar notion introduced independently in [BS21] is not BB-extractable and thus not useful for us.) We still have to overcome the hurdle of providing BB extraction for the US and the public key of the EKU-PKE in the CRS (crs_{enc}). Very briefly, our key idea is to use not necessarily succinct but efficient NIZK proofs *without a CRS* that provide BB extraction for all updates of the CRS elements, i.e., updates of the underlying SNARK (crs_{SNARK}), of the public key of the US scheme (crs_{sig}), and of the public key of the EKU-PKE scheme (crs_{enc}). We choose to base these proofs on Σ-protocols converted to NIZK proofs using either the Fiat-Shamir (FS) [FS87], Fischlin [Fis05] or Unruh [Unr15] approach. While this requires that the updates of all components are Σ-protocol friendly, this holds true for the relations in all known constructions. Interestingly, a byproduct of this approach is that the update proofs for the underlying SNARK CRS become much more efficient to verify (and typically also much smaller). This improvement also carries over to the original LAMASSU framework [ARS20] and can be used to improve their CRS update proofs as well.

Since BB-LAMASSU is BB SE, it is also UC-secure and should therefore realize the NIZK ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$ of [Gro06]. However, so far this ignores the updatable CRS aspect. We recall that in our update proofs of the underlying CRS crs_{SNARK}, of the US, and of the EKU-PKE scheme, we use a FS/Fischlin/Unruh-transformed NIZK. UC however precludes the use of rewinding extractors (i.e., FS). Since we never need to extract from proofs of update correctness of crs_{SNARK} (the simulator always uses the other branch of the OR), that proof can use FS, but the other two parts must rely on the Fischlin or Unruh transforms, which provide straight-line extractors. To formally confirm this intuition, we introduce a new ideal functionality $\mathcal{F}_{\mathsf{up\text{-}CRS}}$ for the updatable CRS generation and then prove that BB-LAMASSU realizes the functionality $\mathcal{F}_{\mathsf{NIZK}}$ in the $\mathcal{F}_{\mathsf{up\text{-}CRS}}$-hybrid model.

2 Preliminaries

Let PPT denote probabilistic polynomial-time. Let $\lambda \in \mathbb{N}$ be the security parameter. All adversaries are stateful. By $y \leftarrow \mathcal{A}(\mathtt{x}; \omega)$ we mean that algorithm $\mathcal{A}$, given an input $\mathtt{x}$ and random coins ω, outputs y. We write $x \leftarrow\!\!\$\ \mathcal{D}$ to denote that x is sampled according to distribution $\mathcal{D}$ or uniformly randomly if $\mathcal{D}$ is a set. Let $\mathsf{RND}(\mathcal{A})$ denote the random tape of $\mathcal{A}$, and let $\omega \leftarrow\!\!\$\ \mathsf{RND}(\mathcal{A})$ denote the random choice of the random coins ω from $\mathsf{RND}(\mathcal{A})$. A PPT $\mathcal{A}$ is able to read only polynomially many (in security parameter λ) symbols of the random tape. We denote by $\mathsf{negl}(\lambda)$ an arbitrary negligible function and write $a \approx_\lambda b$ if $|a - b| \leq \mathsf{negl}(\lambda)$. A bilinear group generator $\mathsf{Pgen}(1^\lambda)$ returns $\mathsf{BG} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \bar{e})$, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are cyclic groups of prime order p and $\bar{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a non-degenerate efficiently computable bilinear map (pairing).

We recall the definitions of key-homomorphic signatures, Schnorr signatures, Σ-protocols, and the Fiat-Shamir, Fischlin, and Unruh transforms in Appendices A.1 to A.5.

Black-box constructions. We consider constructions to be *black-box* if they do not refer to the code of any cryptographic primitives they use, but rather depend only on the primitives' input/output behavior. We therefore call a NIZK extractor black-box if it does not take the adversary as input.

2.1 Non-Interactive Zero-Knowledge

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

We adopt the (SE) updatable NIZK definitions from [Gro16, GKM+18, ARS20]. We consider the updatable CRS setting, meaning 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 (or the initial creation).

In the following we provide a formal definition of an updatable NIZK.

An *updatable NIZK* $\Pi = (\mathsf{KGen}, \mathsf{Ucrs}, \mathsf{Vcrs}, \mathsf{P}, \mathsf{V})$ for $\mathcal{R}$ consists of the following PPT algorithms:

$\mathsf{KGen}_\mathsf{crs}(\mathcal{R})$: On input $\mathcal{R} \in \text{image}(\mathsf{RGen}(1^\lambda))$, outputs CRS $\mathtt{crs}$, a trapdoor $\mathtt{tc}$, and a proof ζ.

$\mathsf{Ucrs}(\mathtt{crs}, \zeta)$: On input $(\mathtt{crs}, \zeta)$ outputs $(\mathsf{up}_\mathtt{tc}, \mathtt{crs}_\mathsf{up}, \zeta_\mathsf{up})$ where $\mathsf{up}_\mathtt{tc}$ and $\mathtt{crs}_\mathsf{up}$ are the update trapdoor and the updated CRS respectively, and ζ_up is a proof for the correctness of the updating procedure.

$\mathsf{Vcrs}(\mathtt{crs}, \zeta)$: On input $(\mathtt{crs}, \zeta)$, returns either 0 (the CRS is ill-formed) or 1 (the CRS is well-formed).

$\mathsf{P}(\mathtt{crs}, \mathtt{x}, \mathtt{w})$: On input $(\mathtt{crs}, \mathtt{x}, \mathtt{w})$, where $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$, output a proof π.

$\mathsf{V}(\mathtt{crs}, \mathtt{x}, \pi)$: On input $(\mathtt{crs}, \mathtt{x}, \pi)$, returns either 0 (reject) or 1 (accept).

$\mathsf{Sim}(\mathtt{crs}, \mathtt{tc}_\mathsf{up}, \mathtt{x})$: On input $(\mathcal{R}, \mathsf{aux}_\mathcal{R}, \mathtt{crs}, \mathtt{tc}, \mathtt{x})$, outputs a simulated proof π. Here, $\mathtt{tc}_\mathsf{up} := \mathtt{tc} \odot \mathsf{up}_\mathtt{tc}$, where depending on the construction the operator $\odot$ might be different operations (like addition, multiplication).

Definition 1. *Let* $\Pi = (\mathsf{KGen}_\mathsf{crs}, \mathsf{Ucrs}, \mathsf{Vcrs}, \mathsf{P}, \mathsf{V})$ *be an updatable non-interactive argument for the relation* $\mathcal{R}$*. Then the argument* Π *is* updatable secure *if it satisfies the following properties:*

Updatable completeness. Π *is* complete for RGen *if for all* λ*,* $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$*, and PPT* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{l}\mathcal{R} \leftarrow \mathsf{RGen}(1^\lambda), (\mathtt{crs}, \mathtt{tc}, \zeta) \leftarrow \mathsf{A}(\mathcal{R}),\\ 1 \leftarrow \mathsf{Vcrs}(\mathtt{crs}, \zeta) :\\ \mathsf{V}(\mathtt{crs}, \mathtt{x}, \mathsf{P}(\mathcal{R}, \mathsf{aux}_\mathcal{R}, \mathtt{crs}, \mathtt{x}, \mathtt{w})) = 1\end{array}\right] = 1.$$

Where ζ *is a proof for the correctness of the generation (or updating) of the CRS.*

Updatable BB **simulation extractability.** Π *is* BB simulation extractable for RGen *if for every PPT* $\mathcal{A}$ *and any subverter* Z*, there exists a PPT extractor* Ext *such that*

$$\Pr\left[\begin{array}{l}\mathcal{R} \leftarrow \mathsf{RGen}(1^\lambda),\\ (\mathtt{crs}, \mathtt{tc} := (\mathtt{tc}_{\mathsf{sim}}, \mathtt{tc}_{\mathsf{ext}}), \zeta) \leftarrow \mathsf{KGen}_{\mathsf{crs}}(\mathcal{R}),\\ \omega_\mathsf{Z} \leftarrow\!\!\$\ \mathsf{RND}(\mathsf{Z}),\\ (\mathtt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}}, \mathsf{aux}_\mathsf{Z}) \leftarrow \mathsf{Z}(\mathtt{crs}, (\zeta_i)_{i=1}^{n}, \omega_\mathsf{Z}),\\ \textbf{if } \mathsf{Vcrs}(\mathtt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}}) = 0 \textbf{ then } \textbf{return } 0,\\ (\mathtt{x}, \pi) \leftarrow \mathcal{A}^{\mathsf{O}(\cdot)}(\mathcal{R}, \mathtt{crs}_{\mathsf{up}}, \mathtt{crs}, \mathsf{aux}_\mathsf{Z}),\\ \mathtt{w} \leftarrow \mathsf{Ext}(\mathcal{R}, \mathtt{crs}_{\mathsf{up}}, \mathtt{crs}; \mathtt{tc}_{\mathsf{ext}}) :\\ (\mathtt{x}, \pi) \notin \mathcal{Q} \wedge (\mathtt{x}, \mathtt{w}) \notin \mathcal{R} \wedge\\ \mathsf{V}(\mathtt{crs}_{\mathsf{up}}, \mathtt{x}, \pi) = 1\end{array}\right] \approx_\lambda 0.$$

Here $\mathsf{RND}(\mathsf{Z}) = \mathsf{RND}(\mathcal{A})$ *and* $(\zeta_i)_{i=1}^{n}$ *for* $n \in \mathbb{N}$ *is a series of proofs for the correctness of the updating procedure. The oracle* $\mathsf{O}(.)$ *represents two oracles* $\mathsf{O}_1(.)$ *and* $\mathsf{O}_2(.)$ *which return* $\pi := \mathsf{Sim}(\mathtt{crs}, \mathtt{tc}_{\mathsf{sim}}, \mathtt{x})$ *and* $\pi := \mathsf{Sim}(\mathtt{crs}_{\mathsf{up}}, \mathtt{tc}_{\mathsf{up,sim}}, \mathtt{x})$ *respectively.* $\mathsf{O}(.)$ *keeps track of all queried* $(\mathtt{x}, \pi)$ *via* $\mathcal{Q}$*. Note that* Z *can also first generate* $\mathtt{crs}$ *and then an honest updater updates it and outputs* $\mathtt{crs}_{\mathsf{up}}$*. In the latter case,* $\mathsf{O}(.) = \mathsf{O}_2(.)$*.*

Remark 1. We note that what we call simulation extractability is often called strong simulation extractability in the literature. Sometimes one encounters a relaxed form called weak simulation extractable, which only requires $\mathtt{x} \notin \mathcal{Q}$ *in the winning condition. We will make it explicit when we talk about this weak form.*

Notice that for the updatable ZK property, one must assume that at least one of the (possibly malicious) updaters does not communicate with the others. This guarantees ZK even if all updaters are malicious (i.e., in the split adversarial model where updating is done by two adversaries $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *who do not share their secret values (*$\mathtt{tc}_1$ *and* $\mathtt{tc}_2$*) with each other), since none of them has access to whole CRS trapdoor* $\mathtt{tc}$ *(containing both* $\mathtt{tc}_1$ *and* $\mathtt{tc}_2$*).*[8]

[8] Alternatively, with a slightly stronger assumption than the split adversarial model, one may simply assume that one of the updates is honestly done.

Π *is* statistically unbounded updatable ZK for RGen *[GKM+18], if for any PPT* Z *there exists a PPT* Ext*, such that for all* $\mathcal{R} \in \mathrm{im}(\mathsf{RGen}(1^\lambda))$*, and computationally unbounded* $\mathcal{A}$*,* $\varepsilon_0^{unb} \approx_\lambda \varepsilon_1^{unb}$*, where*

$$\varepsilon_b^{unb} = \Pr\left[\begin{array}{l}\omega_{\mathsf{Z}} \leftarrow_\$ \mathsf{RND}(\mathsf{Z}), (\mathtt{crs}, \zeta, \mathsf{aux}_{\mathsf{Z}}) \leftarrow \mathsf{Z}(\mathcal{R}, \omega_{\mathsf{Z}}),\\ \mathtt{tc} \leftarrow \mathsf{Ext}(\mathcal{R}, \mathsf{aux}, \mathtt{crs}):\\ \mathsf{Vcrs}(\mathtt{crs}, \zeta) = 1 \ \wedge \mathcal{A}^{\mathsf{O}_b(\cdot,\cdot)}(\mathcal{R}, \mathtt{crs}, \mathsf{aux}_{\mathsf{Z}}) = 1\end{array}\right].$$

Here $\mathsf{RND}(\mathsf{Z}) = \mathsf{RND}(\mathcal{A})$ *and the oracle* $\mathsf{O}_0(\mathtt{x}, \mathtt{w})$ *returns* $\perp$ *(reject) if* $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}$*, otherwise returning* $\mathsf{P}(\mathtt{crs}, \mathtt{x}, \mathtt{w})$*. Similarly,* $\mathsf{O}_1(\mathtt{x}, \mathtt{w})$ *returns* $\perp$ *(reject) if* $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}$*, and otherwise it returns* $\mathsf{Sim}(\mathtt{crs}, \mathtt{tc}, \mathtt{x})$*.* Π *is* perfectly unbounded updatable ZK for RGen *if* $\varepsilon_0^{unb} = \varepsilon_1^{unb}$*.*

In the above, aux is some auxiliary information that depends on the BB-extraction technique (like rewinding or straight-line extraction).

2.2 Updatable Signatures

We recall the notion of updatable signatures from [ARS20] below and include their relevant properties (updatable correctness, updatable strong key hiding, and updatable EUF-CMA) in Appendix A.6. Going forward, let μ be an efficiently computable map from the private key space $(\mathbb{H}, +)$ to the public key space $(\mathbb{E}, \cdot)$ such that for all $\mathsf{csk}, \mathsf{csk}' \in \mathbb{H}$, $\mu(\mathsf{csk} + \mathsf{csk}') = \mu(\mathsf{csk}) \cdot \mu(\mathsf{csk}')$ and for all $(\mathsf{csk}, \mathsf{cpk}) \leftarrow \mathsf{KGen}(1^\lambda)$, $\mathsf{cpk} = \mu(\mathsf{csk})$ (cf. [ARS20, Def. 3]). A key functionality of such schemes is that now signatures produced under any csk can, with the knowledge of csk', be updated to signatures valid under verification key $\mu(\mathsf{csk} + \mathsf{csk}')$.

Updatable zero-knowledge. Definition 2 (Updatable signature schemes). *An updatable signature scheme* $\Sigma = (\mathsf{KGen}, \mathsf{Upk}, \mathsf{Vpk}, \mathsf{Sign}, \mathsf{Verify})$ *is a key-homomorphic [ARS20, Def. 4] signature scheme consisting of the following PPT algorithms:*

$\mathsf{KGen}(1^\lambda)$: *Given a security parameter* λ*, output a signing key* csk*, a verification key* cpk*, a proof* ζ*, and a message space* $\mathcal{M}$*.*

$\mathsf{Upk}(\mathsf{cpk})$: *Given a verification key* cpk*, output an updated verification key* $\mathsf{cpk}_{\mathsf{up}}$ *with associated secret key updating information* $\mathsf{up}_{\mathsf{csk}}$ *and a proof* ζ*. The updated signing key is then* $\mathsf{csk}_{\mathsf{up}} := \mathsf{csk} + \mathsf{up}_{\mathsf{csk}}$*.*

$\mathsf{Vpk}(\mathsf{cpk}, \mathsf{cpk}_{\mathsf{up}}, \zeta)$: *Given a verification key* cpk*, a potentially updated verification key* $\mathsf{cpk}_{\mathsf{up}}$*, and a proof* ζ*, check if* $\mathsf{cpk}_{\mathsf{up}}$ *has been updated correctly. When verifying the original* cpk*, we write* $\mathsf{Vpk}(\mathsf{cpk}, \zeta)$*.*

$\mathsf{Sign}(\mathsf{csk}_{\mathsf{up}}, m)$: *Given a potentially updated secret key* $\mathsf{csk}_{\mathsf{up}}$ *and a message* $m \in \mathcal{M}$*, output a signature* σ*.*

$\mathsf{Verify}(\mathsf{cpk}_{\mathsf{up}}, m, \sigma)$: *Given a potentially updated public key* $\mathsf{cpk}_{\mathsf{up}}$*, a message* $m \in \mathcal{M}$ *and a signature* σ*, output a bit* $b \in \{0, 1\}$*.*

Example of Updatable Signatures. Lamassu [ARS20] uses an updatable signature construction based on the Schnorr signature scheme. Their scheme is instantiated in a bilinear group and uses the pairing to check updates, with a knowledge assumption for extraction. For consistency with our construction of key-updatable public-key encryption (EKU-PKE) in Section 3, which proves update validity via NIZKs, we use the same approach here and with an updatable Schnorr construction in a prime-order multiplicative group $\mathbb{G}$ with generator g. Let (P, V) be a simulation-extractable NIZK for the relation $\mathcal{R} = \{((\mathsf{cpk}, \mathsf{cpk}_{\mathsf{up}}, g), x') : \mathsf{cpk}_{\mathsf{up}} = \mathsf{cpk} \cdot g^{x'}\}$. We recall Schnorr signatures in Appendix A.2 and only give the key update algorithms here:

$\mathsf{Upk}(\mathsf{cpk})$: Set $\mathsf{up}_{\mathsf{csk}} := x' \leftarrow\$ \mathbb{Z}_p$, $\mathsf{cpk}_{\mathsf{up}} := \mathsf{cpk} \cdot g^{x'}$, $\zeta_{\mathsf{up}} \leftarrow \mathsf{P}((\mathsf{cpk}, \mathsf{cpk}_{\mathsf{up}}, g), \mathsf{up}_{\mathsf{cpk}})$ and return $(\mathsf{up}_{\mathsf{csk}}, \mathsf{cpk}_{\mathsf{up}}, \zeta_{\mathsf{up}})$.

$\mathsf{Vpk}(\mathsf{cpk}, \mathsf{cpk}_{\mathsf{up}}, \zeta_{\mathsf{up}})$: Return $\mathsf{V}(\mathtt{crs}, (\mathsf{cpk}, \mathsf{cpk}_{\mathsf{up}}, g), \zeta_{\mathsf{up}})$.

Finally, we present an efficient extractor $\mathsf{Ext}_{\mathsf{Z}}$. If Vpk returns 1 on any input $(\mathsf{cpk}, \mathsf{cpk}_{\mathsf{up}}, \zeta_{\mathsf{up}})$, by the simulation extractability of the NIZK we have an extractor that extracts $\mathsf{up}_{\mathsf{csk}} := x'$ from ζ_{up} s.t. $\mathsf{csk}_{\mathsf{up}} = \mathsf{csk} + \mathsf{up}_{\mathsf{csk}}$ and $\mathsf{cpk}_{\mathsf{up}} = \mathsf{cpk} \cdot g^{\mathsf{up}_{\mathsf{csk}}}$.

2.3 The Lamassu Compiler

The Lamassu [ARS20] compiler lifts any CRS-based NIZK that is knowledge sound (e.g., a SNARK) to a non-BB simulation-extractable version while maintaining compatibility with an updatable CRS. Roughly speaking, Lamassu uses a combination of an updatable EUF-CMA secure signature scheme Σ and a strongly unforgeable one-time signature (sOTS) scheme Σ_{OT} (e.g., Groth's sOTS [Gro06] or Schnorr) together with the folklore OR-trick to obtain simulation-extractability, i.e., achieve non-malleability.

In more detail, during proof generation, the prover uses a fresh keypair $\mathsf{csk}_o, \mathsf{cpk}_o$ of Σ to compute a signature which certifies the public key of an sOTS. Then, it uses the secret key of the sOTS to sign the parts of the proof which must be non-malleable. Crucially, the former signature is provided in plain and thus one does *not* need to encrypt it or prove that it verifies under some verification key in the CRS (e.g., as in [Gro06]). Consequently, the OR clause of the lifted language only requires the shift (this is the trapdoor of the CRS) which adapts signatures valid under the freshly sampled cpk_o to ones valid under the verification key cpk in the CRS (equivalent to the difference $\mathsf{csk} - \mathsf{csk}_o$; the technique was introduced in [DS19]). As it turns out, this feature lays the foundation to support updatability.

Now, given any language $\mathcal{L}$ with NP relation $\mathcal{R}_{\mathcal{L}}$, the language obtained via the compiler is $\mathcal{L}_{\mathsf{lamassu}}$ s.t. $\{\mathtt{x}_{\mathsf{lamassu}} := (\mathtt{x}, \mathsf{cpk}, \mathsf{cpk}_o), \mathtt{w}_{\mathsf{lamassu}} := (\mathtt{w}, \mathsf{csk} - \mathsf{csk}_o)\} \in \mathcal{R}_{\mathcal{L}_{\mathsf{lamassu}}}$ iff:

$$((\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathcal{L}} \vee \mathsf{cpk} = \mathsf{cpk}_o \cdot \mu(\mathsf{csk} - \mathsf{csk}_o)).$$

A proof for $\mathtt{x} \in \mathcal{L}$ is easy to compute given $\mathtt{w}$ such that $(\mathtt{x}, \mathtt{w}) \in \mathcal{L}$, in which case the prover does not need to satisfy the second branch of the OR statement but instead computes the signatures under newly generated keys. To simulate proofs, however, one can set up the CRS so that csk corresponding to cpk is known, compute the "shift amount" $\mathsf{csk} - \mathsf{csk}_o$, and use it as a satisfying witness for the second branch of the OR.

The following theorem captures the properties of LAMASSU applied to any knowledge sound NIZK with updatable CRS:

Theorem 1 ([ARS20]). *Assume that the underlying updatable NIZK (SNARK) scheme satisfies perfect completeness, updatable zero-knowledge, and updatable knowledge soundness. Let Σ be a EUF-CMA secure adaptable key-homomorphic signature scheme and Σ_{OT} a strongly-unforgeable one-time signature scheme. The lifted NIZK construction is a zero-knowledge proof system satisfying perfect completeness, updatable zero-knowledge, and updatable simulation extractability.*

3 Extractable Key-Updatable PKE

Now we introduce a new primitive called extractable key-updatable PKE (EKU-PKE), which is a PKE scheme that allows one to update the keys and provide extractability key update proofs. We will need this primitive to enable encryption of the witness in our NIZK construction (for simulation extractability) in a way that is compatible with an updatable CRS.

There are approaches in the literature for key-updatability [PR18, JMM19] which do not consider extractability. A recent work by Dodis *et al.* [DKW21] additionally considers extractability. While this notion is very close to ours, it only considers possibly dishonest updates. In our case, however, security should hold as long as either the initial key generation or at least one of the updates is performed honestly. We note that as done by Groth *et al.* [GKM+18] for updatable CRS (Lemma 6), we model only a single update, since a single adversarial update implies EKU-PKEs with arbitrarily many adversarial updates.

3.1 Definition and Security

We call a PKE scheme UP *extractable key-updatable (EKU-PKE)* if the key generation is run by an *updatable key generation* scheme and the correctness and black-box extraction properties of the scheme hold for all updated keys that pass verification:

Definition 3 (Updatable key generation). *An updatable key generation scheme* $\mathsf{UP.KGen} = (\mathsf{KGen}, \mathsf{Upk}, \mathsf{Vpk})$ *consists of the following PPT algorithms:*

$\mathsf{KGen}(1^\lambda)$: *Given a security parameter λ, output a secret key* sk*, a public key* pk *and a proof ζ.*

$\mathsf{Upk}(\mathsf{pk}, (\zeta_i)_{i=1}^n)$: *Given a public key* pk *and update proofs for* pk*, output an updated public key* $\mathsf{pk}_{\mathsf{up}}$ *with associated secret key updating information* $\mathsf{up}_{\mathsf{sk}}$ *and a proof* ζ_{up}*.*

$\mathsf{Vpk}(\mathsf{pk}_{\mathsf{up}}, (\zeta_i)_{i=1}^n)$: *Given a potentially updated public key* $\mathsf{pk}_{\mathsf{up}}$ *and a list of update proofs* ζ_i*, output a bit* b *indicating acceptance* $(b = 1)$ *or rejection* $(b = 0)$.

We note that in general the updated $\mathsf{sk}_{\mathsf{up}}$ equals $\mathsf{sk} \odot \mathsf{up}_{\mathsf{sk}}$, where depending on the scheme the operator $\odot$ might represent different operations (e.g., addition, multiplication). For our instantiation later we use multiplication.

Definition 4 ((Perfect) Updatable key correctness). *The* (perfect) updatable key correctness *property requires the following three conditions:*

(i) *for any* $(\mathsf{sk}, \mathsf{pk}, \zeta) \leftarrow \mathsf{KGen}(1^\lambda)$*:* $\mathsf{Vpk}(\mathsf{pk}, \zeta) = 1$

(ii) *for any* $(\mathsf{sk}, \mathsf{pk}, \zeta) \leftarrow \mathsf{KGen}(1^\lambda)$ *and* $(\mathsf{pk}_{\mathsf{up}}, (\zeta_i)_{i=1}^{n+1})$ *such that* $\mathsf{Vpk}(\mathsf{pk}_{\mathsf{up}}, (\zeta_i)_{i=1}^{n+1}) = 1$*, the distributions of* pk *and* $\mathsf{pk}_{\mathsf{up}}$ *are (perfectly) indistinguishable.*

(iii) *for any* $(\mathsf{pk}, (\zeta_i)_{i=1}^n)$ *such that* $\mathsf{Vpk}(\mathsf{pk}, (\zeta_i)_{i=1}^n) = 1$ *and* $(\mathsf{pk}_{\mathsf{up}}, \zeta_{n+1}) \leftarrow \mathsf{Upk}(\mathsf{pk}, (\zeta_i)_{i=1}^n)$*, we have that* $\mathsf{Vpk}(\mathsf{pk}_{\mathsf{up}}, (\zeta_i)_{i=1}^{n+1}) = 1$.

Definition 5 (Updatable black-box extraction). *An updatable key generation scheme* $\mathsf{UP.KGen} = (\mathsf{KGen}, \mathsf{Upk}, \mathsf{Vpk})$ *is black-box extractable if there exists an efficient extractor* Ext *such that for any* $(\mathsf{sk}, \mathsf{pk}, \zeta) \in \mathrm{image}(\mathsf{KGen}(1^\lambda))$ *and any* $(\mathsf{up}_{\mathsf{sk}}, \mathsf{pk}_{\mathsf{up}}, \zeta_{\mathsf{up}}) \in \mathrm{image}(\mathsf{Upk}(\mathsf{pk}, \zeta))$ *where both* $\mathsf{Vpk}(\mathsf{pk}, \zeta) = 1$ *and* $\mathsf{Vpk}(\mathsf{pk}_{\mathsf{up}}, \zeta_{\mathsf{up}}) = 1$ *hold, then for* $\mathsf{sk}_{\mathsf{up}} \leftarrow \mathsf{Ext}(\zeta, \mathsf{pk}, \zeta_{\mathsf{up}}, \mathsf{pk}_{\mathsf{up}})$ *we have that* $(\mathsf{sk}_{\mathsf{up}}, \mathsf{pk}_{\mathsf{up}}, \cdot) \in \mathrm{image}(\mathsf{KGen}(1^\lambda))$.

Now, with the above properties we can take any PKE scheme that satisfies updatable key generation and convert it into an EKU-PKE scheme.

Security properties. Let UP be a EKU-PKE scheme. We now define *key-updatable IND-CPA* security for the EKU-PKE scheme UP and note that one can analogously define *key-updatable IND-PCA* and *key-updatable IND-CCA* security:

```
$\mathsf{Exp}^{\mathsf{up\text{-}cpa}}_{\mathsf{UP},\mathcal{A}}(\lambda)$
$(\mathsf{sk}, \mathsf{pk}, \zeta) \leftarrow \mathsf{UP.KGen}(1^\lambda)$;
$((\mathsf{pk}_{\mathsf{up}}, \zeta_{\mathsf{up}}), m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}, \zeta); b \leftarrow\$ \{0,1\}$;
$r \leftarrow\$ \mathsf{RND}(\mathsf{UP}); \mathsf{c}^* \leftarrow \mathsf{UP.Enc}(\mathsf{pk}_{\mathsf{up}}, m_b; r); b' \leftarrow \mathcal{A}(\mathsf{c}^*)$;
$\textbf{return } (b = b') \wedge \mathsf{UP.Vpk}(\mathsf{pk}_{\mathsf{up}}, \{\zeta, \zeta_{\mathsf{up}}\})$;
```

Note that alternatively, $\mathcal{A}$ can generate the initial pk, which is then updated by an honest updater Upk outputting $\mathsf{pk}_{\mathsf{up}}$, $\mathsf{up}_{\mathsf{sk}}$, and the proof ζ_{up}. In that case, we require that $\mathsf{UP.Vpk}(\mathsf{pk}, \zeta)$ holds.

Definition 6 (Key-updatable IND-CPA security). UP *is* key-updatable IND-CPA secure *if for any PPT adversary* $\mathcal{A}$*,*

$$\mathsf{Adv}^{\mathsf{up\text{-}cpa}}_{\mathsf{UP},\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{up\text{-}cpa}}_{\mathsf{UP},\mathcal{A}}(\lambda) = 1] - 1/2| \approx_\lambda 0.$$

3.2 Instantiation

We present a construction of an EKU-PKE over a prime-order group $(\mathbb{G}, \mathsf{g}, \mathsf{p})$ based on the ElGamal PKE scheme. Thus, our setup outputs only publicly verifiable parameters and does not need to be run by a trusted party. Let ZK be in the set $\{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$ for the relation $\mathcal{R}(\mathtt{x}_{\mathsf{ZK}}, \mathtt{w}_{\mathsf{ZK}})$ where $\mathtt{x}_{\mathsf{ZK}} := (\mathsf{pk}', \mathsf{pk})$, $\mathtt{w}_{\mathsf{ZK}} := w$ such that $\mathsf{pk}' = \mathsf{pk}^w$. The full construction is as follows:

$\mathsf{KGen}(1^\lambda)$: Given a security parameter λ, outputs a secret key sk, public key $\mathsf{pk} := \mathsf{g}^{\mathsf{sk}}$, and its corresponding proof $\zeta_1 := \pi_{\mathsf{ZK}}$ for $(\mathsf{pk}, \mathsf{g})$ and witness sk.

$\mathsf{Upk}(\mathsf{pk}, (\zeta_i)_{i=1}^n)$: Output an updated public key $\mathsf{pk}_{\mathsf{up}} := \mathsf{pk}^{\mathsf{up}_{\mathsf{sk}}}$ with associated secret key updating information $\mathsf{up}_{\mathsf{sk}}$ and a proof ζ_{n+1} that $((\mathsf{pk}_{\mathsf{up}}, \mathsf{pk}), \mathsf{up}_{\mathsf{sk}}) \in \mathcal{R}$.

$\mathsf{Vpk}(\mathsf{pk}, \mathsf{pk}_{\mathsf{up}}, (\zeta_i)_{i=1}^n)$: Given a verification key pk, a potentially updated verification key $\mathsf{pk}_{\mathsf{up}}$, and the proof ζ_{up}, check if $\mathsf{pk}_{\mathsf{up}}$ has been updated correctly by running $\mathsf{V}_{\mathsf{ZK}}((\mathsf{pk}_{\mathsf{up}}, \mathsf{pk}), \zeta_{\mathsf{up}})$. When verifying the original pk, we write $\mathsf{Vpk}(\mathsf{pk}, \zeta)$.

$\mathsf{Enc}(\mathsf{pk}_{\mathsf{up}}, \mathsf{M}; r)$: Given a potentially updated public key $\mathsf{pk}_{\mathsf{up}}$, a message $\mathsf{M} \in \mathbb{G}$, and randomness r, output the ciphertext $\mathsf{c} := (\mathsf{g}^r, \mathsf{M} \cdot \mathsf{pk}_{\mathsf{up}}^r)$.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{up}}, \mathsf{c})$: Given a potentially updated secret key $\mathsf{sk}_{\mathsf{up}}$ and the ciphertext c, output the message $\mathsf{M} := \mathsf{c}_2 / \mathsf{c}_1^{\mathsf{sk}_{\mathsf{up}}}$.

Theorem 2. *Let* $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$ *be a non-interactive proof of knowledge with black-box extraction for the relation* $\mathcal{R}(\mathtt{x}_{\mathsf{ZK}}, \mathtt{w}_{\mathsf{ZK}})$ *and suppose that the* DDH *assumption holds in* $(\mathbb{G}, \mathsf{g}, p)$*. Then the above scheme is a extractable key-updatable PKE.*

Proof. Property (i) of updatable key correctness is straightforward by construction and the completeness of ZK. Similarly, property (ii) follows by construction and by soundness of ZK. We reduce property (iii) to the soundness of the ZK argument. Let $\mathcal{A}$ be the adversary against (iii). Let $\mathcal{B}$ be an adversary against the soundness of ZK with relation $\mathcal{R}(\mathtt{x}_{\mathsf{ZK}}, \mathtt{w}_{\mathsf{ZK}})$ and language $\mathcal{L}_{\mathsf{ZK}}$ with $\mathtt{x}_{\mathsf{ZK}} = (\mathsf{pk}_{\mathsf{up}}, \mathsf{pk})$, $\mathtt{w}_{\mathsf{ZK}} = \mathsf{up}_{\mathsf{sk}}$ such that $\mathsf{Vpk}(\mathsf{pk}, \mathsf{pk}'_{\mathsf{up}}, \zeta'_{\mathsf{up}}) = 1$. $\mathcal{B}$ picks $\mathsf{pk} \leftarrow\!\!\$\ \mathbb{G}$, runs the adversary $\mathcal{A}(\mathsf{pk})$, and obtains $\mathsf{pk}'_{\mathsf{up}}$ with $\zeta'_{\mathsf{up}} = \pi_{\mathsf{ZK}}$ such that $\mathsf{Vpk}(\mathsf{pk}, \mathsf{pk}'_{\mathsf{up}}, \zeta'_{\mathsf{up}}) = 1$ and $(\mathsf{pk}'_{\mathsf{up}}, \mathsf{pk}) \notin \mathcal{L}_{\mathsf{ZK}}$. Therefore, the reduction has the same (non-negligible) advantage in the ZK's soundness game as $\mathcal{A}$ has in the property (iii) game.

Finally, updatable BB-extractability follows directly from the BB-extractability of ZK.

We note that in general, the properties of the NIZK extractor directly translate to the UP extractor, i.e., if ZK provides a straight-line extractor, then UP is also straight-line extractable. Additionally, in Lemma 1, we prove that this construction is key-updatable IND-CPA secure.

Lemma 1. *Let* $(\mathbb{G}, \mathsf{g}, \mathsf{p})$ *be a prime-order group and suppose the* DDH *assumption holds. Let* $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$ *be a non-interactive proof of knowledge with black-box extraction. Then the above scheme is key-updatable IND-CPA secure.*

Proof. We prove IND-CPA security with a sequence of games starting from the standard IND-CPA game, where the adversary has no control over the key, and ending with key-updatable IND-CPA, where $\mathcal{A}$ is able to update pk. The detailed games are as follows:

Game_1 : This is the original IND-CPA experiment.

Game_2 : This game is the same as Game_1, with the difference that $\mathcal{A}$ receives the proof ζ_{ZK} related to the well-formedness of the pk as depicted in Definition 6.

$\mathsf{Game}_1 \rightarrow \mathsf{Game}_2$: This is straightforward from the zero-knowledge property of the NIZK and so the two games are indistinguishable, with $\Pr[\mathsf{Game}_1] \leq \Pr[\mathsf{Game}_2] + \mathsf{negl}(\lambda)$.

Game_3 : This game is the same as Game_2, with the difference that $\mathcal{A}$ updates pk and so she receives the challenge ciphertext c^* under the updated $\mathsf{pk}_{\mathsf{up}}$ as depicted in Definition 6.

$\mathsf{Game}_2 \rightarrow \mathsf{Game}_3$: This is straightforward from property (ii) of updatable key correctness, which states that if Vpk outputs 1, we have that the public keys pk and $\mathsf{pk}_{\mathsf{up}}$ are indistinguishable from each other. This guarantees that c^* has the same distribution under both pk and the updated $\mathsf{pk}_{\mathsf{up}}$. Thus we have $\Pr[\mathsf{Game}_2] \leq \Pr[\mathsf{Game}_3] + \mathsf{negl}(\lambda)$.

EKU-PKE for arbitrary message spaces. For encrypting large witnesses, an EKU-PKE which supports the encryption of arbitrary bit strings is required. As the updatability notions do not require any specific properties on the ciphertexts, a key-updatable PKE for arbitrary message spaces can be obtained by following the hybrid approach [CS03]. Combining an IND-CPA-secure EKU-PKE with an IND-CPA-secure symmetric encryption scheme thus yields a IND-CPA-secure EKU-PKE for arbitrary message spaces.

4 UC-Secure Updatable Circuit-Succinct NIZK

In this section, we present a general framework for UC-secure circuit-succinct NIZKs with a weaker trusted setup (i.e., updatable CRS) using a black-box EKU-PKE defined in Section 3. We recall that in the updatable CRS setting [GKM+18] everyone can update a CRS, removing the trust in the CRS generator at both the prover and verifier side as long either the generation of the CRS or any of its updates are performed honestly (e.g., by the verifier).

Recall that the C∅C∅ framework [KZM+15] lifts any NIZK to a UC-secure NIZK in the CRS model. But UC-secure NIZKs with reduced trust in the CRS generation, e.g., via updatable CRS, are still an open problem. Indeed, to achieve UC-secure NIZKs in such a setting, one needs to guarantee SE for the updatable NIZKs in a *black-box* way. Recall that SE requires (knowledge) soundness to hold even if an adversary can see an arbitrary number of simulated proofs, which they can adaptively obtain on statements of their choice (see Section 2.1 for a rigorous definition).

The LAMASSU framework [ARS20] (see Section 2.3) transforms any updatable SNARK (or NIZK) to a *non*-black-box SE updatable SNARK (resp. NIZK) under some non-falsifiable assumption. More precisely, in the lifted SNARK (NIZK), both the *zero-knowledge* and *SE proofs* are based on non-falsifiable assumptions. It is known that UC-security can not be achieved for a construction under non-falsifiable assumptions.

In this section, we start from the LAMASSU construction and tackle the aforementioned hurdles to UC-security by converting this framework to a black-box version. Then, for the first time, we show how one can achieve UC-secure updatable circuit-succinct NIZKs.

4.1 Black-Box SE Updatable Circuit-Succinct NIZKs

Now, we introduce a framework for black-box SE updatable circuit-succinct NIZKs that builds upon and extends the LAMASSU compiler. Before describing the intuition of our construction, we recall some notation and primitives used in the construction.

- An updatable SNARK or NIZK Π in the CRS model (e.g., Groth *et al.* [GKM+18])
- A BB-extractable EKU-PKE UP (Section 3)
- A BB-extractable updatable signature Σ (Section 2.2)
- A BB-extractable non-interactive proof of knowledge (knowledge sound NIZK) ZK (either FS [FS87], $\mathsf{Fischlin}$ [Fis05] or Unruh [Unr15])

Intuition. We can divide our approach into two parts:

From non-BB *to* BB *extractable updatable NIZK.* In order to satisfy *black-box* extraction, we start with LAMASSU and add the public key $\mathsf{UP.pk}$ of an IND-CPA secure EKU-PKE UP (defined in Section 3) to the CRS. This will be used to encrypt the witness, giving us a black-box extractable version of LAMASSU.

From BB *extraction to* BB *SE updatable NIZK.* To achieve a UC-secure version of LAMASSU, we additionally need to enable proof simulation which is compatible with BB extraction. Thus, we replace the updatable signature of the LAMASSU compiler with a BB-extractable updatable signature Σ (defined in Section 2.2).

Finally, in order to satisfy BB extraction for updates to the lifted CRS (which now includes the underlying CRS, the public key of UP, and the public key of Σ), we require a non-interactive proof of knowledge $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$ that the update is correctly done. This gives us a fully black-box version of the LAMASSU compiler.

Remark 2. To achieve more efficient BB updatable SE NIZKs, we may use $\mathsf{ZK} = \mathsf{FS}$ instead of $\mathsf{Fischlin}$ or Unruh. This construction might be of independent interest for applications of BB-updatable SE NIZKs, but it is not UC-friendly due to the use of rewinding in the extraction phase of FS. We will discuss this more in Section 4.2.

We present the full construction of black-box SE updatable (circuit-succinct) NIZKs in Fig. 2, where $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$ is a BB-extractable NIZK that the update is correctly done. This is in contrast to [GKM+18] and [Lip20] as well as the LAMASSU framework, which reveal some intermediate shares in both groups $\mathbb{G}_1$ and $\mathbb{G}_2$ to construct a CRS verification that the update is correctly done under some non-falsifiable assumptions.

Although our framework, like LAMASSU, does not itself add updatability to the underlying NIZK, one can use techniques similar to those in [GKM+18] (for SNARKs) or [Lip20] (for QA-NIZKs) to transfer any CRS-based NIZK to the updatable setting. Then, starting from the CRS-updatable NIZK, BB-LAMASSU adds black-box SE. Specifically, given any language $\mathcal{L}$ with NP relation $\mathcal{R}_{\mathcal{L}}$, the language obtained via the compiler is $\mathcal{L}'$ s.t. $\{\mathtt{x}' := (\mathtt{x}, \mathsf{c}, \mathsf{pk}_l), \mathtt{w}' := (\mathtt{w}, \omega, \mathsf{csk} - \mathsf{sk}_l)\} \in \mathcal{R}_{\mathcal{L}'}$ iff:

$$\mathsf{c} = \mathsf{UP.Enc}(\mathsf{pk}_{\mathsf{up}}, \mathtt{w}; \omega) \wedge$$
$$((\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathcal{L}} \vee \mathsf{cpk} = \mathsf{pk}_l \cdot \mu(\mathsf{csk} - \mathsf{sk}_l))$$

where cpk is the public key of Σ included in the CRS.

Theorem 3. *Let Π be a NIZK (SNARK) scheme satisfying perfect completeness, computational updatable zero-knowledge, and computational updatable (optionally knowledge) soundness,* UP *an EKU-PKE scheme with message space $\mathcal{M}$ satisfying IND-CPA security and perfect correctness, Σ an EUF-CMA-secure updatable signature scheme, and Σ_{OT} a one-time signature scheme with strong unforgeability. Let $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$ be a non-interactive proof of knowledge with* BB *extraction. Then the construction in Fig. 2 is a NIZK satisfying perfect completeness, updatable zero-knowledge, and* BB *updatable simulation extractability.*

Proof. We follow the outline of the proof of (non-black-box) LAMASSU [ARS20, Thm. 4].

(i: Completeness): This is straightforward from the construction of BB SE updatable NIZKs (SNARKs) in Fig. 2.

If $((\mathtt{crs}, (\zeta_i)_{i=1}^n), \mathtt{x}, \mathtt{w}) \leftarrow \mathcal{A}(1^\lambda)$ and $\mathsf{Vcrs}(\mathtt{crs}, (\zeta_i)_{i=1}^n) = 1 \wedge (\mathtt{x}, \mathtt{w}) \in \mathcal{R}$, then $\mathsf{V}(\mathtt{crs}, \mathtt{x}, \mathsf{P}(\mathtt{crs}, \mathtt{x}, \mathtt{w})) = 1$.

(ii: Updatable zero-knowledge): Underlying the (rewinding or straight-line) extraction property of the ζ_{ZK} suppose that there exists a PPT malicious subverter Z that takes $\mathtt{crs} = (\mathtt{crs}_\Pi, \mathsf{cpk}, \mathsf{pk})$ and $\zeta = (\zeta_{\mathsf{ZK},\Pi}, \zeta_{\mathsf{ZK},\mathsf{cpk}}, \zeta_{\mathsf{ZK},\mathsf{pk}})$ as input and outputs $\mathtt{crs}_{\mathsf{up}} = (\mathtt{crs}_{\Pi,\mathsf{up}}, \mathsf{cpk}_{\mathsf{up}}, \mathsf{pk}_{\mathsf{up}})$ as well as $\zeta_{\mathsf{up}} = (\zeta_{\mathsf{ZK},\Pi,\mathsf{up},i}, \zeta_{\mathsf{ZK},\mathsf{cpk}_{\mathsf{up}},i}, \zeta_{\mathsf{ZK},\mathsf{pk}_{\mathsf{up}},i})_{i=1}^n$ such that $\mathsf{Vcrs}(\mathtt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}}) = 1$ and more precisely $\mathsf{Vpk}(\mathsf{cpk}_{\mathsf{up}}, (\zeta_{\mathsf{ZK},\mathsf{cpk}_{\mathsf{up}},i})_{i=1}^n) = 1$ holds with non-negligible probability.

Then, by using the ζ_{ZK} extractor $\mathsf{Ext}_{\mathsf{ZK}}$, given the statement $\mathtt{x}'$ of the language $\mathcal{L}'$ (more precisely, given $\mathsf{cpk}_{\mathsf{up}}$ of the signature) and the proofs $(\zeta_{\mathsf{ZK},\mathsf{cpk}_{\mathsf{up}},i})_{i=1}^n$ as input, we can output $\mathsf{csk}_{\mathsf{up}}$.[9] For this case $\mathcal{A}$ is the adversary from Fig. 3.

[9] For example for the $\mathsf{Fischlin}$ extractor $\mathsf{Ext}_{\mathsf{Fischlin}}$, given the statement $\mathtt{x}'$ of the language $\mathcal{L}'$, the proofs $(\zeta_{\mathsf{Fischlin},\mathsf{cpk}_{\mathsf{up}},i})_{i=1}^n$, and the list of queries and answers of $Q_H(\mathsf{Z})$ (related to the trapdoor extraction in [Fis05, Theorem 2]) as input, one can recover $\mathsf{csk}_{\mathsf{up}}$.

```
$\mathsf{KGen_{crs}}(\mathcal{R}, \mathsf{aux}_\mathcal{R})$
    - $(\mathtt{crs}_\Pi, \mathtt{tc}_\Pi, \zeta_{\mathsf{ZK},\Pi}) \leftarrow \Pi.\mathsf{KGen}(\mathcal{R}, \mathsf{aux}_\mathcal{R})$;
    - $(\mathsf{csk}, \mathsf{cpk}, \zeta_{\mathsf{ZK,cpk}}) \leftarrow \Sigma.\mathsf{KGen}(1^\lambda)$;
    - $(\mathsf{sk}, \mathsf{pk}, \zeta_{\mathsf{ZK,pk}}) \leftarrow \mathsf{UP.KGen}(1^\lambda)$;$\mathtt{crs} := (\mathtt{crs}_\Pi, \mathsf{cpk}, \mathsf{pk})$;
    - $\mathtt{tc} := (\mathtt{tc}_\Pi, \mathsf{csk}, \mathsf{sk}); \zeta := (\zeta_{\mathsf{ZK},\Pi}, \zeta_{\mathsf{ZK,cpk}}, \zeta_{\mathsf{ZK,pk}})$
    - **return** $(\mathtt{crs}, \mathtt{tc}, \zeta)$;
$\mathsf{Ucrs}(\mathtt{crs}, (\zeta_i)_{i=1}^n)$
    - $(\mathtt{tc}_{\Pi,\mathsf{up}}, \mathtt{crs}_{\Pi,\mathsf{up}}, \zeta_{\mathsf{ZK},\Pi,\mathsf{up}}) \leftarrow \Pi.\mathsf{Ucrs}(\mathtt{crs}_\Pi, (\zeta_{\mathsf{ZK},\Pi,i})_{i=1}^n)$;
    - $(\mathsf{up_{csk}}, \mathsf{cpk_{up}}, \zeta_{\mathsf{ZK,cpk_{up}}}) \leftarrow \Sigma.\mathsf{Upk}(\mathsf{cpk}, (\zeta_{\mathsf{ZK,cpk},i})_{i=1}^n)$;
    - $(\mathsf{up_{sk}}, \mathsf{pk_{up}}, \zeta_{\mathsf{ZK,pk_{up}}}) \leftarrow \mathsf{UP.Upk}(\mathsf{pk}, (\zeta_{\mathsf{ZK,pk},i})_{i=1}^n)$;
    - $\zeta_{\mathsf{up}} := (\zeta_{\mathsf{ZK},\Pi,\mathsf{up}}, \zeta_{\mathsf{ZK,cpk_{up}}}, \zeta_{\mathsf{ZK,pk_{up}}})$;
    - **return** $(\mathtt{crs}_{\mathsf{up}} := (\mathtt{crs}_{\Pi,\mathsf{up}}, \mathsf{cpk_{up}}, \mathsf{pk_{up}}), \zeta_{\mathsf{up}})$;
$\mathsf{Vcrs}(\mathtt{crs}, (\zeta_i)_{i=1}^n)$
    - **if** $\Pi.\mathsf{Vcrs}(\mathtt{crs}_\Pi, (\zeta_{\mathsf{ZK},\Pi,i})_{i=1}^n) = 1 \wedge$
      $\Sigma.\mathsf{Vcpk}(\mathsf{cpk}, (\zeta_{\mathsf{ZK,cpk},i})_{i=1}^n) = 1 \wedge$
      $\mathsf{UP.Vpk}(\mathsf{pk}, (\zeta_{\mathsf{ZK,pk},i})_{i=1}^n) = 1$ **then return** $1$; **else return** $0$;
$\mathsf{P}(\mathtt{crs}_{\mathsf{up}}, \mathtt{x}, \mathtt{w})$
    - $(\mathsf{sk}_l, \mathsf{pk}_l) \leftarrow \Sigma.\mathsf{KGen}(1^\lambda); (\mathsf{sk_{OT}}, \mathsf{pk_{OT}}) \leftarrow \Sigma_{\mathsf{OT}}.\mathsf{KGen}(1^\lambda)$;
    - $\omega \leftarrow\$ \mathbb{Z}_\mathsf{p}; \mathsf{c} \leftarrow \mathsf{UP.Enc}(\mathsf{pk_{up}}, \mathtt{w}; \omega)$;
    - $\pi_\Pi \leftarrow \Pi.\mathsf{P}(\mathtt{crs}_{\mathsf{up}}, (\mathtt{x}, \mathsf{c}, \perp), (\mathtt{w}, \omega, \perp)); \sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_l, \mathsf{pk_{OT}})$;
    - $\sigma_{\mathsf{OT}} \leftarrow \Sigma_{\mathsf{OT}}.\mathsf{Sign}(\mathsf{sk_{OT}}, \pi_\Pi||\mathtt{x}||\mathsf{c}||\mathsf{pk}_l||\sigma)$;
    - **return** $\pi := (\mathsf{c}, \pi_\Pi, \mathsf{pk}_l, \sigma, \mathsf{pk_{OT}}, \sigma_{\mathsf{OT}})$;
$\mathsf{V}(\mathtt{crs}_{\mathsf{up}}, \mathtt{x}, \pi = (\mathsf{c}, \pi_\Pi, \mathsf{pk}_l, \sigma, \mathsf{pk_{OT}}, \sigma_{\mathsf{OT}}))$
    - **if** $\Pi.\mathsf{V}(\mathtt{crs}_{\mathsf{up}}, \mathtt{x}, \mathsf{c}, \pi_\Pi) = 1 \wedge \Sigma.\mathsf{Verify}(\mathsf{pk}_l, \mathsf{pk_{OT}}, \sigma) = 1 \wedge$
      $\Sigma_{\mathsf{OT}}.\mathsf{Verify}(\mathsf{pk_{OT}}, \pi_\Pi||\mathtt{x}||\mathsf{c}||\mathsf{pk}_l||\sigma, \sigma_{\mathsf{OT}}) = 1$ **then return** $1$;
      **else return** $1$;
$\mathsf{Sim}(\mathtt{crs}_{\mathsf{up}}, \mathtt{x}, \mathtt{tc})$
    - $(\mathsf{sk}_l, \mathsf{pk}_l) \leftarrow \Sigma.\mathsf{KGen}(1^\lambda);\ (\mathsf{sk_{OT}}, \mathsf{pk_{OT}}) \leftarrow \Sigma_{\mathsf{OT}}.\mathsf{KGen}(1^\lambda)$;
    - $\omega, z \leftarrow\$ \mathbb{Z}_\mathsf{p}; \mathsf{c} \leftarrow \mathsf{UP.Enc}(\mathsf{pk_{up}}, z; \omega)$;
    - $\pi_{\mathsf{Sim}} \leftarrow \Pi.\mathsf{Sim}(\mathtt{crs}_{\mathsf{up}}, (\mathtt{x}, \mathsf{c}, \mathsf{pk}_l), (z, \omega, \mathsf{csk_{up}}))$;
    - $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_l, \mathsf{pk_{OT}})$;
    - $\sigma_{\mathsf{OT}} \leftarrow \Sigma_{\mathsf{OT}}.\mathsf{Sign}(\mathsf{sk_{OT}}, \pi_{\mathsf{Sim}}||\mathtt{x}||\mathsf{c}||\mathsf{pk}_l||\sigma)$;
    - **return** $\pi := (\mathsf{c}, \pi_{\mathsf{Sim}}, \mathsf{pk}_l, \sigma, \mathsf{pk_{OT}}, \sigma_{\mathsf{OT}})$.
$\mathsf{Ext}(\mathsf{sk_{up}}, \mathsf{c}, \mathtt{crs}, \mathtt{crs}_{\mathsf{up}}, (\zeta_i)_{i=1}^n)$
    - **if** $\mathsf{UP.Vpk}(\mathsf{pk}, (\zeta_{\mathsf{ZK,pk},i})_{i=1}^n) = 0$ **then return** $0$;
  **else return** $\mathtt{w} \leftarrow \mathsf{UP.Dec}(\mathsf{sk_{up}}, \mathsf{c})$.
```

Fig. 2. BB-Lamassu: generic black-box SE updatable (succinct) NIZKs. Changes to Lamassu are indicated with grey boxes.

```
$\mathcal{A}(\mathtt{crs} = (\mathtt{crs}_\Pi, \mathsf{cpk}, \mathsf{pk}), \zeta = (\zeta_{\mathsf{ZK},\Pi}, \zeta_{\mathsf{ZK},\mathsf{cpk}}, \zeta_{\mathsf{ZK},\mathsf{pk}}))$
$(\mathtt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}}) \leftarrow \mathsf{Z}(\mathtt{crs}, \zeta);$
    $\mathsf{Ext}_{\mathsf{ZK}}(\mathsf{cpk}, (\zeta_{\mathsf{ZK},\mathsf{cpk}_{\mathsf{up}},i})_{i=1}^{n}, \mathsf{aux})$
    **return** $\mathsf{csk}_{\mathsf{up}}$.
```

Fig. 3. Extractor and the constructed adversary $\mathcal{A}$ from the updatable ZK proof.

To prove updatable zero-knowledge, we use the extractor $\mathsf{Ext}_{\mathsf{ZK}}$ to obtain the trapdoor $\mathsf{csk}_{\mathsf{up}}$ as explained above and give a simulator Sim (see Fig. 2). When provided $\mathsf{csk}_{\mathsf{up}}$, Sim produces a proof π_{Sim} that has the same distribution as a real proof π generated using the witness $\mathtt{w}$. Recall that due to the OR trick, Sim just needs to prove that it knows the shift $\mathsf{up}_{\mathsf{csk}}$ (which is the trapdoor of the $\mathsf{cpk}_{\mathsf{up}}$) to adapt signatures from cpk to ones valid under verification key $\mathsf{cpk}_{\mathsf{up}}$ in the CRS. Specifically, Sim first chooses $z \leftarrow\!\!\$\ \mathcal{M}$, $\omega \leftarrow\!\!\$\ \mathsf{RND}(\mathsf{Sim})$ and computes $\mathsf{c} \leftarrow \mathsf{UP.Enc}(\mathsf{pk}_{\mathsf{up}}, z; \omega)$. Finally Sim can locally generate $(\mathsf{sk}_l, \mathsf{pk}_l) \leftarrow \Sigma.\mathsf{KGen}(1^\lambda); (\mathsf{sk}_{\mathsf{OT}}, \mathsf{pk}_{\mathsf{OT}}) \leftarrow \Sigma_{\mathsf{OT}}.\mathsf{KGen}(1^\lambda)$ and then compute $\sigma_{\mathsf{OT}} \leftarrow \Sigma_{\mathsf{OT}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{OT}}, \pi_\Pi || \mathsf{x} || \mathsf{c} || \mathsf{pk}_l || \sigma)$. Now the simulated $\pi_{\mathsf{Sim}} = (\mathsf{c}, \pi_{\mathsf{Sim}}, \mathsf{pk}_l, \sigma, \mathsf{pk}_{\mathsf{OT}}, \sigma_{\mathsf{OT}})$ has the same distribution as a real proof $\pi = (\mathsf{c}, \pi_\Pi, \mathsf{pk}_l, \sigma, \mathsf{pk}_{\mathsf{OT}}, \sigma_{\mathsf{OT}})$. Here π_Π is a real proof in the underlying updatable NIZK Π.

(iii: Black-box updatable SE): For the sake of simplicity, let the malicious subverter Z make only a single update after an honest setup, or let Z generate the CRS, after which point we have only a single update by an honest updater.

Recall that based on the (rewinding or straight-line) extraction property of $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$ used in the CRS updates, it is possible to extract the adversary's contribution to the trapdoors csk and sk when the adversary generates the CRS itself. To collapse chains of honest updates into a single honest setup (resp. update) it is convenient that the trapdoor contributions of the setup and update commute in our scheme.

Our proof is based on the non-BB SE proof in [DS19], replacing the underlying NIZK with an updatable NIZK (SNARK) in a black-box manner. We use simulation of the trapdoors of the EUF-CMA-secure updatable signature scheme to simulate proofs. Based on the updatability property, if $\mathcal{A}$ outputs $\mathtt{crs}_{\mathsf{up}} = (\mathtt{crs}_{\Pi,\mathsf{up}}, \mathsf{cpk}_{\mathsf{up}}, \mathsf{pk}_{\mathsf{up}})$ and $(\zeta_{\mathsf{ZK},\Pi,\mathsf{up}}, \zeta_{\mathsf{ZK},\mathsf{cpk}_{\mathsf{up}}}, \zeta_{\mathsf{ZK},\mathsf{pk}_{\mathsf{up}}})$ such that $\mathsf{Vcrs}(\mathtt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}}) = 1$, then by the extractability of ZK, there exists a PPT extractor $\mathsf{Ext}_{\mathsf{ZK}}$ which, given $\mathsf{cpk}_{\mathsf{up}}$, $\mathsf{pk}_{\mathsf{up}}$, and the proofs $(\zeta_{\mathsf{ZK},\mathsf{pk}_{\mathsf{up}}}, \zeta_{\mathsf{ZK},\mathsf{cpk}_{\mathsf{up}}})$, outputs $(\mathsf{sk}_{\mathsf{up}}, \mathsf{csk}_{\mathsf{up}})$.

We note that the SE adversary $\mathcal{A}$ in the *updatable setting*, besides seeing a pair $(\mathtt{crs}, \pi)$, may even have already updated the $\mathtt{crs}$. Thus, here $\mathcal{A}$ has more power than the standard SE adversary in [DS19]. To make the proof more precise, we use the malicious updater Z for updating the $\mathtt{crs}$ and the adversary $\mathcal{A}$ against the SE property. Note that Z and $\mathcal{A}$ can communicate with each other.

We recall the experiment for updatable SE in Fig. 4 and we highlight changes between games by specifying the altered line numbers in the experiment or oracle.

```
$\mathsf{Exp}^{\mathsf{bb\text{-}up\text{-}se}}(\mathcal{A}, \lambda)$
1 : $(\mathtt{crs} = (\mathtt{crs}_\Pi, \mathsf{cpk}, \mathsf{pk}), (\zeta_i)_{i=1}^n, \mathsf{aux}_\mathsf{Z}) \leftarrow \mathsf{Z}(1^\lambda)$;
2 : $(\mathtt{crs}_\mathsf{up}, \zeta_\mathsf{up}) \leftarrow \mathsf{Ucrs}(\mathtt{crs}, (\zeta_i)_{i=1}^n)$;
3 : if $\mathsf{Vcrs}(\mathtt{crs}, (\zeta_i)_{i=1}^n) = 0$ then return $0$
4 : $\mathsf{csk}_\mathsf{up} \leftarrow \mathsf{Ext}_\mathsf{ZK}(\mathsf{cpk}_\mathsf{up}, \zeta_{\mathsf{ZK},\mathsf{cpk}_\mathsf{up}}, \mathsf{aux})$;
5 : $(\mathtt{x}, \pi) \leftarrow \mathcal{A}^{\mathsf{O}(\mathtt{crs}_\mathsf{up}, \mathsf{csk}_\mathsf{up}, \cdot)}(\mathtt{crs}, \mathtt{crs}_\mathsf{up}, \mathsf{aux}_\mathsf{Z})$;
6 : Parse $\pi := (\mathsf{c}, \pi_\Pi, \mathsf{pk}_l, \sigma, \mathsf{pk}_\mathsf{OT}, \sigma_\mathsf{OT})$;
7 : $\mathsf{sk}_\mathsf{up} \leftarrow \mathsf{Ext}_\mathsf{ZK}(\mathsf{pk}_\mathsf{up}, \zeta_{\mathsf{ZK},\mathsf{pk}_\mathsf{up}}, \mathsf{aux})$;
8 : $\mathtt{w} \leftarrow \mathsf{UP.Dec}(\mathsf{sk}_\mathsf{up}, \mathsf{c})$;
9 : if $(\mathtt{x}, \pi) \notin \mathcal{Q} \wedge \mathsf{V}(\mathtt{crs}_\mathsf{up}, \mathtt{x}, \pi) = 1 \wedge (\mathtt{x}, \mathtt{w}) \notin \mathcal{R}$ return $1$.
10 : else return $0$.

$\mathsf{O}(\mathtt{crs}_\mathsf{up}, \mathtt{tc}, \mathtt{x})$
1 : $(\mathsf{sk}_l, \mathsf{pk}_l) \leftarrow \Sigma.\mathsf{KGen}(1^\lambda)$; $(\mathsf{sk}_\mathsf{OT}, \mathsf{pk}_\mathsf{OT}) \leftarrow \Sigma_\mathsf{OT}.\mathsf{KGen}(1^\lambda)$;
2 : $\omega, z \leftarrow\!\$\, \mathbb{Z}_\mathsf{p}$; $\mathsf{c} \leftarrow \mathsf{UP.Enc}(\mathsf{pk}_\mathsf{up}, z, \omega)$;
3 : $\pi_\mathsf{Sim} \leftarrow \Pi.\mathsf{Sim}(\mathtt{crs}_\mathsf{up}, (\mathtt{x}, \mathsf{c}, \mathsf{pk}_l), (z, \omega, \mathtt{tc})$; $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_l, \mathsf{pk}_\mathsf{OT})$;
4 : $\sigma_\mathsf{OT} \leftarrow \Sigma_\mathsf{OT}.\mathsf{Sign}(\mathsf{sk}_\mathsf{OT}, \pi_\mathsf{Sim}||\mathtt{x}||\mathsf{c}||\mathsf{pk}_l||\sigma)$;
5 : $\pi := (\mathsf{c}, \pi_\Pi, \mathsf{pk}_l, \sigma, \mathsf{pk}_\mathsf{OT}, \sigma_\mathsf{OT})$;
6 : $\mathcal{Q} := \mathcal{Q} \cup \{(\mathtt{x}, \pi)\}$; $\mathcal{T} := \mathcal{T} \cup \{\mathsf{pk}_\mathsf{OT}\}$;
7 : return $\pi$;
```

Fig. 4. Experiment $\mathsf{Exp}^{\mathsf{bb\text{-}up\text{-}se}}(\mathcal{A}, \lambda)$ for black-box SE updatable NIZKs.

Game_1: This is the original experiment in Fig. 4.

Game_2: This game is the same as Game_1, with the difference that Z updates the $\mathtt{crs}$ instead of generating it:

Exp, line 1: $(\mathtt{crs}_\Pi, \mathtt{tc}_\Pi, \zeta_{\mathsf{ZK},\Pi}) \leftarrow \Pi.\mathsf{KGen}(1^\lambda)$; $(\mathsf{csk}, \mathsf{cpk}, \zeta_{\mathsf{sk},\mathsf{cpk}}) \leftarrow \Sigma.\mathsf{KGen}(1^\lambda)$; $(\mathsf{sk}, \mathsf{pk}, \zeta_{\mathsf{ZK},\mathsf{pk}}) \leftarrow \mathsf{UP.KGen}(1^\lambda)$; $\mathtt{crs} := (\mathtt{crs}_\Pi, \mathsf{cpk}, \mathsf{pk})$, $\mathtt{tc} := (\mathtt{tc}_\Pi, \mathsf{csk}, \mathsf{sk})$, $\zeta := (\zeta_{\mathsf{sk},\Pi}, \zeta_{\mathsf{sk},\mathsf{cpk}}, \zeta_{\mathsf{sk},\mathsf{pk}})$; **return** $(\mathtt{crs}, \mathtt{tc}, \zeta)$;
Exp, line 2: $(\mathtt{crs}_\mathsf{up}, \zeta_\mathsf{up}, \mathsf{aux}_\mathsf{Z}) \leftarrow \mathsf{Z}(1^\lambda, \mathtt{crs}, (\zeta_i)_{i=1}^n)$;

$\mathsf{Game}_1 \rightarrow \mathsf{Game}_2$: This is straightforward from the property of the updating procedure that if Vcrs outputs 1, then there is an extractor that extracts sk_up and csk_up (i.e., when the adversary updates an honest CRS it is possible to extract the updates with, e.g., the straight-line trapdoor extraction of $\mathsf{Fischlin}$ for UP and Σ) and the zero-knowledge property of the NIZK. Thus, we have $\Pr[\mathsf{Game}_0] \leq \Pr[\mathsf{Game}_1] + \mathsf{negl}(\lambda)$.

Game_3: This game is the same as Game_2, but $\Delta \leftarrow\!\$\, \mathbb{H}$ is replaced in $\mathsf{cpk} = \mu(\Delta) \cdot \mathsf{pk}_l$.

Exp, line 1: $\Delta \leftarrow\!\$\, \mathbb{H}$;
Exp, line 2: $\mathtt{crs} := (\mathtt{crs}_\Pi, \mathsf{cpk} \cdot \mu(\Delta), \mathsf{pk}), \mathtt{tc} := (\mathtt{tc}_\Pi, \mathsf{csk}, \mathsf{sk})$;

Winning condition: Let $\mathcal{Q}$ be the set of $(\mathtt{x}, \pi)$ pairs, let $\mathcal{T}$ be the set of OTS verification keys generated by the oracle O. The game outputs 1 iff: $(\mathtt{x}, \pi) \notin \mathcal{Q} \wedge \mathsf{V}(\mathtt{crs}_\mathsf{up}, \mathtt{x}, \pi) = 1 \wedge \mathsf{pk}_\mathsf{OT} \notin \mathcal{T} \wedge \mathsf{cpk} \cdot \mu(\Delta) = \mathsf{pk}_l \cdot \mu(\Delta) \cdot \mu(\mathsf{csk} - \mathsf{sk}_l)$.

– CRS: On input $(\texttt{start}, \mathsf{sid})$ run $\texttt{crs} \leftarrow \mathsf{KGen}(1^\lambda)$. Send $(\texttt{CRS}, \mathsf{sid}, \texttt{crs})$ to all parties and halt.

Fig. 5. The ideal UC functionality $\mathcal{F}_{\texttt{crs}}$ for UC NIZK common reference string generation [Gro06].

– Proof: On input $(\texttt{prove}, \mathsf{sid}, \mathtt{x}, \mathtt{w})$ from party P ignore if $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}$. Send $(\texttt{prove}, \mathtt{x})$ to $\mathsf{Sim}_{\mathsf{uc}}$ and wait for answer $(\texttt{proof}, \pi)$. Upon receiving the answer store $(\mathtt{x}, \pi)$ and send $(\texttt{proof}, \mathsf{sid}, \pi)$ to P.
– $\mathsf{Verification}$: On input $(\texttt{verify}, \mathsf{sid}, \mathtt{x}, \pi)$ from V check whether $(\mathtt{x}, \pi)$ is stored. If not send $(\texttt{verify}, \mathtt{x}, \pi)$ to $\mathsf{Sim}_{\mathsf{uc}}$ and wait for an answer $(\texttt{witness}, w)$. Upon receiving the answer, check whether $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$ and in that case, store $(\mathtt{x}, \pi)$. If $(\mathtt{x}, \pi)$ has been stored return $(\texttt{verification}, \mathsf{sid}, 1)$ to V, else return $(\texttt{verification}, \mathsf{sid}, 0)$ to V.

Fig. 6. The ideal UC functionality $\mathcal{F}_{\mathsf{NIZK}}$, parameterized by relation $\mathcal{R}$, interacts with adversary $\mathsf{Sim}_{\mathsf{uc}}$ and parties $P_1, \ldots, P_n$ [Gro06].

$\mathsf{Game}_2 \rightarrow \mathsf{Game}_3$: This follows from [ARS20, Theorem 3] and the adaptable and updatable EUF-CMA property of Σ.

4.2 From Black-Box SE Updatable NIZKs to UC-Secure Updatable NIZKs

The notion of simulation extractability (SE) is roughly speaking equivalent to UC-secure (succinct) NIZKs. We now elaborate on the relation between SE NIZKs and UC-Secure NIZKs. Groth [Gro06] showed that the notion of SE can be used to instantiate a NIZK ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$. More precisely, Groth [Gro06] defined two separated ideal functionalities $\mathcal{F}_{\mathsf{CRS}}$ and $\mathcal{F}_{\mathsf{NIZK}}$ (see Figs. 5 and 6).

Similarly, Kosba *et al.* [KZM+15] show that a weak SE secure NIZK can be used to realize a weaker version of the ideal functionality called $\mathcal{F}_{\mathsf{weak\text{-}NIZK}}$. The main difference between the weaker and the stronger version is that the weaker version may permit an adversary to maul an existing proof to a new proof, but for the same statement. Both versions prevent the adversary from mauling a proof to a related statement. Depending on the application, sometimes the weak SE notion suffices in protocol design.

As our framework in Fig. 2 for black-box updatable (strong) SE NIZKs is not in the conventional CRS model but the updatable CRS model, we define a new ideal functionality $\mathcal{F}_{\mathsf{up\text{-}CRS}}$ for the updatable CRS generation in Fig. 7. We employ the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$ [Gro06] for the proof of a correct update. Finally, in Theorem 4 we prove that our framework in Fig. 2 for black-box updatable (strong) SE NIZKs realizes $\mathcal{F}_{\mathsf{NIZK}}$ in the $\mathcal{F}_{\mathsf{up\text{-}CRS}}$-hybrid model. We note that, due to the fact that rewinding (used in the extraction phase in FS) is

- CRS: On input $(\texttt{start}, \mathsf{sid}, \texttt{tc})$ from party P ignore if $\texttt{tc} = \perp$. Generate a $\texttt{crs}$ and do as follows:
  - If P is uncorrupted then send $(\texttt{start}, \texttt{crs})$ to $\mathsf{Sim}_{\mathsf{uc}}$ and wait for answer $(\texttt{proofCRS}, \zeta)$.
  - If P is corrupted then send $(\texttt{start}, \texttt{crs}, \texttt{tc})$ to $\mathsf{Sim}_{\mathsf{uc}}$ and wait for answer $(\texttt{proofCRS}, \zeta)$.

  Upon receiving the answer store $(\mathsf{sid}, \texttt{crs}, \zeta)$ in $Q_{\texttt{crs}}$ and send $(\texttt{proofCRS}, \mathsf{sid}, \texttt{crs}, \zeta)$ to P.
- upCRS: On input $(\texttt{upCRS}, \mathsf{sid}, \texttt{crs}, \texttt{tc}_{\mathsf{up}})$ from party P ignore if $(\mathsf{sid}, \texttt{crs}) \notin Q_{\texttt{crs}}$ or $\texttt{tc} = \perp$. Generate $\texttt{crs}_{\mathsf{up}}$ and do as follows:
  - If P is uncorrupted then send $(\texttt{upCRS}, \texttt{crs}, \texttt{crs}_{\mathsf{up}})$ to $\mathsf{Sim}_{\mathsf{uc}}$ and wait for answer $(\texttt{proofCRS}, \zeta_{\mathsf{up}})$.
  - If P is corrupted then send $(\texttt{upCRS}, \texttt{crs}, \texttt{crs}_{\mathsf{up}}, \texttt{tc})$ to $\mathsf{Sim}_{\mathsf{uc}}$ and wait for answer $(\texttt{proofCRS}, \zeta_{\mathsf{up}})$.

  Upon receiving the answer store $(\mathsf{sid}, \texttt{crs}, \texttt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$ in $Q_{\texttt{crs}}$ and send $(\texttt{proofCRS}, \mathsf{sid}, \texttt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$ to P.
- verCRS: On input $(\texttt{checkCRS}, \mathsf{sid}, \texttt{crs}, \texttt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$ from P check whether $(\mathsf{sid}, \texttt{crs}, \texttt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$ is stored in $Q_{\texttt{crs}}$. If not send $(\texttt{checkCRS}, \texttt{crs}, \texttt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$ to $\mathsf{Sim}_{\mathsf{uc}}$ and wait for an answer $(\texttt{trapdoor}, \texttt{tc}_{\mathsf{up}})$. Upon receiving the answer, check whether $\mathsf{Vcrs}(1^\lambda, \texttt{crs}_{\mathsf{up}}, \zeta := (\texttt{crs}, \texttt{tc}_{\mathsf{up}})) = 1$ and in that case, store $(\texttt{crs}, \texttt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$. If $(\texttt{crs}, \texttt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$ has been stored return $(\texttt{verCRS}, \mathsf{sid}, 1)$ to P, else return $(\texttt{verCRS}, \mathsf{sid}, 0)$ to P.

Fig. 7. The ideal UC functionality $\mathcal{F}_{\mathsf{up\text{-}CRS}}$ for UC updatable CRS generation of NIZKs, interacts with adversary $\mathsf{Sim}_{\mathsf{uc}}$ and parties.

not allowed in the UC model [Can01], we assume $\mathsf{ZK} \in \{\mathsf{Fischlin}, \mathsf{Unruh}\}$ for the instantiation of the ZK proof.

Theorem 4. *Let Π be a NIZK (SNARK) scheme satisfying perfect completeness, computational updatable zero-knowledge, and computational updatable (optionally knowledge) soundness, UP an EKU-PKE scheme with message space $\mathcal{M}$ satisfying IND-CPA security and perfect correctness, Σ an EUF-CMA-secure updatable signature scheme, and Σ_{OT} a one-time signature scheme with strong unforgeability. Let $\mathsf{ZK} \in \{\mathsf{Fischlin}, \mathsf{Unruh}\}$ be a non-interactive proof of knowledge with BB extraction. Then the construction in Fig. 2 securely realizes $\mathcal{F}_{\mathsf{NIZK}}$ in the $\mathcal{F}_{\mathsf{up\text{-}CRS}}$-hybrid model.*

For the proof, we refer to Appendix A.7.

5 Evaluation and Instantiation

We split the evaluation into multiple parts. First we consider the costs for the CRS updates and of the witness encryption and we compare our framework to Lamassu. Finally, in Section 5.2 we apply BB-Lamassu to the updatable SNARK Sonic and present the results of our benchmarks.

5.1 Overheads

Costs of the CRS update. For the CRS update costs, we do not consider the overhead of the CRS update proofs for the underlying SNARK since they depend on the underlying construction. We will discuss this overhead later using Sonic as an example. Observe that in comparison to LAMASSU, the CRS of BB-LAMASSU is extended with an UP public key which is updated in the CRS update. Consequently, the proof of the CRS update is extended with a proof for the UP public key update. For our UP construction from Section 3, this proof is respect to the statement $\mathsf{pk}'_{\mathsf{up}} = \mathsf{pk}_{\mathsf{up}} \cdot g^x$, which can be proven with a simple Σ-protocol.

Note that if the Fiat-Shamir, Fischlin or Unruh transform is applied to such a Σ-protocol, we are able to omit the first message (commitment) as it can be recomputed from challenge and response. Therefore, the proof consists of 2 $\mathbb{Z}_p$ elements for FS and $2s$ $\mathbb{Z}_p$ elements for $\mathsf{Fischlin}$ (where s is a parameter of the latter; see Appendix A.5 for details). The choice of s influences both the size and the runtime of the prover. The smaller s, the smaller the proof, but the harder it is for the prover to find suitable challenge-response pairs. For Unruh, we can set the parameter $t = 1$ since the Σ-protocol has a negligible soundness error and $M = 2$ since it is 2-special sound (again, see Appendix A.5 for transform details). Therefore, for Unruh, the proofs consist of 5 $\mathbb{Z}_p$ elements. Consequently, except for the case $s = 2$, $\mathsf{Fischlin}$ always produces the largest proofs. When instantiating $\mathbb{G}$ with an order of ≈ 256 bits, we obtain proofs of 170 bytes with Unruh. Compared to the 64 bytes for FS, achieving UC-compatible extraction with our technique only incurs a small overhead.

Costs of encrypting the witness. For the costs of proving consistency of the encryption of the witness, we can focus on the number of constraints induced by the statements as a cost metric. As this metric depends on the choice of the involved groups, we choose the SNARK-friendly group Jubjub and can thus lift the number of constraints from [HBHW22] for the evaluation. We split the analysis of encryption with the hybrid application UP into the EKU-PKE (ElGamal) part and the symmetric part.

For the ElGamal-based EKU-PKE, we need to prove $\mathsf{c}_1 = g^r \wedge \mathsf{c}_2 = \mathsf{M} \cdot \mathsf{pk}_{\mathsf{up}}^r$ for witnesses M and r. Statements of the form $y = h \cdot g^{\mathtt{w}}$ for a witness $\mathtt{w}$ with respect to the Jubjub curve group can be expressed with 756 constraints. Proving that $\mathtt{w}$ is in the correct range costs another 252 constraints and that h is a group element costs 4 constraints. Hence, we require at most 1768 constraints.

Selecting a symmetric encryption scheme is more involved. The straightforward choice is AES. Since a mode supporting parallel encryption is preferable as they allow for shallower circuits, we focus on counter (CTR) mode. All other modes require at least the same number of AES evaluations. Since a single AES evaluation is expensive [KPS18], choosing a different block cipher with low multiplicative complexity may be more desirable. While some of those primitives have only been optimized for the use of keyless permutations to construct hashes in the context of SNARKs, sponge-based constructions with a keyless permutation also yield a secure stream cipher [BDPV12].

Table 2. Number of constraints required for symmetric-key encryption for witness of sizes 1 KB and 32 KB.

Symmetric primitive	Mode	# of constraints for	
		1 KB	32 KB
AES128	CTR	748,694	23,878,166
AES256 (estimated)	CTR	1,048,224	33,431,072
POSEIDON-$(1536, 2, 10, 114)$	Sponge	4,020	103,716
LOWMC-$(1602, 256, 1, 1484)$	Sponge	29,680	721,224
GMIMC-$(256, 32, 564)$	CTR	1,128	36,096
GMIMC-$(256, 32, 564)$	Sponge	4,512	41,736
VISION-$(127, 14, 10)$	Sponge	12,600	292,600

Table 2 shows the number of constraints for various symmetric primitives with witness sizes of 1 and 32 KiB. We consider GMIMC-(N, t, R) with a collapsing round function [AGP+19], POSEIDON-(N, t, R_f, R_p) with $x \mapsto x^5$ as the SBox [GKR+21], VISION-(N, t, R) [AAB+20], and LOWMC-(N, k, m, R) [ARS+15], 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. The numbers for AES256 are extrapolated from those of AES128 [KPS18]. The sponge constructions are all instantiated with a capacity of ≈ 512 bits. All keys are chosen to have 256 bits and nonces have 96 bits. Overall, Table 2 shows that even moderately-sized witness can be handled efficiently.

Comparison with LAMASSU. We recall from the evaluation of LAMASSU [ARS20] that the transformation from an updatable zk-SNARK to an updatable SE zk-SNARK comes with an overhead that is bounded by ≈ 32 bytes for the CRS and ≈ 256 bytes for the proofs independent of the concrete circuits. In contrast to LAMASSU, in our case the proofs for the validity of the CRS update need to be extended for the corresponding proof of the updatable signature scheme. By the same calculation we gave above for UP, this requires at most 170 bytes using Unruh.

5.2 Black-Box SE Version of Sonic

Finally, as an example of our generic black-box SE updatable circuit-succinct NIZKs, we provide a black-box SE version of Sonic [MBKM19]. Although implementations of more recent updatable NIZKs than Sonic are available, the goal of our evaluation is to illustrate the overhead introduced by our compiler over an (arbitrary) base scheme and therefore the choice of the base implementation is not significant. We chose to implement our transformation on top of Sonic due to ease of use and availability of the implementation.

Sonic uses an updatable structured reference string (uSRS). While uSRSs are modeled in their paper, this is done in a non-black-box way and we instead model their security in the setting of black-box SE. Here, a uSRS is a reference

Table 3. Runtime of our BB SE succinct NIZK compared to the non-BB SE zk-SNARK obtained via LAMASSU [ARS20] and the base non-SE zk-SNARK Sonic [MBKM19].

Input size (bits)	Scheme	Prove (s)		Vrfy (s)		Helped Vrfy (ms)
Pedersen hash (average over 20 iterations)						
48	*Sonic*	0.371		0.00123		0.844
	Lamassu	0.661		0.00282		0.816
	This work	**5.758**	**(15.52×)**	**0.0226**	**(18.37×)**	**0.719**
384	*Sonic*	1.610		0.00649		0.843
	Lamassu	1.938		0.00804		0.821
	This work	**6.592**	**(4.09×)**	**0.0226**	**(3.48×)**	**0.817**
SHA-256 (average over 10 iterations)						
512	*Sonic*	47.736		0.457		1.30
	Lamassu	49.400		0.455		-0.111
	This work	**56.729**	**(1.18×)**	**0.490**	**(1.07×)**	**0.347**
1024	*Sonic*	76.899		0.727		1.67
	Lamassu	79.517		0.734		-0.444
	This work	**89.510**	**(1.16×)**	**0.770**	**(1.06×)**	1.42
2048	*Sonic*	126.918		1.272		0.822
	Lamassu	126.597		1.277		0.0132
	This work	**155.982**	**(1.23×)**	**1.349**	**(1.06×)**	**0.666**

string with an underlying trapdoor $\mathtt{tc}_\Pi$ which has had a structure function SRS imposed on it. $\mathsf{SRS}(\mathtt{tc}_\Pi)$ is the reference string itself, while $\mathtt{tc}_\Pi$ is the trapdoor.

SRS of Sonic Given generators $\mathsf{g} \in \mathbb{G}_1$, $\mathsf{h} \in \mathbb{G}_2$ and a depth parameter $d \in \mathbb{Z}_p$, the SRS has a trapdoor of $\mathtt{tc}_\Pi := (\alpha, \chi) \in \mathbb{Z}_p^{*2}$. The corresponding structure function is defined as:

$$\mathsf{SRS}(\mathtt{tc}_\Pi) = \left(\{\mathsf{g}^{\chi^i}, \mathsf{h}^{\chi^i}, \mathsf{h}^{\alpha\chi^i}\}_{i=-d}^{i=d}, \{\mathsf{g}^{\alpha\chi^i}\}_{i=-d, i\neq 0}^{i=d}\right)$$

We omit the $\bar{e}(\mathsf{g}, \mathsf{h}^\alpha)$ term presented in [MBKM19], as this can be computed from the rest of the uSRS and is therefore immaterial to the update procedure.

Black-box SE Sonic As we discussed in Section 4.1, in order to add the black-box SE property to the Sonic scheme, we start with the updating procedure on the uSRS of Sonic and give a non-interactive proof of knowledge with black-box extraction that the update is correctly done. This is in contrast to the update in [MBKM19], which reveals some intermediate shares in both source groups to

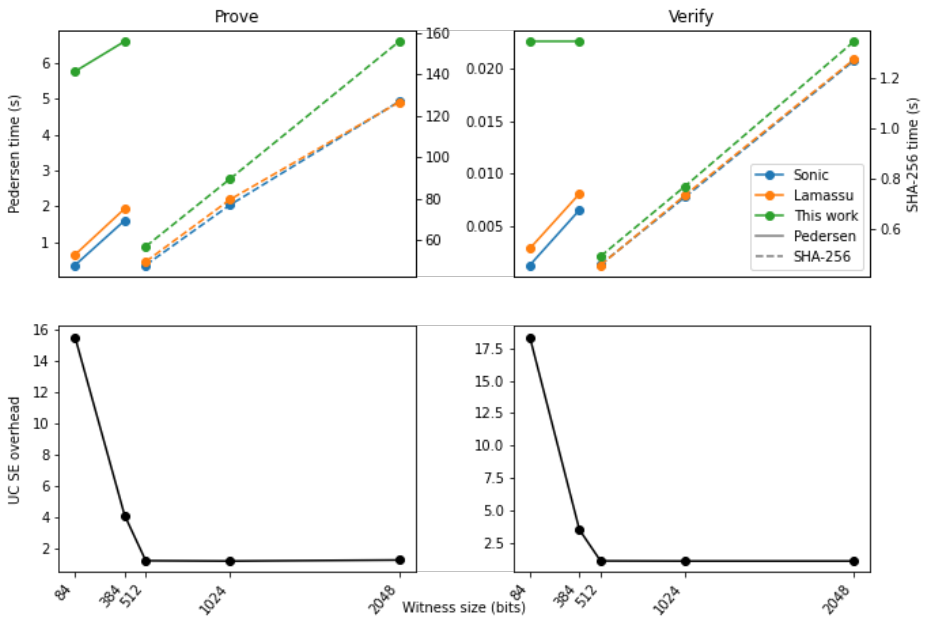

Fig. 8. Runtimes of the Prove and Verify algorithms of our BB SE succinct NIZK compared to the non-BB SE zk-SNARK obtained via LAMASSU [ARS20] and the base non-SE zk-SNARK Sonic [MBKM19]. In the lower part we plot the overhead of our transformation to add BB SE, which decreases as the witness size increases.

show that the update is correctly done and uses a non-falsifiable assumption to extract the updated secret key; thus, is not black-box. We present the changes to the uSRS and its update procedure in Appendix A.8. With these changes we obtain the following result directly from Theorem 3:

Corollary 1. *Assume that Sonic satisfies perfect completeness, updatable computational zero-knowledge, and updatable computational soundness. Let* $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$ *be a non-interactive proof of knowledge with black-box extraction,* UP *be the ElGamal EKU-PKE, and* Σ *be the updatable Schnorr signature scheme. Then the scheme resulting from the application of* BB-LAMASSU *to Sonic is a NIZK satisfying perfect completeness, updatable zero-knowledge, and updatable simulation extractability.*

Implementation We implemented BB-LAMASSU as well as LAMASSU [ARS20] in Rust[10] on top of Sonic[11] with the updatable signature scheme from Section 2.2 and the EKU-PKE from Section 3 (both instantiated over the Jubjub curve group), and Schnorr signatures as sOTS. In Table 3 we report times for

[10] `https://github.com/nglaeser/sonic-ucse/`
[11] `https://github.com/ebfull/sonic`

proving and verifying knowledge of a hash preimage. Averages are taken over 20 iterations for Pedersen and 10 for SHA-256. As in [MBKM19], "Helped Verify" is the marginal cost of verifying an additional proof when proofs are aggregated. This number equals the cost of batch-verifying n proofs minus the cost to verify 1, divided by $n - 1$ (where n is the number of iterations). This number generally decreases as the witness size increases, with some fluctuations due to noise since the marginal costs are very small (on the order of hundreds of μs). The benchmarks were taken on an Intel Xeon 3.8 GHz quad-core CPU with 64 GB RAM.

Figure 8 shows the overhead of adding black-box SE to Sonic. Note that the circuit for the Pedersen hash using the Jubjub curve group is only a few hundred constraints; for larger circuits, such as SHA-256 or even Pedersen hash with larger inputs, the overhead of BB-Lamassu, which scales linearly in the witness size, decreases relative to the cost of processing the original circuit with Sonic.

6 Conclusion

In this work, we present a generic construction of UC-secure NIZKs with an updatable CRS and circuit-succinct proofs, which has been an open problem. While our construction induces some overhead in the runtimes, the evaluation demonstrates that the costs are dominated by the original circuit for moderately-sized witnesses or large circuits. In such regimes our construction can be considered entirely practical.

Acknowledgements. We thank the anonymous reviewers for helpful suggestions, and Sean Bowe and Michael Rosenberg for helpful feedback on the implementation. This work was in part funded by the European Union's Horizon 2020 research and innovation programme under grant agreement n°871473 (Kraken) and n°890456 (SlotMachine), and by the Austrian Science Fund (FWF) and netidee SCIENCE under grant agreement P31621-N38 (Profet). This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 1840340. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Work of B.A. in part done while with Max Planck Institute for Security and Privacy. This work was also supported by the German Federal Ministry of Education and Research BMBF (grant 16KISK038, project 6GEM).

References

AAB$^+$20. Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Trans. Symm. Cryptol.*, 2020(3):1–45, 2020. doi:10.13154/tosc.v2020.i3.1-45.

ABK$^+$21. Michel Abdalla, Manuel Barbosa, Jonathan Katz, Julian Loss, and Jiayu Xu. Algebraic adversaries in the universal composability framework. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 311–341. Springer, Heidelberg, December 2021. doi:10.1007/978-3-030-92078-4_11.

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. doi:10.1007/978-3-319-70700-6_1.

AGP$^+$19. Martin R. Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. Feistel structures for MPC, and more. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019, Part II*, volume 11736 of *LNCS*, pages 151–171. Springer, Heidelberg, September 2019. doi:10.1007/978-3-030-29962-0_8.

AME$^+$21. Lukas Aumayr, Matteo Maffei, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Siavash Riahi, Kristina Hostáková, and Pedro Moreno-Sanchez. Bitcoin-compatible virtual channels. In *2021 IEEE Symposium on Security and Privacy*, pages 901–918. IEEE Computer Society Press, May 2021. doi:10.1109/SP40001.2021.00097.

ARS$^+$15. Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_17.

ARS20. Behzad Abdolmaleki, Sebastian Ramacher, and Daniel Slamanig. Lift-and-shift: Obtaining simulation extractable subversion and updatable SNARKs generically. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1987–2005. ACM Press, November 2020. doi:10.1145/3372297.3417228.

BCCT12. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In Shafi Goldwasser, editor, *ITCS 2012*, pages 326–349. ACM, January 2012. doi:10.1145/2090236.2090263.

BCR$^+$19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019. doi:10.1007/978-3-030-17653-2_4.

BDPV12. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 320–337. Springer, Heidelberg, August 2012. doi:10.1007/978-3-642-28496-0_19.

BFM88. Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988. doi:10.1145/62212.62222.

BFS16. Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee

Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53890-6_26.

BKSV21. Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-snark. In *Financial Cryptography (1)*, volume 12674 of *LNCS*, pages 457–475. Springer, 2021.

BPR20. Karim Baghery, Zaira Pindado, and Carla Ràfols. Simulation extractable versions of groth's zk-snark revisited. In *CANS*, volume 12579 of *LNCS*, pages 453–461. Springer, 2020.

BPW12. David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 626–643. Springer, Heidelberg, December 2012. doi:10.1007/978-3-642-34961-4_38.

BS21. Karim Baghery and Mahdi Sedaghat. Tiramisu: Black-box simulation extractable nizks in the updatable CRS model. In *CANS*, volume 13099 of *LNCS*, pages 531–551. Springer, 2021.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. doi:10.1109/SFCS.2001.959888.

CFF$^+$21. Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 3–33. Springer, Heidelberg, December 2021. doi:10.1007/978-3-030-92078-4_1.

CFQ19. Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2075–2092. ACM Press, November 2019. doi:10.1145/3319535.3339820.

CHM$^+$20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Heidelberg, May 2020. doi:10.1007/978-3-030-45721-1_26.

CS03. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

DDO$^+$01. Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Heidelberg, August 2001. doi:10.1007/3-540-44647-8_33.

DFGK14. George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct NIZK arguments. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 532–550. Springer, Heidelberg, December 2014. doi:10.1007/978-3-662-45611-8_28.

DKW21. Yevgeniy Dodis, Harish Karthikeyan, and Daniel Wichs. Updatable public key encryption in the standard model. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 254–285.

Springer, Heidelberg, November 2021. doi:10.1007/978-3-030-90456-2_9.

DP92. Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *33rd FOCS*, pages 427–436. IEEE Computer Society Press, October 1992. doi:10.1109/SFCS.1992.267809.

DS19. David Derler and Daniel Slamanig. Key-homomorphic signatures: definitions and applications to multiparty signatures and non-interactive zero-knowledge. *Des. Codes Cryptogr.*, 87(6):1373–1413, 2019.

EKKV22. Felix Engelmann, Thomas Kerber, Markulf Kohlweiss, and Mikhail Volkhov. Zswap: zk-snark based non-interactive multi-asset swaps. *Proc. Priv. Enhancing Technol.*, 2022(4):507–527, 2022. doi:10.56553/popets-2022-0120.

Fis05. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Heidelberg, August 2005. doi:10.1007/11535218_10.

FKMV12. Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 60–79. Springer, Heidelberg, December 2012. doi:10.1007/978-3-642-34931-7_5.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.

Fuc18. Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, Heidelberg, March 2018. doi:10.1007/978-3-319-76578-5_11.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013. doi:10.1007/978-3-642-38348-9_37.

GKK$^+$22. Chaya Ganesh, Hamidreza Khoshakhlagh, Markulf Kohlweiss, Anca Nitulescu, and Michał Zając. What makes fiat–shamir zksnarks (updatable srs) simulation extractable? In *International Conference on Security and Cryptography for Networks*, pages 735–760. Springer, 2022.

GKM$^+$18. Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96878-0_24.

GKO$^+$23. Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Witness-succinct universally-composable snarks. In *EUROCRYPT (2)*, volume 14005 of *Lecture Notes in Computer Science*, pages 315–346. Springer, 2023.

GKR$^+$21. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021.

GLS$^+$21. Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and post-quantum SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/1043, 2021. https://eprint.iacr.org/2021/1043.

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. doi:10.1007/978-3-319-63715-0_20.

GMM$^+$22. Noemi Glaeser, Matteo Maffei, Giulio Malavolta, Pedro Moreno-Sanchez, Erkan Tairi, and Sri Aravinda Krishnan Thyagarajan. Foundations of coin mixing services. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1259–1273. ACM Press, November 2022. doi:10.1145/3548606.3560637.

GMR85. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985. doi:10.1145/22145.22178.

GOP$^+$22. Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Fiat-shamir bulletproofs are non-malleable (in the algebraic group model). In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 397–426. Springer, Heidelberg, May / June 2022. doi:10.1007/978-3-031-07085-3_14.

GOS12. Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *Journal of ACM*, pages 1–11, 2012.

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. doi:10.1007/11935230_29.

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. doi:10.1007/978-3-642-17373-8_19.

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. doi:10.1007/978-3-662-49896-5_11.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008. doi:10.1007/978-3-540-78967-3_24.

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. doi:10.1145/1993636.1993651.

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

HBHW22. Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification: Version 2022.2.18 [nu5 proposal], 2022.

JMM19. Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 159–188. Springer, Heidelberg, May 2019. doi:10.1007/978-3-030-17653-2_6.

JR13. Charanjit S. Jutla and Arnab Roy. Shorter quasi-adaptive NIZK proofs for linear subspaces. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2013. doi:10.1007/978-3-642-42033-7_1.

KKK21a. Thomas Kerber, Aggelos Kiayias, and Markulf Kohlweiss. Composition with knowledge assumptions. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 364–393, Virtual Event, August 2021. Springer, Heidelberg. doi:10.1007/978-3-030-84259-8_13.

KKK21b. Thomas Kerber, Aggelos Kiayias, and Markulf Kohlweiss. Mining for privacy: How to bootstrap a snarky blockchain. In Nikita Borisov and Claudia Díaz, editors, *FC 2021, Part I*, volume 12674 of *LNCS*, pages 497–514. Springer, Heidelberg, March 2021. doi:10.1007/978-3-662-64322-8_24.

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. doi:10.1109/SP.2016.55.

KMSV21. Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 98–127. Springer, Heidelberg, December 2021. doi:10.1007/978-3-030-92078-4_4.

KNYY19. Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Exploring constructions of compact NIZKs from various assumptions. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 639–669. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26954-8_21.

KNYY20. Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Compact NIZKs from standard assumptions on bilinear maps. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 379–409. Springer, Heidelberg, May 2020. doi:10.1007/978-3-030-45727-3_13.

KPS18. Ahmed E. Kosba, Charalampos Papamanthou, and Elaine Shi. xJsnark: A framework for efficient verifiable computation. In *2018 IEEE Symposium on Security and Privacy*, pages 944–961. IEEE Computer Society Press, May 2018. doi:10.1109/SP.2018.00018.

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Heidelberg, December 2010. doi:10.1007/978-3-642-17373-8_11.

KZM$^+$15. Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093, 2015. https://eprint.iacr.org/2015/1093.

LCOK21. Jeonghyuk Lee, Jaekyung Choi, Hyunok Oh, and Jihye Kim. Privacy-preserving identity management system. *IACR Cryptol. ePrint Arch.*, page 1459, 2021. URL: https://eprint.iacr.org/2021/1459.

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. Springer, Heidelberg, March 2012. doi:10.1007/978-3-642-28914-9_10.

Lip13. Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 41–60. Springer, Heidelberg, December 2013. doi:10.1007/978-3-642-42033-7_3.

Lip19. Helger Lipmaa. Simulation-extractable snarks revisited. Cryptology ePrint Archive, Report 2019/612, 2019. https://eprint.iacr.org/2019/612.

Lip20. Helger Lipmaa. Key-and-argument-updatable QA-NIZKs. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 645–669. Springer, Heidelberg, September 2020. doi:10.1007/978-3-030-57990-6_32.

Lip22. Helger Lipmaa. A unified framework for non-universal snarks. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I*, volume 13177 of *Lecture Notes in Computer Science*, pages 553–583. Springer, 2022. doi:10.1007/978-3-030-97121-2_20.

LR22. Anna Lysyanskaya and Leah Namisa Rosenbloom. Universally composable Σ-protocols in the global random-oracle model. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 203–233. Springer, Heidelberg, November 2022. doi:10.1007/978-3-031-22318-1_8.

MBKM19. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019. doi:10.1145/3319535.3339817.

NRBB22. Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. *IACR Cryptol. ePrint Arch.*, page 1592, 2022. URL: https://eprint.iacr.org/2022/1592.

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. doi:10.1109/SP.2013.47.

PR18. Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96884-1_1.

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996. doi:10.1007/3-540-68339-9_33.

RWGM22. Michael Rosenberg, Jacob White, Christina Garman, and Ian Miers. `zk-creds`: Flexible anonymous credentials from zksnarks and existing identity infrastructure. Cryptology ePrint Archive, Paper 2022/878, 2022. https://eprint.iacr.org/2022/878. URL: https://eprint.iacr.org/2022/878.

RZ21. Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 774–804, Virtual Event, August 2021. Springer, Heidelberg. doi:10.1007/978-3-030-84242-0_27.

Sah99. Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999. doi:10.1109/SFFCS.1999.814628.

Sah01. Amit Sahai. Simulation-sound non-interactive zero knowledge. Technical report, IBM RESEARCH REPORT RZ 3076, 2001.

Sch90. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990. doi:10.1007/0-387-34805-0_22.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Heidelberg, August 2020. doi:10.1007/978-3-030-56877-1_25.

SW14. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014. doi:10.1145/2591796.2591825.

TMM21. Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. A^{2}L: Anonymous atomic locks for scalability in payment channel hubs. In *2021 IEEE Symposium on Security and Privacy*, pages 1834–1851. IEEE Computer Society Press, May 2021. doi:10.1109/SP40001.2021.00111.

TMM22. Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. Universal atomic swaps: Secure exchange of coins across all blockchains. In *IEEE S&P*, pages 1299–1316. IEEE, 2022.

Unr15. Dominique Unruh. Non-interactive zero-knowledge proofs in the quantum random oracle model. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 755–784. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46803-6_25.

A Omitted Definitions and Primitives

A.1 Key-Homomorphic Signatures

We recall the definition of key-homomorphic signatures as introduced in [DS19]. Parts of this section are taken verbatim from [ARS20]. 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 it is required that group operations, inversions, membership testing as well as sampling from the uniform distribution are efficient.

Definition 7 (Secret Key to Public Key Homomorphism). *A signature scheme Σ 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 $\mathsf{sk}, \mathsf{sk}' \in \mathbb{H}$ it holds that $\mu(\mathsf{sk} + \mathsf{sk}') = \mu(\mathsf{sk}) \cdot \mu(\mathsf{sk}')$, and for all $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}$, it holds that $\mathsf{pk} = \mu(\mathsf{sk})$.*

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

Definition 8 (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* Adapt*, defined as:*

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

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

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

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 Adapt is known.

Definition 9 (Adaptability of Signatures). *A key-homomorphic signature scheme provides adaptability of signatures, if for every $\lambda \in \mathbb{N}$ and every message $m \in \mathcal{M}$, it holds that*

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

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

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

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

A.2 Schnorr Signatures

We recall the Schnorr signature scheme [Sch90] together with the Adapt algorithm and a common setup.

Definition 10. *The Schnorr signature scheme $\Sigma = (\mathsf{Pgen}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Adapt})$ consists of the following PPT algorithms:*

$\mathsf{PGen}(1^\lambda)$: *Given a security parameter λ, it outputs a prime order group $(\mathbb{G}, \mathsf{g}, p) \leftarrow \mathsf{GGen}(1^\lambda)$ and a hash function $H \leftarrow\!\$ \{H_k\}_{k \in \mathcal{K}}$.*

$\mathsf{KGen}(\mathsf{pp} = ((\mathbb{G}, \mathsf{g}, p), H))$: *Given public parameters* pp*, it ouputs a secret key* $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$ *and public key* $\mathsf{pk} \leftarrow \mathsf{g}^{\mathsf{sk}}$*.*

$\mathsf{Sign}(\mathsf{sk}, \mathsf{M})$: *Given a secret key* sk *and a message* $\mathsf{M} \in \{0,1\}^*$*, it samples* $r \leftarrow\!\!\$\ \mathbb{Z}_p$*, computes* $R \leftarrow \mathsf{g}^r$*,* $c \leftarrow H(R \| m)$*,* $y \leftarrow r + \mathsf{sk} \cdot c$*, and outputs a signature* $\sigma \leftarrow (c, y)$*.*

$\mathsf{Verify}(\mathsf{pk}, \mathsf{M}, \sigma = (c, y))$: *Given a public key* pk*, a message* M*, and a signature* σ*, it outputs* 1 *if* $c = H(\mathsf{pk}^{-c}\mathsf{g}^y, \mathsf{M})$ *and* 0 *otherwise.*

$\mathsf{Adapt}(\mathsf{pk}, \mathsf{M}, \sigma = (c, y), \Delta)$: *Given a public key* pk*, a message* M*, a signature* σ*, and a key update* $\Delta \in \mathbb{Z}_p$*, it computes* $\mathsf{pk}' \leftarrow \mathsf{pk} \cdot \mathsf{g}^{\Delta}$*,* $y' \leftarrow y + c \cdot \Delta$*, and outputs* $\sigma' = (c, y')$*.*

The signature scheme is EUF-CMA-secure in the random oracle model (ROM) under the discrete logarithm problem in $\mathbb{G}$ [PS96] and satisfies the signature adaptability notion of [ARS20].

A.3 Σ-Protocols

A Σ-protocol for language $\mathcal{L}$ is an interactive three move protocol between a prover and a verifier, where the prover proves knowledge of a witness $\mathtt{w}$ to $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathcal{L}}$. They are defined as follows:

Definition 11. *A* Σ*-protocol for language* $\mathcal{L}$ *is an interactive three-move protocol between a PPT prover* $\mathsf{P} = (\mathsf{Commit}, \mathsf{Prove})$ *and a PPT verifier* $\mathsf{V} = (\mathsf{Challenge}, \mathsf{Verify})$*, where* P *makes the first move and transcripts are of the form* $(\mathsf{com}, \mathsf{ch}, \mathsf{resp}) \in \mathsf{COM} \times \mathsf{CH} \times \mathsf{R}$*. They satisfy the following properties:*

Completeness: *A* Σ*-protocol is complete, if for all security parameters* λ*, and for all* $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathcal{L}}$*, it holds that*

$$\Pr\left[\langle \mathsf{P}(\mathtt{x}, \mathtt{w}), \mathsf{V}(\mathtt{x}) \rangle = 1\right] = 1.$$

s**-Special Soundness:** *A* Σ*-protocol* s*-is special sound, if there exists a PPT extractor* Ext *so that for all* $\mathtt{x}$*, and for all sets of accepting transcripts* $\{(\mathsf{com}, \mathsf{ch}_i, \mathsf{resp}_i)\}_{i \in [s]}$ *with respect to* $\mathtt{x}$ *where* $\mathsf{ch}_i \neq \mathsf{ch}_j$ *for* $i \neq j$*, generated by any PPT algorithm, it holds that*

$$\Pr\left[\begin{array}{l} \mathtt{w} \leftarrow \mathsf{Ext}\left(\mathtt{x}, \{(\mathsf{com}, \mathsf{ch}_i, \mathsf{resp}_i)\}_{i \in [s]}\right) : \\ (\mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathcal{L}} \end{array}\right] \geq 1 - \varepsilon(\lambda).$$

Special Honest-Verifier Zero-Knowledge: *A* Σ*-protocol is special honest-verifier zero-knowledge, if there exists a PPT simulator* Sim *so that for every* $\mathtt{x} \in \mathcal{L}$ *and every challenge* $\mathsf{ch} \in \mathsf{CH}$*, it holds that a transcript* $(\mathsf{com}, \mathsf{ch}, \mathsf{resp})$*, where* $(\mathsf{com}, \mathsf{resp}) \leftarrow \mathsf{Sim}(\mathtt{x}, \mathsf{ch})$ *is indistinguishable from a transcript resulting from an honest execution of the protocol.*

A.4 Fiat-Shamir Transformation

Given Σ-protocol for language $\mathcal{L}$, one can obtain a NIZK by applying the Fiat-Shamir transform [FS87]. Essentially, the transform removes the interaction between the prover and the verifier by using a hash function H (modeled as a random oracle) to obtain the challenge. That is, the algorithm $\mathsf{Challenge}$ obtains the challenge as $H(\mathsf{com}, \mathtt{x})$. We formally recall this stronger variant of the Fiat-Shamir transform [FKMV12, BPW12]. The original variant of the transform does not include $\mathtt{x}$ in the challenge generation.

Definition 12 (FS transform). *Let $(\mathsf{P}_\Sigma, \mathsf{V}_\Sigma)$ be Σ-protocol for relation $\mathcal{R}$ and H a random oracle mapping to CH. Define a NIZK for relation $\mathcal{R}$ in the random oracle model as follows:*

$\mathsf{P}_{\mathsf{FS}}(\mathtt{x}, \mathtt{w})$: *Start P_Σ on $(\mathtt{x}, \mathtt{w})$, obtain the commitment com, answer with $\mathsf{ch} \leftarrow H(\mathsf{com}, \mathtt{x})$. Obtain resp and return $\pi \leftarrow (\mathsf{com}, \mathsf{resp})$.*

$\mathsf{V}_{\mathsf{FS}}(\mathtt{x}, \pi)$: *Parse π as $(\mathsf{com}, \mathsf{resp})$. Start V_Σ on $\mathtt{x}$ and send com as first message to the verifier. When V_Σ outputs ch, reply with resp and output 1 if V_Σ accepts and 0 otherwise.*

For that transform, we require the min-entropy μ of the commitment com to be such that $2^{-\mu}$ is negligible in the security parameter λ. Furthermore, its challenge space CH needs to exponentially large in the security parameter, which can always be achieved by parallel repetition of the protocol.

A.5 Non-Interactive Proofs of Knowledge with Straight-line Extractors

Fischlin [Fis05] showed how to turn three-move proofs of knowledge into non-interactive ones in the random oracle model. Unlike the classical Fiat-Shamir transformation, Fischlin's construction ($\mathsf{Fischlin}$) supports a straight-line extractor which outputs the witness from such a non-interactive proof instantaneously, without having to rewind or fork. Additionally, the communication complexity of Fischlin's construction is significantly lower than for previous proofs with straight-line extractors. In the following we recall $\mathsf{Fischlin}$ construction.

The starting point for $\mathsf{Fischlin}$ is a Σ-protocol with logarithmic challenge length ℓ. Note that such proofs can be easily constructed from proofs with smaller challenge length d by combining ℓ/d parallel executions. $\mathsf{Fischlin}$ consists of s repetitions of the base protocol, where in each repetition i, the prover is allowed to search through challenges and responses to find a tuple $(x, \mathsf{com}, i, \mathsf{ch}, \mathsf{resp})$ whose b least significant bits of the hash are $\vec{0}^b$ for a small b. Alternatively, let H only have b output bits which can always be achieved by cutting off the leading bits. Instead of demanding that all s hash values equal $\vec{0}^b$, it gives the honest prover more flexibility and let the verifier also accept proofs $(\mathsf{com}_i, \mathsf{ch}_i, \mathsf{resp}_i)_{i=1}^s$ such that the sum of the s hash values $H(\mathtt{x}, \vec{\mathsf{com}}, i, \mathsf{ch}_i, \mathsf{resp}_i)$ (viewed as natural numbers) does not exceed some parameter S. With this we can bound the prover's number of trials in each execution by 2^t for another parameter t, slightly larger than b, and guarantee that the prover terminates in strict polynomial time.

Definition 13 (Fischlin **construction [Fis05]).** *Let* $(\mathsf{P}_\Sigma, \mathsf{V}_\Sigma)$ *be a* Σ*-protocol with challenges of* $\ell = \ell(k) = O(\log k)$ *bits for relation* $\mathcal{R}$*. Define the parameters* b*,* s*,* S*,* t *(as functions of* k*) for the number of test bits, repetitions, maximum sum and trial bits such that* $bs = \omega(\log k)$*,* $2^{t-b} = \omega(\log k)$*,* $b, s, t = O(\log k)$*,* $S = O(s)$ *and* $b \le t \le \ell$*. Define the following non-interactive proof system for relation* $\mathcal{R}$ *in the random oracle model, where the random oracle maps to* b *bits.*

$\mathsf{P}(\mathtt{x}, \mathtt{w})$: *First run the prover* P_Σ *on* $(\mathtt{x}, \mathtt{w})$ *in* s *independent repetitions to obtain* s *commitments* $\vec{\mathsf{com}} = (\mathsf{com}_1, \cdots, \mathsf{com}_s)$*. Then* P *does the following for each repetition* i*: for each* $\mathsf{ch}_{ij} = 0, 1, \cdots, 2^t - 1$ *(viewed as* t*-bit strings) it lets* P_Σ *compute the final responses* $\mathsf{resp}_{ij} = \mathsf{resp}_{ij}(\mathsf{ch}_{ij})$*, until it finds the first one such that* $H(\mathtt{x}, \vec{\mathsf{com}}, i, \mathsf{ch}_{ij}, \mathsf{resp}_{ij}) = \vec{0}^b$*; if no such tuple is found then* P *picks the first one for which the hash value is minimal among all* 2^t *hash values. The prover finally outputs* $\pi = (\mathsf{com}_i, \mathsf{ch}_{ij}, \mathsf{resp}_{ij})_{i=1}^s$.

$\mathsf{V}(\mathtt{x}, \pi)$: *Accepts if and only if* V_Σ *accepts* $\mathtt{x}$ *with* $(\mathsf{com}_i, \mathsf{ch}_i, \mathsf{resp}_i)$ *for each* $i = 1, \cdots, s$*, and if* $\sum_{i=1}^s H(\mathtt{x}, \vec{\mathsf{com}}, i, \mathsf{ch}_i, \mathsf{resp}_i) \le S$*.*

Fischlin has a small completeness error. For deterministic verifiers this error can be removed by standard techniques, e.g., by letting the prover check on behalf of the verifier that the proof is valid before outputting it.

Theorem 5 ([Fis05]). *Let* $(\mathsf{P}_\Sigma, \mathsf{V}_\Sigma)$ *be a* Σ *protocol for relation* $\mathcal{R}$*. Then the scheme* Fischlin *is a non-interactive zero-knowledge proof of knowledge for relation* $\mathcal{R}$ *(in the random oracle model) with a straight-line extractor.*

Unruh [Unr15] adapted Fischlin's strategy to obtain simulation-extractable NIZKs in the quantum ROM (QROM) that provide a straight-line extractor and avoid the completeness error. At a high level, Unruh's transform works as follows: Given a s-special-sound Σ-protocol, integers t and $M \ge s$, a statement $\mathtt{x}$ and a random permutation G, the prover will repeat the first phase of the Σ-protocol t times. For each of the t runs, it produces proofs to M different randomly selected challenges. The prover applies G to each of the so-obtained responses. The prover then selects the responses to publish for each round of the Σ-protocol by querying the random oracle on the statement, all commitments, all challenges and all permuted responses. We formally define it below.

Definition 14 (Unruh **transform).** *Let* $(\mathsf{P}_\Sigma, \mathsf{V}_\Sigma)$ *be* s*-special sound* Σ*-protocol for relation* $\mathcal{R}$ *and* H *a random oracle mapping to* $[M]^t$ *and* G *be permutation of* Σ*'s response space. Define a NIZK for relation* $\mathcal{R}$ *in the random oracle model as follows:*

$\mathsf{P}_{\mathsf{Unruh}}(\mathtt{x}, \mathtt{w})$: *1. For* $i \in [t]$*: Start* P_Σ *on* $(\mathtt{x}, \mathtt{w})$ *and obtain commitment* com_i*. Then, for* $j \in [M]$*, set* $\mathsf{ch}_{i,j} \leftarrow\!\!\$\ \mathsf{CH} \setminus \{\mathsf{ch}_{i,1}, \ldots, \mathsf{ch}_{i,j-1}\}$ *and obtain response* $\mathsf{resp}_{i,j}$ *for challenge* $\mathsf{ch}_{i,j}$*. Set* $\vec{\mathsf{ch}}_i \leftarrow (\mathsf{ch}_{i,j})_{j \in [M]}$

2. For $i, j \in [t] \times [M]$*, set* $g_{i,j} \leftarrow G(\mathsf{resp}_{i,j})$*. Set* $\vec{g}_j \leftarrow (g_{i,j})_{j \in [M]}$

3. Let $(J_1, \ldots, J_t) \leftarrow H((\mathsf{com}_i)_{i \in [t]}, (\vec{\mathsf{ch}}_i)_{i \in [t]}, (\vec{g}_i)_{i \in [t]})$*.*

4. Return $\pi \leftarrow ((\mathsf{com}_i)_{i \in [t]}, (\vec{ch}_i)_{i \in [t]}, (\vec{g}_i)_{i \in [t]}, (\mathsf{resp}_{i,J_i})_{i \in [t]})$*.*

$\mathsf{V}_{\mathsf{Unruh}}(x, \pi)$: *Parse* π *as*

$$((\mathsf{com}_i)_{i\in[t]}, (\vec{\mathsf{ch}}_i)_{i\in[t]}, (\vec{g}_i)_{i\in[t]}, (\mathsf{resp}_i)_{i\in[t]}).$$

1. *Let* $(J_1, \dots, J_t) \leftarrow H((\mathsf{com}_i)_{i\in[t]}, (\vec{\mathsf{ch}}_i)_{i\in[t]}, (\vec{g}_i)_{i\in[t]})$.
2. *For* $i \in [t]$ *check that all* $\mathsf{ch}_{i,1}, \dots, \mathsf{ch}_{i,M}$ *are pairwise distinct.*
3. *For* $i \in [t]$ *check whether* V_Σ *accepts the proof with respect to* $\mathtt{x}$*, commitment* com_i*, challenge* ch_{i,J_i} *and response* resp_i*.*
4. *For* $i \in [t]$ *check* $g_{i,J_i} = G(\mathsf{resp}_i)$*.*
5. *Output* 1 *if all checks succeeded and* 0 *otherwise.*

Theorem 6 ([Unr15]). *Let* $(\mathsf{P}_\Sigma, \mathsf{V}_\Sigma)$ *be a* Σ*-protocol for relation* $\mathcal{R}$*. Then the scheme* Unruh *is a non-interactive zero-knowledge proof of knowledge for relation* $\mathcal{R}$ *(in the random oracle model) with a straight-line extractor.*

In general, the overhead of Unruh is $t \cdot M$ for the prover and in the proof size. The verifier, however, has to invoke the verifier of the Σ-protocol only t times.

A.6 Properties of Updatable Signatures

Definition 15 (Updatable correctness). *A signature scheme* Σ *is* updatable correct*, if for all* $m \in \mathcal{M}$*, all* $(\mathsf{csk}, \mathsf{cpk}, \zeta) \leftarrow \mathsf{KGen}(1^\lambda)$ *and* $(\mathsf{up}_{\mathsf{csk}}, \mathsf{cpk}_{\mathsf{up}}, \zeta_{\mathsf{up}}) \leftarrow \mathsf{Upk}(\mathsf{cpk})$ *such that* $\mathsf{Vpk}(\mathsf{cpk}, \mathsf{cpk}_{\mathsf{up}}, \zeta_{\mathsf{up}}) = 1$*, we have* $\mathsf{Verify}(\mathsf{cpk}, m, \mathsf{Sign}(\mathsf{csk}, m)) = 1$ *and* $\mathsf{Verify}(\mathsf{cpk}_{\mathsf{up}}, m, \mathsf{Sign}(\mathsf{csk} + \mathsf{up}_{\mathsf{csk}}, m)) = 1$.

Definition 16 (Updatable strong key hiding). *For all* $(\mathsf{csk}, \mathsf{cpk}) \leftarrow \mathsf{KGen}(1^\lambda)$ *and* $(\mathsf{up}_{\mathsf{csk}}, \mathsf{cpk}_{\mathsf{up}}, \zeta_{\mathsf{up}}) \leftarrow \mathsf{Upk}(\mathsf{cpk})$*, it holds that* $(\mathsf{csk}, \mathsf{cpk}) \approx_\lambda (\mathsf{csk}_{\mathsf{up}}, \mathsf{cpk}_{\mathsf{up}}) \in \mathsf{KGen}(1^\lambda)$ *(where* $\mathsf{csk}_{\mathsf{up}} := \mathsf{csk} + \mathsf{up}_{\mathsf{csk}}$*) if one of the following settings holds:*

- cpk *was honestly generated and the key update verifies, i.e.,* $(\mathsf{csk}, \mathsf{cpk}) \leftarrow \mathsf{KGen}(1^\lambda)$ *and* $\mathsf{Vpk}(\mathsf{cpk}, \mathsf{cpk}_{\mathsf{up}}, \zeta_{\mathsf{up}}) = 1$*; or*
- cpk *verifies and the key update was honest, i.e.,* $\mathsf{Vpk}(\mathsf{cpk}, \zeta) = 1$ *and* $(\mathsf{up}_{\mathsf{csk}}, \mathsf{cpk}_{\mathsf{up}}, \zeta_{\mathsf{up}}) \leftarrow \mathsf{Upk}(\mathsf{cpk})$*.*

Definition 17 (Updatable EUF-CMA). *A signature scheme* Σ *is updatable* EUF-CMA *secure, if, for any PPT subverter* Z*, there exists a PPT extractor* $\mathsf{Ext}_{\mathsf{Z}}$ *s.t. for all PPT adversaries* $\mathcal{A}$

$$\Pr\left[\begin{array}{c} (\mathsf{csk}, \mathsf{cpk}, \zeta) \leftarrow \mathsf{KGen}(1^\lambda), \\ (\mathsf{cpk}_{\mathsf{up}}, \zeta_{\mathsf{up}}, \mathsf{aux}_{\mathsf{Z}}) \leftarrow \mathsf{Z}(\mathsf{cpk}), \\ \mathsf{up}_{\mathsf{csk}} \leftarrow \mathsf{Ext}_{Z}(\mathsf{cpk}_{\mathsf{up}}), \\ (m^\star, \sigma^\star) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{cpk}_{\mathsf{up}}, \mathsf{aux}_{\mathsf{Z}}): \\ \mathsf{Vpk}(\mathsf{cpk}, \mathsf{cpk}_{\mathsf{up}}, \zeta_{\mathsf{up}}) = 1 \wedge \\ \mathsf{cpk}_{\mathsf{up}} = \mathsf{cpk} \cdot \mu(\mathsf{up}_{\mathsf{sk}}) \wedge \\ m^\star \notin \mathcal{Q}^{\mathsf{Sign}} \wedge \mathsf{Verify}(\mathsf{cpk}_{\mathsf{up}}, m^\star, \sigma^\star) = 1 \end{array}\right] \approx_\lambda 0,$$

where $\mathcal{O} = \mathsf{Sign}(\mathsf{csk}, \cdot), \mathsf{Sign}(\mathsf{csk} + \mathsf{up}_{\mathsf{csk}}, \cdot)$*, the environment keeps track of the queries to the signing oracle via* $\mathcal{Q}^{\mathsf{Sign}}$*. Note that* Z *can also generate the initial* cpk*, which an honest updater* Upk *then updates, outputting* $\mathsf{cpk}_{\mathsf{up}}$*,* $\mathsf{up}_{\mathsf{csk}}$*, and the proof* ζ_{up}*. Then we require that* $\mathsf{Vpk}(\mathsf{cpk}, \zeta) = 1$ *and we extract* csk *by running* $\mathsf{Ext}_{\mathsf{Z}}$ *on* cpk.

Remark 3. The above definition of updatable EUF-CMA is adapted to black-box extractors, whereas the definition given in [ARS20] is with respect to non-black-box extractors.

A.7 Proof of Theorem 4

In this section, we prove Theorem 4.

Theorem 4. *Let Π be a NIZK (SNARK) scheme satisfying perfect completeness, computational updatable zero-knowledge, and computational updatable (optionally knowledge) soundness,* UP *an EKU-PKE scheme with message space $\mathcal{M}$ satisfying IND-CPA security and perfect correctness, Σ an EUF-CMA-secure updatable signature scheme, and Σ_{OT} a one-time signature scheme with strong unforgeability. Let* $\mathsf{ZK} \in \{\mathsf{Fischlin}, \mathsf{Unruh}\}$ *be a non-interactive proof of knowledge with* BB *extraction. Then the construction in Fig. 2 securely realizes $\mathcal{F}_{\mathsf{NIZK}}$ in the $\mathcal{F}_{\mathsf{up\text{-}CRS}}$-hybrid model.*

Proof. Let $\mathcal{A}$ be a non-uniform polynomial time adversary. We describe an ideal adversary $\mathsf{Sim}_{\mathsf{uc}}$ so no non-uniform polynomial time environment can distinguish whether it is running in the $\mathcal{F}_{\mathsf{up\text{-}CRS}}$-hybrid model with parties $P_1, \ldots, P_n$ and adversary $\mathcal{A}$ or in the ideal process with $\mathcal{F}_{\mathsf{NIZK}}$, $\mathsf{Sim}_{\mathsf{uc}}$ and dummy parties $\hat{P}_1, \ldots, \hat{P}_n$.

$\mathsf{Sim}_{\mathsf{uc}}$ starts by invoking a copy of $\mathcal{A}$. It will run a simulated interaction of $\mathcal{A}$, the parties, and the environment. In particular, whenever the simulated $\mathcal{A}$ communicates with the environment, $\mathsf{Sim}_{\mathsf{uc}}$ just passes this information along. And whenever $\mathcal{A}$ corrupts a party P_i, $\mathsf{Sim}_{\mathsf{uc}}$ corrupts the corresponding dummy party $\hat{P}_i$.

Simulating uncorrupted initial CRS generator in $\mathcal{F}_{\mathsf{up\text{-}CRS}}$. Suppose $\mathsf{Sim}_{\mathsf{uc}}$ receives $(\mathtt{start}, \mathtt{crs})$ from $\mathcal{F}_{\mathsf{up\text{-}CRS}}$. This means that some dummy party $\hat{P}$ received input $(\mathtt{start}, \mathsf{sid}, \mathtt{tc})$, where $\mathtt{tc} \neq \perp$. We must simulate the output a real party (updater) P would make, however. We create $\zeta_{\mathsf{up}} \leftarrow \mathsf{Sim}_{\mathsf{ZK}}(\mathtt{crs})$ and return $(\mathtt{proofCRS}, \zeta_{\mathsf{up}})$ to $\mathcal{F}_{\mathsf{up\text{-}CRS}}$. $\mathcal{F}_{\mathsf{up\text{-}CRS}}$ subsequently sends $(\mathtt{proofCRS}, \mathsf{sid}, \mathtt{crs}, \zeta)$ to $\hat{P}$ and we deliver this message so it is output to the environment.

Simulating uncorrupted updater P in $\mathcal{F}_{\mathsf{up\text{-}CRS}}$: Suppose $\mathsf{Sim}_{\mathsf{uc}}$ receives $(\mathtt{upCRS}, \mathsf{sid}, \mathtt{crs}, \mathtt{crs}_{\mathsf{up}})$ from $\mathcal{F}_{\mathsf{up\text{-}CRS}}$. This means that some dummy party $\hat{P}$ received input $(\mathtt{upCRS}, \mathsf{sid}, \mathtt{crs}, \mathtt{tc}_{\mathsf{up}})$, where $(\mathsf{sid}, \mathtt{crs}) \in Q_{\mathtt{crs}}$ and $\mathtt{tc} \neq \perp$. We must simulate the output a real party (updater) P would make, however. We create $\zeta_{\mathsf{up}} \leftarrow \mathsf{Sim}_{\mathsf{ZK}}(\mathtt{crs}, \mathtt{crs}_{\mathsf{up}})$ and return $(\mathtt{proofCRS}, \zeta_{\mathsf{up}})$ to $\mathcal{F}_{\mathsf{up\text{-}CRS}}$. The functionality $\mathcal{F}_{\mathsf{up\text{-}CRS}}$ subsequently sends $(\mathtt{proofCRS}, \mathsf{sid}, \mathtt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$ to $\hat{P}$ and we deliver this message so it is output to the environment.

Simulating uncorrupted update checker P in $\mathcal{F}_{\mathsf{up\text{-}CRS}}$: Suppose $\mathsf{Sim}_{\mathsf{uc}}$ receives $(\mathtt{checkCRS}, \mathtt{crs}, \mathtt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$ from $\mathcal{F}_{\mathsf{up\text{-}CRS}}$. This means an honest dummy party (update checker) $\hat{P}$ has received $(\mathtt{checkCRS}, \mathsf{sid}, \mathtt{crs}, \mathtt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$ from the environment. $\mathsf{Sim}_{\mathsf{uc}}$ checks the proof, $b \leftarrow \mathsf{Vcrs}(\mathtt{crs}, \mathtt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$. If invalid, it sends $(\mathtt{trapdoor}, \mathsf{no}\ \mathtt{tc}_{\mathsf{up}})$ to $\mathcal{F}_{\mathsf{up\text{-}CRS}}$ and delivers the consequent message $(\mathtt{verCRS},$

$\mathsf{sid}, 0)$ to $\hat{P}$, who outputs this rejection to the environment. Otherwise, if the update argument is valid we must try to extract a trapdoor $\mathtt{tc}_{\mathsf{up}}$. If $(\mathtt{crs}, \mathtt{crs}_{\mathsf{up}})$ has ever been proved by an honest updater that was later corrupted, we will know the $\mathtt{tc}_{\mathsf{up}}$ and do not need to run the following extraction procedure. If the trapdoor is not known already, $\mathsf{Sim}_{\mathsf{uc}}$ lets $\mathtt{tc}_{\mathsf{up}} \leftarrow \mathsf{Ext}_{\mathsf{ZK}}(\mathtt{crs}, \zeta_{\mathsf{ZK}}))$. If $\mathtt{crs}$, $\mathtt{crs}_{\mathsf{up}}$, and $\mathtt{tc}_{\mathsf{up}}$ are not consistent ($\mathtt{tc}_{\mathsf{up}}$ is invalid), it sets $\mathtt{tc}_{\mathsf{up}} = \mathsf{no}\ \mathtt{tc}_{\mathsf{up}}$. It sends $(\mathtt{trapdoor}, \mathtt{tc}_{\mathsf{up}})$ to $\mathcal{F}_{\mathsf{up\text{-}CRS}}$ and delivers the resulting output message to the update checker $\hat{P}$, who outputs it to the environment. We will later argue that the probability of the proof being valid, yet $\mathsf{Sim}_{\mathsf{uc}}$ being unable to supply a good $\mathtt{tc}_{\mathsf{up}}$ to $\mathcal{F}_{\mathsf{up\text{-}CRS}}$ is negligible. This means that with overwhelming probability, when ζ_{up} is an acceptable UC NIZK argument for $(\mathtt{crs}, \mathtt{crs}_{\mathsf{up}})$, we input a valid trapdoor $\mathtt{tc}_{\mathsf{up}}$ to $\mathcal{F}_{\mathsf{up\text{-}CRS}}$.

Simulating corruption in $\mathcal{F}_{\mathsf{up\text{-}CRS}}$: Suppose a simulated party P_i is corrupted by $\mathcal{A}$. Then we must simulate the transcript of P_i. We start by corrupting $\hat{P}_i$, thereby learning all UC proofs it has verified. It is straightforward to simulate P_i's internal tapes when running these verification processes. We also learn all updates $(\mathtt{crs}, \mathtt{crs}_{\mathsf{up}})$ that it has proved, together with their corresponding trapdoors $\mathtt{tc}_{\mathsf{up}}$. Recall that the UC NIZK arguments ζ_{up} have been provided by $\mathsf{Sim}_{\mathsf{uc}}$. Since we erased all other data, we can simulate the tape of P_i.

Simulating uncorrupted prover in $\mathcal{F}_{\mathsf{NIZK}}$: Suppose $\mathsf{Sim}_{\mathsf{uc}}$ receives $(\mathtt{prove}, \mathtt{x})$ from $\mathcal{F}_{\mathsf{NIZK}}$. This means that some dummy party $\hat{P}$ received input $(\mathtt{prove}, \mathsf{sid}, \mathtt{x}, \mathtt{w})$ where $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$. We must simulate the output a real party P would make, but we may not know $\mathtt{w}$. We create $\pi \leftarrow \mathsf{Sim}(\mathtt{crs}_{\mathsf{up}}, \mathtt{x}, \mathtt{tc}_{\mathsf{up}})$ and send $(\mathtt{proof}, \pi)$ to $\mathcal{F}_{\mathsf{NIZK}}$. $\mathcal{F}_{\mathsf{NIZK}}$ subsequently sends $(\mathtt{proof}, \mathsf{sid}, \pi)$ to $\hat{P}$ and we deliver this message so it is output to the environment.

Simulating uncorrupted verifiers: Suppose $\mathsf{Sim}_{\mathsf{uc}}$ receives the message $(\mathtt{verify}, \mathtt{x}, \pi)$ from $\mathcal{F}_{\mathsf{NIZK}}$. This means an honest dummy party $\hat{V}$ has received $(\mathtt{verify}, \mathsf{sid}, \mathtt{x}, \pi)$ from the environment. $\mathsf{Sim}_{\mathsf{uc}}$ checks the proof: $b \leftarrow \mathsf{V}(\mathtt{crs}_{\mathsf{up}}, \mathtt{x}, \pi)$. If invalid, it sends $(\mathtt{witness}, \mathsf{no\ witness})$ to $\mathcal{F}_{\mathsf{NIZK}}$ and delivers the message $(\mathtt{verification}, \mathsf{sid}, 0)$ to $\hat{V}$, who outputs this rejection to the environment. Otherwise, if the UC NIZK argument is valid, we must try to extract a witness $\mathtt{w}$. If $\mathtt{x}$ has ever been proved by an honest prover that was later corrupted, we will know the witness and do not need to run the following extraction procedure. If the witness is not known already $\mathsf{Sim}_{\mathsf{uc}}$ lets $\mathtt{w} \leftarrow \mathsf{UP.Dec}(\mathtt{tc}_{\mathsf{up}}, \mathsf{c})$. If $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}$ it sets $\mathtt{w} = \mathsf{no\ witness}$. It sends $(\mathtt{witness}, \mathtt{w})$ to $\mathcal{F}_{\mathsf{NIZK}}$. It delivers the resulting output message to $\hat{V}$, who outputs it to the environment. We will later argue that the probability of the proof being valid, yet $\mathsf{Sim}_{\mathsf{uc}}$ not being able to supply a good witness to $\mathcal{F}_{\mathsf{NIZK}}$ is negligible. This means that with overwhelming probability, when π is an acceptable UC NIZK argument for $\mathtt{x}$, we input a valid witness $\mathtt{w}$ to $\mathcal{F}_{\mathsf{NIZK}}$.

Simulating corruption: Assume a simulated party P_i is corrupted by $\mathcal{A}$. Then we must simulate the transcript of P_i. We start by corrupting $\hat{P}_i$, thereby learning all UC NIZK arguments it has verified. It is straightforward to simulate P_i's internal tapes when running these verification processes. We also learn all statements $\mathtt{x}$ that it has proved together with the corresponding witnesses $\mathtt{w}$. Recall that the UC NIZK arguments π have been provided by $\mathsf{Sim}_{\mathsf{uc}}$. Since we erased all other data, we can simulate the tape of P_i.

Hybrids. We argue that no environment can distinguish between the adversary $\mathcal{A}$ running with parties executing the UC NIZK protocol in the $\mathcal{F}_{\mathsf{up\text{-}CRS}}$-hybrid model and the ideal adversary $\mathsf{Sim}_{\mathsf{uc}}$ running in the $\mathcal{F}_{\mathsf{NIZK}}$-hybrid model with dummy parties. In order to do so we define several hybrid experiments and show that the environment cannot distinguish between any of them.

$\mathsf{H0}$: This is the $\mathcal{F}_{\mathsf{up\text{-}CRS}}$-hybrid model running with adversary $\mathcal{A}$ and parties $P_1, \ldots, P_n$.

$\mathsf{H1}$: We modify $\mathsf{H0}$ by running $(\mathtt{crs}, \mathtt{tc}, \zeta) \leftarrow \mathsf{KGen}(1^\lambda)$.[12] Given $\mathtt{crs}$ and $\mathtt{crs}_{\mathsf{up}}$, we create the proofs of uncorrupted updaters as $\zeta_{\mathsf{up}} \leftarrow \mathsf{Sim}_{\mathsf{ZK}}(\mathtt{crs}, \mathtt{crs}_{\mathsf{up}})$ and the proofs of uncorrupted provers as $\pi \leftarrow \mathsf{Sim}(\mathtt{crs}_{\mathsf{up}}, \mathtt{x}, \mathtt{tc}_{\mathsf{up}})$. By zero-knowledge, this experiment is indistinguishable from $\mathsf{H0}$.

$\mathsf{H2}$: Consider the case where an honest party (update checker) P and honest party V receive $(\mathtt{checkCRS}, \mathsf{sid}, \mathtt{crs}, \mathtt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$ and $(\mathtt{verify}, \mathsf{sid}, \mathtt{x}, \pi)$, respectively. Suppose ζ_{up} and π are indeed acceptable UC NIZK proofs and not simulated. We run $\mathtt{tc}_{\mathsf{up}} \leftarrow \mathsf{Ext}_{\mathsf{ZK}}(\mathtt{crs}, \zeta_{\mathsf{ZK}})$ and $\mathtt{w} \leftarrow \mathsf{UP.Dec}(\mathsf{c}, \mathsf{sk}_{\mathsf{up}})$. If $\mathtt{tc}_{\mathsf{up}}$ is invalid and $(\mathtt{w}, \mathtt{x}) \notin \mathcal{R}$, give up in the simulation. By the SE property there is negligible probability that we will ever give up, so $\mathsf{H2}$ is indistinguishable from $\mathsf{H1}$.

$\mathsf{H3}$: This is the ideal process running with $\mathcal{F}_{\mathsf{up\text{-}CRS}}$, $\mathsf{Sim}_{\mathsf{uc}}$, and $\mathcal{F}_{\mathsf{NIZK}}$. Inspection shows that $\mathsf{H2}$ and $\mathsf{H3}$ are identical and therefore perfectly indistinguishable to the environment.

A.8 Black-box SE Version of Sonic

In order to satisfy black-box SE, we follow the framework presented in Section 4.1: We first add the public key pk of the IND-CPA-secure EKU-PKE UP to the SRS and use a BB SE NIZK $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$ to prove update correctness. Then, we use a combination of an EUF-CMA-secure updatable signature scheme Σ with BB extraction and an sOTS scheme Σ_{OT} to add non-malleability, together with the folklore OR-trick to enable simulation of proofs. The final SRS contains Sonic's original SRS, the public key pk of UP, and the public key cpk of Σ.

Assume Sonic [MBKM19] with the SRS as in Section 5.2, the ElGamal EKU-PKE from Section 3 with public key $\mathsf{pk} = \mathsf{g}^{\mathsf{sk}}$ and secret key sk, and the Schnorr updatable signature with BB extraction from Section 2.2 with public key $\mathsf{cpk} = \mathsf{g}^{\mathsf{csk}}$ and signing key csk. The SRS update proof procedure of the black-box SE Sonic works as follows:

– Choose $\mathsf{up}_{\mathtt{tc}} := (\mathsf{up}_\alpha, \mathsf{up}_\chi) \leftarrow\!\!\$\ \mathbb{Z}_p^{*2}$ and compute $\mathsf{srs}_{\mathsf{up}} :=$

$$\big(\{(\mathsf{g}^{\chi^i})^{\mathsf{up}_\chi^i}, (\mathsf{h}^{\chi^i})^{\mathsf{up}_\chi^i}, (\mathsf{h}^{\alpha\chi^i})^{\mathsf{up}_\alpha \mathsf{up}_\chi^i}\}_{i=-d}^{i=d},$$
$$\{(\mathsf{g}^{\alpha\chi^i})^{\mathsf{up}_\alpha \mathsf{up}_\chi^i}\}_{i=-d, i\neq 0}^{i=d}\big)$$

[12] Without loss of generality, we assume the initial $\mathtt{crs}$ is honestly generated and the updating procedure is possibly maliciously executed.

together with a proof $\zeta_{\mathsf{ZK},\Pi,\mathsf{up}}$ that this computation is correctly done. More precisely, the proof $\zeta_{\mathsf{ZK},\Pi,\mathsf{up}}$ is for the language $\mathcal{L}_1 :=$

$$\left\{ \mathsf{srs}_{\mathsf{up}} \middle| \begin{array}{c} \exists (\mathsf{up}_\chi, \mathsf{up}_\alpha) \in \mathbb{Z}_p^{*2} \colon \mathsf{srs}_{\mathsf{up}} = \\ \left(\{\mathsf{srs}_1^{\mathsf{up}_\chi^i}, \mathsf{srs}_2^{\mathsf{up}_\alpha \mathsf{up}_\chi^i}\}_{i=-d}^{i=d}, \{\mathsf{srs}_3^{\mathsf{up}_\alpha \mathsf{up}_\chi^i}\}_{i=-d, i\neq 0}^{i=d} \right) \end{array} \right\}$$

where $\mathsf{srs}_1 = (\mathsf{g}^{\chi^i}, \mathsf{h}^{\chi^i})$, $\mathsf{srs}_2 = \mathsf{h}^{\alpha\chi^i}$, and $\mathsf{srs}_3 = \mathsf{g}^{\alpha\chi^i}$.

– Choose $\mathsf{up}_{\mathsf{sk}} \leftarrow\!\!\$\ \mathbb{Z}_p^*$ and compute $\mathsf{pk}_{\mathsf{up}} := \mathsf{pk}^{\mathsf{up}_{\mathsf{sk}}} = (\mathsf{g}^{\mathsf{sk}})^{\mathsf{up}_{\mathsf{sk}}}$ together with a proof $\zeta_{\mathsf{ZK},\mathsf{pk},\mathsf{up}}$ that this computation is correctly done. More precisely, the proof $\zeta_{\mathsf{ZK},\mathsf{pk},\mathsf{up}}$ is for the language

$$\mathcal{L}_2 := \{\mathsf{pk}_{\mathsf{up}} | \exists \mathsf{up}_{\mathsf{sk}} \in \mathbb{Z}_p^* \colon \mathsf{pk}_{\mathsf{up}} = \mathsf{pk}^{\mathsf{up}_{\mathsf{pk}}}\}.$$

– Choose $\mathsf{up}_{\mathsf{csk}} \leftarrow\!\!\$\ \mathbb{Z}_p^*$, compute $\mathsf{cpk}_{\mathsf{up}} := \mathsf{cpk}^{\mathsf{up}_{\mathsf{cpk}}} = (\mathsf{g}^{\mathsf{csk}})^{\mathsf{up}_{\mathsf{csk}}}$ together with a proof $\zeta_{\mathsf{ZK},\mathsf{cpk},\mathsf{up}}$ that this computation was correctly done. More precisely, the proof $\zeta_{\mathsf{ZK},\mathsf{cpk},\mathsf{up}}$ is for the language

$$\mathcal{L}_3 := \{\mathsf{cpk}_{\mathsf{up}} | \exists \mathsf{up}_{\mathsf{csk}} \in \mathbb{Z}_p^* \colon \mathsf{cpk}_{\mathsf{up}} = \mathsf{cpk}^{\mathsf{up}_{\mathsf{csk}}}\}.$$