Universally Composable NIZKs: Circuit-Succinct, Non-Malleable and CRS-Updatable

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

[1] Max Planck Institute for Security and Privacy, Bochum, Germany
`abdolmaleki.behzad.ir@gmail.com`
[2] University of Maryland, College Park, USA
`nglaeser@umd.edu`
[3] 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 (so called zk-SNARKs) increasingly see real-world adoption in large and complex systems.
A requirement that turns out to be important for NIZKs is ensuring non-malleability of proofs, which can be achieved via the property of simulation extractability (SE). Moreover, many zk-SNARKs require a trusted setup, i.e., a common reference string (CRS), and in practice it is desirable to reduce the trust in the CRS generation. Latter can be achieved via the notions of subversion or updatable CRS. Another important property when deployed in large and complex systems is the secure composition of protocols, e.g., via using the Universal Composability (UC) framework. Relying on the UC frameworks allows to arbitrarily and securely compose protocols in a modular way.
In this work, we are interested in whether zk-SNARKs can provide all these desired properties. This is a tricky task as the UC framework rules out several natural techniques for such a construction. Our main result is to show that achieving these properties is indeed possible in a generic and modular way when slightly relaxing the succinctness properties of zk-SNARKs 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 will argue that for various practical applications of zk-SNARKs this overhead is perfectly tolerable. Our starting point is a framework by Abdolmaleki *et al.* called Lamassu (ACM CCS'20) which we extend in several directions. Moreover, we implement our compiler on top of Sonic (ACM CCS'19) and provide benchmarks as well as a discussion on the choice of the required primitives.

1 Introduction

Non-Interactive Zero-Knowledge proofs (NIZKs) [GMR85, BFM88] are a fascinating and powerful primitive. They allow a party 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 credentials and digital currencies) as well as multi-party computation protocols.

Due to their numerous potential applications in privacy-preserving cryptocurrencies and blockchains in general, short NIZKs with efficient verification (at the cost of tolerating a less efficient prover), so called zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs) [BCCT12], have attracted a tremendous amount of research within the last decade. Enormous research effort has been put into developing efficient zk-SNARKs, e.g., [Gro10, Lip12, GGPR13, PHGR13, Lip13, DFGK14, Gro16, BCR$^+$19, MBKM19, GWC19, CHM$^+$20, GLS$^+$21], enabling proofs of statements that are not efficiently realizable with NIZKs for algebraic languages. The development of different types and optimized versions of existing zk-SNARKs is meanwhile exploding and progress is extremely fast-paced. There is also an increasing interest in composing zk-SNARKs in a modular way [CFQ19, CFF$^+$21], as well as generalized frameworks for their construction [RZ21]. Despite their well-known drawback of relying on assumptions that are non-falsifiable [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, toolchains around zk-SNARKs enable non-cryptography experts to easily express a statement to be proven in a familiar programming language and the corresponding implementations are generated automatically. This is possible due to the vast amount of available tools.[4]

Desirable properties of zk-SNARKs for secure adoption. Making the use and adoption of zk-SNARKs in (complex) applications easy from an engineering perspective entails the risk that the security of the entire system breaks due to the lack of some property provided by the used zk-SNARK. Two such important but not readily available properties are: *i)* *non-malleability* and *ii)* *composability*. Non-malleability means that from a given proof it is neither possible to obtain another valid proof for the same statement nor a new proof for a potentially related statement from a given proof. This can be guaranteed via the strong soundness notion of *simulation extractability* (SE) [Sah99, Sah01], which is currently intensively studied for specific zk-SNARKs [GM17, Lip19, BPR20, BKSV21, GOP$^+$22, GKK$^+$22]. Composability means that a zk-SNARK can be arbitrarily composed with other cryptographic primitives into more complex system while the security properties are still guaranteed to hold. A prominent tool to achieve this is the universal composability (UC) framework [Can01], which is very popular for modeling security for blockchain-based systems, e.g., [KMS$^+$16, AME$^+$21, TMM21, TMM22]. Ideally, it is possible to devise a generic technique for providing the SE property in the UC framework that works for a large class

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

of (if not all) zk-SNARKs and thereby makes minimal assumptions on the underlying SNARK. As we will discuss soon, this is not straightforward. Even more so, when taking the following aspect into account.

There is a large class of zk-SNARKs that require a trusted setup, i.e., the generation of a common reference string (CRS) where the CRS generator is trusted to delete the trapdoor of the CRS after the setup. Consequently, when devising a generic technique, it is also desirable that it helps to reduce the required trust. While this is possible via ceremonies [KMSV21], they are cumbersome in practice. An alternative approach is to rely on the notion of subversion NIZK [BFS16] or (zk-)SNARKs [ABLZ17, Fuc18]. Unfortunately, for zk-SNARKs this approach can only provide guarantees for the prover. A technique that represents a viable middle ground is the use of an updatable CRS [GKM+18], where everyone can update a CRS and updates can be verified by anyone. As long as one operation – the CRS creation or one of its updates – have been performed honestly, the prover can be sure that zero-knowledge holds in the presence of a malicious CRS generator and the verifier can be sure that soundness holds. This concept is getting increasingly popular [MBKM19, GWC19, CHM+20, RZ21, CFF+21, Lip22].

Hurdles to overcome. zk-SNARKs that do not require a CRS but have a transparent setup instead, e.g., [BCR+19, Set20], use the Fiat-Shamir (FS) heuristic [FS87] to obtain non-interactivity in the random oracle model (ROM) [BR93]. There are also popular zk-SNARKs that use a CRS and the FS heuristic for achieving non-interactivity [MBKM19, GWC19, CHM+20]. While [MBKM19, GWC19, CHM+20] support an updatable CRS, due to the use of FS these protocols require a rewinding extractor and are thus not compatible with the UC framework. Moreover, while there are results on SE for certain types of three-round public-coin interactive arguments [FKMV12], the aforementioned protocols are multi-round protocols and the study of SE in this setting is ongoing [GKK+22]. It would be possible to switch to alternative transformations that are straight-line extractable [Fis05, Unr15], but besides incurring a performance penalty, their application to such protocols has also not been studied for these types of proofs.

When moving to zk-SNARKs that avoid the use of FS and directly rely on knowledge assumptions, e.g., [Gro10, Gro16, Lip22], and/or require knowledge assumptions for their CRS update functionality, other issues emerge. The use of knowledge assumptions is not compatible with the UC framework per se. Though there is some recent progress into the direction of knowledge assumptions [KKK21] and the use of algebraic adversaries [ABK+21], i.e., the algebraic group model [FKL18], in UC, the results are still not generically applicable without restrictions. So the use of knowledge assumptions in UC still needs to be considered problematic. When it comes to the SE aspect, the popular zk-SNARK due to Groth [Gro16] only satisfies a weak notion of SE [BKSV21] or one requires specifically crafted designs [GM17] to achieve SE. In case of updatability of the CRS, there are either specific designs (or modifications of existing designs) [GKM+18, Lip19] and only very recently SE has been shown for some popular multi-round protocols that use FS and have an updatable CRS [GKK+22]. Nevertheless, this still leaves the problem of using knowledge assumptions for the CRS update functionality and thus no general UC compatibility.

At this point we can conclude that achieving SE in the UC framework and support for updatable CRS (if required) in the general case is a challenging task that requires a lot of protocol specific work. Moreover, for some zk-SNARKs the current state of knowledge does not allow for providing these properties.

Overcoming the hurdles. A central problem that underlies many issues discussed above is relying on non-black-box extractors, i.e., either rewinding extractors for FS or the direct use of knowledge assumptions. One well-known technique to overcome these issues when having a CRS available is to extend the CRS by a public key, to include an encryption of the witness in the proof and to extend the original statement to also show that the correct witness is encrypted [DP92]. This trick can be combined with the classical OR trick to enable unbounded simulation of proofs [DDO+01] and together this gives the SE property with a straight-line extractor. This has been previously used in the C∅C∅-framework [KZM+15] to generically obtain SE-NIZKs from NIZKs that are secure in the UC framework. Recently in [ARS20] the C∅C∅-framework has been revisited and tailored to the use with zk-SNARKs. In particular, [ARS20] uses the non-black-box extractor of the underlying zk-SNARK rather than an encryption of the witness (and thus keeps the zk-SNARKs succinct as they are, modulo some small constant overhead) and builds upon a different technique to support an unbounded simulation of proofs from [DS19]. In [ARS20] it is then shown that latter technique can be made compatible with updatable CRS and thus it yields updatable SE-SNARKs generically via a framework called Lamassu. Unfortunately, the non-black-box extraction approach makes it incompatible with the UC framework. So while switching to a black-box extractor can solve this issue at the cost of increasing the proof size by the size of the encrypted witness, there is a problem that remains with respect to updatability of the CRS. In particular, the used public-key encryption (PKE) scheme also needs to be compatible with updatability. Recent work [BS21] tries to overcome this issue by introducing what they call PKE with updatable keys. However, despite claiming to provide a black-box approach, one critical issue with the approach in [BS21] is that updatability of the PKE is based on a non-black-box extractor. In particular, extractability of the PKE with updatable keys relies on a concrete knowledge assumption and thus makes it incompatible with the UC framework.

So the state of affairs is unsatisfactory and our aim is to provide a *generic approach that is fully black-box* and thus circumvents all these problems and allows to build UC secure SE-NIZKs (with updatable CRS) from zk-SNARKs generically.

Relaxing the succinctness of zk-SNARKs. In CRS-based NIZK proofs the proof size is linear in the size of the circuit C computing the NP relation with either a multiplicative, e.g., [GOS12], or an additive overhead [KNYY19] (latter are called compact NIZK). Making them more compact either requires to rely on heavy machinery, i.e., iO [SW14] or knowledge assumption [Gro10, Lip12, GGPR13] as done within zk-SNARKs. The latter allows to obtain so called *circuit-* and *witness- succinct* zk-SNARKs where the typical succinctness requirement is that the proof size is $\mathsf{poly}(\lambda, \log|\mathsf{C}|)$ with λ being the security parameter (and this is in fact even independent of the witness). A weaker notion of the succinctness is called *circuit-succinctness* [KZM+15, KMS+16] which is a NIZK with proof size $\mathsf{poly}(\lambda, \log|\mathsf{C}|) + |\mathtt{w}|$. In other words, the size of the proofs and verification time are (quasi-)linear in the witness size $|\mathtt{w}|$, but sublinear in size of the circuit that encodes the language. As discussed above, we aim at constructing UC secure SE-NIZKs (with updatable CRS) that are circuit-succinct, which is an open problem.

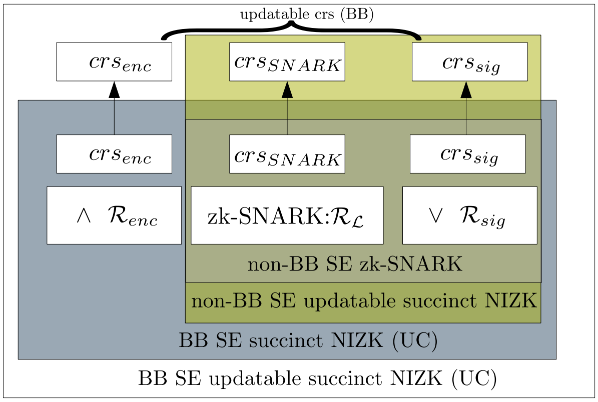
On the size of witnesses in practice. As mentioned above, achieving black-box extraction requires us to additionally encrypt the witness and thus we need to additionally include the resulting ciphertext in the proofs, incurring an additive overhead to the proof size. By using a hybrid cryptosystem for this task, e.g. combining a PKE with a symmetric encryption scheme, the size grows exactly by the witness-independent part required for the PKE and the length of the witness w (potentially with some additional small constant overhead from the symmetric encryption scheme). Consequently, our approach is not suitable for applications with huge witnesses like scalability solutions in blockchains (e.g., ZK-Rollups). However, there are many practical applications where we often deal with relatively small witnesses. One prominent example is the witnesses of the Sapling output or Spend circuits from Zcash [HBHW22]. They 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 context of self-sovereign identity [LCOK21] or anonymous credentials [RWGM22]. Blockchain applications include blockchain-based e-voting (e.g., the recently launched Vocdoni[5] or proofs of assets or swaps in cryptocurrencies [EKKV22]. Furthermore, the Merkle memberships proofs as used by Filecoin for proofs of replication[6] start from small nodes which serve as witnesses and thus our techniques also seem applicable in this context.

Concurrent and independent work. In a recent concurrent and independent work, Ganesh et al. [GKO+22] present an approach that avoids the linear dependency on the size of the witness and obtain *witness-succinct* UC SNARKs in the global random oracle model (GROM). The core idea is to replace the encryption of the witness with a succinct commitment that is straight-line extractable. While their overall approach is generic, they in particular show that the KZG polynomial commitment [KZG10] (more precisely the deterministic variant in [CHM+20]) provides all the required properties. They cleverly use

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

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

them to encode the witness as the coefficients of the polynomial and combine it with Fischlin's approach [Fis05] to obtain straight-line extractability. Unfortunately, Ganesh et al. 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 prove of the KZG eval algorithm and the witness encoding. For security parameter λ, they require at least λ elements from $\mathbb{G}_1$ and and $\mathbb{F}$ for the polynomial commitment evaluation and λ elements from $\mathbb{F}$ for the evaluation of the polynomial. Assuming a security parameter of 128 bits and consequently a pairing-friendly elliptic curve group $\mathbb{G}_1$ of at least 381 bits and a finite field of at least 256 bits, the constant overhead is at least 12.5 KB. Our approach in contrast only needs a single call to a PKE and a



updatable crs (BB)
crs_{enc}
crs_{SNARK}
crs_{sig}
crs_{enc}
crs_{SNARK}
crs_{sig}
$\wedge$ $\mathcal{R}_{enc}$
zk-SNARK:$\mathcal{R}_{\mathcal{L}}$
$\vee$ $\mathcal{R}_{sig}$
non-BB SE zk-SNARK
non-BB SE updatable succinct NIZK
BB SE succinct NIZK (UC)
BB SE updatable succinct NIZK (UC)

Fig. 1. Overview of our approach.

symmetric encryption and the constant overhead is below 0.6 KB (cf. Section 5). For large witnesses, their approach (ignoring computational costs) will clearly be superior when it comes to proof size due to being witness-succinct, but for witnesses up to 10 KB our approach is competitive. Finally, we want to mention that due to the CRS of KZG polynomial commitments, in contrast to our approach the approach by Ganesh et al. in [GKO+22] does not yield *updatable* UC SNARKs.

1.1 Our Contributions

Our contributions can be summarized as follows:

Framework for BB SE updatable succinct NIZK. We present a framework for black-box (BB) simulation extractable (SE) succinct NIZK with updatable CRS (see Figure 1). As mentioned above, for achieving BB extractability we need to relax the succinctness of SNARKs to that of succinct NIZKs [KNYY20] since we need to encrypt the witness in the proof (which adds $\mathcal{R}_{enc}$). Our starting point is the LAMASSU framework [ARS20], which represents a compiler that takes any updatable SNARK and transforms it into an SE updatable SNARK. Here it is important to mention that LAMASSU uses the non-BB extractor provided by the underlying SNARK and the notion of updatable signatures (US) for the simulation of proofs. The simulation is enabled by the OR trick [DS19] based on key-homomorphic signatures (which adds $\mathcal{R}_{sig}$). They are turned into US to support an updatable CRS. Also for the extraction from updates of the US, a non-BB extractor based on a knowledge assumption is required to finally obtain SE updatable SNARKs. Consequently, we have to overcome the hurdle to provide BB extraction for the used US as well as the new public key in the CRS crs_{enc} used for encrypting the witness. Very briefly, our key idea to providing BB extraction 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., crs_{SNARK} of the underlying SNARK, the public key of the US scheme (crs_{sig}) and the public key of the encryption scheme (crs_{enc}). We choose to base them on Σ-protocols converted to NIZK proofs either using the Fiat-Shamir (FS) [FS87], the Fischlin [Fis05] or the Unruh [Unr15] approach. While this requires that the updates of all components are Σ-protocol friendly, this holds true for all the existing relations in known constructions. Interestingly, by using this approach for the CRS of the underlying SNARK, we can make the verification of the update proofs much more efficient and typically the update proofs are also shorter than the original ones. This improvement also carries over to the original LAMASSU framework in [ARS20] and can be used to improve their CRS update proofs. We now basically use the same idea for achieving BB extraction for US and the encryption. For the latter we introduce a novel public-key encryption (PKE) primitive which we call extractable key-updatable PKE (EKU-PKE) and show an efficient instantiation.[7] We call the resulting framework BB-LAMASSU.

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.[8] While there are some recent results [KKK21, ABK+21] pointing to a promising direction of directly analyzing SNARKs in UC, we want to proceed via a generic approach that does not depend on the underlying SNARK. Since BB-LAMASSU provides BB extraction to achieve SE, following Groth [Gro06] it is possible to securely instantiate a NIZK ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$. However, so far this ignores the updatable CRS aspect. We recall that in our update proof of the CRS of the SNARK, the US and the encryption scheme, we use a FS/Fischlin/Unruh-transformed NIZK. Due to the nature of UC and our need to extract from proofs, we are prevented from rewinding extractors (as is the case for FS). Since we never need to extract from proofs of update

[7] Note that [BS21] introduced a similar notion of PKE with updatable keys independently. However, they are not focusing on BB extraction and thus this is not useful to us.

[8] In an independent work, Kerber *et al.* [KKK20] defined a functionality for updatable SRS to perform this secure generation in a distributed manner. But they do not investigate the UC-security of the whole NIZK construction.

correctness of the CRS of the underlying SNARK (we can always simulate using the OR trick), we can use FS there. But for the other two parts we need to 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.

Implementation and evaluation. In order to demonstrate the applicability of the BB-Lamassu framework, we provide a detailed analysis of the induced overheads. Except for the witness-dependent cost for encrypting the witness (which is equivalent to the size of the witness), we are able to reduce overheads imposed by the updatable signature scheme both in storage and runtime compared to Lamassu. In concrete instantiations, the overheads are 32 bytes for the CRS, 170 bytes for the CRS update, and 256 bytes plus the size of the witness for the proof. To estimate the runtime costs, we explore possible SNARK-friendly choices for the symmetric encryption used in a hybrid version of our EKU-PKE. For witness sizes observed in practical applications such as Zcash, BB-Lamassu adds well below 10,000 additional constraints.

To give one concrete application of BB-Lamassu, we describe how it can be applied to Sonic [MBKM19]. Specifically, we discuss modifications to Sonic's CRS update to make the update proofs UC-compatible. Thereby, we replace all the pairing equations with the equivalent amount of discrete logarithm relations that are proven using a FS/Fischlin/Unruh NIZK, which significantly reduces the runtime of the CRS update verifier. Additionally, we discuss the overheads of BB-Lamassu on top of Sonic. For the circuit of a preimage of SHA-256, which is interesting for Merkle-tree membership proofs, the overhead is a factor of 1.07. Our evaluation shows that as the circuits get more complex, proving and verifying the original circuit dominates the overall performance costs and the overhead converges to the size of the witness.

2 Preliminaries

Let PPT denote probabilistic polynomial-time. Let $\lambda \in \mathbb{N}$ be the security parameter. All adversaries will be stateful. By $y \leftarrow \mathcal{A}(\mathtt{x}; \omega)$ we denote the fact that $\mathcal{A}$, given an input $\mathtt{x}$ and random coins ω, outputs y. By $x \leftarrow\!\!\$\ \mathcal{D}$ we denote that x is sampled according to distribution $\mathcal{D}$ or uniformly randomly if $\mathcal{D}$ is a set. Let $\mathsf{RND}(\mathcal{A})$ 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 $\mathsf{RND}(\mathcal{A})$.[9] We denote by $\mathsf{negl}(\lambda)$ an arbitrary negligible function. We 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 three additive cyclic groups of prime order p, and $\bar{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate efficiently computable bilinear map (pairing).

[9] Assuming the given value of λ, a PPT $\mathcal{A}$ is able to read only polynomially many (in security parameter λ) symbols of the random tape.

We recall the definitions of key-homomorphic signatures, Schnorr signatures, NIZKs, Σ-protocols, the Fiat-Shamir, Fischlin and Unruh transforms in Appendices A.1 to A.6.

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

2.1 Updatable Signatures

We recall the notion of updatable signatures from [ARS20] below include their relevant properties (updatable correctness, updatable strong key hiding, and updatable EUF-CMA) Appendix A.7. For the remainder of the paper, 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]).

Definition 1 (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. [ARS20] gives a Schnorr signature-based construction which is instantiated in a bilinear group to provide update checkability and which uses a knowledge assumption for extraction. For consistency with the construction of key-updatable public-key encryption (EKU-PKE) in Section 3, which proves update validity via NIZKs, here we use the same approach by presenting an instantiation of updatable Schnorr in a prime-order multiplicative group $\mathbb{G}$ with generator g combined with a NIZK. We recall the the Schnorr signature scheme in Appendix A.2 and here only focus on the algorithms for the key update. Let (P, V) be a simulation-extractable NIZK for the relation $\mathcal{R} = \{((\mathsf{cpk}, \mathsf{cpk}_{\mathsf{up}}, g), x') : \mathsf{cpk}_{\mathsf{up}} = \mathsf{cpk} \cdot g^{x'}\}$.

$\mathsf{Upk}(\mathsf{cpk})$: 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 $\mathsf{Ext}_{\mathsf{Z}}$. Note that 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 $\mathsf{cpk}_{\mathsf{up}} = \mathsf{cpk} \cdot g^{\mathsf{up}_{\mathsf{csk}}}$.

2.2 The Lamassu Compiler

The Lamassu [ARS20] compiler lifts any CRS-based NIZK that is sound (or knowledge sound) to a non-BB simulation-extractable version. 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 a strongly unforgeable variant of Schnorr) to add the required non-malleability guarantees to the underlying NIZK proof system Π. It also uses the folklore OR-trick to add simulation soundness. In particular, during proof generation one computes a signature which certifies the public key of an sOTS instance using a freshly sampled signing key csk of Σ. Then, one uses the secret key of the sOTS to sign the parts of the proof which must be non-malleable. The distinguishing feature is that the signature is provided in plain and thus one does *not* need to encrypt it or prove that it verifies with a verification key in the CRS (e.g., as done in [Gro06]). Consequently, in the OR part of the proof, the prover only needs to prove that it knows the shift of the secret key csk (this shift is the trapdoor of the CRS) which adapts signatures from the freshly sampled cpk_o to ones valid under the verification key cpk in the CRS (a technique which 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:

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

The relation associated with $\mathcal{L}_{\mathsf{lamassu}}$ is designed so that the additional clause introduced via the OR-trick is the "shift amount" required to shift the signatures under cpk_o to signatures under a fixed key cpk in the CRS. A proof for $\mathtt{x} \in \mathcal{L}$ is easy to compute when given $\mathtt{w}$ such that $(\mathtt{x}, \mathtt{w}) \in \mathcal{L}$, and in this case, one does not need a satisfying second clause in the OR statement but can instead compute the signatures under newly generated keys. To simulate proofs, however, one can set up the CRS in a way that csk corresponding to cpk is known, compute the "shift amount", and use it as a satisfying witness for the other clause in the OR statement.

Finally, for Lamassu applied to CRS updatable NIZK proof systems, [ARS20] show the following:

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

3 Extractable Key-Updatable PKE

Now we introduce extractable key-updatable PKE (EKU-PKE), which are PKE schemes that allow one to update the keys and provide extractability of key updates. 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 does only consider possibly dishonest updates. The security guarantees of a scheme secure in our notion, however, 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 (using Lemma 6), we model only a single update, as a single adversarial update implies EKU-PKEs with arbitrarily many 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 $\mathsf{Vpk}()$:

Definition 2 (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, it outputs 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*, it outputs an updated public key* $\mathsf{pk_{up}}$ *with associated secret key updating information* $\mathsf{up_{sk}}$ *and a proof ζ_{up}.*

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

We note that in general the final $\mathsf{sk_{up}} := \mathsf{sk} \odot \mathsf{up_{sk}}$, in which depending on the scheme the operator $\odot$ might represent different operations (e.g., addition, multiplication). For our instantiation here we use multiplication.

Definition 3 ((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 4 (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* $(\mathsf{sk}, \mathsf{pk}, \zeta) \in \text{image}(\mathsf{KGen}(1^\lambda))$*,* $(\mathsf{up}_{\mathsf{sk}}, \mathsf{pk}_{\mathsf{up}}, \zeta_{\mathsf{up}}) \in \text{image}(\mathsf{Upk}(\mathsf{pk}, \zeta))$ *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* $(\mathsf{sk}_{\mathsf{up}}, \mathsf{pk}_{\mathsf{up}}, \cdot) \in \text{image}(\mathsf{KGen}(1^\lambda))$*.*

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

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

Note that $\mathcal{A}$ can also generate the initial pk, which an honest updater Upk then updates and outputs $\mathsf{pk}_{\mathsf{up}}$, $\mathsf{up}_{\mathsf{sk}}$, and the proof ζ_{up}. Then we require that $\mathsf{Vpk}(\mathsf{pk}, \zeta) = 1$.

Definition 5 (Key-updatable IND-CPA). 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* UP 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 it 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.

$\mathsf{Upk}(\mathsf{pk}, (\zeta_i)_{i=1}^{n})$: It outputs 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 $\zeta_{n+1} = \pi_{\mathsf{ZK}}$ 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} it checks 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 potentially updated public key $\mathsf{pk}_{\mathsf{up}}$, a message $\mathsf{M} \in \mathbb{G}$, and a randomness r it outputs the ciphertext $\mathsf{c} := (\mathsf{g}^r, \mathsf{M} \cdot \mathsf{pk}_{\mathsf{up}}^r)$.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{up}}, \mathsf{c})$: Given potentially updated secret key $\mathsf{sk}_{\mathsf{up}}$ and the ciphertext c, it outputs 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, updatable key correctness-(ii) follows by construction and by soundness of ZK. We reduce updatable key correctness-(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 $\mathsf{pk} \leftarrow\$ \mathbb{G}$. She 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 the $\mathcal{A}$ has in the updatable key correctness-(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 we have $\mathcal{A}$ that 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, but the only difference is that $\mathcal{A}$ receives the proof ζ_{ZK} related to the well-formedness of the pk as depicted in Definition 5.

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

$\mathsf{Game}_3:$ This game is the same as Game_2, but the only difference is that $\mathcal{A}$ updates pk and so she receives the cipher c^* under the updated $\mathsf{pk}_{\mathsf{up}}$ as depicted in Definition 5.

$\mathsf{Game}_2 \rightarrow \mathsf{Game}_3:$ 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, a EKU-PKE which supports the encryption of large 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 key-updatable IND-CPA-secure EKU-PKE with an IND-CPA-secure symmetric encryption scheme thus yields a key-updatable 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 based on black-box EKU-PKE defined in Section 3. Our focus is the setting of updatable CRS introduced by Groth *et al.* [GKM+18]. We recall that in this setting everyone can update a CRS and besides removing the trust at the prover side, also the trust on the CRS generator at the verifier side is removed as long either the generation of the CRS or any of its updates are performed honestly (e.g., by the verifier).

To construct UC-secure NIZKs in the CRS model, Kosba *et al.* [KZM+15] proposed a framework that lifts any NIZKs to UC-secure NIZKs in the CRS model. But UC-secure NIZKs with a reduction in the trusted CRS generation, e.g., via updatable CRS, is still an open problem. Indeed, to achieve UC-secure NIZKs in such a setting, one needs to guarantee the simulation extractability [Sah99, Sah01]) for the updatable NIZKs in a *black-box* way.[10] Abdolmaleki *et al.* [ARS20] proposed the LAMASSU framework (cf. Section 2.2) which allows one to transform an updatable NIZK (SNARK) to a non-black-box SE updatable NIZK (SNARK) under some non-falsifiable assumption. More precisely, with LAMASSU, one can obtain non-black-box SE updatable NIZKs (SNARK)

[10] Recall that these notions require soundness and knowledge soundness respectively to hold even if an adversary can see an arbitrary number of simulated proofs which they can adaptively obtain on statements of their choice.

such that 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 [ARS20] 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 building upon and extends the LAMASSU compiler [ARS20]. 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 key-updatable public-key encryption UP (cf. Section 3)
- A BB-extractable updatable signature Σ (cf. Section 2.1)
- A BB-extractable non-interactive proof of knowledge (knowledge sound NIZK) ZK (either FS [FS87], Fischlin [Fis05] or Unruh [Unr15])

Intuition. We can divide our approach into two parts:

*From non-*BB *to* BB *simulation extractable updatable NIZK.* The LAMASSU compiler [ARS20] adds SE and updatability. In order to satisfy *black-box* SE, we add the public key UP.pk of an IND-CPA secure extractable key-updatable PKE UP (Section 3) to the CRS used for encrypting the witness. This gives us a black-box SE version of LAMASSU compiler.

Adding UC-security. To achieve a UC-secure version of LAMASSU, we need to work with BB extraction. Thus, we replace the updatable signature of the LAMASSU compiler with a BB-extractable updatable signature Σ (defined in Section 2.1). Then, in order to satisfy BB extraction for the updating procedures of the underlying CRS and of the public key of UP, 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.

More precisely, starting from any CRS-based NIZK that is only sound instead of knowledge sound, we transfer it to the updatable setting so that one can update the CRS (i.e., similar to the technique in [GKM+18] for SNARKs and [Lip20] for QA-NIZKs). We also give a BB-extractable NIZK (using $\mathsf{ZK} \in \{\mathsf{Fischlin}, \mathsf{Unruh}\}$) 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. Now, given any language $\mathcal{L}$ with NP relation $\mathcal{R}_\mathcal{L}$, the language obtained via the BB-LAMASSU compiler is $\mathcal{L}'$ s.t. $\{\mathtt{x}' := (\mathtt{x}, \mathsf{c}, \mathsf{cpk}, \mathsf{cpk}_o), \mathtt{w}' := (\mathtt{w}, \omega, \mathsf{csk} - \mathsf{csk}_o)\} \in \mathcal{R}_{\mathcal{L}'}$ iff:

$$\mathsf{c} = \mathsf{UP.Enc}(\mathsf{pk}_{\mathsf{up}}, \mathtt{w}; \omega) \wedge$$
$$((\mathtt{x}, \mathtt{w}) \in \mathcal{R}_\mathcal{L} \vee \mathsf{cpk} = \mathsf{cpk}_o \cdot \mu(\mathsf{csk} - \mathsf{csk}_0)).$$

We present the full construction of black-box SE updatable (circuit-succinct) NIZKs in Fig. 2, where $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$. Notice that the subverter Z could be either the algorithm $\mathsf{KGen}_{\mathsf{crs}}$ or the updater algorithm Ucrs.

Remark 1. For achieving more efficient black-box updatable SE NIZKs in Fig. 2, 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.

Theorem 3. *Assume that the underlying NIZK (SNARK) scheme satisfies perfect completeness, computational zero-knowledge, and computational (knowledge) soundness. Let* $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$ *be non-interactive proofs of knowledge with* BB *extraction. Assume the extractable key-updatable encryption scheme* UP *with message space* $\mathcal{M}$ *is IND-CPA-secure and perfectly correct. Let* Σ *be a EUF-CMA secure adaptable key-homomorphic signature scheme, and that the one-time signature scheme* Σ_{OT} *is strongly unforgeable. Then the construction in Fig. 2 is a zero-knowledge proof system satisfying perfect completeness, updatable zero-knowledge, and updatable simulation sound extractability.*

Proof. **(i: Completeness):** This is straight forward 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, outputs $\mathsf{csk}_{\mathsf{up}}$. For this case adversary $\mathcal{A}$ is the adversary from Fig. 3. 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, outputs $\mathsf{csk}_{\mathsf{up}}$.

To prove updatable zero-knowledge, we use the extractor $\mathsf{Ext}_{\mathsf{ZK}}$ to obtain the trapdoor $\mathsf{csk}_{\mathsf{up}}$ and give a simulator Sim (see Fig. 2). Sim first chooses $z \leftarrow\!\!\$\ \mathcal{M}$, $\omega \leftarrow\!\!\$\ \mathsf{RND}(\mathsf{Sim})$, computes $\mathsf{c} \leftarrow \mathsf{UP.Enc}(\mathsf{pk}_{\mathsf{up}}, z; \omega)$, and produces proofs π_{Sim} when provided $\mathsf{csk}_{\mathsf{up}}$ such that any proof π_{Sim} has the same distribution as a real proof π_Π generated by the witness $\mathtt{w}$.[11] Finally Sim can generate locally $(\mathsf{sk}_l, \mathsf{pk}_l) \leftarrow \Sigma.\mathsf{KGen}(1^\lambda)$; $(\mathsf{sk}_{\mathsf{OT}}, \mathsf{pk}_{\mathsf{OT}}) \leftarrow \Sigma_{\mathsf{OT}}.\mathsf{KGen}(1^\lambda)$ and then compute $\sigma_{\mathsf{OT}} \leftarrow \Sigma_{\mathsf{OT}}.\mathsf{Sign}(\mathsf{sk}_{\mathsf{OT}}, \pi_\Pi || \mathsf{x} || \mathsf{c} || \mathsf{pk}_l || \sigma)$ such that $\pi = (\mathsf{c}, \pi_{\mathsf{Sim}}, \mathsf{pk}_l, \sigma, \mathsf{pk}_{\mathsf{OT}}, \sigma_{\mathsf{OT}})$

[11] Recall that in the OR part of the proof one just needs to prove that one 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.

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

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

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

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

has the same distribution as a real proof $\pi = (\mathsf{c}, \pi_\Pi, \mathsf{pk}_l, \sigma, \mathsf{pk}_{\mathsf{OT}}, \sigma_{\mathsf{OT}})$. Note that π_{Sim} is the simulated proof and π_Π is the real proof in the original updatable NIZK.

(iii: Black-box updatable (strong) simulation extractability): 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 the $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$ protocol 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] where we replace the underlying NIZK with an updatable NIZK (SNARK) but in a black-box manner. We use simulation the trapdoors of the EUF-CMA secure adaptable key-homomorphic 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 straight-line or rewinding extraction of the ZK proofs for the EKU-PKE scheme and the updatable signature scheme, 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 $(\mathsf{sk}_{\mathsf{up}}, \mathsf{csk}_{\mathsf{up}})$. For example, for the case of the straight-line extraction of $\mathsf{Fischlin}$, there exists a PPT extractor $\mathsf{Ext}_{\mathsf{Fischlin}}$ which, given $\mathsf{cpk}_{\mathsf{up}}$, $\mathsf{pk}_{\mathsf{up}}$, the proofs $(\zeta_{\mathsf{Fischlin},\mathsf{pk}_{\mathsf{up}}}, \zeta_{\mathsf{Fischlin},\mathsf{cpk}_{\mathsf{up}}})$, and the list of queries and answers of $Q_H(\mathcal{A})$, 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 by pointing to the line numbers in the experiment or the oracle, where the ZK proof could be either FS, $\mathsf{Fischlin}$, or Unruh.

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

Game_2: This game is the same as Game_1, but the only difference is that Z updates the $\mathtt{crs}$.

```
$\mathsf{Exp}^{\mathsf{bb\text{-}up\text{-}se}}(\mathcal{A}, \lambda)$

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

$\mathsf{O}(\mathtt{crs}_\mathsf{up}, \mathtt{tc}, \mathtt{x})$

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 : $\pi_\mathsf{Sim} \leftarrow \Pi.\mathsf{Sim}(\mathtt{crs}_\mathsf{up}, \mathtt{x}, \mathsf{c}, (z, \bot); \mathtt{tc}_{\mathsf{cpk}_\mathsf{up}});\ \sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_l, \mathsf{pk}_\mathsf{OT});$
4 : $\sigma_\mathsf{OT} \leftarrow \Sigma_\mathsf{OT}.\mathsf{Sign}(\mathsf{sk}_\mathsf{OT}, \pi_\mathsf{Sim}||\mathsf{x}||\mathsf{c}||\mathsf{pk}_l||\sigma);$
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 : $\textbf{return } \pi;$
```

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

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** $(\mathtt{crs}, \zeta)$;
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 output 1, then there is an extractor that extracts sk_up and csk_up (i.e., the straight-line trapdoor extraction of Fischlin for updatable pk and cpk, that it is possible to extract them when the adversary updates an honest CRS) 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 $\mathsf{cpk} = \mu(\Delta) \cdot \mathsf{pk}_o$.

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

$\mathsf{Game}_2 \rightarrow \mathsf{Game}_3$: 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 sound 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).

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

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

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

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, first, we need to 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 ideal functionality 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 $\mathcal{F}_{\mathsf{NIZK}}$

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

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

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

in the $\mathcal{F}_{\mathsf{up\text{-}CRS}}$ model. We note that, due to the fact that the rewinding technique (in the extraction phase in FS) is not allowed in the UC model [Can01], we assume that $\mathsf{ZK} \in \{\mathsf{Fischlin}, \mathsf{Unruh}\}$ for the instantiation of the ZK proof.

Theorem 4. *For* $\mathsf{ZK} \in \{\mathsf{Fischlin}, \mathsf{Unruh}\}$*, the protocol 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 $\mathtt{tc} \neq \bot$. 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 gets 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 $\mathtt{tc} \neq \bot$. 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 gets output to the environment.

Simulating uncorrupted updating 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 updating 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 dummy party updating checker $\hat{P}$ that outputs this rejection to the environment. Otherwise, if the updating 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 $\mathcal{F}_{\mathsf{up\text{-}CRS}}$. It delivers the resulting output message to the updating checker $\hat{P}$ that outputs it to the environment. We will later argue that the probability of the proof being valid, yet us not being able to supply a good $\mathtt{tc}_{\mathsf{up}}$ to $\mathcal{F}_{\mathsf{up\text{-}CRS}}$ is negligible. That means with overwhelming probability we input a valid trapdoor $\mathtt{tc}_{\mathsf{up}}$ to $\mathcal{F}_{\mathsf{up\text{-}CRS}}$ when ζ_{up} is an acceptable UC NIZK argument for $(\mathtt{crs}, \mathtt{crs}_{\mathsf{up}})$.

Simulating corruption in $\mathcal{F}_{\mathsf{up\text{-}CRS}}$: Suppose a simulated party P_i is corrupted by $\mathcal{A}$. Then we have to 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 updating $(\mathtt{crs}, \mathtt{crs}_{\mathsf{up}})$ that it has proved together with the corresponding trapdoor $\mathtt{tc}_{\mathsf{up}}$. Recall that the UC NIZK arguments ζ_{up} have been provided by $\mathsf{Sim}_{\mathsf{uc}}$. Since we erased all other data, we can therefore 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, however, we may not know $\mathtt{w}$. We create $\pi \leftarrow \mathsf{Sim}(\mathtt{crs}_{\mathsf{up}}, \mathtt{x}, \mathtt{tc}_{\mathsf{up}})$ and return $(\mathtt{proof}, \pi)$ 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 gets 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

$(\texttt{verification}, \mathsf{sid}, 0)$ to $\hat{V}$ that 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 $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$ it sets $\mathtt{w} = \mathsf{no\ witness}$. It sends $(\texttt{witness}, \mathtt{w})$ to $\mathcal{F}_{\mathsf{NIZK}}$. It delivers the resulting output message to $\hat{V}$ that outputs it to the environment. We will later argue that the probability of the proof being valid, yet us not being able to supply a good witness to $\mathcal{F}_{\mathsf{NIZK}}$ is negligible. That means with overwhelming probability we input a valid witness $\mathtt{w}$ to $\mathcal{F}_{\mathsf{NIZK}}$ when π is an acceptable UC NIZK argument for $\mathtt{x}$.

Simulating corruption: Assume a simulated party P_i is corrupted by $\mathcal{A}$. Then we have to 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 x that it has proved together with the corresponding witnesses w. Recall, the UC NIZK arguments π have been provided by $\mathsf{Sim}_{\mathsf{uc}}$. Since we erased all other data, we can therefore 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] and given $\mathtt{crs}$ and $\mathtt{crs}_{\mathsf{up}}$, creating 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 the zero-knowledge property the adversary this experiment is indistinguishable from $\mathsf{H0}$.

$\mathsf{H2}$: Consider the case where an honest party updating checker P and honest party V receive $(\texttt{checkCRS}, \mathsf{sid}, \mathtt{crs}, \mathtt{crs}_{\mathsf{up}}, \zeta_{\mathsf{up}})$ and $(\texttt{verify}, \mathsf{sid}, \mathtt{x}, \pi)$ respectively. Suppose ζ_{up} and π are indeed an acceptable UC NIZK proofs and ζ_{up} and π are not in the 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 in process $\mathsf{H2}$ and $\mathsf{H3}$ we are computing the different parts of the protocol in the same way. $\mathsf{H2}$ and $\mathsf{H3}$ are therefore perfectly indistinguishable to the environment.

[12] Without loss of generality, let us assume that the initial $\mathtt{crs}$ is honestly generated and then the updating procedure can be possibly maliciously done.

5 Evaluation and Instantiation

We split the evaluation into multiple parts. First we consider the costs for the CRS updates, the costs of the witness encryption and compare our framework to Lamassu. Finally, in Section 5.2 we apply BB-Lamassu to 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 SNARK specific overhead of the update proofs for the updatable CRS of the underlying SNARK. We will later discuss this with 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 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 the Unruh transform is applied to such a Σ-protocol, we are able to omit the 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 $\mathsf{Fischlin}$. 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 pick $t = 1$ since the Σ-protocol has a negligible soundness error and $M = 2$ since it is 2-special sound. Therefore, 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 cost metric. As this metric depends on the choice of the involved groups, we choose the involved groups to be SNARK-friendly such as Jubjub and can thus lift the number of constraints from [HBHW22] for the evaluation. We split the analysis of encryption with UP into the ElGamal as well as the symmetric encryption part. For UP, 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 Jubjub curve 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 with some mode of operation. Since a mode supporting parallel encryption is preferable as they allow for shallower circuits, we focus on counter (CTR) mode. All other modes require the same number or more AES evaluations. A single AES evaluation is expensive [KPS18], hence choosing one of the low multiplicative complexity block ciphers may be more desirable. While

Table 1. 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

some of those primitives have only been optimized for the use of keyless permutations to construct hashes in the context of SNARKs, Sponge-based construction with a keyless permutation yield a secure stream cipher [BDPV12].

Table 1 depicts the number of constraints for symmetric primitives and witness sizes of 1 and 32 KiB, respectively. As symmetric primitives 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 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 1 shows that also 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, the proofs for the validity of the CRS update need to be extended for the corresponding proof of the updatable signature scheme. As 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 (succinct) NIZKs, we provide a black-box SE version of Sonic [MBKM19]. While updatable structured reference strings (uSRSs)[13] are modeled in [MBKM19], this is done in a

[13] An SRS is a CRS created by sampling from some complex distribution, often involving a sampling algorithm with internal randomness that must not be revealed, since it would create a trapdoor that enables creation of convincing proofs for false statements. The SRS may be non-universal (depend on the specific relation) or universal (independent of the relation) [MBKM19].

Table 2. 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)		Verify (s)		Helped Verify (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**

non black-box way and we model their security in the setting of black-box SE. Here, a uSRS is a reference 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 Sonic, 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

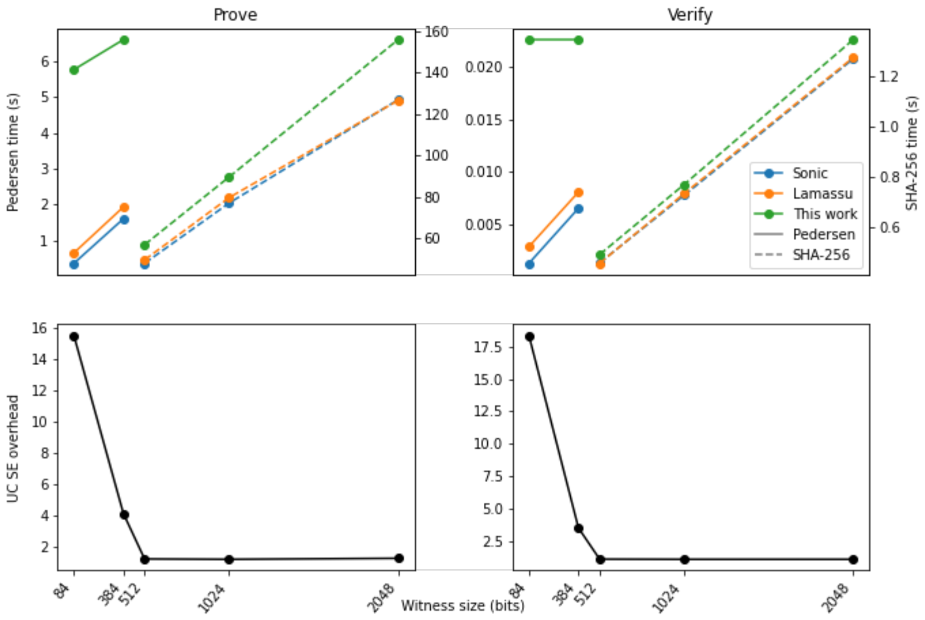


Fig. 8. Runtimes of the Prove and Verify algorithms of our BB SE succinct NIZK compared to the non-BB SE zk-SNARK obtained via Lamassu [ARS20] and the base non-SE zk-SNARK Sonic [MBKM19]. In the lower part we plot the overhead of our transformation to add BB SE, which decreases as the witness size increases.

extraction that the update is correctly done.[14] We present the changes to the uSRS and the update procedure in Appendix B. With these changes we then obtain the following result directly from Theorem 3:

Corollary 1. *Assume that the Sonic scheme satisfies perfect completeness, computational zero-knowledge, and computational (knowledge) soundness. Let* $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$ *be a non-interactive proof of knowledge with black-box extractors,* UP *be the extractable key-updatable ElGamal encryption scheme, and* Σ *be the updatable Schnorr signature scheme. Then the black-box SE Sonic is a zero-knowledge proof system satisfying perfect completeness, updatable zero-knowledge, and updatable simulation sound extractability.*

Implementation We have implemented BB-Lamassu as well as Lamassu [ARS20] in Rust[15] on top of Sonic[16] with the updatable signature scheme from Section 2.1

[14] The update in [MBKM19] reveals some intermediate shares in both source groups in order to construct a SRS verification that the update is correctly done and then uses a non-falsifiable assumption to extract the updated secret key, thus, is not black-box.

[15] `https://github.com/nglaeser/sonic-ucse/`

[16] `https://github.com/ebfull/sonic`

and the key-updatable public key encryption scheme from Section 3 (both instantiated over the Jubjub curve), and Schnorr signatures as sOTS. The required OR constraint of the transformation, which is of the form $h' = h \vee c' = c$, is represented in Sonic's constraint system as the linear constraint $(h' - h)(c' - c) = 0$.

We report in Table 2 times for proving and verifying knowledge of a hash preimage and illustrate them in Figure 8. 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, but with some fluctuations which are 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 is only a few hundred contraints. For larger circuits, such as SHA-256 or even Pedersen hash with larger inputs, the overhead of BB-Lamassu, which is on the order of 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 prover and verifier runtime, 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 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. 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.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. doi:10.1145/168588.168596.

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.

FKL18. Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96881-0_2.

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 Michal Zajac. What makes fiat-shamir zksnarks (updatable SRS) simulation extractable? In Clemente Galdi and Stanislaw Jarecki, editors, *Security and Cryptography for Networks - 13th International Conference, SCN 2022, Amalfi, Italy, September 12-14, 2022, Proceedings*, volume 13409 of *Lecture Notes in Computer Science*, 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$^+$22. Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Witness-succinct universally-composable snarks. *IACR Cryptol. ePrint Arch.*, page 1618, 2022. URL: https://eprint.iacr.org/2022/1618.

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.

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.

KKK20. Thomas Kerber, Aggelos Kiayias, and Markulf Kohlweiss. Mining for privacy: How to bootstrap a snarky blockchain. Cryptology ePrint Archive, Report 2020/401, 2020.

KKK21. 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.

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.

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.

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 6 (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 7 (Key-Homomorphic Signatures). *A signature scheme is called key-homomorphic, if it provides a secret key to public key homomorphism and an additional PPT algorithm* 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 8 (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 9. *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 DLP in $\mathbb{G}$ [PS96] and satisfies the signature adaption notion of [ARS20].

A.3 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 steps 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) X-NIZK definitions from [Gro16, GKM+18, ARS20] where $X \in \{\mathsf{trusted}, \mathsf{updatable}\}$. In other words, besides considering the standard setting with a trusted CRS generation, we also capture the updatable CRS setting. Trusted means generated by a trusted third party, or even a more general MPC protocol, and, updatable means that an adversary can adaptively generate sequences of CRSs and arbitrarily interleave its own malicious updates into them. The only constraints on the final CRS are that it is well formed and that at least one honest participant has contributed to it by providing an update (or the initial creation).

In the following we provide a formal definition of X-NIZK.

A *X-NIZK* $\Pi = (\mathsf{KGen}, \mathsf{Ucrs}, \mathsf{Vcrs}, \mathsf{P}, \mathsf{V})$ for $\mathcal{R}$ consists of the following PPT algorithms (it contains Ucrs and Vcrs when $X = \mathsf{update}$):

$\mathsf{KGen}_{\mathsf{crs}}(\mathcal{R})$: On input $\mathcal{R} \in \mathrm{image}(\mathsf{RGen}(1^\lambda))$, outputs CRS $\mathtt{crs}$, a trapdoor $\mathtt{tc}$, and a proof ζ.[17]

[17] If $X = \mathsf{trusted}$, then $\zeta = \bot$ and we may omit it.

$\mathsf{Ucrs}(\mathtt{crs}, \zeta)$: 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).

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

Definition 10. *Let* $\Pi = (\mathsf{KGen}_{\mathsf{crs}}, \mathsf{Ucrs}, \mathsf{Vcrs}, \mathsf{P}, \mathsf{V})$ *be a non-interactive argument for the relation* $\mathcal{R}$*. Then the argument* Π *is* X*-secure for* $X \in \{\mathsf{trusted}, \mathsf{updatable}\}$*, if it satisfies the following properties:*

X-Completeness. Π *is* complete for RGen*, if for all* λ*,* $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$*, and PPT algorithms* 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. If* $X = \mathsf{trusted}$ *then* A *is* $\mathsf{KGen}_{\mathsf{crs}}$ *and* $\zeta = \perp$ *and* A *is adversary* $\mathcal{A}$ *otherwise.*

X-BB simulation extractability. *For* $X = \mathsf{trusted}$*,* Π *is* BB simulation extractable for RGen*, if for every PPT* $\mathcal{A}$*, there exists a PPT extractor* Ext*,*

$$\Pr\left[\begin{array}{l}\mathcal{R} \leftarrow \mathsf{RGen}(1^\lambda),\\ (\mathtt{crs}, \mathtt{tc} := (\mathtt{tc}_{\mathsf{sim}}, \mathtt{tc}_{\mathsf{ext}})) \leftarrow \mathsf{KGen}_{\mathsf{crs}}(\mathcal{R}),\\ (\mathtt{x}, \pi) \leftarrow \mathcal{A}^{\mathsf{O}(\cdot)}(\mathcal{R}, \mathtt{crs}),\\ \mathtt{w} \leftarrow \mathsf{Ext}(\mathcal{R}, \mathtt{crs}; \mathtt{tc}_{\mathsf{ext}}):\\ (\mathtt{x}, \pi) \notin \mathcal{Q} \wedge (\mathtt{x}, \mathtt{w}) \notin \mathcal{R} \wedge\\ \mathsf{V}(\mathtt{crs}, \mathtt{x}, \pi) = 1\end{array}\right] \approx_\lambda 0.$$

Here, $\mathsf{O}(\mathtt{x})$ *returns* $\pi := \mathsf{Sim}(\mathtt{crs}, \mathtt{tc}_{\mathsf{sim}}, \mathtt{x})$ *and keeps track of all queries and the result,* $(\mathtt{x}, \pi)$*, via* $\mathcal{Q}$*. For* $X = \mathsf{updatable}$*,* Π *is* BB simulation extractable for RGen*, if for every PPT* $\mathcal{A}$ *and any subverter* Z*, there exists a PPT* *extractor* Ext,

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

Here $\mathsf{RND}(\mathsf{Z}) = \mathsf{RND}(\mathcal{A})$ *and* $(\zeta_i)_{i=1}^{n}$ *for* $n \in \mathbb{N}$ *is a number 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 2. 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.*

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

$$\varepsilon_b^{unb} = \Pr\Big[(\mathtt{crs}, \mathtt{tc}) \leftarrow \mathsf{KGen}_{\mathsf{crs}}(\mathcal{R}) \colon \mathcal{A}^{\mathsf{O}_b(\cdot,\cdot)}(\mathcal{R}, \mathtt{crs}) = 1\Big].$$

Here, oracle $\mathsf{O}_0(\mathtt{x}, \mathtt{w})$ *returns* $\perp$ *(reject) if* $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}$, *and otherwise it returns* $\mathsf{P}(\mathtt{crs}, \mathtt{x}, \mathtt{w})$. *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 ZK for RGen *if we require* $\varepsilon_0^{unb} = \varepsilon_1^{unb}$.
Notice that for the updatable ZK property, one needs to assume that at least one of the (possibly malicious) updaters does not communicate to the others. This guarantees ZK property, if all updataers are malicious (i.e. let assume the split adversarial model in a way that the updating is done by two advertisers $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *but they do not share their secret values (*$\mathtt{tc}_1$ *and* $\mathtt{tc}_2$*) to each other), as none of them has access to whole CRS trapdoor* $\mathtt{tc}$ *(containing both* $\mathtt{tc}_1$ *and* $\mathtt{tc}_2$*).*[18]

[18] Or with a bit stronger assumption that the split adversarial model, one may simply assume that one of the updating is honestly done.

For $X = \mathsf{updatable}$, Π is statistically unbounded X-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 $\mathsf{RND}(\mathsf{Z}) = \mathsf{RND}(\mathcal{A})$, the oracle $\mathsf{O}_0(\mathtt{x}, \mathtt{w})$ returns $\perp$ (reject) if $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}$, and otherwise it returns $\mathsf{P}(\mathtt{crs}, \mathtt{x}, \mathtt{w})$. 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 X-ZK for RGen *if one requires that $\varepsilon_0^{unb} = \varepsilon_1^{unb}$.*

We note that depending the BB-extraction technique (like rewinding or straight-line extraction) aux is some auxilary information.

A.4 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*

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

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.5 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 (modelled 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 achived by parallel repetition of the protocol.

A.6 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* $b \leq t \leq \ell$*. 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* $\sum_{i=1}^s H(\mathtt{x}, \vec{\mathsf{com}}, i, \mathsf{ch}_i, \mathsf{resp}_i) \leq S$*.*

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 $M \geq s$, 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* $(J_1,\ldots,J_t) \leftarrow H((\mathsf{com}_i)_{i\in[t]}, (\vec{\mathsf{ch}}_i)_{i\in[t]}, (\vec{g}_i)_{i\in[t]})$.
2. *For* $i \in [t]$ *check that all* $\mathsf{ch}_{i,1},\ldots,\mathsf{ch}_{i,M}$ *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.7 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* Ext_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* Ext_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.

B Black-box SE Version of Sonic

In order to satisfy black-box SE, we first add the public key $\mathsf{UP.pk}$ of the IND-CPA secure extractable key-updatable PKE UP in the SRS and give a simulation-extractable NIZK with black-box extraction (such as FS, $\mathsf{Fischlin}$, and Unruh) that the update is correctly done. Then, following the framework presented in Section 4.1, we use a combination of an updatable EUF-CMA secure updatable signature scheme Σ with BB extraction and a strongly unforgeable one-time signature (sOTS) scheme Σ_{OT} to add the required non-malleability guarantees to Sonic together with the folklore OR-trick to enable simulation of proofs. The final SRS contains Sonic's original SRS, the public key $\mathsf{UP.pk}$ of the updatable IND-CPA secure extractable key-updatable PKE UP, and the public key cpk of the updatable EUF-CMA secure updatable signature scheme Σ.

Assume Sonic [MBKM19] with the SRS as in Section 5.2, the extractable key-updatable ElGamal encryption scheme from Section 3 with the public key $\mathsf{pk} = \mathsf{g}^{\mathsf{sk}}$ and the secret key sk, and the updatable Schnorr signatures with BB extraction from Section 2.1 with the public key $\mathsf{cpk} = \mathsf{g}^{\mathsf{csk}}$ and the signing key csk. Then, we describe the SRS update-proof procedure of the black-box SE Sonic as follows:

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

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

together with a proof $\zeta_{\mathsf{ZK},\Pi,\mathsf{up}}$ that this computation is correctly done, where $\mathsf{ZK} \in \{\mathsf{FS}, \mathsf{Fischlin}, \mathsf{Unruh}\}$. 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

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

– 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 $\zeta_{\mathsf{ZK},\mathsf{cpk},\mathsf{up}}$ such that this computation was correctly done. More precisely, the proof $\zeta_{\mathsf{ZK},\mathsf{cpk},\mathsf{up}}$ is computed for the language

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