How to Use SNARKs in Universally Composable Protocols

Ahmed Kosba[†] Zhichao Zhao[*] Andrew Miller[†] Yi Qian[‡] Hubert Chan[*]
Charalampos Papamanthou[†] Rafael Pass[‡] abhi shelat[•] Elaine Shi[‡]

†: UMD ∗: HKU ‡: Cornell •: UVA

Abstract

The past several years have seen tremendous advances in practical, general-purpose, non-interactive proof systems called SNARKs. These building blocks are efficient and convenient, with multiple publicly available implementations, including tools to compile high-level code (e.g., written in C) to arithmetic circuits, the native representation used by SNARK constructions. However, while we would like to use these primitives in UC-secure protocols—which are provably-secure even when composed with other arbitrary concurrently-executing protocols—the SNARK definition is not directly compatible with this framework, due to its use of non black-box knowledge extraction. We show several constructions to transform SNARKs into UC-secure NIZKs, along with benchmarks and an end-to-end application example showing that the added overhead is tolerable.

Our constructions rely on embedding cryptographic algorithms into the SNARK proof system. Ordinarily, cryptographic constructions are chosen and tuned for implementation on CPUs or in hardware, not as arithmetic circuits. We therefore also explore SNARK-friendly cryptography, describing several protocol parameterizations, implementations, and performance comparisons for encryption, commitments, and other tasks. This is also of independent interest for use in other SNARK-based applications.

1 Introduction

Succinct Non-Interactive ARguments of Knowledge (SNARKs) [9,24] are cryptographic building blocks that allow a prover to produce *computationally sound* proofs for any NP statements (under the pre-processing model). The proofs are *succinct*, and verification takes time proportional to the length of the inputs and outputs, rather than the length of the computation. Known SNARK constructions typically also offer a *zero-knowledge* option (henceforth referred to as zk-SNARKs), i.e., a prover can prove that an NP statement is true, without revealing any information about the witness.

Despite its well-known drawback of relying on non-falsifiable assumptions, SNARKs are an attractive object to practitioners due to its (relative) practical efficiency as well as universality. Recent developments in SNARKs have been exciting, and are exemplary of pushing cryptography research towards practical adoption. Specifically, various SNARK implementations have been made open source, and compiler support has also caught up that allows the compilation of general programs to SNARK implementations [6, 22, 37, 43].

SNARKs have numerous applications. A line of research focused on verifiable outsourcing of data and computation to an untrusted cloud provider [6, 21, 22, 30, 37, 41, 43–46]. More recently, several works also started adopting zk-SNARKs for multi-party protocols to achieve security against potentially malicious players — in particular, numerous applications of zk-SNARKs

have been identified in the in cryptocurrency space [5, 23, 35]. In comparison with adopting customized zero-knowledge proofs or alternative constructions of generic zero-knowledge proofs, zk-SNARKs offer numerous advantages: 1) due to their universality and available compiler support, they allow rapid protocol prototyping; 2) they do not require full-scale protocol redesign when the functionality requirements change (as is often needed for customized protocols), since it is easy to modify a zk-SNARK to prove a different statement; and 3) their relative efficiency and non-interactivity make zk-SNARKs broadly applicable in numerous applications.

SNARK lifting for UC-secure protocols. Despite the attractiveness of zk-SNARKs to practitioners, unfortunately zk-SNARKs often cannot be readily adopted in composable, simulation-secure protocols, due to a gap in the security offered by zk-SNARKs and the security often required by universally composable (UC) protocols [14, 15, 17]. Specifically, UC-secure protocols would often require *simulation sound extractable* zero-knowledge proofs. In other words, suppose that a simulator answers an adversary's queries on polynomially many, possibly false statements — nevertheless, whenever the adversary submits a proof for a new statement, the simulator will be able to extract a valid witness except with negligible failure probability. Unfortunately, known zk-SNARKs do not offer such strong soundness properties. Despite the fact that SNARKs allow "knowledge extraction", SNARKs' knowledge extractor is too weak for many UC-secure protocols, since SNARKs' extractor is *non-blackbox*, and requires knowledge of the concrete algorithm of the adversarial prover. By contrast, in UC-secure protocols, the simulator must extract witnesses without knowing the environment's algorithm.

1.1 Our Results and Contributions

SNARK-lifting transformations optimized for concrete efficiency. Our work proposes *efficient* SNARK-lifting transformations that allow us to transform zk-SNARKs to zero-knowledge proofs with *simulation sound extractability*, such that they could be adopted in UC-secure protocols.

Although the theoretical feasibility of lifting ordinary, computationally-sound, non-interactive zero-knowledge proofs (NIZKs) to *simulation sound extractability* ones appears to be folklore knowledge[1], to the best of our knowledge, ours is the *first* endeavor to optimize such SNARK-lifting transformations for *concrete efficiency*. To achieve this goal, we tailor our transformations specifically for SNARKs (e.g., Section A.2), and moreover, we make new contributions in designing SNARK-friendly cryptographic primitives as explained below.

SNARK-friendly cryptography. To optimize SNARK-lifting transformations and attain concrete efficiency, we investigated "SNARK-friendly cryptographic primitives". Specifically, SNARK-lifting transformations would typically require evaluating cryptographic primitives such as encryption, commitments, and/or signatures schemes as part of a SNARK circuit. Existing cryptographic primitives are typically selected and optimized for performance on modern processor hardware, and may therefore be suboptimal when expressed as an arithmetic circuit for use within a SNARK. "SNARK-friendly cryptography" refers to a class of cryptographic building blocks (e.g., encryption, commitments, signatures) that are efficient as arithmetic circuits.

We note that while we primarily use SNARK-friendly cryptography to improve the concrete performance of our SNARK-lifting constructions, the SNARK-friendly primitives can be of independent interest and practical in other (non-simulation-secure) applications as well.

Evaluation and open source. Our evaluation results suggest promising performance gains. For individual primitives such as encryption, we achieve **10x** to **40x** speedup in comparison

[1]For example, the work by De Santis et al. transforms an ordinary NIZK to a simulation sound NIZK [40]. Their construction can be modified to achieve *simulation sound extractability*, which is a stronger property than simulation soundness.

with a straightforward, unoptimized implementation. For our SNARK-lifting transformation as a whole — assuming that the application-specific NP language has a small witness-validating circuit — we demonstrate **5x** performance gains in comparison with a naive, unoptimized SNARK-lifting transformation. Finally, in realistic, end-to-end applications where the application-specific NP language has a moderate-sized circuit [29], we demonstrate a **2x** improvement over an unoptimized implementation. In this case, the speedup is smaller because our optimizations were mainly related to a small fraction of the entire circuit (which includes the application-specific NP language).

Our security-lifted zk-SNARKs are generally useful in many multi-party cryptographic protocols. To make our work reusable by the community, we are in the process of open sourcing our code and implementations.

Caveats and future work. Our SNARK-lifting transformations result in zero-knowledge proofs whose sizes are linear in the witness, but still independent of the size of the computation. We note that there is also no known simulation sound extractable zero-knowledge proof construction that achieves proof size sublinear in the witness length (even under non-standard assumptions). We leave it as a future research question how or whether we can achieve this.

1.2 Related Work

Universally composable protocols. Universal Composability (UC) was initially proposed by Canetti [14] and later improved in subsequent works [15, 17]. At a high level, UC gives a formal definition framework that tells us when and how cryptographic building blocks may be considered as "idealized, composable boxes" in protocol design. In principle, UC may allow designing larger-scale, secure systems in a modular manner.

Non-interactive zero-knowledge proofs. Non-interactive zero-knowledge proofs were first proposed by Blum, Feldman, and Micali [11], and later extended to multi-theorem by Blum et al. [12].

Sahai [39] was the first to construct a one-time, simulation-sound NIZK scheme. De Santis et al. [40] subsequently provide unbounded simulation-sound NIZKs, allowing the adversary to access many simulated proofs of possibly false statements. Both schemes above are not practical. Simulation soundness is a slightly weaker condition than simulation sound extractability—the latter requires that even after seeing a polynomial number of simulated proofs of possibly false statements, whenever a polynomial-time adversary produces a valid proof, an extractor can extract a valid witness except with negligible failures. It has been observed [26, 27] that simulation sound extractable NIZKs are UC-secure NIZKs by Canetti's definition [14] in the presence of a static adversary. Groth et al. [27] construct perfect NIZK arguments for circuit satisfiability using bilinear groups. They also extend their scheme to construct UC-secure NIZKs. Groth [26] also gave more practical, simulation-sound extractable NIZK constructions for an NP language for bilinear groups.

All of the above simulation-sound extractable NIZKs are not *succinct*—namely the size of the proof is proportional to the size of the witness verification circuit $|C|$ that encodes the language. In this context, succinctness has been shown not to be possible by Gentry and Wichs [25] unless non-falsifiable assumptions are adopted. The line of research on SNARKs adopt non-falsifiable assumptions to attain succinctness as well as practical efficiency [6, 21, 22, 24, 37, 43].

Subsequent works that apply our SNARK-lifting transformations. Our SNARK-lifting transformations yield proofs proportional to the witness size (but independent of the circuit size $|C|$) and requires encoding into the SNARK circuit various primitives like signatures or encryption schemes, for which we manifest various efficient circuits.

Subsequent works [28, 29] have since adopted our SNARK-lifting transformation. For exam-

ple, recent works that adopt SNARKs in cryptocurrency protocols [28, 29] used our SNARK-lifting transformations to attain UC security in protocol design. These works adopted an earlier, naive version of our construction described in Section 3.1 of this paper, and without SNARK-friendly cryptography optimizations.

Since then, we have attained better transformations and SNARK-friendly implementations, allowing us to demonstrate 2x performance improvements in end-to-end applications such as Hawk [29]. To the best of our knowledge, we are the first to consider efficient SNARK circuits for such SNARK-lifting transformations, which can be useful in a broad class of composable protocols.

2 Preliminaries

Notation. In the remainder of the paper, $f(\lambda) \approx g(\lambda)$ means that there exists a negligible function $\nu(\lambda)$ such that $|f(\lambda) - g(\lambda)| < \nu(\lambda)$.

2.1 Non-Interactive Zero-Knowledge Proofs

A non-interactive zero-knowledge proof system (NIZK) for an NP language $\mathcal{L}$ consists of the following algorithms:

- $\mathsf{crs} \leftarrow \mathcal{K}(1^\lambda, \mathcal{L})$, also written as $\mathsf{crs} \leftarrow \mathsf{KeyGen}_{\text{nizk}}(1^\lambda, \mathcal{L})$: Takes in a security parameter λ, a description of the language $\mathcal{L}$, and generates a common reference string crs.
- $\pi \leftarrow \mathcal{P}(\mathsf{crs}, \mathsf{stmt}, w)$: Takes in crs, a statement stmt, a witness w such that $(\mathsf{stmt}, w) \in \mathcal{L}$, and produces a proof π.
- $b \leftarrow \mathcal{V}(\mathsf{crs}, \mathsf{stmt}, \pi)$: Takes in a crs, a statement stmt, and a proof π, and outputs 0 or 1, denoting accept or reject.
- $(\widehat{\mathsf{crs}}, \tau, \mathsf{ek}) \leftarrow \widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$: Generates a simulated common reference string $\widehat{\mathsf{crs}}$, trapdoor τ, and extract key ek
- $\pi \leftarrow \widehat{\mathcal{P}}(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt})$: Uses trapdoor τ to produce a proof π without needing a witness

Perfect completeness. A NIZK system is said to be perfectly complete, if an honest prover with a valid witness can always convince an honest verifier. More formally, for any $(\mathsf{stmt}, w) \in R$, we have that

$$\Pr\left[\begin{array}{c} \mathsf{crs} \leftarrow \mathcal{K}(1^\lambda, \mathcal{L}),\ \pi \leftarrow \mathcal{P}(\mathsf{crs}, \mathsf{stmt}, w) : \\ \mathcal{V}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1 \end{array}\right] = 1$$

Computational zero-knowlege. Informally, a NIZK system is computationally zero-knowledge, if the proof does not reveal any information about the witness to any polynomial-time adversary. More formally, a NIZK system is said to computationally zero-knowledge, if for all non-uniform polynomial-time adversary $\mathcal{A}$, we have that

$$\begin{aligned} & \Pr\left[\mathsf{crs} \leftarrow \mathcal{K}(1^\lambda, \mathcal{L}) : \mathcal{A}^{\mathcal{P}(\mathsf{crs}, \cdot, \cdot)}(\mathsf{crs}) = 1\right] \\ \approx\ & \Pr\left[(\widehat{\mathsf{crs}}, \tau, \mathsf{ek}) \leftarrow \widehat{\mathcal{K}}(1^\lambda, \mathcal{L}) : \mathcal{A}^{\widehat{\mathcal{P}}_1(\widehat{\mathsf{crs}}, \tau, \cdot, \cdot)}(\widehat{\mathsf{crs}}) = 1\right] \end{aligned}$$

In the above, $\widehat{\mathcal{P}}_1(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt}, w)$ verifies that $(\mathsf{stmt}, w) \in \mathcal{L}$, and if so, outputs $\widehat{\mathcal{P}}(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt})$ which simulates a proof without knowing a witness. Otherwise, if $(\mathsf{stmt}, w) \notin \mathcal{L}$, the experiment aborts.

Computational soundness. A NIZK scheme for the language $\mathcal{L}$ is said to be computationally sound, if for all polynomial-time adversaries $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} \mathsf{crs} \leftarrow \mathcal{K}(1^\lambda, \mathcal{L}), (\mathsf{stmt}, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) : \\ (\mathcal{V}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1) \wedge (\mathsf{stmt} \notin \mathcal{L}) \end{array}\right] \approx 0$$

Simulation sound extractability. Simulation sound extractability says that even after seeing many simulated proofs, whenever the adversary makes a new proof, a simulator is able to extract a witness. Simulation extractability implies simulation soundness and non-malleability, since if the simulator can extract a valid witness from an adversary's proof, the statement must belong to the language. More formally, a NIZK system is said to be simulation sound extractable, if there exists a polynomial-time algorithm $\mathcal{E}$, such that for any polynomial-time adversary $\mathcal{A}$, it holds that

$$\Pr\left[\begin{array}{l}(\widehat{\mathsf{crs}},\tau,\mathsf{ek})\leftarrow\widehat{\mathcal{K}}(1^\lambda,\mathcal{L});\\(\mathsf{stmt},\pi)\leftarrow\mathcal{A}^{\widehat{\mathcal{P}}(\widehat{\mathsf{crs}},\tau,\cdot)}(\widehat{\mathsf{crs}},\mathsf{ek});\\w\leftarrow\mathcal{E}(\widehat{\mathsf{crs}},\mathsf{ek},\mathsf{stmt},\pi):\mathsf{stmt}\notin Q\text{ and}\\\qquad\qquad(\mathsf{stmt},w)\notin\mathcal{L}\text{ and }V(\widehat{\mathsf{crs}},\mathsf{stmt},\pi)=1\end{array}\right]\approx 0$$

In the above, Q is the list of simulation queries. Here the $\widehat{\mathcal{K}}$ is identical to the zero-knowledge simulation setup algorithm when restricted to the first two terms.

Note that in the above definition, the adversary may be able to fake a (different) proof for a statement that has been queried, however, it is not able to forge a proof for any other invalid statement. There is a natural strengthening of the above notion where the adversary cannot even fake a different proof for a statement queried (in fact, it is this stronger notion that is given as the default in [26]). We define and give constructions for this later in Section 5. In Hawk [29], however, it is shown that the weaker notion defined above suffices for a typical UC application; therefore we focus on this notion first.

2.2 Succinct Non-Interactive ARguments of Knowledge (SNARKs)

A SNARK is a NIZK scheme that is perfectly complete, computationally zero-knowledge, and with the additional properties of being succinct and having a knowledge extractor (which is a stronger property than soundness):

Succinctness. A SNARK is said to be succinct if an honestly generated proof has $\mathsf{poly}(\lambda)$ bits and that the verification algorithm $\mathcal{V}(\mathsf{crs},\mathsf{stmt},\pi)$ runs in $\mathsf{poly}(\lambda)\cdot O(|\mathsf{stmt}|)$ time.

Knowledge extraction. Knowledge extraction property says that if a proof generated by an adversary is accepted by the verifier, then the adversary "knows" a witness for the given instance. Formally, a SNARK for language $\mathcal{L}$ satisfies the knowledge extraction property *iff:*

For all polynomial-time adversary $\mathcal{A}$, there exists a polynomial-time extractor $\mathcal{E}$, such that for all uniform advice string z,

$$\Pr\left[\begin{array}{ll}\mathsf{crs}\leftarrow\mathcal{K}(1^\lambda,\mathcal{L})\\(\mathsf{stmt},\pi)\leftarrow\mathcal{A}(\mathsf{crs},z) & :\ \begin{array}{l}\mathcal{V}(\mathsf{crs},\mathsf{stmt},\pi)=1\\(\mathsf{stmt},a)\notin\mathcal{R}_\mathcal{L}\end{array}\\a\leftarrow\mathcal{E}(\mathsf{crs},z)\end{array}\right]\approx 0$$

Note that the knowledge extraction property implies computationally soundness (defined for NIZK), as a valid witness is extracted.

Remarks about knowledge extraction. It has been shown that a SNARK knowledge extractor cannot exist for all distributions of advice strings [10, 13]. In this paper, we will first give SNARK-lifting transformations that do not rely on the SNARKs' knowledge extractor (see Sections 3.1 and 4.1). However, we show in Section 4.2 that although the UC simulator cannot directly make use of the SNARK's non-blackbox extractor, by making use of the SNARK's knowledge extractor in other ways in security reductions, we can further optimize our SNARK-lifting transformation for concrete performance. In such cases, we will assume that a SNARK knowledge extractor exists for the particular distributions on advice strings that we care about in this paper (despite the impossibility result for general distributions).

3 Basic SNARK Lifting Transformation

3.1 Lifting from an Ordinary NIZK

In this section, we show a construction that transforms any NIZK to one that satisfies simulation sound extractability.

Typically, a NIZK only guarantees soundness, which simply means that if the verifier accepts a proof, then the statement must be in the language. However, in many cases, what we actually desire is to gurantee that the prover actually "knows" a valid witness. For example, given a collision-resistant hash function, it is necessarily true that a collision *exists*, though to actually *compute* such a collision would be an impressive feat. The definition of simulation sound extractability captures the desired knowledge property - given a valid proof (and the trapdoor produced during setup), the extractor algorithm can efficiently compute a witness.

Intuition. Our first construction makes use of an asymmetric signature scheme and encryption scheme, the public keys for which are embedded in the setup parameters, and the private keys for which are embedded in the trapdoor. The idea is to force every prover to encrypt a witness and a signature, at least one of which must be legitimate. While an honest prover will simply provide a valid witness, the simulated prover will use the trapdoor to provide a signature. The extractor can simply use the trapdoor decryption key to recover a valid witness (or at least a signature) from the proof. This guarantees that an adversary who breaks the system can be leveraged to either break the soundness of the underlying NIZK or the unforgeability of the signature scheme.

Construction. In the following, assume Σ is an unforgeable signature scheme, and $(\mathsf{KeyGen}_{\mathrm{Enc}}, \mathsf{Enc}, \mathsf{Dec})$ is a perfectly correct public key encryption scheme.

- $\mathcal{K}(1^\lambda, \mathcal{L})$: Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$. Run $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$.
  Let $\mathcal{L}'$ be the following language: $((\mathsf{stmt}, c), (r, w, \sigma)) \in \mathcal{L}'$ *iff*

$$(c = \mathsf{Enc}(\mathsf{pk}_e, (w, \sigma), r)) \wedge$$
$$((\mathsf{stmt}, w) \in \mathcal{L} \vee (\Sigma.\mathcal{V}(\mathsf{pk}, \mathsf{stmt}, \sigma) = 1))$$

  Run $\mathsf{nizk.crs} \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$.
  Publish $\mathsf{crs} := (\mathsf{nizk.crs}, \mathsf{pk}, \mathsf{pk}_e)$ as the common reference string.
- $\mathcal{P}(\mathsf{crs}, \mathsf{stmt}, w)$: Parse $\mathsf{crs} := (\mathsf{nizk.crs}, \mathsf{pk})$. Choose random r, and compute $c := \mathsf{Enc}(\mathsf{pk}_e, (w, \perp), r)$.
  Call $\pi := \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}, c), (r, w, \perp))$, and output $\pi' := (c, \pi)$.
- $\mathcal{V}(\mathsf{crs}, \mathsf{stmt}, \pi')$: Parse $\pi' := (c, \pi)$, and output $\mathsf{nizk}.\mathcal{V}(\mathsf{nizk.crs}, (\mathsf{stmt}, c), \pi)$.
- $\widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$: Run the honest $\mathcal{K}$ algorithm, but retain the signing key sk as the simulation trapdoor $\tau := \mathsf{sk}$. The extraction key $\mathsf{ek} := \mathsf{sk}_e$, the simulated $\widehat{\mathsf{crs}} := \mathsf{crs} = (\mathsf{nizk.crs}, \mathsf{pk}, \mathsf{pk}_e)$.
- $\widehat{\mathcal{P}}(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt})$: the simulator calls $\pi := \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}, c), (\perp, \perp, \sigma))$ where $\sigma := \Sigma.\mathsf{Sign}(\mathsf{sk}, \mathsf{stmt})$ and c is an encryption of $(\perp, \sigma)$. Output (c, π).

Theorem 1. *Assume that the underlying NIZK scheme satisfies perfect completeness, computational soundness, and computational zero-knowlege, that the signature scheme satisfies existential unforgeability under chosen message attack, and that the encryption scheme is semantically secure and perfectly correct, then the above construction is a zero-knowledge proof system satisfying perfect completeness, computational zero-knowledge, and simulation sound extractability.*

The proof for the above theorem is relatively simple, which we defer to appendix.

In the appendix, we also discuss two minor optimizations for this construction: in one, we reduce the size of the ciphertext by encrypting either the witness or signature, but not both; and if the underlying ciphertext is a SNARK, we can use the non-blackbox knowledge extractor to avoid needing to encrypt the signature at all.

3.2 Optimizations

This basic SNARK-lifting construction can be further optimized for concrete performance. We defer the details of these optimizations to Appendix A, and instead focus on describing a more efficient construction that avoids evaluating signatures inside a SNARK circuit, but instead relies on pseudorandom functions and commitments.

4 Improved SNARK Lifting Transformations

The previous section demonstrates the possibility of upgrading any NIZK to an SSE-NIZK. However, the construction relies on the use of signature scheme within an arithmetic circuit, which limits its performance.

In this section, we first show a more efficient construction that avoids the use of signature schemes (Section 4.1). Next, in Section 4.2 we describe a SNARK-specific optimization that leverages the SNARK's non-blackbox knowledge extractor — this may seem counter-intuitive at first sight since a UC simulator must extract without knowing the environment's algorithm; however, we show that the SNARK's non-blackbox extractor can be used to improve the efficiency of the lifting transformation, even though the UC simulator never uses the SNARK's knowledge extractor to extract the witness.

4.1 Lifting from an Ordinary NIZK

Our construction makes use of a pseudo-random function and a perfectly-binding commitment scheme, which together replace the original signature scheme.

Intuition. Recall the intuition for a pseudorandom function f: without the knowledge of the key a, $f_a(\cdot)$ behaves like a true random function. However, given a, one can compute $f_a(\cdot)$ easily. In order to use this in lieu of a signature, we include a commitment to a in the public parameters, and keep a (and the commitment opening) as the trapdoor.

We then design a transformed language such that a prover with a correct witness can pass; otherwise, the (simulated) prover must give $f_a(\mathsf{stmt})$ and an opening of the commitment to the same a.

Notation. We adopt several conventions for the convenience of readers following along with our experiment proofs. Sometimes we need placeholders that correspond to different variables in the experiments. To avoid confusing the names, we reserve $z_0, z_1, \ldots$ for this purpose. For example, z_3 replaces witness "w" in Expt_0, although z_3 need not be a valid witness. To highlight the difference between successive experiments, we color the new line red, and may reproduce the ~~previous line it replaces~~, striken through.

Construction. Let $\{f_s : \{0,1\}^* \to \{0,1\}^\lambda\}_{s\in\{0,1\}^\lambda}$ be a pseudo-random function family, let comm be a perfectly binding commitment scheme, and let $(\mathsf{KeyGen}_{\mathrm{Enc}}, \mathsf{Enc}, \mathsf{Dec})$ be a semantically secure encryption scheme.

To simplify our description, we assume Enc and comm both take exactly λ random bits as randomness and that the witness for $\mathcal{L}$ is exactly λ bits; it is straightforward to adapt the proof when they are of different lengths.

Note that in the language $\mathcal{L}'$, c must be a correct encryption of some w and μ, which allows the extractor to decrypt. For a statement-witness pair to be valid, either a witness in $\mathcal{R}_\mathcal{L}$ is provided or an opening to ρ together with the value of $f_a(\mathsf{stmt})$ is provided, where a is the opened value of ρ (from crs).

For language $\mathcal{L}$ with NP relation $\mathcal{R}_\mathcal{L}$, let $\mathcal{L}'$ be the language defined as $\big((\mathsf{stmt}, c, \mathsf{pk}_e, \rho), (\mu, r, r', w, a)\big) \in$

$\mathcal{R}_{\mathcal{L}'}$ *iff*:

$$c = \mathsf{Enc}(\mathsf{pk}_e, (w, \mu); r) \wedge \Big((\mathsf{stmt}, w) \in \mathcal{R}_{\mathcal{L}} \vee \big(\rho = \mathsf{comm}(a; r') \wedge \mu = f_a(\mathsf{stmt})\big)\Big)$$

Our SSE-NIZK construction is then defined as follows:

- $\mathcal{K}(1^\lambda, \mathcal{L})$:

  $\mathsf{nizk.crs} \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;
  $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$;
  return $\mathsf{crs} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$.

- $\mathcal{P}(\mathsf{crs}, \mathsf{stmt}, w)$:

  Parse $\mathsf{crs} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$; Abort if $(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$;
  $z_0, z_1, z_2, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, (w, z_0); r_1)$;
  $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}, c, \mathsf{pk}_e, \rho), (z_0, r_1, z_1, w, z_2))$;
  return $\pi := (c, \mathsf{nizk}.\pi)$.

- $\mathcal{V}(\mathsf{crs}, \mathsf{stmt}, \pi)$:

  Parse $\mathsf{crs} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$ and $\pi := (c, \mathsf{nizk}.\pi)$;
  Call $\mathsf{nizk}.\mathcal{V}(\mathsf{nizk.crs}, (\mathsf{stmt}, c, \mathsf{pk}_e, \rho), \mathsf{nizk}.\pi)$.

- $\widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$: Run $\mathcal{K}$ to get $\widehat{\mathsf{crs}} := \mathsf{crs}$, but keep trapdoor $\tau := (s_0, r_0)$, extraction key $\mathsf{ek} := \mathsf{sk}_e$.

- $\widehat{\mathcal{P}}(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt})$:

  Parse $\widehat{\mathsf{crs}} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$ and $\tau := (s_0, r_0)$;
  $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $\mu = f_{s_0}(\mathsf{stmt})$; $c = \mathsf{Enc}(\mathsf{pk}_e, (z_3, \mu); r_1)$;
  $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}, c, \mathsf{pk}_e, \rho), (\mu, r_1, r_0, z_3, s_0))$;
  return $\pi := (c, \mathsf{nizk}.\pi)$.

- We also define the extractor here:
  $\mathcal{E}(\widehat{\mathsf{crs}}, \mathsf{ek}, \mathsf{stmt}, \pi)$: Parse $\pi := (c, \mathsf{nizk}.\pi)$; $(w, \mu) \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$; return w.

Theorem 2. *Assume that the underlying NIZK scheme satisfies perfect completeness, computational soundness, computational zero-knowledge, and that the encryption scheme is semantically secure and perfectly correct, and that the pseudo-random function family is secure, and that the commitment scheme is perfectly binding and computational hiding, then the above construction is a zero-knowledge proof system satisfying perfect completeness, computational zero-knowledge, and simulation sound extractability.*

Proof. Completeness is obvious.

Proof of simulation sound extractability.

Lemma 1. *The construction is simulation sound extractable.*

Proof. We define the simulation soundness extractablility experiment as follows:

$\underline{\mathsf{Expt}_0 \text{ (Actual game)}}$:

1. Setup:

   $\mathsf{nizk.crs} \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;
   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x)$:

$z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $\mu = f_{s_0}(\mathsf{stmt}_x)$; $c = \mathsf{Enc}(\mathsf{pk}_e, (z_3, \mu); r_1)$;
$\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho), (\mu, r_1, r_0, z_3, s_0))$;
return $\pi := (c, \mathsf{nizk}.\pi)$.

3. $(\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{O(\cdot)}(\widehat{\mathsf{crs}}, \mathsf{ek})$;
4. Parse $\pi := (c, \mathsf{nizk}.\pi)$; $(w, \mu) \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$;
5. Let Q be the set of stmt_x queried by $\mathcal{A}$.
   Output 1 *iff*: (1) $\mathsf{stmt} \notin Q$; and (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$; and (3) $(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$.

Note that this is exactly the definition. We next show that $\Pr[\mathsf{Expt}_0] = \mathsf{negl}(\lambda)$ by a series of hybrid games, which proves simulation sound extractablility.

Expt_1 (Relax return condition):

1. Setup:
   $\mathsf{nizk.crs} \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;
   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$.
2. Define function $O(\mathsf{stmt}_x)$:
   $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $\mu = f_{s_0}(\mathsf{stmt}_x)$; $c = \mathsf{Enc}(\mathsf{pk}_e, (z_3, \mu); r_1)$;
   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho), (\mu, r_1, r_0, z_3, s_0))$;
   return $\pi := (c, \mathsf{nizk}.\pi)$.
3. $(\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{O(\cdot)}(\widehat{\mathsf{crs}}, \mathsf{ek})$;
4. Parse $\pi := (c, \mathsf{nizk}.\pi)$; $(w, \mu) \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$;
5. Let Q be the set of stmt_x queried by $\mathcal{A}$.
   Output 1 *iff*: (1) $\mathsf{stmt} \notin Q$; and (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$; and (3) $f_{s_0}(\mathsf{stmt}) = \mu$.

Claim 1. *If the underlying encryption scheme is perfectly correct, and that the commitment scheme scheme is perfectly binding, and that the underlying NIZK is computationally sound, then we have* $\Pr[\mathsf{Expt}_0] \leq \Pr[\mathsf{Expt}_1] + \mathsf{negl}(\lambda)$.

Proof. From the (computational) soundness of the underlying NIZK, we know that $(\mathsf{stmt}, c, \mathsf{pk}_e, \rho) \in \mathcal{L}'$ holds except for negligible probability.

Since the underlying encryption scheme is perfectly correct, the decrypted (w, μ) is the only possible values that encrypts to c, hence it is unique for all valid witnesses. Given $(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$, we consider all such valid witnesses for $\mathcal{R}_{\mathcal{L}'}$, there must exist s_0', r_0' such that: (1) $\rho = \mathsf{comm}(s_0'; r_0')$; (2) $\mu = f_{s_0'}(\mathsf{stmt})$. From the perfectly binding property of the underlying commitment scheme, all witnesses must use the unique value $s_0' = s_0$ (recall s_0 is from setup).

Hence, assuming soundness holds, we have $f_{s_0} = \mu$. □

Expt_2 (Use simulation setup):

1. Setup:
   $(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\widehat{\mathcal{K}}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;
   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.
2. Define function $O(\mathsf{stmt}_x)$:
   $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $\mu = f_{s_0}(\mathsf{stmt}_x)$; $c = \mathsf{Enc}(\mathsf{pk}_e, (z_3, \mu); r_1)$;
   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}_1(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho), (\mu, r_1, r_0, z_3, s_0))$;
   Equivalent: $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho))$;

return $\pi := (c, \mathsf{nizk}.\pi)$.

3. $(\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{O(\cdot)}(\widehat{\mathsf{crs}}, \mathsf{ek})$;
4. Parse $\pi := (c, \mathsf{nizk}.\pi)$; $(w, \mu) \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$;
5. Let Q be the set of stmt_x queried by $\mathcal{A}$.
   Output 1 *iff*: (1) $\mathsf{stmt} \notin Q$; and (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$; and (3) $f_{s_0}(\mathsf{stmt}) = \mu$.

Claim 2. *Assuming the underlying NIZK is computationally zero-knowledge, we have* $\Pr[\mathsf{Expt}_1] \leq \Pr[\mathsf{Expt}_2] + \mathsf{negl}(\lambda)$.

Proof. Given any polynomial adversary $\mathcal{A}$, we construct the following adversary $\mathcal{B}$ for the zero-knowledge game of the underlying NIZK.

Run Expt_2, with $\mathsf{nizk}.\widehat{\mathsf{crs}}$ obtained from the zero-knowledge game. Replace all calls to $\mathsf{nizk}.\widehat{\mathcal{P}}_1$ with oracle calls to the game. Note that we do not have $\mathsf{nizk}.\tau$, which is not used anymore.

Observe that $\mathsf{Expt}_1, \mathsf{Expt}_2$ corresponds to running the game with honest and simulated setup/prover, respectively. I.e., $\Pr[\mathcal{B}] = \Pr[\mathsf{Expt}_1]$ with honest setup and $\Pr[\mathcal{B}] = \Pr[\mathsf{Expt}_2]$ with simulation setup. As the underlying NIZK is computationally zero-knowledge, and that $\mathsf{Expt}_2^{\mathcal{A}}$ runs in polynomial time, the claim holds. □

Expt_3 (Separate s_0):

1. Setup:
   $(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\widehat{\mathcal{K}}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;
   $s_0, s_0', r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0'; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.
2. Define function $O(\mathsf{stmt}_x)$:
   $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $\mu = f_{s_0}(\mathsf{stmt}_x)$; $c = \mathsf{Enc}(\mathsf{pk}_e, (z_3, \mu); r_1)$;
   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho))$;
   return $\pi := (c, \mathsf{nizk}.\pi)$.
3. $(\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{O(\cdot)}(\widehat{\mathsf{crs}}, \mathsf{ek})$;
4. Parse $\pi := (c, \mathsf{nizk}.\pi)$; $(w, \mu) \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$;
5. Let Q be the set of stmt_x queried by $\mathcal{A}$.
   Output 1 *iff*: (1) $\mathsf{stmt} \notin Q$; and (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$; and (3) $f_{s_0}(\mathsf{stmt}) = \mu$.

Claim 3. *If the underlying commitment scheme is computationally hiding, we have* $\Pr[\mathsf{Expt}_2] \leq \Pr[\mathsf{Expt}_3] + \mathsf{negl}(\lambda)$.

Proof. By the hiding property of the commitment scheme, we know that for all polynomial adversary $\mathcal{B}$, we have

$$\Pr[x_0, x_1 \xleftarrow{\$} \{0,1\}^\lambda; b \xleftarrow{\$} \{0,1\} : \mathcal{B}(x_0, x_1, \mathsf{comm}(x_b)) = b] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

Consider the following adversary $\mathcal{B}$: Run $\mathsf{Expt}_2^{\mathcal{A}}$, except with $s_0 = x_1, s_0' = x_0$ and $\rho = \mathsf{comm}(x_b)$. Return the output of Expt_2.

Observe that getting a commitment of s_0 actually corresponds to $\mathsf{Expt}_2^{\mathcal{A}}$, while s_0' corresponds to $\mathsf{Expt}_3^{\mathcal{A}}$. Hence by the hiding property, we have

$$\begin{aligned}
\Pr[B = b] =& \frac{1}{2}\Pr[\mathcal{B} = 0 | b = 0] + \frac{1}{2}\Pr[\mathcal{B} = 1 | b = 1] \\
=& \frac{1}{2}(1 - \Pr[\mathsf{Expt}_3]) + \frac{1}{2}\Pr[\mathsf{Expt}_2] \\
\leq& \frac{1}{2} + \mathsf{negl}(\lambda)
\end{aligned}$$

which gives $\Pr[\mathsf{Expt}_2] - \Pr[\mathsf{Expt}_3] \leq \mathsf{negl}(\lambda)$. □

Expt_4 (Replace PRF):

Let F be a true random function.

1. Setup:
$$(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\widehat{\mathcal{K}}(1^\lambda, \mathcal{L}'); \ (\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda);$$
$$\cancel{s_0}, s'_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda; \ \rho := \mathsf{comm}(s'_0; r_0); \ \widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho).$$
2. Define function $O(\mathsf{stmt}_x)$:
$$z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda; \ \mu \leftarrow F(\mathsf{stmt}_x); \ c = \mathsf{Enc}(\mathsf{pk}_e, (z_3, \mu); r_1);$$
$$\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho));$$
$$\mathsf{return} \ \pi := (c, \mathsf{nizk}.\pi).$$
3. $(\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{O(\cdot)}(\widehat{\mathsf{crs}}, \mathsf{ek})$;
4. Parse $\pi := (c, \mathsf{nizk}.\pi)$; $(w, \mu) \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$;
5. Let Q be the set of stmt_x queried by $\mathcal{A}$.
Output 1 *iff*: (1) $\mathsf{stmt} \notin Q$; and (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$; and (3) $F(\mathsf{stmt}) = \mu$.

Claim 4. *If the underlying pseudo-random function family $\{f_s\}_{s\in\{0,1\}^\lambda}$ is secure, then we have* $\Pr[\mathsf{Expt}_3] \le \Pr[\mathsf{Expt}_4] + \mathsf{negl}(\lambda)$.

Proof. By the security of the underlying pseudo-random function family, no polynomial-time algorithm can distinguish a PRF from a true random function F.

We construct the following adversary $\mathcal{B}$ for the security game of PRF: Run $\mathsf{Expt}_4^{\mathcal{A}}$, replace each call to $F(\cdot)$ with an oracle call to the game. Return the output of Expt_4.

Observe that we have $\Pr[\mathcal{B} = 1|\text{Run with } \mathsf{PRF}] = \Pr[\mathsf{Expt}_3]$ and also $\Pr[\mathcal{B} = 1|\text{Run with } \mathsf{Random}] = \Pr[\mathsf{Expt}_4]$, which completes the proof. □

Claim 5. *We have* $\Pr[\mathsf{Expt}_4] \le 2^{-\lambda}$.

Proof. Since we have $\mathsf{stmt} \notin Q$, we can view $F(\mathsf{stmt})$ as newly generated random bits independent from μ. The result follows. □

The above claims complete the proof for simulation sound extractability. □

Proof of computational zero-knowledge. We prove the following lemma first:

Lemma 2. *If* $(\mathsf{KeyGen}_{Enc}, \mathsf{Enc}, \mathsf{Dec})$ *is a semantically secure encryption scheme, then for all polynomial adversary $\mathcal{A}$, we have the following:*

$$\Pr\left[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{Enc}(1^\lambda) : \mathcal{A}^{O^0(\mathsf{pk},\cdot,\cdot)}(\mathsf{pk}) = 1\right]$$
$$\approx \Pr\left[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{Enc}(1^\lambda) : \mathcal{A}^{O^1(\mathsf{pk},\cdot,\cdot)}(\mathsf{pk}) = 1\right]$$

where $O^b(\mathsf{pk}, m_0, m_1)$: *Abort if* $|m_0| \neq |m_1|$. *Return* $\mathsf{Enc}(pk, m_b; U_\lambda)$.

Proof. Let t^* be the polynomial bound on the maximum number of queries. We define oracles $O_t(m_0, m_1)$: Abort if $|m_0| \neq |m_1|$. Return $\mathsf{Enc}(\mathsf{pk}, m_1; U_\lambda)$ for the the first t queries and $\mathsf{Enc}(\mathsf{pk}, m_0; U_\lambda)$ otherwise.

Observe that $O_0 = O^0$ and $O_{t^*} = O^1$. We next prove O_t and O_{t+1} are indistinguishable by the security game of the encryption.

We construct the following adversary $\mathcal{B}$ for the security game of the encryption: Get pk from the game; Run $\mathcal{A}^{O_t(\cdot,\cdot)}$ with the $(t+1)$-th query answered by oracle call to the game. Return the output of $\mathcal{A}$. Observe that $\Pr[\mathcal{B}|\text{ansered by } O^0] = \Pr[\mathcal{A}^{O_t}]$ and $\Pr[\mathcal{B}|\text{ansered by } O^1] = \Pr[\mathcal{A}^{O_{t+1}}]$

By the cipher-text indistinguishability of the encryption scheme, we have $\Pr[\mathcal{A}^{O_t}] \approx \Pr[\mathcal{A}^{O_{t+1}}]$ □

Lemma 3. *The construction is computational zero-knowledge.*

Proof. We prove zero-knowledge by showing any adversary $\mathcal{A}$ can not distinguish a series of hybrid games.

Expt_0 (actual game):

1. Setup (just $\widehat{\mathcal{K}}$):

   $\mathsf{nizk.crs} \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x, w)$: (just $\widehat{\mathcal{P}}_1$)

   Abort if $(\mathsf{stmt}_x, w) \notin \mathcal{R}_\mathcal{L}$; $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $\mu = f_{s_0}(\mathsf{stmt}_x)$; $c = \mathsf{Enc}(\mathsf{pk}_e, (z_3, \mu); r_1)$;

   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho), (\mu, r_1, r_0, z_3, s_0))$;

   return $\pi := (c, \mathsf{nizk}.\pi)$.

3. $b \leftarrow \mathcal{A}^{O(\cdot,\cdot)}(\widehat{\mathsf{crs}})$; Output b.

Expt_1 (use simluation setup):

1. Setup:

   $(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\widehat{\mathcal{K}}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x, w)$:

   Abort if $(\mathsf{stmt}_x, w) \notin \mathcal{R}_\mathcal{L}$; $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $\mu = f_{s_0}(\mathsf{stmt}_x)$; $c = \mathsf{Enc}(\mathsf{pk}_e, (z_3, \mu); r_1)$;

   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}_1(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho), (\mu, r_1, r_0, z_3, s_0))$;

   Equivalent: $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho))$;

   return $\pi := (c, \mathsf{nizk}.\pi)$.

3. $b \leftarrow \mathcal{A}^{O(\cdot,\cdot)}(\widehat{\mathsf{crs}})$; Output b.

Claim 6. *If the underlying NIZK is zero-knowledge, then we have* $\Pr[\mathsf{Expt}_0] \leq \Pr[\mathsf{Expt}_1] + \mathsf{negl}(\lambda)$.

Proof. Consider the following adversary $\mathcal{B}$ for the zero-knowledge game: Get crs from the game; Run Expt_1 with crs; Replace all calls to $\widehat{\mathcal{P}}_1$ with oracle calls. Output the same as Expt_1

By the zero-knowledge property, we have $\Pr[\mathcal{B}|\text{honest}] \approx \Pr[\mathcal{B}|\text{simulated}]$. Observe that $\Pr[\mathcal{B}|\text{honest}] = \Pr[\mathsf{Expt}_0]$ while $\Pr[\mathcal{B}|\text{simulated}] = \Pr[\mathsf{Expt}_1]$. □

Expt_2 (encrypt true witness):

1. Setup:

   $(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\widehat{\mathcal{K}}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x, w)$:

   Abort if $(\mathsf{stmt}_x, w) \notin \mathcal{R}_\mathcal{L}$; ~~$z_3$~~, $z_0, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; ~~$\mu = f_{s_0}(\mathsf{stmt}_x)$~~; $c = \mathsf{Enc}(\mathsf{pk}_e, (w, z_0); r_1)$;

   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho))$;

   return $\pi := (c, \mathsf{nizk}.\pi)$.

3. $b \leftarrow \mathcal{A}^{O(\cdot,\cdot)}(\widehat{\mathsf{crs}})$; Output b.

Claim 7. *If the underlying encryption scheme is secure, we have* $\Pr[\mathsf{Expt}_1] \approx \Pr[\mathsf{Expt}_2]$.

Proof. We construct the following adversary $\mathcal{B}$ for the game in Lemma 2: Get pk from the game; Run Expt_2 with pk; Replace Enc with oracle calls to the game (providing both $(z_3, \mu), (w, z_0)$ as two messages). Output as Expt_2.

By Lemma 2, we have $\Pr[\mathcal{B}|\text{encrypting } w] \approx \Pr[\mathcal{B}|\text{encrypting } z_3]$. And observe they corresponds to Expt_2 and Expt_1. □

Expt_3 (Use $\mathsf{nizk}.\widehat{\mathcal{P}}_1$):

1. Setup:

   $(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\widehat{\mathcal{K}}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x, w)$:

   Abort if $(\mathsf{stmt}_x, w) \notin \mathcal{R}_\mathcal{L}$; $z_0, z_1, z_2, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $\mu \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, (w, z_0); r_1)$;

   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}_1(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho), (z_0, r_1, z_1, w, z_2))$;

   return $\pi := (c, \mathsf{nizk}.\pi)$.

3. $b \leftarrow \mathcal{A}^{O(\cdot,\cdot)}(\widehat{\mathsf{crs}})$; Output b.

Claim 8. *We have* $\Pr[\mathsf{Expt}_3] = \Pr[\mathsf{Expt}_2]$.

Proof. The two probabilities are equal, by the definition of $\mathsf{nizk}.\widehat{\mathcal{K}}_1$ and that (z_0, r_1, z_1, w, z_2) is valid. □

Expt_4 (Back to $\mathcal{P}$):

1. Setup:

   $\mathsf{nizk.crs} \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x, w)$:

   Abort if $(\mathsf{stmt}_x, w) \notin \mathcal{R}_\mathcal{L}$;

   $z_0, z_1, z_2, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, (w, z_0); r_1)$;

   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho), (z_0, r_1, z_1, w, z_2))$;

   return $\pi := (c, \mathsf{nizk}.\pi)$.

3. $b \leftarrow \mathcal{A}^{O(\cdot,\cdot)}(\widehat{\mathsf{crs}})$; Output b.

Claim 9. *If the underlying NIZK is zero-knowledge, then we have* $\Pr[\mathsf{Expt}_4] \approx \Pr[\mathsf{Expt}_3]$.

Proof. We construct the following adversary $\mathcal{B}$ for the zero-knowledge game: Get crs from the game; Run $\mathsf{Expt}_4^\mathcal{A}$ with crs; Replace $\mathsf{nizk}.\mathcal{P}$ with oracle calls. Return the output of $\mathsf{Expt}_4^\mathcal{A}$.

By the zero-knowledge property, we have $\Pr[\mathcal{B}|\text{honest}] \approx \Pr[\mathcal{B}|\text{simulated}]$. And observe the two probabilities correspond two $\Pr[\mathsf{Expt}_4]$ and $\Pr[\mathsf{Expt}_3]$. □

Note that the last experiment is just the honest prover, which completes the proof of the zero-knowledge property.

□

□

4.2 SNARK-Specific Optimizations

We now show that the non-blackbox extractor from SNARK can be used to simplify and improve the performance of the construction in Section 4.1.

Intuition. This may seem counter-intuitive: the SNARK knowledge extractor is inherently incompatible with the simulation-based definition, since the extractor assumes knowledge of the entire program code and auxilary inputs of the prover, which are not available to the simulator in the SSE-NIZK game. As we shall see, while our proof relies on a reduction to the SNARK knowledge extraction game, the simulator itself never uses this extractor.

Note that in the construction in Section 4.1, the encryption of μ is not actually used (in the extractor). It is useful only because we wish to leverage the adversary to break the pseudo-random function family, and thus must extract the value of $f_a(\mathsf{stmt})$ from the proof returned by the adversary.

When the underlying NIZK is a SNARK, we can actually omit the encryption of μ. The rationale is that in our reduction, we can use the SNARK's non-blackbox extractor to produce a valid witness instead of extracting it from the ciphertext. The argument can follow a similar argument to break the pseudo-random function family, given an adversary for the constructed SSE-NIZK.

We stress that although we use the non-blackbox extractor (in the proof), our simulator extractor is still blackbox.

Construction. For language $\mathcal{L}$ with NP relation $\mathcal{R}_{\mathcal{L}}$, let $\mathcal{L}'$ be the language $\big((\mathsf{stmt}, c, \mathsf{pk}_e, \rho), (\mu, r, r', w, a)\big) \in \mathcal{R}_{\mathcal{L}'}$ *iff*:

$$c = \mathsf{Enc}(\mathsf{pk}_e, w; r) \wedge \Big((\mathsf{stmt}, w) \in \mathcal{R}_{\mathcal{L}} \vee \big(\rho = \mathsf{comm}(a; r') \wedge \mu = f_a(\mathsf{stmt})\big)\Big)$$

Our SSE-NIZK from SNARK construction is defined as follows:

- $\mathcal{K}(1^\lambda, \mathcal{L})$:

  $\mathsf{snark.crs} \leftarrow \mathsf{snark}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

  $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; return $\mathsf{crs} := (\mathsf{snark.crs}, \mathsf{pk}_e, \rho)$.

- $\mathcal{P}(\mathsf{crs}, \mathsf{stmt}, w)$:

  Parse $\mathsf{crs} := (\mathsf{snark.crs}, \mathsf{pk}_e, \rho)$; Abort if $(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$;

  $z_0, z_1, z_2, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, w; r_1)$;

  $\mathsf{snark}.\pi \leftarrow \mathsf{snark}.\mathcal{P}(\mathsf{snark.crs}, (\mathsf{stmt}, c, \mathsf{pk}_e, \rho), (z_0, r_1, z_1, w, z_2))$;

  return $\pi := (c, \mathsf{snark}.\pi)$.

- $\mathcal{V}(\mathsf{crs}, \mathsf{stmt}, \pi)$:

  Parse $\mathsf{crs} := (\mathsf{snark.crs}, \mathsf{pk}_e, \rho)$ and $\pi := (c, \mathsf{snark}.\pi)$;

  Call $\mathsf{snark}.\mathcal{V}(\mathsf{snark.crs}, (\mathsf{stmt}, c, \mathsf{pk}_e, \rho), \mathsf{snark}.\pi)$.

- $\widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$: Run $\mathcal{K}$ to get $\widehat{\mathsf{crs}} := \mathsf{crs}$, but keep trapdoor $\tau := (s_0, r_0)$, extraction key $\mathsf{ek} := \mathsf{sk}_e$.

- $\widehat{\mathcal{P}}(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt})$:

  Parse $\widehat{\mathsf{crs}} := (\mathsf{snark.crs}, \mathsf{pk}_e, \rho)$ and $\tau := (s_0, r_0)$;

  $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $\mu = f_{s_0}(\mathsf{stmt})$; $c = \mathsf{Enc}(\mathsf{pk}_e, z_3; r_1)$;

  $\mathsf{snark}.\pi \leftarrow \mathsf{snark}.\mathcal{P}(\mathsf{snark.crs}, (\mathsf{stmt}, c, \mathsf{pk}_e, \rho), (\mu, r_1, r_0, z_3, s_0))$;

  return $\pi := (c, \mathsf{snark}.\pi)$.

- We also define the extractor here:
  $\mathcal{E}(\widehat{\mathsf{crs}}, \mathsf{ek}, \mathsf{stmt}, \pi)$: Parse $\pi := (c, \mathsf{snark}.\pi)$; $w \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$; return w.

Theorem 3. *Assume that the underlying SNARK scheme satisfies perfect completeness, computational zero-knowledge, and knowledge extraction (which implies soundness), and that the encryption scheme is semantically secure and perfectly correct, and that the pseudo-random function family is secure, and that the commitment scheme is perfectly binding and computational hiding, then the above construction is a zero-knowledge proof system satisfying perfect completeness, computational zero-knowledge, and simulation sound extractability.*

Proof. Completeness is obvious. Next we prove simulation sound extractability and zero-knowledge.

Proof of simulation sound extractable.

Lemma 4. *The construction is simulation sound extractable.*

Proof. We define the simulation soundness extractablility experiment as follows:

$\underline{\mathsf{Expt}_0}$: Actual game.

We next show that $\Pr[\mathsf{Expt}_0] = \mathsf{negl}(\lambda)$ by a series of hybrid games, which proves simulation sound extractablility.

$\underline{\mathsf{Expt}_1^{\mathcal{A}}}$: We view all **probabilistic** polynomial-time algorithms as **deterministic** polynomial-time algorithms, that takes in the random bits as a parameter. E.g., the key generators, provers, adversaries. Hence in the experiment, we (implicitly) need to sample (and store) all the random bits used, called R. Consider the point where the adversary returns a statement and a proof. By definition, there exists a $\mathsf{snark}.\mathcal{E}$, that extracts the witness. (Here we view the experiment till this point as the snark extraction adversary, and the randomness R as the advice string.) Change return condition to the following:

$(\mu, \perp, \perp, \perp, \perp) \leftarrow \mathsf{snark}.\mathcal{E}_{\mathcal{A}}(\mathsf{snark.crs}, R)$:

(1) $\mathsf{stmt} \notin Q$; and (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$; and (3) ~~$(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$~~ $f_{s_0}(\mathsf{stmt}) = \mu$.

Claim 10. *If the underlying encryption scheme is perfectly correct, and that the commitment scheme scheme is perfectly binding, and that the underlying SNARK satisfies the knowledge extraction property, then we have* $\Pr[\mathsf{Expt}_0] \leq \Pr[\mathsf{Expt}_1] + \mathsf{negl}(\lambda)$.

Proof. From the knowledge extraction property of the underlying SNARK, we know that $(\mathsf{stmt}, c, \mathsf{pk}_e, \rho) \in \mathcal{L}'$ and that $(\mu, \perp, \perp, \perp, \perp)$ is a valid witness for it, except for negligible probability.

Since the underlying encryption scheme is perfectly correct, the decrypted w is the only possible values that encrypts to c, hence it is unique for all valid witnesses. Given $(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$, we consider all such valid witnesses for $\mathcal{R}_{\mathcal{L}'}$, there must exist s_0', r_0' such that: (1) $\rho = \mathsf{comm}(s_0'; r_0')$; (2) $\mu = f_{s_0'}(\mathsf{stmt})$. From the perfectly binding property of the underlying commitment scheme, all witnesses must use the unique value $s_0' = s_0$ (recall s_0 is from setup).

Hence, assuming soundness holds, we have $f_{s_0} = \mu$. □

We also list the following hybrids, the claims are essentially the same as the ones in construction in Section 4.1.

$\underline{\mathsf{Expt}_2}$: We change to use simulation setup, i.e. $\mathsf{snark}.\widehat{\mathcal{K}}$ and $\mathsf{snark}.\widehat{\mathcal{P}}_1$. And note that $\mathsf{snark}.\widehat{\mathcal{P}}$ and $\mathsf{snark}.\widehat{\mathcal{P}}_1$ are equivalent as we used a valid witness for $\mathcal{L}'$, which allows us to further relax $\mathsf{snark}.\widehat{\mathcal{P}}_1$ to $\mathsf{snark}.\widehat{\mathcal{P}}$ (ignore the witnesses).

Claim 11. *Assuming that the underlying SNARK is computationally zero-knowledge, we have* $\Pr[\mathsf{Expt}_1] \leq \Pr[\mathsf{Expt}_2] + \mathsf{negl}(\lambda)$.

$\underline{\mathsf{Expt}_3}$: Change the setup: we commit to a different s_0'. i.e. $s_0, s_0', r_0 \xleftarrow{\$} \{0,1\}^{\lambda}$; $\rho := \mathsf{comm}(s_0'; r_0)$;

Claim 12. *If the underlying commitment scheme is computationally hiding, we have* $\Pr[\mathsf{Expt}_2] \leq \Pr[\mathsf{Expt}_3] + \mathsf{negl}(\lambda)$.

Expt_4: In the setup, s_0 is no longer generated. We replace each f_{s_0} with a true random function F. Note that this also change the return condition to: (3) $F(\mathsf{stmt}) = \mu$.

Claim 13. *If the underlying pseudo-random function family* $\{f_s\}_{s\in\{0,1\}^\lambda}$ *is secure, then we have* $\Pr[\mathsf{Expt}_3] \leq \Pr[\mathsf{Expt}_4] + \mathsf{negl}(\lambda)$.

Claim 14. *We have* $\Pr[\mathsf{Expt}_4] \leq 2^{-\lambda}$.

The above claims complete the proof for simulation sound extractability. □

Proof of computational zero-knowledge.

Lemma 5. *The construction sastisfies computational zero-knowledge.*

Proof. We prove zero-knowledge by showing any adversary $\mathcal{A}$ can not distinguish a series of hybrid games.

Expt_0: Actual game.

Expt_1: We change to use simulation setup, i.e. $\mathsf{snark}.\widehat{\mathcal{K}}$ and $\mathsf{snark}.\widehat{\mathcal{P}}_1$. And note that $\mathsf{snark}.\widehat{\mathcal{P}}$ and $\mathsf{snark}.\widehat{\mathcal{P}}_1$ are equivalent as we used a valid witness for $\mathcal{L}'$, which allows us to further relax $\mathsf{snark}.\widehat{\mathcal{P}}_1$ to $\mathsf{snark}.\widehat{\mathcal{P}}$ (ignore the witnesses).

Claim 15. *If the underlying SNARK is zero-knowledge, then we have* $\Pr[\mathsf{Expt}_0] \leq \Pr[\mathsf{Expt}_1] + \mathsf{negl}(\lambda)$.

Expt_2: We change to encrypt the true witness, i.e. ~~$c = \mathsf{Enc}(\mathsf{pk}_e, z_3; r_1)$~~ $c = \mathsf{Enc}(\mathsf{pk}_e, w; r_1)$. Also note that μ is not needed any more.

Claim 16. *If the underlying encryption scheme is secure, we have* $\Pr[\mathsf{Expt}_1] \approx \Pr[\mathsf{Expt}_2]$.

Expt_3: We change back to use $\mathsf{snark}.\widehat{\mathcal{P}}_1$, by providing (z_0, r_1, z_1, w, z_2) as a valid witness, i.e., $\mathsf{snark}.\pi \leftarrow \mathsf{snark}.\widehat{\mathcal{P}}_1(\mathsf{snark}.\widehat{\mathsf{crs}}, \mathsf{snark}.\tau, (\mathsf{stmt}_x, c, \mathsf{pk}_e, \rho), (z_0, r_1, z_1, w, z_2))$.

Expt_4: We change back to the honest setting. We use $\mathsf{snark}.\mathcal{K}$ and $\mathsf{snark}.\mathcal{P}$ to generate the crs and the proofs.

Claim 17. *If the underlying SNARK is zero-knowledge, then we have* $\Pr[\mathsf{Expt}_4] \approx \Pr[\mathsf{Expt}_3]$.

This completes the proof of the zero-knowledge property.

□

□

4.3 From SSE-NIZK to UC-Secure NIZKs

While our motivation is to construct a tool suitable for use in UC secure applications, our construction itself does not require any details from the UC framework. In Hawk, [29] it is shown that a (weak) SSE-NIZK is sufficient for use in UC-secure applications — this uses the SSE-NIZK definition directly, instead of an ideal functionality specification (i.e., the UC composition theorem is not used). It also has been observed elsewhere [16, 26] that a (strong) SSE-NIZK scheme can realize a UC ideal functionality with a static adversary. In the Appendix, we describe (without proof) how to weaken the ideal functionality $\mathcal{F}_{\text{NIZK}}$ so that it can be realized from (weak) SSE-NIZK.

5 A Stronger Version

In this section, we define a strengthened version of simulation sound extractability and provide a construction.

The original definition of simulation sound extractability says that if the adversary does not know a witness for a statement, he can only prove that statement if the he has previously submitted this statement as an oracle query (i.e., to the simulated prover). In our strengthened definition, which we call "strongly simulation sound extractable", we further constrain the adversary to only produce *statement-proof pairs* that have been previously queried. In other words, under the weaker definition, it is possible for an adversary to generate *novel* proofs for previously-queried statements; this is precluded by the stronger definition.

Strongly simulation sound extractable. We say a NIZK for a language $\mathcal{L}$ is strongly simulation sound extractable *iff* there exists an extractor $\mathcal{E}$ such that for all polynomial-time adversary $\mathcal{A}$, the following holds:

$$\Pr\left[\begin{array}{ll} (\widehat{\mathsf{crs}}, \tau, \mathsf{ek}) \leftarrow \widehat{\mathcal{K}}(1^\lambda) & (\mathsf{stmt}, \pi) \notin Q \text{ and} \\ (\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{\widehat{\mathcal{P}}(\widehat{\mathsf{crs}}, \tau, \cdot)}(\widehat{\mathsf{crs}}, \mathsf{ek}) & : \ (\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}} \text{ and} \\ w \leftarrow \mathcal{E}(\widehat{\mathsf{crs}}, \mathsf{ek}, \mathsf{stmt}, \pi) & \mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1 \end{array}\right] = \mathsf{negl}(\lambda)$$

where Q is the set of statement-proof pairs generated by the oracle calls to $\widehat{\mathcal{P}}$.

Intuition. Like before, a prover must always provide an encryption of a (possibly bogus) witness. Our construction makes use of a one-time signature scheme. A pair of one-time signing/verification keys are generated for each prove. Compared with Section 4.1, the difference is that instead of $f_a(\mathsf{stmt})$, a simulated prover is required to provide $\mu = f_a(\mathsf{pk})$. Then we require the prover to sign the statement together with the proof, the cipher-text, and μ. Briefly, due to the security of signature scheme, the adversary must use a different pk from the ones returned from oracle queries. Thus, in order for a statement to pass the verifier without a proper witness, the prover must generate $f_a(\mathsf{pk})$ without the knowledge of a (thus breaking the pseudo-random function).

The Construction. Given a language $\mathcal{L}$ with NP relation $\mathcal{R}_{\mathcal{L}}$, let $\mathcal{L}'$ be the language that $\big((\mathsf{stmt}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r, r', w, a)\big) \in \mathcal{R}_{\mathcal{L}'}$ *iff*:

$$c = \mathsf{Enc}(\mathsf{pk}_e, w; r) \wedge \Big((\mathsf{stmt}, w) \in \mathcal{R}_{\mathcal{L}} \vee \big(\mu = f_a(\mathsf{pk}_s) \wedge \rho = \mathsf{comm}(a; r')\big)\Big)$$

Next we show the construction from NIZK to strong SSE-NIZK.

- $\mathcal{K}(1^\lambda, \mathcal{L})$:

  $\mathsf{nizk.crs} \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

  $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$. return $\mathsf{crs} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$.

- $\mathcal{P}(\mathsf{crs}, \mathsf{stmt}, w)$:

  Parse $\mathsf{crs} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$; Abort if $(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$;

  $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$;

  $z_0, z_1, z_2, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, w; r_1)$;

  $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}, c, z_0, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r_1, z_1, w, z_2))$;

  $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, z_0, \mathsf{nizk}.\pi))$;

  return $\pi := (c, z_0, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.

- $\mathcal{V}(\mathsf{crs}, \mathsf{stmt}, \pi)$:

  Parse $\mathsf{crs} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$ and $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$;

Abort if $\mathsf{Verify}(\mathsf{pk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi), \sigma) = 0$;
Call $\mathsf{nizk}.\mathcal{V}(\mathsf{nizk.crs}, (\mathsf{stmt}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), \mathsf{nizk}.\pi)$.

- $\widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$: Run $\mathcal{K}$ to get $\widehat{\mathsf{crs}} := \mathsf{crs}$, but keep trapdoor $\tau := (s_0, r_0)$, extraction key $\mathsf{ek} := \mathsf{sk}_e$.
- $\widehat{\mathcal{P}}(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt})$:

  Parse $\widehat{\mathsf{crs}} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$ and $\tau := (s_0, r_0)$;
  $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu = f_{s_0}(\mathsf{pk}_s)$; $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, z_3; r_1)$;
  $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r_1, r_0, z_3, s_0))$;
  $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$; return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.
- We also define the extractor here:
  $\mathcal{E}(\widehat{\mathsf{crs}}, \mathsf{ek}, \mathsf{stmt}, \pi)$: Parse $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$; $w \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$; return w.

Theorem 4. *Assume that the underlying NIZK scheme satisfies perfect completeness, computational soundness, computational zero-knowledge, that the encryption scheme is semantically secure and perfectly correct, that the pseudo-random function family is secure, that the commitment scheme is perfectly binding and computational hiding, and that the one-time signature scheme is strongly unforgeable. Then the above construction is a zero-knowledge proof system satisfying perfect completeness, computational zero-knowledge, and strongly simulation sound extractability.*

The proof uses a similar idea of that in Section 4, which we defer to appendix.

6 SNARK-Friendly Cryptography

What is efficient and not efficient for SNARKs. Known SNARK constructions model computation as algebraic circuits modulo a large prime p. Standard implementations and parameter choices for cryptographic primitives are targeted at modern hardware platforms with different constraints than SNARKs. For example, some algebraic operations, like addition and constant-scalar multiplication of field elements $\mathbb{F}_p$, which are expensive in hardware, are essentially free in a SNARK; however, while XORing two 32-bit numbers takes a single cycle on an ordinary CPU, this is far more costly in an arithmetic circuit.

The following observations guide our choices:

- Addition and multiplications by constants in the field $\mathbb{F}_p$, where p is the SNARK field order, are almost for free, and in particular are much cheaper than multiplication. Reducing the number of multiplication gates is the main optimization criteria.
- Bit-level operations in an arithmetic circuit is expensive, as splitting a field element into n bits requires $n + 1$ multiplication gates.
- However, once we have a binary representation of a value, operations like Rotation and Shift are free.
- Random-access lookup tables, such as those used in many S-Box symmetric cryptography implementations, are likely a bottleneck. Typically, there are two main approaches to implement lookup tables in a circuit. The first approach is a linear scan to select one element, which results in $O(n)$ cost. The other approach involves using a permutation network to sort a sequence of memory accesses by address and providing a proof of consistency; the complexity of this depends on the number of the accesses to the array. [42]
- Verification can be simpler than forward computation. The SNARK circuits do not always have to compute the result, but can instead represent a verification algorithm. For example, a multiplicative inverse circuit does not have to encode the computation of the inverse, but can instead consist of a single multiplication constraint on the value provided by the prover.

Designing SNARK-friendly cryptographic primitives. We explore the following strategies for designing SNARK-friendly cryptographic primitives.

1. **Protocol- and algebraic-level choices.** First, for the same cryptographic building block (e.g., encryption, signature), we explore building it from different algebraic building blocks such as RSA and Diffie-Hellman. Certain algebraic structures and operations are by nature more efficient when encoded as SNARK circuits. We also explore various choices such as using public-key encryption vs hybrid encryption
2. **Circuit-level optimizations.** Once a scheme is fixed, we perform numerous optimizations at the circuit level to reduce the concrete circuit size.

We now describe our protocol choices and optimizations for the cryptographic tasks needed in our SNARK-lifting constructions from Sections 3 and 4.

6.1 Encryption

As the costs of all our SNARK-lifting constructions are dominated by the public-key encryption of the witness, we focus most of our efforts on this task.

6.1.1 Public-key encryption.

Hawk [29] and Gyges [28] use a naïve implementation of RSA with OAEP, despite the fact this encoding is poorly suited to arithmetic circuits.

We explore the following strategies for optimizing public-key encryption as a SNARK circuit.

- **Circuit-level optimizations for RSA.** We start by implementing an optimized circuit for RSA encryption. The essential challenge with RSA is that the arithmetic operations are over integers mod n, where n is larger (e.g., 1024 bits) than the SNARK field order p (typically a 254-bit prime). We represent integers mod n as sixteen 64-bit elements. To multiply a pair of such integers $z := x * y \bmod n$, we construct a circuit that verifies $x * y = q * n + z$, where q and z are 16 64-bit elements provided as witnesses by the prover. The optimizations we added over Hawk's implementation was to use a more efficient approach for long integer equality checks when the chunks are not aligned, and reducing the number of comparisons needed.
- **Diffie-Hellman encryption in a SNARK-friendly field extension.** Instead of relying on RSA as the main PKE scheme, we investigate another scheme based on the Discrete-Logarithm problem in Extension Fields, and use it for symmetric key exchange. Since p is only 254-bit prime, the DL problem in $\mathbb{F}_p$ will not be hard, therefore an extension $\mathbb{F}_{p^\mu}$ will be used instead. This idea is mainly inspired by the construction in Pinocchio coin [23]. The key exchange circuit has two generators in that case $g, h \in \mathbb{F}_{p^\mu}$, where $\langle g \rangle = \langle h \rangle$ is a large multiplicative subgroup of order $q | p^\mu - 1$. We select q to be a factor of the μ-th cyclotomic polynomial $\mathbf{\Phi}_\mu(x)$ when evaluated at $x = p$ as in [31].

  The extension field construction requires us to search for large primes that divide $\mathbf{\Phi}_\mu(p)$. In our implementation using libsnark [8], in order to get about 80-bit level of security, we set μ to be 4, and choose q to be the 398-bit prime factor of the $\mathbf{\Phi}_4(p)$, where p is the SNARK field order of libsnark. For higher security when $\mu = 6$, we found a 313-bit prime order subgroup for the extension field. However, to get higher security levels (i.e., $\mu > 6$), this may require expensive factorization. Additionally, RSA can be used to encrypt small messages in a straightforward way, but for the extension field scheme, it is difficult to find an appropriate way to encode messages as subgroup elements.
- **Hybrid encryption.** Finally, we consider the use of hybrid encryption, where we use the public-key encryption to encrypt a symmetric key, and then use the symmetric key to encrypt

the plaintext. To explore this option, we additionally explored various options of symmetric-key encryption as we describe below.

6.1.2 Symmetric-key encryption.

After encrypting an ephemeral secret using RSA or an extension field-based scheme, we use a SHA-256 circuit to derive a secret symmetric key and a secret initialization vector. Then, a symmetric encryption is performed in CBC mode using a block cipher.

Choosing a standard block cipher like AES is a poor choice for SNARKs due to its complexity. For example, using a naïve implementation of AES using snarklib [18], one AES-128 block requires more than 1 million gates, which is several orders of magnitude more expensive than our baseline from Hawk. This high cost is mainly because naïve implementations use inefficient look up tables for S-boxes, as well as similarly unoptimized procedures. In our optimized implementation, we substantially reduce the overhead for memory accesses by using a customized efficient technique for the S-Box, and remove other look up tables when more efficient alternatives can be used (Further detail will be covered in a later version). Our more efficient implementation costs about 23k gates per block, and about 4.6k gates in the initial key expansion phase (we are also investigating more optimizations in an ongoing work).

To achieve more practical performance, we looked for lightweight ciphers according to the criteria we described in the beginning of the section, and found two promising ciphers, Speck and Chaskey, which were proposed recently. Speck was proposed in 2013 [3] by NSA, and in 2015, no attacks have been found so far [4]. Chaskey was proposed in [36], where its security was proven in the standard model. We use a more secure version of Chaskey called Chaskey-LTS, which uses 16 rounds instead of 8 to achieve long-term security. These ciphers have more SNARK-friendly implementations compared to AES, but the disadvantage of using these ciphers is that they are new compared to AES. We plan to investigate more lightweight ciphers in the future as well.

6.1.3 Micro-benchmarks

Table 1 provides the micro-benchmarks for the public key and symmetric key schemes discussed above, compared to their naïve implementations when possible. It should be noted that for PKE schemes, we assumed that the public key is hardcoded in the circuit, which is suitable for our purposes in the transformations. If the public keys are not hardcoded, the cost for the field extension circuit will be about 35k gates instead, but it will result into minor difference in the RSA case. As noted in the table, the cost of Field Extension is **40x** better than the RSA case. Similarly for the block ciphers, the table shows about 4x better cost for Speck and Chaskey compared to an optimized version of AES. We also compared with our optimized version of AES (23k multiplication gates) to an auto-generated SNARK circuit using snarklib [18] ($>$ 1 million multiplication gates). In this case, our optimized version of AES implementation is at least **40x** better than the auto-generated version.

Table 1: The number of multiplication gates of PKE and Symmetric-key Cipher Schemes. Numbers between (.) represent naive implementation cost, when significant.

PKE Scheme	Cost for Key Exchange	**Block Cipher**	Cost per Block
RSA-1024	210k (330k)	AES 128	23k (1m)
Field Extension ($\mu = 4$)	5k	Speck 128	6k
		Chaskey LTS 128	5k

Table 2 provides the cost of encrypting 200 bytes using all the above schemes (after optimizations). One interesting observation to make here is that the cost of RSA with AES encryption

is worse than using RSA directly. Furthermore, the extension field with the two lightweight ciphers provide better performance than the other techniques with about **3x** speedup. Note that the speedup here is lower than the previous table, due to the cost of key derivation after key exchange.

Table 2: The number of gates for encrypting 200 bytes for all schemes.

	Total Cost	Cost Per Bit	Cost Ratio
RSA only	420k	262.5	3.39
RSA + AES	563k	351.88	4.54
RSA + Speck	346k	216.49	2.79
RSA + Chaskey LTS	329k	205.26	2.65
Field Extension + AES	358k	223.75	2.89
Field Extension + Speck	141k	88.36	1.14
Field Extension + Chaskey LTS [Baseline]	**124k**	**77.50**	**1.0**

6.2 Pseudo-random functions / Commitments

In our implementation, we instantiate PRFs and Commitments using an efficient SHA-256 circuit. An efficient SHA-256 circuit costs about 27k gates, while its naïve implementation using SNARK compilers costs more than 40k gates. The optimizations are mainly achieved by representing Boolean operations efficiently, and careful circuit design. A previous similar implementation and a detailed discussion of SHA-256 optimizations can be found in [5].

6.3 Collision Resistant Hash Functions

Lattice-based cryptography, including Ajtai's collision resistant hash, are promising for use in SNARKs [7]. However, existing estimates of concrete security for such schemes only extend to lattices over small finite fields, but do not *a priori* apply to lattices constructed over a SNARK's (much larger) native field. In the Appendix, we establish that these estimates do indeed apply and show how to parameterize lattice-based schemes.

6.4 Signatures

For the digital signature scheme, we use an optimized RSA signature verification circuit using the PKCS-1 standard. As stated earlier, SNARK circuits do not necessarily have to compute, and since the signature verification in RSA is cheaper (due to the small public exponent), we adopt a signature verification circuit instead, and apply the same optimizations we applied for the RSA Encryption circuit.

We currently use SHA-256 to hash the message to be signed. For very long messages, this could be improved by composing a SNARK-friendly collision-resistant hash with a single SHA-256 block in the end.

6.5 End-to-end Application

Speedup of the entire SNARK-lifting transformation. We evaluate the speedup of our optimized SNARK-lifting transformation in comparison with a naive, unoptimized baseline which implements the scheme in Section 3.1. To do this, we isolate the effect of the application-specfic circuit for the NP language, by considering a very small application-specific circuit. In

Table 3: Keygen / Proof Computation Time (in seconds) for the pour and freeze circuits from Hawk [29]

	Transformation in Section 3, A.2							
	Hyb. Enc. w/ Field Ext.			Hyb. Enc. w/ RSA			RSA	Naive RSA
	Chaskey LTS	Speck	AES	Chaskey LTS	Speck	AES		Baseline [29]
Pour (Keygen)	125.2	128.1	156.2	152.9	151.8	181.3	163.1	218.1
Pour (Proof)	41.0	41.3	55.3	49	48.5	65.0	53.4	80.8
Freeze (Keygen)	96.7	96.7	126.0	120.9	123.9	149.41	132.6	188.1
Freeze (Proof)	31.8	32.0	42.7	40.2	40.5	50.36	40.7	63.6
	Transformation in Section 4.2							
	Hyb. Enc. w/ Field Ext.			Hyb. Enc. w/ RSA			RSA	
	Chaskey LTS	Speck	AES	Chaskey LTS	Speck	AES		
Pour (Keygen)	**106.9**	**110.0**	132.4	126.4	130.1	157.8	141.4	
Pour (Proof)	**40.4**	**40.4**	43.1	40.6	40.7	55.5	46.6	
Freeze (Keygen)	**78.1**	**81.1**	110.1	104.9	107.9	129.5	116.7	
Freeze (Proof)	**26.7**	**26.9**	42.2	39.8	39.8	42.8	40.2	

this case, our implementation shows **5x** performance improvement. In particular, our optimized version adopts the more superior SNARK-lifting transformation described in Section 4.2, with SNARK-friendly cryptography primitives such as the encryption using field extension and Chaskey or Speck ciphers.

In the context of a realistic, cryptocurrency application. As mentioned earlier in Section 1.2 subsequent works [28, 29] that use SNARKs for cryptocurrency applications have since adopted our SNARK-lifting transformations. However, these works were conducted before we had the opportunity to optimize our transformations and make them SNARK-friendly. We therefore use those earlier implementations as a baseline of comparison.

To show the enhancement introduced by our techniques in a real application, we apply the transformations proposed in this paper and the SNARK-friendly cryptography on sample two circuits from the Hawk system [29] at 80-bit security level (using RSA-1024 and field extension at $\mu = 4$). The two circuits selected were pour and freeze which mainly enable users to spend or commit to coins with hidden values. Note that the circuits for pour and freeze include many other components for Merkle trees and commitment verification, while the soundness components are just a subset. The original Hawk paper [29] used the transformation in Section 3 with the optimization in A.2, using only unoptimized RSA for encryption and signature. This will be the baseline for comparison.

In Table 3, we study the SNARK key generation and proof construction time under the transformations in sections 3 using the optimization in A.2, and the transformation in 4.2, and compare all the encryption schemes for both circuits. We used libsnark to run the evaluation on a single processor of 2.2 GHz with 6GB memory in the worse case. The table shows that applying the second transformation with the hybrid encryption scheme that uses field extension and Chaskey LTS can achieve at least **2x** better overall performance for both circuits compared to the baseline. The reason why the speedup is smaller in this case is because our application required verifying an NP language that is modeled by a moderately large circuit containing about 300-400K multiplication gates (excluding our transformations). Therefore, our optimizations were mainly related to a small fraction of the circuit — and it is outside the scope of this paper to optimize the application-specific circuit for the NP language.

7 Conclusion

We have shown several ways to ugprade a SNARK (or an ordinary NIZK) to an SSE-NIZK, suitable for use in a composable simulation-based security framework. This is of immediate practical use, as there are already efficient and general implementations of SNARKs, and numerous applications requiring them. Along the way, we've identified protocol choices, parameterizations ad optimized implementations for several cryptographic tasks. We show 10x to 40x performance improvement for implementing individual cryptographic primitives (e.g., encryption) over SNARK. Altogether, our optimized SNARK-lifting transformation gains 5x concrete performance in comparison with a naive implementation. Given the wide range of applications building on SNARKs, we hope our work inspires further effort to develop SNARK-friendly cryptography, and also that it encourages wider use of simulation-based security specifications.

Acknowledgments

We gratefully acknowledge Jonathan Katz for helpful technical discussions about the zero-knowledge proof constructions. This work is funded in part by NSF grants CNS-1314857, CNS-1453634, CNS-1518765, CNS-1514261, a Packard Fellowship, a Sloan Fellowship, two Google Faculty Research Awards, a VMWare Research Award, as well as grants from the DARPA Safeware and DARPA Brandeis programs. This work was done in part while a subset of the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant CNS-1523467.

References

[1] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Cryptology ePrint Archive*, Report 2015/046, 2015.

[2] Anonymized. Hawk: Programming private smart contracts. Forthcoming manuscript.

[3] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. `http://eprint.iacr.org/`.

[4] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. Simon and speck: Block ciphers for the internet of things. Cryptology ePrint Archive, Report 2015/585, 2015. `http://eprint.iacr.org/`.

[5] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on. IEEE*. IEEE, 2014.

[6] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO*, 2013.

[7] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Scalable zero knowledge via cycles of elliptic curves. In *Advances in Cryptology–CRYPTO 2014*, pages 276–294. Springer, 2014.

[8] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *USENIX Security*, 2014.

[9] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, pages 326–349, 2012.

[10] N. Bitansky, R. Canetti, O. Paneth, and A. Rosen. On the existence of extractable one-way functions. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 505–514, 2014.

[11] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 103–112, 1988.

[12] M. Blum, A. D. Santis, S. Micali, and G. Persiano. Non-interactive zero knowledge. *SIAM Jornal of Computation*, 1991.

[13] E. Boyle and R. Pass. Limits of extractability assumptions with distributional auxiliary input. *IACR Cryptology ePrint Archive*, 2013:703, 2013.

[14] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

[15] R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *Theory of Cryptography*, pages 61–85. Springer, 2007.

[16] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 494–503. ACM, 2002.

[17] R. Canetti and T. Rabin. Universal composition with joint state. In *CRYPTO*, 2003.

[18] J. Carlsson. snarklib: a c++ template library for zero knowledge proofs. https://github.com/jancarlsson/snarklib.

[19] Y. Chen and P. Q. Nguyen. Bkz 2.0: Better lattice security estimates. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073, pages 1–20. Springer, December 2011.

[20] Y. Chen and P. Q. Nguyen. Bkz 2.0: Better lattice security estimates (full version), 2012.

[21] A. Chiesa, E. Tromer, and M. Virza. Cluster computing in zero knowledge. In *Advances in Cryptology-EUROCRYPT 2015*, pages 371–403. Springer, 2015.

[22] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur. Geppetto: Versatile verifiable computation. In *Proceedings of the 36th IEEE Symposium on Security and Privacy, S&P*, volume 15, 2014.

[23] G. Danezis, C. Fournet, M. Kohlweiss, and B. Parno. Pinocchio Coin: building Zerocoin from a succinct pairing-based proof system. In *PETShop*, 2013.

[24] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Eurocrypt*, 2013.

[25] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *ACM symposium on Theory of computing (STOC)*, 2011.

[26] J. Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *Proceedings of the 12th International Conference on Theory and Application of Cryptology and Information Security*, ASIACRYPT'06, pages 444–459, 2006.

[27] J. Groth, R. Ostrovsky, and A. Sahai. Perfect non-interactive zero knowledge for NP. In *Proceedings of the 24th Annual International Conference on The Theory and Applications of Cryptographic Techniques*, EUROCRYPT'06, 2006.

[28] A. Juels, A. Kosba, and E. Shi. The ring of gyges: Using smart contracts for crime. Manuscript, 2015.

[29] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. Cryptology ePrint Archive, Report 2015/675, 2015. `http://eprint.iacr.org/`.

[30] A. E. Kosba, D. Papadopoulos, C. Papamanthou, M. F. Sayed, E. Shi, and N. Triandopoulos. Trueset: Nearly practical verifiable set computations. In *Usenix Security Symposium*, 2014.

[31] A. K. Lenstra. Using cyclotomic polynomials to construct efficient discrete logarithm cryptosystems over finite fields. In *Information Security and Privacy*, pages 126–138. Springer, 1997.

[32] R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In A. Kiayias, editor, *CT-RSA, Lecture Notes in Computer Science*, volume 6558, pages 319–339, Berlin, February 2011. Springer.

[33] C. C. Martin R. Albrecht, J.-C. Faugère, R. Fitzpatrick, and L. Per-ret. On the complexity of the bkw algorithm on lwe. In *Designs, Codes and Cryptography*, volume 74, pages 325–354, 2015.

[34] D. Micciancio and O. Regev. Lattice-based cryptography. In *Bernstein et al*, pages 147–191, 2009.

[35] A. Miller, E. Shi, A. Kosba, and J. Katz. Nonoutsourceable Scratch-Off Puzzles to Discourage Bitcoin Mining Coalitions (preprint), 2014.

[36] N. Mouha, B. Mennink, A. Van Herrewege, D. Watanabe, B. Preneel, and I. Verbauwhede. Chaskey: An efficient mac algorithm for 32-bit microcontrollers. In *Selected Areas in Cryptography–SAC 2014*, pages 306–323. Springer, 2014.

[37] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *S & P*, 2013.

[38] M. Rückert and M. Schneider. Estimating the security of lattice-based cryptosystems. *Cryptology ePrint Archive*, Report 2010/137, 2010.

[39] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 543–, 1999.

[40] A. D. Santis, G. D. Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust non-interactive zero knowledge. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 566–598, 2001.

[41] S. Setty, V. Vu, N. Panpalia, B. Braun, A. Blumberg, and M. Walfish. Taking proof-based verified computation a few steps closer to actual practicality. In *USENIX security*, 2012.

[42] R. S. Wahby, S. Setty, Z. Ren, A. J. Blumberg, and M. Walfish. Efficient ram and control flow in verifiable outsourced computation. In *NDSS*, 2015.

[43] R. S. Wahby, S. T. V. Setty, Z. Ren, A. J. Blumberg, and M. Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014*, 2015.

[44] M. Walfish and A. J. Blumberg. Verifying computations without reexecuting them. *Communications of the ACM*, 58(2):74–84, 2015.

[45] Y. Zhang, J. Katz, and C. Papamanthou. Integridb: Verifiable SQL for outsourced databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1480–1491, 2015.

[46] Y. Zhang, C. Papamanthou, and J. Katz. ALITHEIA: towards practical verifiable graph processing. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 856–867, 2014.

Appendix

A Optimizations for Basic SNARK Lifting Transformation

A.1 Witness Optimization

In this section, we show an improvement of Section 3.1. The idea is simple:

Note in the construction of Section 3.1, for a witness to be valid for $\mathcal{R}_{\mathcal{L}'}$, either w or σ is valid, but not both. Hence, instead of encrypting both (concatenation), we only need to encrypt the one we actually use.

For language $\mathcal{L}$ with NP relation $\mathcal{R}_{\mathcal{L}}$, let $\mathcal{L}'$ be $\big((\mathsf{stmt}, c, \mathsf{pk}_s, \mathsf{pk}_e), (r, w)\big) \in \mathcal{R}_{\mathcal{L}'}$ *iff*:

$$c = \mathsf{Enc}(\mathsf{pk}_e, w; r) \wedge \big((\mathsf{stmt}, w) \in \mathcal{R}_{\mathcal{L}} \vee \mathsf{Verify}(\mathsf{pk}_s, \mathsf{stmt}, w) = 1\big)$$

The construction is defined as follows:

- $\mathcal{K}(1^\lambda, \mathcal{L})$:

  $\mathsf{snark.crs} \leftarrow \mathsf{snark}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;
  $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; return $\mathsf{crs} := (\mathsf{snark.crs}, \mathsf{pk}_s, \mathsf{pk}_e)$.

- $\mathcal{P}(\mathsf{crs}, \mathsf{stmt}, w)$:

  Parse $\mathsf{crs} := (\mathsf{snark.crs}, \mathsf{pk}_s, \mathsf{pk}_e)$; Abort if $(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$;
  $z_0, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, w; r_1)$;
  $\mathsf{snark}.\pi \leftarrow \mathsf{snark}.\mathcal{P}(\mathsf{snark.crs}, (\mathsf{stmt}, c, \mathsf{pk}_s, \mathsf{pk}_e), (r_1, w))$;
  return $\pi := (c, \mathsf{snark}.\pi)$.

- $\mathcal{V}(\mathsf{crs}, \mathsf{stmt}, \pi)$:

  Parse $\mathsf{crs} := (\mathsf{snark.crs}, \mathsf{pk}_s, \mathsf{pk}_e)$ and $\pi := (c, \mathsf{snark}.\pi)$;
  Call $\mathsf{snark}.\mathcal{V}(\mathsf{snark.crs}, (\mathsf{stmt}, c, \mathsf{pk}_s, \mathsf{pk}_e), \mathsf{snark}.\pi)$.

- $\widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$: Run $\mathcal{K}$ to get $\widehat{\mathsf{crs}} := \mathsf{crs}$, but keep trapdoor $\tau := \mathsf{sk}_s$, extraction key $\mathsf{ek} := \mathsf{sk}_e$.

- $\widehat{\mathcal{P}}(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt})$:

  Parse $\widehat{\mathsf{crs}} := (\mathsf{snark.crs}, \mathsf{pk}_s, \mathsf{pk}_e)$ and $\tau := \mathsf{sk}_s$;
  $z_1, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, \sigma; r_1)$; $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, \mathsf{stmt})$
  $\mathsf{snark}.\pi \leftarrow \mathsf{snark}.\mathcal{P}(\mathsf{snark.crs}, (\mathsf{stmt}, c, \mathsf{pk}_s, \mathsf{pk}_e), (r_1, \sigma))$;
  return $\pi := (c, \mathsf{snark}.\pi)$.

- We also define the extractor here:
  $\mathcal{E}(\widehat{\mathsf{crs}}, \mathsf{ek}, \mathsf{stmt}, \pi)$: Parse $\pi := (c, \mathsf{snark}.\pi)$; $w \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$; return w.

We next prove the construction is a SSE-NIZK.

Theorem 5. *Assume that the underlying NIZK scheme satisfies perfect completeness, computational soundness, and computational zero-knowlege, that the signature scheme satisfies existential unforgeability under chosen message attack, and that the encryption scheme is semantically secure and perfectly correct, then the above construction is a zero-knowledge proof system satisfying perfect completeness, computational zero-knowledge, and simulation sound extractability.*

Proof. Completeness is obvious. Next we show zero-knowledge and simulation sound extractability.

Lemma 6. *The construction is zero-knowledge.*

Proof. The lemma can be proved similarly as previous one. □

Lemma 7. *The construction is simulation sound extractable.*

Proof. The proof follows the same idea as the previous proof, We only sketch the hybrid games:

$\underline{\mathsf{Expt}_0}$: Actual game.

$\underline{\mathsf{Expt}_1}$: Change the return condition to, $w \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$:

(1) $\mathsf{stmt} \notin Q$; (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$;

(3) ~~$(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$~~ $\mathsf{Verify}(\mathsf{pk}_s, \mathsf{stmt}, w) = 1$

We argue that $\Pr[\mathsf{Expt}_0] \leq \Pr[\mathsf{Expt}_1] + \mathsf{negl}(\lambda)$: By the soundness of the underlying NIZK, we know that $(\mathsf{stmt}, c, \mathsf{pk}_s, \mathsf{pk}_e) \in \mathcal{L}'$ except for negligible probability. Hence we only focus on such cases. By the perfectly correctness of the underlying encryption scheme, all valid witnesses must use the unique w decrypted.

If $(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$ then we must have $\mathsf{Verify}(\mathsf{pk}_s, \mathsf{stmt}, w) = 1$, which gives $\Pr[\mathsf{Expt}_0] \leq \Pr[\mathsf{Expt}_1] + \mathsf{negl}(\lambda)$.

Next we argue that $\Pr[\mathsf{Expt}_1^{\mathcal{A}}] \leq \mathsf{negl}(\lambda)$: Consider otherwise, then we use $\mathsf{Expt}_1^{\mathcal{A}}$ as an adversary for the security game of the signature scheme: Get pk_s from the game; Run $\mathsf{Expt}_1^{\mathcal{A}}$ with the same pk_s; Replace signature with oracle calls (hence sk_s is no longer); Output the w decrypted. □

□

A.2 SNARK-Specific Optimization

In this section, we show a modified construction based on a SNARK, which improves the one in Section 3.1 by the use of its non-blackbox extractor.

For language $\mathcal{L}$ with NP relation $\mathcal{R}_{\mathcal{L}}$, let $\mathcal{L}'$ be the language defined as $\big((\mathsf{stmt}, c, \mathsf{pk}_s, \mathsf{pk}_e), (r, w, \sigma)\big) \in \mathcal{R}_{\mathcal{L}'}$ *iff*:

$$c = \mathsf{Enc}(\mathsf{pk}_e, w; r) \wedge \big((\mathsf{stmt}, w) \in \mathcal{R}_{\mathcal{L}} \vee \mathsf{Verify}(\mathsf{pk}_s, \mathsf{stmt}, \sigma) = 1\big)$$

The construction is defined as follows:

- $\mathcal{K}(1^\lambda, \mathcal{L})$:

  $\mathsf{snark.crs} \leftarrow \mathsf{snark}.\mathcal{K}(1^\lambda, \mathcal{L}')$;
  $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$; $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$;
  return $\mathsf{crs} := (\mathsf{snark.crs}, \mathsf{pk}_s, \mathsf{pk}_e)$.

- $\mathcal{P}(\mathsf{crs}, \mathsf{stmt}, w)$:

  Parse $\mathsf{crs} := (\mathsf{snark.crs}, \mathsf{pk}_s, \mathsf{pk}_e)$; Abort if $(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$;
  $z_0, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, w; r_1)$;
  $\mathsf{snark}.\pi \leftarrow \mathsf{snark}.\mathcal{P}(\mathsf{snark.crs}, (\mathsf{stmt}, c, \mathsf{pk}_s, \mathsf{pk}_e), (r_1, w, z_0))$;
  return $\pi := (c, \mathsf{snark}.\pi)$.

- $\mathcal{V}(\mathsf{crs}, \mathsf{stmt}, \pi)$:

  Parse $\mathsf{crs} := (\mathsf{snark.crs}, \mathsf{pk}_s, \mathsf{pk}_e)$ and $\pi := (c, \mathsf{snark}.\pi)$;
  Call $\mathsf{snark}.\mathcal{V}(\mathsf{snark.crs}, (\mathsf{stmt}, c, \mathsf{pk}_s, \mathsf{pk}_e), \mathsf{snark}.\pi)$.

- $\widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$: Run $\mathcal{K}$ to get $\widehat{\mathsf{crs}} := \mathsf{crs}$, but keep trapdoor $\tau := \mathsf{sk}_s$, extraction key $\mathsf{ek} := \mathsf{sk}_e$.

- $\widehat{\mathcal{P}}(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt})$:

  Parse $\widehat{\mathsf{crs}} := (\mathsf{snark.crs}, \mathsf{pk}_s, \mathsf{pk}_e)$ and $\tau := \mathsf{sk}_s$;

$z_1, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, z_1; r_1)$; $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, \mathsf{stmt})$
$\mathsf{snark}.\pi \leftarrow \mathsf{snark}.\mathcal{P}(\mathsf{snark.crs}, (\mathsf{stmt}, c, \mathsf{pk}_s, \mathsf{pk}_e), (r_1, z_1, \sigma))$;
return $\pi := (c, \mathsf{snark}.\pi)$.

- We also define the extractor here:
  $\mathcal{E}(\widehat{\mathsf{crs}}, \mathsf{ek}, \mathsf{stmt}, \pi)$: Parse $\pi := (c, \mathsf{snark}.\pi)$; $w \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$; return w.

We next prove the construction is a SSE-NIZK.

Theorem 6. *Assume that the underlying SNARK scheme satisfies perfect completeness, and computational zero-knowledge, and proof of knowledge properties (which implies soundness), and that the signature scheme satisfies existential unforgeability under chosen message attack, and that the encryption scheme is semantically secure and perfectly correct, then the above construction is a zero-knowledge proof system satisfying perfect completeness, computational zero-knowledge, and simulation sound extractability.*

Proof. Completeness is obvious. Next we prove zero-knowledge and simulation sound extractability.

Lemma 8. *The construction is zero-knowledge.*

Proof. The proof essentially follows the same as the previous proof. We only sketch the hybrids below:

$\underline{\mathsf{Expt}_0}$: Actual game.

$\underline{\mathsf{Expt}_1}$: Change to use simulation setup, i.e., $\mathsf{snark}.\mathcal{K}$ and $\mathsf{snark}.\widehat{\mathcal{P}}$.

$\underline{\mathsf{Expt}_2}$: We change to encrypt the true witness, i.e. ~~c is an encryption of $(\perp, \sigma)$~~ c is an encryption of $(w, \perp)$.

$\underline{\mathsf{Expt}_3}$: Change back to use honest setup. Note that this can be done as we provide a valid witness in $\mathcal{R}_{\mathcal{L}}$.

□

Lemma 9. *The construction is simulation sound extractable.*

Proof. The proof follows the same idea as the previous proof, with the following modifications:

We can no longer decrypt the proof to get the signature, however we can use the blackbox extractor from the SNARK. This would allow us to argue that: if the extracted signature is valid (and the statement has not been queried), then we can break the signature scheme.

We only sketch the hybrid games:

$\underline{\mathsf{Expt}_0}$: Actual game.

$\underline{\mathsf{Expt}_1^{\mathcal{A}}}$: Similar to Expt_1 in Lemma 4, we view all probabilistic polynomial-time algorithms as deterministic ones with random bits as a parameter. We store the random bits used called R. Change the return condition to:

$w \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$; $(w_0, \perp, \sigma_0) \leftarrow \mathsf{snark}.\mathcal{E}_{\mathcal{A}}(\mathsf{snark.crs}, R)$:

(1) $\mathsf{stmt} \notin Q$; (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$;

(3) ~~$(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$~~ $\mathsf{Verify}(\mathsf{pk}_s, \mathsf{stmt}, \sigma_0) = 1$

We argue that $\Pr[\mathsf{Expt}_0] \leq \Pr[\mathsf{Expt}_1] + \mathsf{negl}(\lambda)$: By the (non-blackbox) extractability of the SNARK and the fact that $\mathcal{A}$ runs in polynomial-time, we have that $(w_0, \perp, \sigma_0)$ is a valid witness except for negligible probability. Hence we focus on the case the $(w_0, \perp, \sigma_0)$ is valid.

By the perfectly correctness of the encryption scheme, we have $w = w_0$. Also note that given $(w_0 = w, \perp, \sigma)$ is valid, if $(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$ then we must have $\mathsf{Verify}(\mathsf{pk}_s, \mathsf{stmt}, \sigma_0) = 1$, which gives $\Pr[\mathsf{Expt}_0] \leq \Pr[\mathsf{Expt}_1] + \mathsf{negl}(\lambda)$.

Next we argue that $\Pr[\mathsf{Expt}_1^{\mathcal{A}}] \leq \mathsf{negl}(\lambda)$: Note that for all polynomial $\mathcal{A}$, there exists a polynomial-time algorithm $\mathsf{snark}.\mathcal{E}_{\mathcal{A}}$, where the subscript indicates that it uses non-blackbox access to the adversary $\mathcal{A}$.

Consider the following adversary for the signature scheme: Get pk_s from the game; Run $\mathsf{Expt}_1^{\mathcal{A}}$ with the same pk_s; Replace signature with oracle calls (hence sk_s is no longer); Output $(\mathsf{stmt}, \sigma_0)$, where stmt is given by the adversary $\mathcal{A}$ and σ_0 is given by the (SNARK) extractor. Since the signature scheme is unforgeable and that stmt has not been queried, we have that $\Pr[\mathsf{Expt}_1] \leq \mathsf{Verify}(\mathsf{pk}_s, \mathsf{stmt}, \sigma_0) \leq \mathsf{negl}(\lambda)$. □

□

B Omitted Proofs

B.1 Proof of Theorem 1

Proof of Theorem 1. The proof of perfect completeness is obvious. We now show that this transformation gives a zero knowledge and simulation sound extractable NIZK.

Proof of zero-knowledge. We now show that no polynomial-time adversary $\mathcal{A}$ can win the zero knowledge game except with negligible probability.

We construct the following hybrid games:

$\mathsf{Expt}_{\mathsf{A}}$**.** $\widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$ and $\widehat{\mathcal{P}}_1(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt}, w)$ are run as defined. Recall that $\widehat{\mathcal{P}}_1$ checks the witness w and then calls $(c, \pi) \leftarrow \widehat{\mathcal{P}}(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt})$.

$\mathsf{Expt}_{\mathsf{B}}$**.**

- $\widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$: Use the underlying simulator setup algorithm $\mathsf{nizk}.\widehat{\mathcal{K}}$. Return the simulated $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}, \mathsf{pk}_e)$, and trapdoor $\tau := (\mathsf{nizk}.\tau, \mathsf{sk})$.
- $\widehat{\mathcal{P}}_1(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt}, w)$: Abort if $(\mathsf{stmt}, w) \notin \mathcal{L}$. Output (c, π), where c is an encryption of $(\perp, \sigma)$ and $\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}, c))$.

$\mathsf{Expt}_{\mathsf{C},t}$**, where** t **is a polynomial function of** λ**.**

- $\widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$: Same as before.
- $\widehat{\mathcal{P}}_1(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt}, w)$: Check if $(\mathsf{stmt}, w) \in \mathcal{L}$, and abort otherwise. For the first $t-1$ queries, let c be an encryption of $(w, \perp)$, and output (c, π) where $\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}, c))$. However, for the t^{th} and all subsequent queries, behave the same as in $\mathsf{Expt}_{\mathsf{B}}$.

$\mathsf{Expt}_{\mathsf{D}}$**.**

- $\widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$: Same as before.
- $\widehat{\mathcal{P}}_1(\widehat{\mathsf{crs}}, \perp, \mathsf{stmt}, w)$: Check if $(\mathsf{stmt}, w) \in \mathcal{L}$, and abort otherwise. Let c be an encryption of $(w, \perp)$, and output (c, π) where $\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\widehat{\mathsf{crs}}, \tau, (\mathsf{stmt}, c))$.

$\mathsf{Expt}_{\mathsf{E}}$**.**

- $\widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$: Actually just run $\mathcal{K}(1^\lambda, \mathcal{L})$, and output $(\mathsf{crs}, \perp, \perp)$.

- $\widehat{\mathcal{P}}_1(\mathsf{crs}, \bot, \mathsf{stmt}, w)$: Actually just run $\mathcal{P}(\mathsf{crs}, \mathsf{stmt}, w)$.

First, Expt_A and Expt_B are indistinguishable by reduction to zero knowledge property of the underlying NIZK. Next, notice that Expt_B and $\mathsf{Expt}_{\mathsf{C},1}$ are identical. We can also prove for every polynomial function $t \geq 1$ that $\mathsf{Expt}_{\mathsf{C},t}$ and $\mathsf{Expt}_{\mathsf{C},t+1}$ are indistinguishable by a reduction to the semantic security of the underlying encryption scheme. Suppose $\mathcal{A}$ distinguishes between $\mathsf{Expt}_{\mathsf{C},t}$ and $\mathsf{Expt}_{\mathsf{C},t+1}$. Then we can construct an adversary $\mathcal{A}'$ for the semantic security game as follows:

- Generate $(\widehat{\mathsf{crs}}, \tau, ek) \leftarrow \widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$
- Call $b \leftarrow A^{\widehat{\mathcal{P}}^\dagger(\widehat{\mathsf{crs}}, \tau, \cdot, \cdot)}$, where $\widehat{\mathcal{P}}^\dagger(\widehat{\mathsf{crs}}, \tau, \mathsf{stmt}, w)$ checks if $(stmt, w) \in \mathcal{L}$, and then does as follows:
  - For the first $t-1$ queries, behave as in $\mathsf{Expt}_{\mathsf{C},\mathsf{t}}$.
  - For the $(t+1)^{th}$ and all subsequent queries, behave as in $\mathsf{Expt}_{\mathsf{C},t+1}$.
  - On the t^{th} query, let $\sigma := \Sigma.\mathsf{Sign}(\mathsf{sk}, \mathsf{stmt})$, and choose $m_1 := (w, \bot)$ and $m_2 := (\bot, \sigma)$ as the adversary's plaintexts in the sematic security game. Let $c^\dagger$ be the resulting challenge ciphertext obtained from the semantic security game. Output $(c^\dagger, \pi^\dagger)$, where $\pi^\dagger \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\widehat{\mathsf{crs}}, \tau, (\mathsf{stmt}, c^\dagger))$.
- Output b

This reduction succeeds because conditioned on the challenger choosing m_1 or m_2, the resulting distribution is identical to $\mathsf{Expt}_{\mathsf{C},t}$ or $\mathsf{Expt}_{\mathsf{C},t+1}$ respectively. Next, suppose $\mathcal{A}$ is able to distinguish between Expt_B and Expt_D. Let $t^* > 1$ be a polynomial function that bounds the number of oracle calls made by $\mathcal{A}$. Notice that $\mathsf{Expt}_{\mathsf{C},t^*}$ is identical to Expt_D when run with $\mathcal{A}$. By induction using the argument above, $\mathsf{Expt}_{\mathsf{C},1}$ is indistinguishable from $\mathsf{Expt}_{\mathsf{C},t^*}$. Finally, Expt_D is indistinguishable from Expt_E because the underlying proof system is zero knowledge.

Proof of simulation sound extractability. We construct the following extractor:

- $\mathcal{E}(\widehat{\mathsf{crs}}, \mathsf{ek}, \mathsf{stmt}, \pi')$: parse $\pi' := (c, \pi)$, and let $(w, \sigma) := \mathsf{Dec}(\mathsf{sk}_e, c)$. Output w.

We now show that no polynomial-time adversary $\mathcal{A}$ can win the simulation sound extractable game except with negligible probability.

Given that the encryption scheme is perfectly correct, we can assume that the (w, σ) decrypted by $\mathcal{E}$ is the unique plaintext that produces c as a ciphertext.

Suppose σ is a valid signature, but stmt is not one of the statements signed via an oracle query to $\widehat{\mathcal{P}}$. Since the underlying signature scheme is unforgeable, this event occurs with negligible probability.

Otherwise, suppose w is not a valid witness under language $\mathcal{L}$ (and hence (stmt, c) is not a statement in $\mathcal{L}'$). This reduces to breaking the computational soundness of the underlying NIZK. Note that the adversary $\mathcal{A}'$ for the computational soundness game must take in a real crs and must generate a valid proof for a false statement. We can therefore construct $\mathcal{A}'(\mathsf{crs})$ from $\mathcal{A}$ as follows: First, $\mathcal{A}'$ runs $(\widehat{\mathsf{crs}}, \tau, \mathsf{ek}) \leftarrow \widehat{\mathcal{K}}(1^\lambda, \mathcal{L})$, but then discards $\widehat{\mathsf{crs}}$. The crs provided as input will be used instead. Notice that crs and $\widehat{\mathsf{crs}}$ here are identically distributed, since both the soundness game and $\widehat{\mathcal{K}}$ both run the ordinary setup $\mathsf{nizk}.\mathcal{K}$ directly. Next, $\mathcal{A}'$ runs $(\mathsf{stmt}, (c, \pi)) \leftarrow \mathcal{A}^{\widehat{\mathcal{P}}(\mathsf{crs}, \tau, \cdot)}$ as in the simulation sound extractable game, and returns $((\mathsf{stmt}, c), \pi)$ to the computational soundness game, winning both games with similar probability. Notice that the oracle $\widehat{\mathcal{P}}(\mathsf{crs}, \tau, \cdot)$ does not have access to any trapdoor for the underlying NIZK and cannot generate false proofs. □

B.2 Proof of Theorem 4

Proof of Theorem 4. Completeness is obvious, we next prove it is also strongly simulation sound extractable and zero-knowledge.

Strong simulation sound extractability.

Lemma 10. *The construction is strongly simulation sound extractable.*

Proof. The game for strong simulation sound extractability is defined as follows:

Expt_0 (Actual game):

1. Setup:
   $\mathsf{nizk.crs} \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;
   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$. $\widehat{\mathsf{crs}} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$.
2. Define function $O(\mathsf{stmt}_x)$:
   $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu = f_{s_0}(\mathsf{pk}_s)$;
   $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, z_3; r_1)$;
   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}_x, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r_1, r_0, z_3, s_0))$; $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$;
   return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.
3. $(\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{O(\cdot)}(\widehat{\mathsf{crs}}, \mathsf{ek})$;
4. Parse $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$; $w \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$;
5. Let Q be the set of statement-proof pairs generated by $O(\cdot)$
   Output 1 *iff*: (1) $(\mathsf{stmt}, \pi) \notin Q$; and (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$; and (3) $(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$.

Expt_1 (Relax return condition):

1. Setup:
   $\mathsf{nizk.crs} \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;
   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$.
2. Define function $O(\mathsf{stmt}_x)$:
   $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu = f_{s_0}(\mathsf{pk}_s)$;
   $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, z_3; r_1)$;
   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}_x, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r_1, r_0, z_3, s_0))$; $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$;
   return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.
3. $(\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{O(\cdot)}(\widehat{\mathsf{crs}}, \mathsf{ek})$;
4. Parse $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$; $w \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$;
5. Let Q be the set of statement-proof pairs and T be the set of verification keys generated by $O(\cdot)$. The experiment outputs 1 *iff*:
   (1) $(\mathsf{stmt}, \pi) \notin Q$; and (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$; and (3) $\mathsf{pk}_s \notin T$; and (4) $\mu = f_{s_0}(\mathsf{pk}_s)$.

Claim 18. *If the underlying one-time signature scheme is strongly unforgeable, and that the underlying NIZK is sound, then we have* $\Pr[\mathsf{Expt}_0] \leq \Pr[\mathsf{Expt}_1] + \mathsf{negl}(\lambda)$.

Proof. Note that if $(\mathsf{stmt}, \pi) \notin Q$ and "pk_s has been generated by $O(\cdot)$", then the $(\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi)$ (from stmt and π) is a valid message/signature pair. Hence by the unforgeability of the signature scheme, we know that $(\mathsf{stmt}, \pi) \notin Q$ and "pk_s has been generated by $O(\cdot)$" happens with negligible probability, which allows us to focus on $\mathsf{pk}_s \notin T$.

The decrypted w is unique for all valid witnesses. Further, if some witness is valid for $\mathcal{L}'$ and that $(\mathsf{stmt}, w) \notin \mathcal{R}_{\mathcal{L}}$, we know it must be the case that there exists some s_0', such that ρ is a valid commitment of s_0' and that $\mu = f_{s_0'}(\mathsf{pk}_s)$, which implies $\mu = f_{s_0}(\mathsf{pk}_s)$, by the perfectly binding property. □

Expt_2 (Use simulation setup):

1. Setup:

   $(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x)$:

   $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu = f_{s_0}(\mathsf{pk}_s)$;

   $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, z_3; r_1)$;

   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}_1(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}_x, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r_1, r_0, z_3, s_0))$;

   Equivalent: $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, (\mathsf{stmt}_x, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho))$;

   $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$;

   return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.

3. $(\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{O(\cdot)}(\widehat{\mathsf{crs}}, \mathsf{ek})$;

4. Parse $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$; $w \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$;

5. Let Q be the set of statement-proof pairs and T be the set of verification keys generated by $O(\cdot)$. The experiment outputs 1 *iff*:
   (1) $(\mathsf{stmt}, \pi) \notin Q$; and (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$; and (3) $\mathsf{pk}_s \notin T$; and (4) $\mu = f_{s_0}(\mathsf{pk}_s)$.

Claim 19. *If the underlying NIZK is zero-knowledge, then we have* $\Pr[\mathsf{Expt}_1] \leq \Pr[\mathsf{Expt}_2] + \mathsf{negl}(\lambda)$.

Proof. By the zero-knowledge property, no polynomial-time algorithm can distinguish an honest setup from an simulation setup. Also note that our experiment runs in polynomial time. This completes the proof. □

Expt_3 (Separate s_0):

1. Setup:

   $(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

   $s_0, s_0', r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0'; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x)$:

   $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu = f_{s_0}(\mathsf{pk}_s)$; $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, z_3; r_1)$;

   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, (\mathsf{stmt}_x, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho))$;

   $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$;

   return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.

3. $(\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{O(\cdot)}(\widehat{\mathsf{crs}}, \mathsf{ek})$;

4. Parse $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$; $w \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$;

5. Let Q be the set of statement-proof pairs and T be the set of verification keys generated by $O(\cdot)$. The experiment outputs 1 *iff*:
   (1) $(\mathsf{stmt}, \pi) \notin Q$; and (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$; and (3) $\mathsf{pk}_s \notin T$; and (4) $\mu = f_{s_0}(\mathsf{pk}_s)$.

Claim 20. *If the underlying commitment scheme is computationally hiding, then we have* $\Pr[\mathsf{Expt}_2] \leq \Pr[\mathsf{Expt}_3] + \mathsf{negl}(\lambda)$.

Proof. By the hiding property, no polynomial algorithm can distinguish the commitment of two elements. □

Expt_4 (Replace PRF):

Let F be a true random function.

1. Setup:

   $(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

   ~~s_0~~, $s_0', r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0'; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x)$:

   $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu = F(\mathsf{pk}_s)$;

   $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, z_3; r_1)$;

   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, (\mathsf{stmt}_x, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho))$;

   $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$;

   return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.

3. $(\mathsf{stmt}, \pi) \leftarrow \mathcal{A}^{O(\cdot)}(\widehat{\mathsf{crs}}, \mathsf{ek})$;

4. Parse $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$; $w \leftarrow \mathsf{Dec}(\mathsf{ek}, c)$;

5. Let Q be the set of statement-proof pairs and T be the set of verification keys generated by $O(\cdot)$. The experiment outputs 1 *iff*:
   (1) $(\mathsf{stmt}, \pi) \notin Q$; and (2) $\mathcal{V}(\widehat{\mathsf{crs}}, \mathsf{stmt}, \pi) = 1$; and (3) $\mathsf{pk}_s \notin T$; and (4) $\mu = F(\mathsf{pk}_s)$.

Claim 21. *If the underlying* PRF *is secure, then we have* $\Pr[\mathsf{Expt}_3] \leq \Pr[\mathsf{Expt}_4]$.

Proof. Since the PRF is secure, no polynomial-time algorithm can distinguish F from f_{s_0}.

We convert Expt_4 to an adversary for the security game of PRF, similar to the proof of Claim 13. It corresponds to Expt_4 and Expt_3 when running with random function and PRF, respectively. This completes the proof. □

Claim 22. *We have* $\Pr[\mathsf{Expt}_4] \leq 2^{-\lambda}$.

Proof. Since $\mathsf{pk}_s \notin T$, we know that $F(\mathsf{pk}_s)$ has not been queried before. Hence we may view $F(\mathsf{pk}_s)$ as newly generated random bits independent from μ. This completes the proof. □

□

Computationally Zero-knowledge.

Lemma 11. *The construction is computationally zero-knowledge.*

Proof. We prove this by showing a series of indistinguishable hybrids, where the first one is the simulation setup and the last one is the honest setup.

Expt_0 (actual game):

1. Setup:

   $\mathsf{nizk.crs} \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x, w)$:

   Abort if $(\mathsf{stmt}_x, w) \notin \mathcal{R}_{\mathcal{L}}$; $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu = f_{s_0}(\mathsf{pk}_s)$;

   $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, z_3; r_1)$;

   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r_1, r_0, z_3, s_0))$;

   $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$; return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.

3. $b \leftarrow \mathcal{A}^{O(\cdot,\cdot)}(\widehat{\mathsf{crs}})$; Output b.

Expt_1 (use simulation setup):

1. Setup:
   $(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\widehat{\mathcal{K}}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;
   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.
2. Define function $O(\mathsf{stmt}_x, w)$:
   Abort if $(\mathsf{stmt}_x, w) \notin \mathcal{R}_{\mathcal{L}}$;
   $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu = f_{s_0}(\mathsf{pk}_s)$;
   $z_3, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, z_3; r_1)$;
   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}_1(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r_1, r_0, z_3, s_0))$;
   Equivalent: $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho))$;
   $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$; return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.
3. $b \leftarrow \mathcal{A}^{O(\cdot,\cdot)}(\widehat{\mathsf{crs}})$; Output b.

Claim 23. *If the underlying NIZK is zero-knowledge, then we have* $\Pr[\mathsf{Expt}_1] \approx \Pr[\mathsf{Expt}_0]$.

Expt_2 (Encrypt true witness):

1. Setup:
   $(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\widehat{\mathcal{K}}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;
   $s_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.
2. Define function $O(\mathsf{stmt}_x, w)$:
   Abort if $(\mathsf{stmt}_x, w) \notin \mathcal{R}_{\mathcal{L}}$; $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu = f_{s_0}(\mathsf{pk}_s)$;
   ~~z_3~~, $r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, w; r_1)$;
   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho))$;
   $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$; return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.
3. $b \leftarrow \mathcal{A}^{O(\cdot,\cdot)}(\widehat{\mathsf{crs}})$; Output b.

Claim 24. *If the underlying encryption scheme is semantically secure, then we have* $\Pr[\mathsf{Expt}_2] \approx \Pr[\mathsf{Expt}_1]$.

Proof. By Claim 2, no polynomial-time algorithm can distinguish an oracle that always encrypt the first message from one that always encrypt the second.

We construct the following adversary: Run Expt_2 but get pk_e from the game; Replace generating c with the oracle call, with the two messages being z_3 and w.

Note that encrypting z_3 is identical to the normal Expt_2, while encrypting w is identical to running Expt_1. The claim follows. □

Expt_3 (Separate s_0):

1. Setup:
   $(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\widehat{\mathcal{K}}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;
   $s_0, s_0', r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s_0'; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.
2. Define function $O(\mathsf{stmt}_x, w)$:
   Abort if $(\mathsf{stmt}_x, w) \notin \mathcal{R}_{\mathcal{L}}$; $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu = f_{s_0}(\mathsf{pk}_s)$;

$r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, w; r_1)$;

$\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho))$;

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$; return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.

3. $b \leftarrow \mathcal{A}^{O(\cdot,\cdot)}(\widehat{\mathsf{crs}})$; Output b.

Claim 25. *If the underlying commitment scheme is computationally hiding, then we have* $\Pr[\mathsf{Expt}_3] \approx \Pr[\mathsf{Expt}_2]$.

Expt_4 (Replace PRF):

1. Setup:

$(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\widehat{\mathcal{K}}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

~~s_0~~, $s'_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s'_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x, w)$:

Abort if $(\mathsf{stmt}_x, w) \notin \mathcal{R}_{\mathcal{L}}$;

$(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu \xleftarrow{\$} \{0,1\}^\lambda$;

$r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, w; r_1)$;

$\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho))$;

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$; return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.

3. $b \leftarrow \mathcal{A}^{O(\cdot,\cdot)}(\widehat{\mathsf{crs}})$; Output b.

Claim 26. *If the underlying* PRF *is secure and that the underlying one-time signature scheme is unforgeable, then we have* $\Pr[\mathsf{Expt}_4] \approx \Pr[\mathsf{Expt}_3]$.

Proof. First note that the generated pk_s's are distinct except for negligible probability, as otherwise it would break the one-time signature scheme.

Also, we can replace f_{s_0} in Expt_3 with a true random function F, which is identical to Expt_4 when the pk_s's are all distinct. □

Expt_5 (Use $\mathsf{nizk}.\widehat{\mathcal{P}}_1$):

1. Setup:

$(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau) \leftarrow \mathsf{nizk}.\widehat{\mathcal{K}}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

$s'_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s'_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x, w)$:

Abort if $(\mathsf{stmt}_x, w) \notin \mathcal{R}_{\mathcal{L}}$;

$(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu \xleftarrow{\$} \{0,1\}^\lambda$;

$z_1, z_2, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, w; r_1)$;

$\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\widehat{\mathcal{P}}_1(\mathsf{nizk}.\widehat{\mathsf{crs}}, \mathsf{nizk}.\tau, (\mathsf{stmt}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r_1, z_1, w, z_2))$;

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$; return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.

3. $b \leftarrow \mathcal{A}^{O(\cdot,\cdot)}(\widehat{\mathsf{crs}})$; Output b.

Claim 27. *We have* $\Pr[\mathsf{Expt}_5] = \Pr[\mathsf{Expt}_4]$.

Expt_6 (Use $\mathcal{P}$):

1. Setup:

$\mathsf{nizk.crs} \leftarrow \mathsf{nizk}.\mathcal{K}(1^\lambda, \mathcal{L}')$; $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{KeyGen}_{\mathrm{Enc}}(1^\lambda)$;

$s'_0, r_0 \xleftarrow{\$} \{0,1\}^\lambda$; $\rho := \mathsf{comm}(s'_0; r_0)$; $\widehat{\mathsf{crs}} := (\mathsf{nizk.crs}, \mathsf{pk}_e, \rho)$.

2. Define function $O(\mathsf{stmt}_x, w)$:

   Abort if $(\mathsf{stmt}_x, w) \notin \mathcal{R}_{\mathcal{L}}$; $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{KeyGen}_{\mathrm{Sig}}(1^\lambda)$; $\mu \xleftarrow{\$} \{0,1\}^\lambda$;

   $z_1, z_2, r_1 \xleftarrow{\$} \{0,1\}^\lambda$; $c = \mathsf{Enc}(\mathsf{pk}_e, w; r_1)$;

   $\mathsf{nizk}.\pi \leftarrow \mathsf{nizk}.\mathcal{P}(\mathsf{nizk.crs}, (\mathsf{stmt}, c, \mu, \mathsf{pk}_s, \mathsf{pk}_e, \rho), (r_1, z_1, w, z_2))$;

   $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_s, (\mathsf{stmt}, c, \mu, \mathsf{nizk}.\pi))$; return $\pi := (c, \mu, \mathsf{nizk}.\pi, \mathsf{pk}_s, \sigma)$.

3. $b \leftarrow \mathcal{A}^{O(\cdot,\cdot)}(\widehat{\mathsf{crs}})$; Output b.

Claim 28. *If the underlying NIZK is zero-knowledge, then we have* $\Pr[\mathsf{Expt}_6] \approx \Pr[\mathsf{Expt}_5]$.

Note that the last experiment is exactly the definition. This completes the proof for the zero-knowledge part.

□

□

B.3 An Ideal Functionality for (Weak) SSE-NIZKs

In Section 2, we defined (weak) SSE-NIZK, and in Section 5 defined a strengthened definition. The difference between these two definitions is about whether the adversary *without a witness* is able to generate *new* proofs for true statements that have already been proven by an honest party *with a witness*. Although the weaker definition is shown in Hawk [29] to be sufficient for UC-secure applications, the ideal functionality $\mathcal{F}_{\text{WEAK-NIZK}}$ as defined in [26] is only shown to be realizable by the stronger definition. We therefore now provide a slightly-weakened variation of $\mathcal{F}_{\text{WEAK-NIZK}}$ that allows the adversary to generate new proofs for already-proven statements, and therefore should be realizable from a (weak) SSE-NIZK. However, we do not provide a proof, nor do we prove that applications using the functionality from [26] are still secure when using our functionality instead.

Functionality $\mathcal{F}_{\text{WEAK-NIZK}}(sid, \mathcal{L})$

Prove: On input $(\texttt{prove}, \mathsf{stmt}, w)$ from party P ignore if $(\mathsf{stmt}, w) \notin \mathcal{L}$. Send $(\texttt{prove}, \mathsf{stmt})$ to $\mathcal{A}$ and wait for answer $(\texttt{proof}, \pi)$. Upon receiving the answer store $(\texttt{proof}, \mathsf{stmt}, \pi)$ and send $(\texttt{proof}, \pi)$ and send $(\texttt{proof}, \pi)$ to P.

Maul: On input $(\texttt{maul}, \mathsf{stmt}, \pi)$ from A, ignore unless (stmt, x) is already stored for some π'. Record (stmt, π) and send $\texttt{ok}$ to $\mathcal{A}$.

Verify: On input $(\texttt{verify}, \mathsf{stmt}, \pi)$ from party P, check whether (stmt, π) is stored. If not, then send $(\texttt{verify}, \mathsf{stmt}, \pi)$ to $\mathcal{A}$ and wait for an answer $(\texttt{witness}, w)$. Upon receiving the answer, check whether $(\mathsf{stmt}, w) \in \mathcal{L}$ and in that case, store (stmt, π). If (stmt, π) has been stored return $(\texttt{verification}, 1)$ to P, and otherwise return $(\texttt{verification}, 0)$ to P.

C The Concrete Hardness of SIS and Ajtai Hash

The Ajtai hash is a lattice-based collision-resistant hash function that shows great promise for use in SNARK-friendly cryptographic applications (e.g., Merkle trees, just for one example). [2, 7].

Ben-Sasson et al. have suggested a parameterization that they conjecture to be secure based on heuristic extrapolation of earlier estimates [7]. However, this extrapolation is delicate, and a lack of confidence in this heuristic has apparently stalled its use (e.g., it was not been used in Zerocash [5], a practical real-world application for which this construction would provide a significant performance enhancement).

The collision-resistance of the Ajtai hash relies on the hardness of the SIS problem. Hence, it is natural to estimate the hardness of Ajtai hash by evaluating its corresponding SIS problem. So far, the best concrete security analysis of SIS is based on empirical observation of the running time of the best known attack algorithms (LLL,BKZ,BKZ2.0) over a range of parameterizations [1, 20, 33]. However, to reap the performance benefits of using this within a SNARK, we must choose a parameterization that lies outside the range covered in this experiment (i.e., the arity of the lattice must be a large prime $q \sim 2^{254}$, the native field of the SNARK, whereas prior experiments cover only small prime $n^2 \leq p \leq n^8$). Therefore it is not apparent a priori if the concrete security estimates can safely be extrapolated here.

In this note, we review the concrete security analysis for SIS, and reproduce (in part) the empirical experiments from [38], extended to the parameter ranges appropriate for SNARK-friendly crypto. We provide a step-by-step algorithm for concrete security analysis of arbitrary SIS instance, and use the algorithm to estimate the security of Ajtai hash functions.

Based on our experiments and calculations, we conclude that:

1. It is indeed reasonable to extrapolate from prior concrete hardness estimates of SIS based primitives, such as Ajtai hash, even in the SNARK-friendly parameter ranges.
2. To achieve 80 bits of security (against the best-known attacks today), we should increase the dimension of Ajtai hash to $n = 3$, rather than $n = 1$ as suggested in [7] (The authors use d instead of n in [7]).

C.1 Preliminaries

Notations. We denote with log the logarithm to base 2. Vectors and matrices are written in boldface, e.g., $\mathbf{v}$ and $\mathbf{M}$. We use $|\mathbf{v}|_p$ to denote the l_p norm of vector $\mathbf{v}$. In particular, we denote with $|\mathbf{v}|$ the l_2 norm of vector $\mathbf{v}$.

Definition 1 (Lattice)**.** *A (full-dimensional) lattice in $\mathbb{R}^m$ is a discrete subgroup $L = \{\mathbf{Bx} \,|\, \mathbf{x} \in \mathbb{Z}^m\}$, where typically $\mathbf{B} = [\mathbf{b_1}, \ldots, \mathbf{b_m}] \in \mathbb{Z}^{m \times m}$ is a matrix of linearly independent vectors. The matrix $\mathbf{B}$ is a basis of the lattice L and we write $L = L(\mathbf{B})$. The rank of a lattice L is the rank of the basis matrix $\mathbf{B}$. If the rank equals m, we say that L is full-rank..*

The Shortest Vector Problem (SVP) is probably the most fundamental hard problem in lattice literature [34].

Definition 2 (Shortest Vector Problem (SVP))**.** *Given a basis $\mathbf{B}$ of L and an approximation factor $\gamma \geq 1$, the task of SVP is to find a set $\mathbf{v} \in L$ such that $|\mathbf{v}| \leq \gamma L_0$, where L_0 denotes the shortest vector in lattice L.*

We are only concerned with a type of lattices called "q-array" lattice. Note that every q-ary lattice is full-rank.s

Definition 3 (q-ary)**.** *A lattice L is called a q-ary if $q\mathbb{Z} \subseteq L$.*

For $q \in \mathbb{N}$ and $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, we define two most important q-arys.

$$\Lambda_q(\mathbf{A}) = \{\mathbf{w} \in \mathbb{Z}^n \,|\, \exists \mathbf{e} \in \mathbb{Z}^m \mathbf{A}^\top \mathbf{e} = \mathbf{w} \,(mod\ q)\} \tag{1}$$

$$\Lambda_q^\perp(\mathbf{A}) = \{\mathbf{v} \in \mathbb{Z}^m \,|\, \mathbf{Av} = \mathbf{0} \,(mod\ q)\} \tag{2}$$

Lemma 12. *Let q be a prime and $m = O(n\log(n))$. With high probability, the rows of $\mathbf{A}$ are linearly independent over $\mathbb{Z}_q$ and $det(\Lambda_q^{\perp}(\mathbf{A})) = q^n$.*

The Short Integer Solution (SIS) problem is defined over the q-arry $\Lambda_q^{\perp}(\mathbf{A})$.

Definition 4 (Short Integer Solution (SIS))**.** *Given $n, m, q \in \mathbb{N}$, a randomly picked $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, and a norm bound $1 \leq \beta < q$, the SIS problem, denoted as $SIS(n, m, q, \beta)$, is to find $\mathbf{v} \in \Lambda_q^{\perp}(\mathbf{A})$ with $0 < |\mathbf{v}| \leq \beta$.*

Definition 5 (Ajtai Hash)**.** *Given $n, m, q \in \mathbb{N}$, a randomly picked $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, the Ajtai Hash $h_{Ajtai(n,m,q)} : \{0,1\}^m \to \mathbb{Z}_q^n$ is defined as*

$$h(\mathbf{x}) = \mathbf{A}\mathbf{x} \ (mod\ q) \tag{3}$$

C.2 High Level Idea

Our goal is to study the concrete hardness of the Ajtai hash function. Specifically, given an instance $h_{Ajtai(n,m,q)}$, what level of bit-security does it satisfy?

The collision-resistance of the Ajtai hash reduces to SIS: In particular, if the adversary can find a collision for $h_{Ajtai(n,m,q)}$, then it would also break $SIS(n, m, q, \sqrt{m})$. Hence, it is natural to estimate the hardness of the Ajtai hash by evaluating its corresponding SIS problem. We focus on the security analysis of the SIS problem next.

The best known algorithm for solving SIS relies on Lattice Reduction Algorithms such as LLL and BKZ. Given the original input lattice basis $\mathbf{B}$, the goal of lattice basis reduction is to find a basis $\bar{\mathbf{B}}$ with short, nearly orthogonal vectors. If the basis after reduction $\bar{\mathbf{B}}$ is of "good quality", the shortest row vector in the reduced basis can be a relatively good estimation of the shortest vector in the lattice. We adopt the convention that the first non-zero vector, denoted as $\bar{\mathbf{b_0}}$, is the shortest vector in the reduced basis $\bar{\mathbf{B}}$. Intuitively, given $SIS(n, m, q, \beta)$, the lattice reduction based attack keeps reducing the basis of the q-array $\Lambda_q^{\perp}(\mathbf{A})$ until its $\bar{\mathbf{b_0}}$ is $\leq \beta$.

In practice, finding the shortest vector for hard lattice instances requires at least exponential running time. The hardness of SIS is estimated by the time cost for the lattice reduction algorithm to find a basis with a certain level of "quality". The quality of a basis B of m-dimensional lattice L is characterized by the root Hermite factor δ_0, which is defined such that $|\mathbf{b_0}| = \delta_0^m |det(L)|^{1/m}$. Notice that the q-array $\Lambda_q^{\perp}(\mathbf{A})$ of $SIS(n, m, q, \beta)$ has determinant $det(\Lambda_q^{\perp}(\mathbf{A})) = q^n$. The concrete hardness of $SIS(n, m, q, \beta)$ is estimated by the time cost the for lattice reduction algorithm to find a basis with $\delta_0 = (\beta/q^{n/m})^{1/m}$.

C.3 Lattice Reduction Algorithm: BKZ, BKZ2.0

Several works in literature have studied the running time of BKZ [1,20,32,38]. Specifically, they provide estimated running time for BKZ/BKZ2.0 to find a basis with certain δ_0 root Hermite factor. The running time below is measured by clock cycles.

1. Ruckert and Schneider [38] provides a table of δ_0 and BKZ running time based the estimation by extrapolation on "dollar-days" cost.
2. Lindner and Peikert [32] gives the estimate on BKZ as: $T_{LP11} = 2^{1.8/\log\delta_0 - 78.9}$
3. Albrecht et al. [33] extrapolate a model similar to [32] on BKZ2.0 as: $T_{ACF15} = 2^{0.009/\log^2\delta_0 + 4.1}$
4. Chen and Nguyen [20] provide a simulation-based estimate for BKZ2.0. We denote the estimated running time as $T_{CN12}(m^*, \delta_0)$ (The simulation also takes the lattice dimension m^* as input).

Notice that the experiments in the above works are run with relatively small prime modulo p. In order to show that the empirical results also apply under the setting where the prime modulo

n	δ_0	$m = 2n \log q$	m^*	$\lambda = \log T_{ACF15}$	$\lambda = \log T_{CN12}$
1	1.0139	508	114	26.8	27.8
2	1.0085	1016	204	63.8	60.6
3	1.0064	1524	289	111.1	107.5
4	1.0052	2032	379	167.2	162.0

Table 4: Security level of the Ajtai Hash for $n = 1, 2, 3, 4$. Prime modulo is set to $q = 2^{254}$. We set m equal to $2n \log q$ (i.e., as appropriate for a Merkle tree application).

is very large $q >> p$. We repeat the experiments in [38] on BKZ with $q = 2^{254}$. Specifically, we run the BKZ reduction algorithm on q-array of randomly generated SIS instance with various (n, m, β) settings. Our running time matches theirs except for a constant speed up (due to a faster CPU) whenever the root Hermite factors under the two settings are equivalent.

BKZ2.0 [19] is an upgraded version of BKZ. These improvements include early termination, extreme pruning, limiting the enumeration radius to the Gaussian Heuristic, and local block pre-processing [19]. We assume with confidence that the empirical results for BKZ2.0 apply in the setting with large prime q. We use the BKZ2.0 estimations T_{ACF15} and T_{CN12} in the security analysis next.

Set $q = 2^{254}$. We evaluate the concrete security level of the Ajtai hash $h_{Ajtai(n,m,q)}$ for $n = 1, 2, 3, 4$ by calling $\mathbf{Security - SIS}$ on input $n, q, \sqrt{m}$. Note that we require input length $m > n \log q$ in order to obtain a hash function that compresses its input. Also, $m \geq 2n \log q$ is required for Merkle Tree applications. See Table 4.

C.4 Calculating The Bit-Security

C.4.1 Choose the optimal sample size m^*

The root Hermite factor δ_0 measures the quality of a lattice basis. Specifically, assume q-array lattice $\Lambda_q^{\perp}(\mathbf{A})$ of $SIS(n, m, q, \beta)$ has root Hermite factor δ_0, its shortest basis vector $\mathbf{b_0}$ satisfies

$$|\mathbf{b_0}| = \delta_0^m q^{n/m}. \tag{4}$$

The goal of the lattice reduction algorithm is to find the shortest possible $\mathbf{b_0}$. The right hand side is minimized when

$$m = m^* = \sqrt{\frac{n \log q}{\log \delta_0}}. \tag{5}$$

This optimal value m^* is the optimal sample size for lattice reduction algorithm, and is sometimes called the "optimal sub-dimension" [34]. Following many other works [1, 20, 32, 33, 38], we assume m^* is always chosen for BKZ/BKZ2.0.

C.4.2 Calculate the bit-security of SIS

Recall that the security level of $SIS(n, m, q, \beta)$ can be estimated by the time cost for lattice reduction algorithm to find a basis with $\delta_0 = (\beta/q^{n/m})^{1/m}$. The problem here is that m is chosen as the optimal value $m^* = \sqrt{\frac{n \log q}{\log \delta_0}}$, which is an expression of δ_0. Following the arguments in [38], we give a simple expression of the optimal m^* that is only dependent on (n, q, β) for SIS problem.

$$m^* = \lceil \frac{2n \log q}{\log \beta} \rceil. \tag{6}$$

Fixing δ_0 value, the minimum of $\delta_0^m q^{n/m}$ achieves when $m = m^* = \sqrt{\frac{n \log q}{\log \delta_0}}$. Equivalently, fixing $m = m^*$, some lattice reduction algorithm can find a basis with $\delta_0 = 2^{n \log q / m^{*2}}$. Therefore, there is some lattice reduction algorithm can find vector of length $\delta_0^{m^*} q^{n/m^*} = 2^{n \log q / m^*} q^{n/m^*} = q^{2n/m^*}$ in m^*-dimensional lattice. It requires $q^{2n/m^*} \leq \beta$ to break $SIS(n, m, q, \beta)$, which gives $m^* = \lceil \frac{2n \log q}{\log \beta} \rceil$.

Combining the above arguments gives us the following algorithm for calculating the bit-security of the SIS problem.

Input : (n, q, β)
Output: λ
Compute $m^* = \lceil \frac{2n \log q}{\log \beta} \rceil$;
Compute $\delta_0 = (\beta / q^{n/m^*})^{1/m^*}$;
Compute T_{BKZ} using the BKZ2.0 estimation either by $T_{ACF15}(\delta_0)$ or $T_{CN12}(m^*, \delta_0)$;
Return $\lambda = \log T_{BKZ}$.

Algorithm 1: Calculate the bit-security of $SIS(n, m, q, \beta)$