Proof of Exponentiation: Enhanced Prover Efficiency for Algebraic Statements

Zhuo Wu[1,2], Shi Qi[1,2], Xinxuan Zhang[1,2], Yi Deng[1,2*], Kun Lai[3], Hailong Wang[4]

[1*]State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.
[2]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China.
[3]Independent Researcher.
[4]Digital Technologies, Ant Group.

*Corresponding author(s). E-mail(s): deng@iie.ac.cn;
Contributing authors: wuzhuo@iie.ac.cn; qishi@iie.ac.cn; zhangxinxuan@iie.ac.cn; me@imlk.top; whl383799@antgroup.com;

Abstract

Recent years have seen the widespread adoption of zkSNARKs constructed over small fields, including but not limited to, the Goldilocks field, small Mersenne prime fields, and tower of binary fields. Their appeal stems primarily from their efficacy in proving computations with small bit widths, which facilitates efficient proving of general computations and offers significant advantages, notably yielding remarkably fast proving efficiency for tasks such as proof of knowledge of hash preimages. Nevertheless, employing these SNARKs to prove algebraic statements (e.g., RSA, ECDSA signature verification) presents efficiency challenges, particularly in critical applications like zk-bridges and zkVMs that require verifying standard cryptographic primitives.

To address this problem, we first define a new circuit model: arithmetic circuits with additional *exponentiation gates*. These gates serve as fundamental building blocks for establishing more intricate algebraic relations. Then we present a *Hash-committed Commit-and-Prove (HCP)* framework to construct Non-interactive Zero-knowledge (NIZK) proofs for the satisfiability of these circuits. Specifically, when proving knowledge of group exponentiations in discrete logarithm hard groups and RSA groups, compared to verifying complex group exponentiations within SNARK circuits, our approach requires proving only more lightweight computations within the SNARK, such as zk-friendly hash functions (e.g., Poseidon hash function). The number of these lightweight computations depends solely on the security parameter. This differentiation leads to substantial speedups for the prover relative to direct SNARK methods, while maintaining competitive proof size and verification cost.

Keywords: zkSNARK, Sigma Protocol, Algebraic Statement

1 Introduction

Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zkSNARKs) are extremely short and non-interactive zero-knowledge proofs [1]. Over the past decade, SNARKs have garnered significant interest, leading to the development of a wide range of efficient constructions [2–11]. Various SNARKs designed to prove arithmetic circuit satisfiability have made great strides in efficiency and scalability.

Motivation. A significant challenge persists in proving algebraic statements, including but not limited to, elliptic curve operations and exponentiation of large integers. This challenge arises because algebraic operations, which are often defined over large prime fields or different modulus, cannot be directly represented or efficiently computed within the native field arithmetic of the SNARK. This limitation is particularly acute when the native field modulus is relatively small, as is frequently the case for SNARKs optimized for proving statements about computations involving small bit widths (such as cryptographic hash functions like SHA256). Examples of such small fields include tower of binary fields [12], small Mersenne primes [13], or the Goldilocks field [14].[1] Consequently, when employing these SNARKs to prove such algebraic statements, these operations must be emulated using additions and multiplications within the native field arithmetic circuits, known as the "Non-Native Arithmetic"(NNA).[2] This imposes significant prover overhead, primarily stemming from the substantial number of constraints required to emulate a large field using operations in a small field. Small-field based SNARKs have become increasingly crucial for practical real-world applications in recent years. This trend is driven by the fact that mapping general computation, particularly operations on small bit widths, onto circuits over very large fields leads to considerable redundancy and inefficiency. Crucially, the reliance on small-field SNARKs is driven by the stringent performance requirements of modern infrastructure, such as zero-knowledge Virtual Machines (zkVMs) and cross-chain bridges. These systems depend fundamentally on the superior recursion efficiency and hardware-friendly arithmetic of small fields to achieve scalability. However, they face an unavoidable conflict: they are frequently required to verify standard, external cryptographic primitives (e.g., ECDSA signatures for blockchain consensus or RSA for TLS handshakes) that are mathematically misaligned with small fields. Reverting to

[1]Notably, the field-agnostic SNARKs [9, 11, 15, 16] can also operate over small fields.

[2]Specifically, verifying a single group exponentiation (e.g., on Secp256k1) inside a SNARK (e.g., over BN254) requires simulating field arithmetic for a different modulus. This introduces a heavy overhead: a single group addition can cost hundreds of constraints, and a full scalar multiplication (requiring ≈ 256 additions/doublings) results in tens of thousands of constraints, becoming the dominant prover bottleneck.

large-field SNARKs solely to accommodate these algebraic operations is often infeasible, as it would sacrifice the critical gains in recursion depth and hash throughput that define modern high-performance proving systems.

Therefore, this work explores new approaches to improve prover efficiency for algebraic statements, towards all SNARKs that are highly efficient for proving hash preimages. The hash functions can be zk-friendly hash functions (e.g., Poseidon hash function [17]) or cryptographic hash functions (e.g., SHA256) as long as they can be efficiently verified within SNARKs. Consequently, our proposed method inherently extends its benefits to SNARKs over large fields; we provide experimental results demonstrating this broader applicability in Section 5. We clarify the meaning of "efficient for proving hash preimages" in the conclusion in Section 5.

1.1 Background

We begin by considering a proof system that is highly efficient for proving hash preimages. The challenge is how to efficiently prove algebraic statements within this fixed native field proof system. We identify the most computationally expensive operation within these algebraic statements as group exponentiation. The subsequent problem is therefore how to efficiently prove statements about group exponentiation within a proof system requiring field emulation. Note that this refers not to proving the correctness of the exponentiation computation itself, but to proving statements about the exponentiation, such as demonstrating knowledge of x and y such that $g^x = y$ or knowledge of g and y such that $g^x = y$.

State-of-the-art methods. To begin, let us review the existing methods for tackling the problems mentioned above. xJsnark [18] and the CRT method [19–21], optimize the emulation of arithmetic modulo n' using additions and multiplications modulo n, where $n \neq n'$. The Windowed method [22] and Eagen et al. [23] each propose techniques that reduce the number of multiplications required to prove exponentiations. [3] Alternatively to directly proving algebraic statements using SNARKs via field emulation in arithmetic circuits, another approach is to employ *Commit-and-Prove*(CP) SNARKs [24–32]. For example, to prove knowledge of x such that "SHA256(G^x) $= y$" where the algebraic part is "$G^x = y'$" and the non-algebraic part is "SHA256(y') $= y$". Such composite statements can be proven by CP-SNARKs. This approach typically involves using Pedersen commitments as external commitments, constructing corresponding Sigma protocols to prove algebraic operations, while non-algebraic operations are executed within SNARKs. Except the above works, optimizations also exist for specific cases, including leveraging pairings in bilinear groups [33], utilizing cycles of curves for recursive proofs [34, 35], binding the Schnorr protocol in Circuits [36] and proving statements of values hidden in linearly homomorphic primitives [37]. Notably, this work considers general scenarios.

Field Emulation. One approach is to directly embed the group exponentiation operation within the SNARK circuits. Techniques optimized for NNA, as previously

[3]Typically, proving the group exponentiation g^x can be done by proving approximately $\log_2 |x|$ group multiplications.

mentioned, can be employed to implement non-native field multiplication. Group multiplications can then be composed of several non-native field additions and multiplications. Proving a group exponentiation $y = g^x$ then effectively decomposes into verifying a sequence of these group multiplications. A standard binary exponentiation method, for instance, necessitates proving up to $2\log_2|x|$ group multiplications within the native field circuit. The Windowed methods can reduce the number of point additions for elliptic curve scalar multiplication, the overall prover efficiency gain is often not substantial.

Sigma protocols under Pedersen Commitment. Alternatively, another approach is to use Pedersen-committed Sigma protocols for proving algebraic statements, one can select a discrete-log hard group with a generator of a specific order for the Pedersen commitments. By aligning this order with the modulus of the finite field over which the algebraic statements are defined, the homomorphic properties of Pedersen commitments (specifically, their additive homomorphism and the ability to prove multiplicative relationships between committed values) naturally simulate the operations within that algebraic domain. As an example, to prove $x_1 + x_2 = x_3 \mod n$, one can select generator g of order n, then $g^{x_1} \cdot g^{x_2} = g^{x_3}$ implies the addition equation. Building upon this, sigma protocols can be further designed for algebraic statements [24, 26, 28, 38].

The practical implementation of Pedersen commitments introduces several considerations, notably concerning the need to select generators within a discrete logarithm hard group of a specified order. This selection process can be infeasible in certain scenarios, such as when employing complex multiplication methods within elliptic curve settings. Alternatively, one could utilize large prime-order subgroups of groups $\mathbb{Z}_p^*$. However, for the discrete logarithm problem in such groups, subexponential-time algorithms are known to exist, this potentially necessitating very large modulus (such as 3072 bits) to ensure sufficient security. If the group order of the generator does not match the modulus of the algebraic statement's field, field emulation under Pedersen commitments becomes necessary (e.g., the ddlog protocol in [26] necessitates a huge amount of range proofs). The need to ensure the consistency of a set of witnesses between SNARKs and Pedersen commitments must also be considered. Although the overhead of "glue" proofs is very small when the SNARK used is based on elliptic curves [28, 30], the cost of such "glue" proofs needs to be considered when leveraging SNARKs based on Merkle tree commitments that use small fields.

Our approach. We observe that by combining SNARK and sigma protocols through hash functions, we can construct a new form of NIZKs for a broad spectrum of algebraic statements without relying on Pedersen commitments. This approach improves the efficiency of proof generation by reducing the size of the verification circuit required to prove algebraic operations in SNARKs. Specifically, to prove claims about group exponentiation g^x, we eliminate the need to verify $\log|x|$ group multiplications within the circuit. Instead, the verification circuit is only required to verify a few number of group multiplications proportional to the security parameter, along with very lightweight and easily provable computations, such as zk-friendly hash functions. This leads to a substantial optimization in the prover efficiency.

1.2 Contributions

This work introduces a new technique to construct NIZKs for algebraic statements that achieves practical high efficiency. Specifically, we focus on proving algebraic relations in discrete logarithm hard groups and RSA groups. Instead of relying on Sigma protocols under Pedersen commitments or embedding all algebraic operations in SNARK circuits, we construct proofs via a new *Hash-committed Commit-and-Prove* framework. We also provide concrete implementations to demonstrate its feasibility and performance.

Hash-committed Commit-and-Prove framework. We start with the definition of *Exponentiation Gate* (EXP gate), which performs group exponentiation over a given cyclic group. EXP gates serve as fundamental components for constructing various algebraic statements. Then we develop a *Hash-committed Commit-and-Prove* (HCP) framework for proving the satisfiability of arithmetic circuits composed of ADD, MUL and EXP gates.

Sigma Argument of Knowledge. We develop a special type of Sigma protocol termed *Sigma Argument of Knowledge* (Sigma AoK) to efficiently prove EXP gates. These protocols enhance prover efficiency and minimize the proof size to only a few group elements and a SNARK proof. We then propose two optimizations to further accelerate performance by exploiting the unique structure of our proposed Sigma AoKs.

Our results. Compared to directly embedding complete exponentiation operations into SNARK circuits (as shown in Table 1, potentially requiring hundreds or even thousands of group additions, point doublings, or large integer modular multiplications within the SNARK circuit), our approach dramatically reduces the number of expensive group operations necessary within the SNARK circuit by decomposing the exponentiation and utilizing Sigma AoKs (as presented in Tables 2 and 3). For discrete logarithm hard groups, the required group additions and doublings are reduced by nearly a factor of 10. For RSA groups, the number of large integer modular multiplications is reduced by tens of times.

1.3 Techniques

In public key cryptography, exponentiations within a group are core operations. Among algebraic statements, the most computationally expensive part is proving knowledge of group exponentiations. For example, to prove knowledge of an ECDSA signature verification, which involves verifying the equation $G^s \overset{?}{=} G^{u_1} \cdot P^{u_2}$ where G is the base point, P is the public key, and s, u_1, u_2 are scalars. Apart from this equation, the remaining parts of signature verification can be efficiently verified by SNARKs. While a single group multiplication can be efficiently verified within SNARKs, proving the correctness of a group exponentiation involves several hundred to a few thousand multiplications.

To address this bottleneck, we first introduce EXP gates to demonstrate the group exponentiations in algebraic statements. The EXP gates are parameterized by two key members: an Abelian group $(\mathbb{G}, \circ)$ and a finite integer set I. Specifically, let $g_1 \in \mathbb{G}$, $x \in I$ and $g_2 \in \mathbb{G}$. The inputs to an EXP gate consist of g_1 and x, with the output being g_2, corresponding to the operation $g_1^x = g_2$.

We are particularly concerned with cyclic groups. Many cryptographic systems rely on the hardness of certain computational problems, most notably the Discrete Logarithm Problem . Therefore, we focus on groups $(\mathbb{G}_p, \circ)$ with a prime order p and $I = \mathbb{F}_p$. The group $\mathbb{G}_p$ can represent multiplicative groups of finite fields or elliptic curve groups where the discrete logarithm problem is hard. We describe the following relations $\{\mathcal{R}^i_{\mathsf{dl}}\}_{i\in[4]}$, where $g_1, g_2 \in \mathbb{G}_p$ and $x \in \mathbb{F}_p$.

$$\mathcal{R}^i_{\mathsf{dl}} := \{\mathbf{st}_i : g_1^x = g_2\}$$

$$\mathbf{st}_0 = ((g_1, g_2); x), \mathbf{st}_1 = (g_1; (g_2, x)), \mathbf{st}_2 = (x; (g_1, g_2)), \mathbf{st}_3 = (\varnothing; (g_1, g_2, x))$$

These four relations are sufficient to encompass valuable algebraic statements within the discrete logarithm hard groups, since all other relations either can be directly computed by the verifier or can be transformed into one of these four forms. For example, if both g_1 and x are private inputs while g_2 is public, this configuration is equivalent to proving $g_2^{-x} = g_1$, corresponding to $\mathcal{R}^1_{\mathsf{dl}}$. We construct specially designed Sigma protocols (Sigma AoKs) for $\{\mathcal{R}^i_{\mathsf{dl}}\}_{i\in[4]}$, denoted as $\{\Pi^i_{\mathsf{dl}}\}_{i\in[4]}$. These protocols are then used to build proofs for other cryptographic schemes based on discrete logarithm hard problem, which can be applied to practical industrial applications.

Another common setting is to consider n as an RSA modulus, a given exponent e (with $\gcd(e, \varphi(n)) = 1$) and d such that $de \equiv 1 \bmod \varphi(n)$. We describe a type of EXP gates targeting the RSA setting: $\mathbb{G} = \mathbb{Z}_n^*$, $I = \{e, d\}$. In this case, we consider relations $\{\mathcal{R}^i_{\mathsf{rsa}}\}_{i\in[2]}$, where $g_1, g_2 \in \mathbb{Z}_n^*$ and $x \in \{e, d\}$.

$$\mathcal{R}^i_{\mathsf{rsa}} := \{\mathbf{st}_i : g_1^x = g_2\}$$
$$\mathbf{st}_0 = ((x, g_2); g_1), \mathbf{st}_1 = (x; (g_1, g_2))$$

Similarly, we construct specially designed Sigma protocols (Sigma AoKs) for these relations, denoted as $\{\Pi^i_{\mathsf{rsa}}\}_{i\in[2]}$. These protocols are then used to build proofs of knowledge for RSA encryption/decryption and signature schemes, which can be applied to practical industrial applications. The relationships described here are detailed in Section 4.

Decomposing arithmetic with exponentiation circuit. Our starting point is an arithmetic circuit composed of ADD and MUL gates over a field $\mathbb{F}_q$, along with EXP gates.[4] We leverage the EXP gates to represent the computationally most expensive part of algebraic statements, namely group exponentiation. The ADD and MUL gates are used to handle the rest of computation except for group exponentiations. Proving the satisfiability of such a circuit thus completes the proof of the entire algebraic

[4]When describing an EXP gate within the $\mathbb{F}_q$ arithmetic circuit, the inputs and output elements are represented by a set of wires over $\mathbb{F}_q$.

statements. To prove the satisfiability of this circuit, denoted as $\mathcal{C}_{\mathsf{orig}}$, which comprises these three types of gates, we first need to decompose it using a circuit compiler.

On input $\mathcal{C}_{\mathsf{orig}}$ and a hash function H, the compiler $\mathsf{Compil}(\mathsf{H}, \mathcal{C}_{\mathsf{orig}})$ outputs a standard arithmetic circuit $\mathcal{C}_{\mathsf{std}}$ comprising ADD and MUL gates over $\mathbb{F}_q$, along with a "exponentiation with hash" relation $\mathcal{C}_{\mathsf{eh}}$: $g_1^x = g_2 \wedge \mathsf{H}(\mathbf{w}, r)$, where r is a fixed length random string, $\mathbf{w}$ are the witnesses consisting of one to three variables among x, g_1 and g_2. By utilizing a common SNARK for $\mathcal{C}_{\mathsf{std}}$ and customized Sigma AoKs for $\mathcal{C}_{\mathsf{eh}}$, we can successfully prove the satisfiability of $\mathcal{C}_{\mathsf{orig}}$. An example is shown in Figure 1. In most cases, H refers to a zk-friendly hash function like Poseidon. This provides us with the following properties that we will exploit in the subsequent protocol construction:

- It is very efficient to prove preimages of zk-friendly hash functions in SNARKs.[5] And we can achieve linkage without an additional "glue" proof.
- It offers collision resistance and computational hiding properties, as detailed in Section 2.2.

Proving exponentiation with hash relation. We first address how to prove $g_1^x = g_2$ within SNARKs, where x is private and g_1, g_2 are public within discrete logarithm hard groups. One solution is to directly use SNARKs to prove the correctness of the generation of the response in the Schnorr protocol [36]. However, if either g_1 or g_2 also serves as a witness, it is difficult to generate such proofs. To address this, we propose a "Ping-Pong" alternating approach. From a high-level perspective, the proving process begins with group exponentiations in a SNARK, transitions to our specially designed Sigma protocol, and then returns to a SNARK for group multiplications and several lightweight computations, such as zk-friendly hash. This process transforms the originally complex and massive SNARK circuit encoding entire group exponentiation into a lightweight relation with enhanced parallel structure, significantly reducing the circuit size. We can also understand this method in another perspective, that is our construction utilizes a common SNARK and Sigma AoKs to prove the satisfiability of arithmetic circuits with EXP gates, for every EXP gate we initialize a corresponding Sigma AoK to prove the decomposed relation as mentioned above. And inside the Sigma AoK, we leverage a SNARK to prove a specified relation which is far more lightweight than the group exponentiation operation. So we can just prove this relation combined with the ADD and MUL gates in the original arithmetic circuits by utilizing a single SNARK proof generation. The prover cost in the Sigma AoK is dominated by the SNARK proof generation of that specified relation. Therefore, our approach can be viewed as essentially providing a smaller verification circuit for group exponentiation. These Sigma AoKs serve to "curtail SNARK circuit size". We observe that the inherent three-round interaction of Sigma protocols can, while preserving equivalent security guarantees, fundamentally assist in diminishing the computational overhead associated with SNARK proof generation in the context of group exponentiation. Specifically, proving one scalar multiplication(one group exponentiation operation) on the P-256 curve typically requires up to 256 point

[5] While the SNARKs considered in this work are capable of efficiently proving hash over binary fields, such as SHA256 and Keccak256, proving zk-friendly hash offers even greater efficiency.

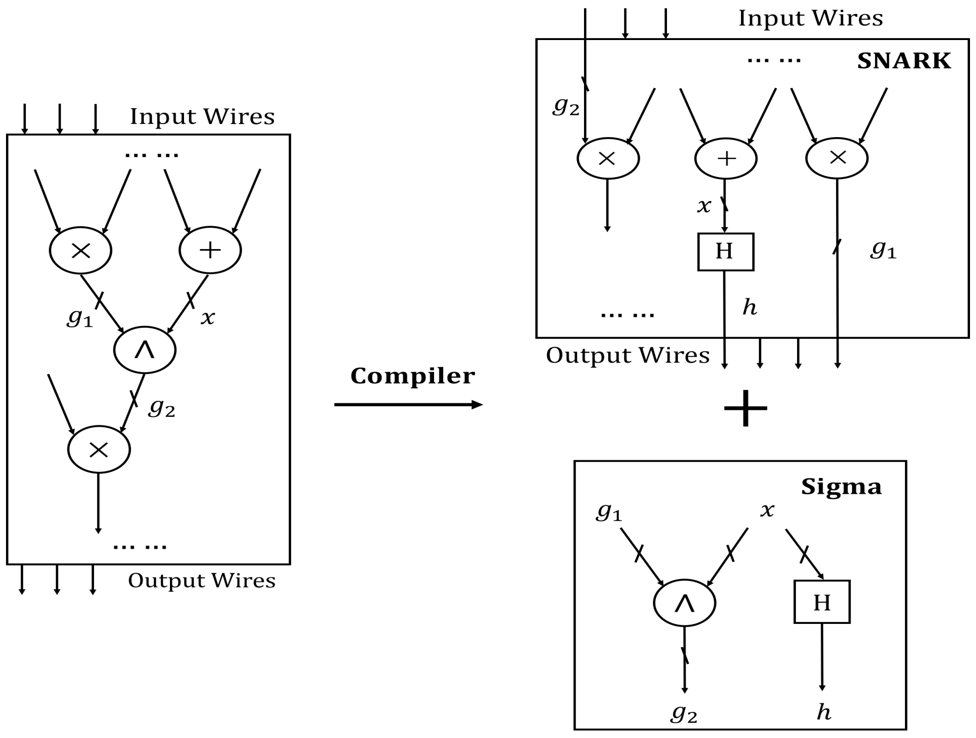

Fig. 1: An EXP gate "$\wedge$" enforcing the relation $g_1 \wedge x = g_2$ ($g_1^x = g_2$), where x is private. H represents the hash function circuit over $\mathbb{F}_q$. Cases where g_1 or g_2 are private variables are handled in a similar manner. In other words, concerning g_1, g_2, and x: variables that are private are hashed; public variables that do not appear in any other ADD or MUL gates are discarded, and public variables that do appear in other ADD or MUL gates are used unmodified

additions and 256 point doublings. After employing the above method, we only need to encode fewer than 50 point additions in the SNARK circuit, where the concrete number is proportional to the security parameter.

For a standard Sigma protocol, it generates the transcript (a, c, z), where a commits the temporary randomness k, c represents the challenge from the verifier, and z typically represents a computation $z(\mathbf{w})$ that serves as the response. The verifier checks whether $\mathsf{V}(\mathbf{s}, a, c, z) = 1$, with $\mathbf{s}$ represents the public part of the statement. An important observation is that if the challenge space is small enough (such as $c \in \{0, 1\}$), proving the relation $\{\mathsf{V}(\mathbf{s}, a, c, z) = 1 \wedge z(\mathbf{w}) \wedge \mathsf{H}(\mathbf{w}, r)\}$ directly using SNARKs has become efficient, fueled by advancements in zk-friendly hash functions and evolving SNARK ecosystems. For protocols like Schnorr [39] and Guillou-Quisquater [40], which inherently involve information about group exponentiation operations, we can make part of $\mathbf{s}$ private and use SNARKs to prove the verification expressions $\mathsf{V}(\mathbf{s}, a, c, z) = 1$ of these Sigma protocols. This allows us to prove relations like $\{\mathcal{R}^i_{\mathsf{dl}}\}_{i\in[4]}$ and $\{\mathcal{R}^i_{\mathsf{rsa}}\}_{i\in[2]}$. By repeatedly proving the same statement for multiple challenges c, we can reduce the knowledge soundness error.

This approach enhances prover efficiency by significantly more than tenfold compared to directly embedding group exponentiation operations into the SNARK circuits. And proving $\{\mathsf{V}(\mathbf{s}, a, c, z) = 1 \wedge z(\mathbf{w}) \wedge \mathsf{H}(\mathbf{w}, r)\}$ achieves linkage without the necessity for an additional "glue" proof due to the hash function. Additionally, compared to simply parallel repetition, we can further optimize efficiency by exploiting the specific structure of this protocol as following:

- *Precompute and Reuse.* We can adjust the challenge space and the number of parallel repetitions by precomputing a small set of group multiplications. Subsequent executions can then reuse these precomputed values, there by minimizing the circuit size that must be verified in SNARKs.
- *Proof aggregation for uniform circuits.* Since the parallel repetitions involve proving different inputs of the same circuit, we can leverage techniques such as folding schemes [41] to accelerate the prover efficiency for uniform circuits.

2 Preliminaries

In this section, we fix our notation and recall the essential definitions to understand this paper. Then, we propose the hiding property that the hash function needs, which plays a crucial role in our constructions.

Notations. We let λ denote the security parameter, $\mathsf{negl}(\lambda)$ denote a negligible function in λ, Π denote a proof system and $\mathcal{R}$ denote the circuit/relation being proven, $[i]$ denote the set of natural numbers $\{0, 1, \ldots, i-1\}$.

In the discrete logarithm setting, we let $\mathbb{G}_p$ denote a discrete logarithm hard cyclic group $(\mathbb{G}_p, \circ)$ with order p and a generator G, G^x denote the exponentiation over $\mathbb{G}_p$ with an integer x. In the RSA setting, for example in RSA signatures, we consistently use the notation "$P^x = Q \mod n$" to represent the signature verification "$\Sigma^e = m \mod n$", where the signature P and message Q are elements of $\mathbb{Z}_n^*$ and the public key x is coprime with $\phi(n)$.

2.1 Zero-Knowledge Proofs

Definition 1 (Sigma protocol [42]) A 3-move public-coin protocol $\langle \mathsf{P}, \mathsf{V}\rangle$ is said to be a Sigma protocol for relation $\mathcal{R}$, if it satisfies the following properties:

- *Completeness:* If P and V follow the protocol on instance x and witness w to P where $(x, w) \in \mathcal{R}$, then V always accepts the transcript.
- *Special soundness:* There exists a probabilistic polynomial time extractor Ext that on input any instance x and two accepting transcripts (a, c, z) and (a, c', z'), with $c \neq c'$, outputs a witness w for x.
- *Special honest-verifier zero-knowledge (SHVZK):* There exists a polynomial time simulator Sim which on input x and random challenge c outputs a transcript (a, c, z) that is indistinguishable from the one generated by an honest interaction between P and V on (common) input x.

Due to the use of zkSNARKs, our constructions of Sigma protocols satisfy only a relaxed version of special soundness property, namely the *computational special soundness* property [43]. This property states that, given a pair of accepting transcripts (a, c, z) and (a, c', z') with $c \neq c'$ produced by an efficient adversary, the extractor succeeds in extracting with a probability negligibly to 1 (here we might let the extractor to get full access to the adversary's state). It is evident that computational special soundness also implies knowledge soundness, and the knowledge error is determined by the size of challenge space. Since the challenge space size in this paper changes from 2 to 2^λ, we directly consider the knowledge property for clarifying its knowledge error and define a new notion called *Sigma Argument of Knowledge*, which replaces the special soundness property of Sigma protocol with knowledge soundness.

Definition 2 (Sigma Argument of Knowledge (Sigma AoK)) A 3-move public-coin protocol is said to be a Sigma Argument of Knowledge for relation $\mathcal{R}$, if it satisfies completeness, SHVZK, and the following property:

- *Knowledge soundness:* An interactive protocol with soundness error δ is knowledge sound, if there exists a polynomial time extractor Ext such that for any instance x and for any polynomial time (potentially malicious) prover P^*, if

  $$\Pr[\langle \mathsf{P}^*, \mathsf{V}\rangle(x) = 1] \geq \delta(|x|) + \epsilon(|x|)$$

  for some non-negligible ϵ, then Ext can extract a witness w such that $(x, w) \in \mathcal{R}$ via accessing P^* in a black-box manner, except with a negligible error.

In the Common Reference String (CRS) model, all parties are assumed to have access to a common string, which is drawn from a carefully defined distribution. In particular, our Sigma AoKs are constructed under the CRS model. Therefore, we consider the adaptive version of security, which allows the selection of the instance x to depend on the CRS.

Definition 3 (zero-knowledge Succinct Non-interactive Argument of Knowledge (zkSNARK)) A zkSNARK for an NP relation $\mathcal{R}$ in the CRS model consists of a triple of polynomial time algorithms $(\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ defined as follows:

- $\mathsf{Setup}(1^\lambda, \mathcal{R})$ takes a security parameter λ and outputs a $\mathsf{crs} \in \{0,1\}^*$.
- $\mathsf{P}(\mathsf{crs}, x; w)$ takes on input the crs, instance x and witness w, and outputs an argument π.
- $\mathsf{V}(\mathsf{crs}, x, \pi)$ takes on input the crs, instance x and proof π, and outputs either 1 accepting the transcript or 0 rejecting it.

The algorithms above satisfy the following properties:

- *Completeness:* If P and V follow the protocol on instance x and witness w to P where $(x, w) \in \mathcal{R}$, then V always accepts the transcript.
- *Knowledge soundness:* A non-interactive protocol is knowledge sound, if for any polynomial time (potentially malicious) prover P^*, there exists a polynomial time extractor Ext, given full access to P^*'s random coins, such that:

$$\Pr\left[\begin{matrix}\mathsf{V}(\mathsf{crs}, x, \pi) = 1 \\ \wedge (x, w) \notin \mathcal{R}\end{matrix}\ \middle|\ \begin{matrix}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}); \\ (x, \pi) \leftarrow \mathsf{P}^*(\mathsf{crs}; r); \\ w \leftarrow \mathsf{Ext}(\mathsf{crs}, x, \pi, r)\end{matrix}\right] \leq \mathsf{negl}(|x|)$$

- *Zero-knowledge:* There exists a polynomial time simulator $(\mathsf{S}_1, \mathsf{S}_2)$ such that, for any polynomial time adversaries $(\mathsf{A}_1, \mathsf{A}_2)$, the absolute value of the following probability is negligible:

$$\Pr\left[\begin{matrix}(x, w) \in \mathcal{R} \wedge \\ \mathsf{A}_2(\pi, \mathsf{st}) = 1\end{matrix}\ \middle|\ \begin{matrix}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}); \\ (x, w, \mathsf{st}) \leftarrow \mathsf{A}_1(\mathsf{crs}); \\ \pi \leftarrow \mathsf{P}(\mathsf{crs}, x, w)\end{matrix}\right]$$
$$- \Pr\left[\begin{matrix}(x, w) \in \mathcal{R} \wedge \\ \mathsf{A}_2(\pi, \mathsf{st}) = 1\end{matrix}\ \middle|\ \begin{matrix}(\mathsf{crs}, \tau) \leftarrow \mathsf{S}_1(1^\lambda, \mathcal{R}); \\ (x, w, \mathsf{st}) \leftarrow \mathsf{A}_1(\mathsf{crs}); \\ \pi \leftarrow \mathsf{S}_2(\mathsf{crs}, \tau, x)\end{matrix}\right]$$

- *Succinctness:* For any x and w, the length of the transcript π is given by $|\pi| = \mathsf{poly}(\lambda) \cdot \mathsf{polylog}(|x| + |w|)$.

2.2 Hash Functions

Definition 4 (Hiding hash function) A hiding hash function is a hash function family $(\mathsf{Gen}, \mathsf{H})$, such that for any $\mathsf{H}_\eta : \{0,1\}^* \times \{0,1\}^\lambda \to \{0,1\}^n$ generated by $\eta \leftarrow \mathsf{Gen}(1^\lambda)$ satisfies the following two properties:

- *Collision resistance:* For any probabilistic polynomial time adversary A,

$$\Pr\left[\begin{matrix}x_0 \neq x_1 \wedge \\ \mathsf{H}_\eta(x_0, r_0) = \mathsf{H}_\eta(x_1, r_1)\end{matrix}\ \middle|\ \begin{matrix}\eta \leftarrow \mathsf{Gen}(1^\lambda); \\ (x_0, r_0, x_1, r_1) \leftarrow \mathsf{A}(\eta)\end{matrix}\right] \leq \mathsf{negl}(\lambda)$$

- *Computational hiding:* For any equal-length $x_0, x_1 \in \{0,1\}^*$, the following two distributions are computationally indistinguishable.

$$\{\mathsf{H}_\eta(x_0, r_0) | r_0 \xleftarrow{\$} \{0,1\}^\lambda\}_\eta \overset{c}{\approx} \{\mathsf{H}_\eta(x_1, r_1) | r_1 \xleftarrow{\$} \{0,1\}^\lambda\}_\eta$$

Instantiation of hiding hash function. We use H to represent the instantiated hiding hash function H_η. The hiding property is implied by the random oracle model (ROM) because $\mathsf{H}(x, r)$ is a uniform distribution. Therefore, if a collision-resistant hash is believed to securely instantiate a random oracle in the real world, then it should also be considered secure to instantiate a hiding hash function.

The construction of our interactive protocol relies on the assumption of the existence of a hiding hash function, an assumption weaker than the ROM. We subsequently apply the Fiat-Shamir transformation in the ROM to obtain a non-interactive version. We emphasize that the hash function employed in our construction is distinct from the Fiat-Shamir hash, which allows us to avoid the complexities of recursive security when proving security in the ROM.

3 Hash-Committed Commit-and-Prove Framework

In this section, we first define an *Arithmetic with Exponentiation Circuit Satisfiability* relation $\mathcal{R}_{\mathsf{orig}}$ (representing the original circuit to be proven) for the arithmetic C-SAT that includes EXP gates. Then, we provide a circuit compiler that decomposes the relation circuit $\mathcal{R}_{\mathsf{orig}}$ into a standard arithmetic C-SAT relation $\mathcal{R}_{\mathsf{std}}$ and some algebraic exponentiation relations $\mathcal{R}_{\mathsf{eh}}$. Finally, we give a general description of our protocol framework.

3.1 Arithmetic with Exponentiation Circuit Satisfiability

We extend the arithmetic C-SAT problem over a field $\mathbb{F}_q$ by incorporating exponentiation operations over a cyclic group $\mathbb{G}_p$, without representing them as arithmetic over $\mathbb{F}_q$. This circuit features three types of gates: ADD and MUL over $\mathbb{F}_q$, and EXP gates over $\mathbb{G}_p$ defined as follows:

Definition 5 (Exponentiation gate) Let $(\mathbb{G}_p, \circ)$ be a cyclic group, and $I \subseteq \mathbb{Z}$ be a finite set.[6] An EXP gate over $\mathbb{G}_p$ is parameterized by two inputs: a group element $P \in \mathbb{G}_p$ and an exponent $x \in I$, producing an output $Q \in \mathbb{G}_p$ such that $P^x = Q$.

EXP gates model exponentiation operations in various cryptographic protocols and are a core component of circuits involving group operations. Since cryptographic algorithms are typically defined over a single cyclic group, we provide an arithmetic circuit that includes EXP gates over a single cyclic group. This structure can be naturally extended to circuits that include operations across multiple cyclic groups.

Definition 6 (Arithmetic with exponentiation circuit satisfiability) Let $\mathbb{F}_q$ be a field and $\mathbb{G}_p$ be a cyclic group. An arithmetic with exponentiation circuit $\mathcal{R}_{\mathsf{orig}}$ operates on wires $\omega = (\mathbf{s}; \mathbf{w})$, satisfying:

$$\mathcal{R}_{\mathsf{orig}} := \left\{ (\mathbf{s}; \mathbf{w}) : \begin{array}{c} \forall i \in I_{\mathsf{add}}, \quad \omega_{\mathrm{out}(i)} = \omega_{\mathrm{in}_1(i)} + \omega_{\mathrm{in}_2(i)} \pmod{q} \land \\ \forall j \in I_{\mathsf{mul}}, \quad \omega_{\mathrm{out}(j)} = \omega_{\mathrm{in}_1(j)} \cdot \omega_{\mathrm{in}_2(j)} \pmod{q} \land \\ \forall k \in I_{\mathsf{exp}}, \quad P_k^{x_k} = Q_k \quad \text{where:} \\ Q_k := \mathcal{L}(\hat{Q}_k),\ P_k := \mathcal{L}(\hat{P}_k),\ x_k := \mathcal{L}(\hat{x}_k) \end{array} \right\}$$

Where:

- $\omega := (\mathbf{s}; \mathbf{w})$ represents all wires, with $\mathbf{s}$ for public wires and $\mathbf{w}$ for private wires.
- $I_{\mathsf{add}}, I_{\mathsf{mul}}, I_{\mathsf{exp}}$ are index sets for ADD, MUL, and EXP gates.
- $\mathrm{out}(i)$, $\mathrm{in}_1(i)$, $\mathrm{in}_2(i)$ denote functions that map the gate indices i to the corresponding output and input wire indices.
- $\hat{Q}_k, \hat{P}_k$ and $\hat{x}_k$ are vectors over $\mathbb{F}_q$ that uniquely represent the original group variables Q_k, P_k and x_k, respectively.[7]
- $\mathcal{L}$ is the linear mapping that reconstructs original variables from their representation of $\mathbb{F}_q$ elements.

[6] The exponent variable in operations over cyclic group $\mathbb{G}_p$ is typically an element of $\mathbb{F}_p$. However, exceptions exist, such as RSA accumulator schemes, where the exponent belongs to a specific set of integers.

[7] For example, to represent a Secp256k1 element, which is 256-bits, using 4 variables of 64-bits each in the native field $\mathbb{F}_q$ where the bit length of q is 64.

For $\hat{Q}_k$ in the gate EXP_k, we restrict the wires in $\hat{Q}_k$ to be entirely composed of either the public wires in $\mathbf{s}$ or the private wires in $\mathbf{w}$. This distinction indicates whether the group element Q_k is public or private in the statement. The same applies to $\hat{P}_k$ and $\hat{x}_k$.

The standard arithmetic C-SAT relation and the relation $\mathcal{R}_{\mathsf{orig}}$ can be mutually reduced. Specifically, $\mathcal{R}_{\mathsf{orig}}$ is equivalent to the standard arithmetic C-SAT when there are no EXP gates, establishing it as an NP-complete relation. We can encompass nearly all mainstream constraint systems into the relation $\mathcal{R}_{\mathsf{orig}}$, including R1CS, PLONK, AIR, QAP, and layered arithmetic circuits, by incorporating these constraint systems along with EXP gates.

3.2 Decompose Circuit

Given an arithmetic with exponentiation circuit $\mathcal{R}_{\mathsf{orig}}(\mathbf{s};\mathbf{w})$[8], we demonstrate how to apply a circuit compiler to decompose it into two relations. If both relations are satisfied, this establishes that $\mathcal{R}_{\mathsf{orig}}(\mathbf{s};\mathbf{w})$ is satisfied. From a high level, this compiler processes x, g_1, g_2 by applying hashing to those belonging to the witness set. For public values among x, g_1, g_2, they are retained if used as inputs or outputs in other addition or multiplication gates within the circuit $\mathcal{C}_{\mathsf{orig}}$; otherwise, public values that do not appear in such gates are discarded.

Circuit Compiler: Denote $I_{\mathsf{am}} = I_{\mathsf{add}} \cup I_{\mathsf{mul}}$. For the input circuit $\mathcal{R}_{\mathsf{orig}}(\mathbf{s};\mathbf{w})$ over arithmetic field $\mathbb{F}_q$ and cyclic group $\mathbb{G}_p$, a security parameter λ, and an initialized hash function H, the compiler $\mathsf{Compil}(\mathsf{H}, \mathcal{R}_{\mathsf{orig}}(\mathbf{s};\mathbf{w}))$ operates as follows:

1. Leave the ADD gates and MUL gates over $\mathbb{F}_q$ unchanged.
2. For every $k \in I_{\mathsf{exp}}$, define two empty vectors $\widetilde{\mathbf{s}}_k$ and $\widetilde{\mathbf{w}}_k$. Then proceed to modify ω and append variables to $\widetilde{\mathbf{s}}_k$ and $\widetilde{\mathbf{w}}_k$ in the following way:
   a. If $\hat{P}_k \subseteq \mathbf{w}$, then append $\widetilde{\mathbf{w}}_k \leftarrow \widetilde{\mathbf{w}}_k \cup \hat{P}_k$. Otherwise, append $\widetilde{\mathbf{s}}_k \leftarrow \widetilde{\mathbf{s}}_k \cup \hat{P}_k$, and if $\hat{P}_k \nsubseteq \{\omega_{\mathsf{out}(I_{\mathsf{am}})} \cup \omega_{\mathsf{in}_1(I_{\mathsf{am}})} \cup \omega_{\mathsf{in}_2(I_{\mathsf{am}})}\}$ then remove $\omega \leftarrow \omega \setminus \hat{P}_k$.[9]
   b. If $\hat{Q}_k \subseteq \mathbf{w}$, then append $\widetilde{\mathbf{w}}_k \leftarrow \widetilde{\mathbf{w}}_k \cup \hat{Q}_k$. Otherwise, append $\widetilde{\mathbf{s}}_k \leftarrow \widetilde{\mathbf{s}}_k \cup \hat{Q}_k$, and if $\hat{Q}_k \nsubseteq \{\omega_{\mathsf{out}(I_{\mathsf{am}})} \cup \omega_{\mathsf{in}_1(I_{\mathsf{am}})} \cup \omega_{\mathsf{in}_2(I_{\mathsf{am}})}\}$ then remove $\omega \leftarrow \omega \setminus \hat{Q}_k$.
   c. If $\hat{x}_k \subseteq \mathbf{w}$, then append $\widetilde{\mathbf{w}}_k \leftarrow \widetilde{\mathbf{w}}_k \cup \hat{x}_k$. Otherwise, append $\widetilde{\mathbf{s}}_k \leftarrow \widetilde{\mathbf{s}}_k \cup \hat{x}_k$, and if $\hat{x}_k \nsubseteq \{\omega_{\mathsf{out}(I_{\mathsf{am}})} \cup \omega_{\mathsf{in}_1(I_{\mathsf{am}})} \cup \omega_{\mathsf{in}_2(I_{\mathsf{am}})}\}$ then remove $\omega \leftarrow \omega \setminus \hat{x}_k$.
3. For every $k \in I_{\mathsf{exp}}$, compile the hash circuit $\mathsf{H}(\widetilde{\mathbf{w}}_k, r_k) = h_k$, where r_k is a random variable in $\{0,1\}^\lambda$, then obtain an arithmetic circuit $\mathsf{H}^k_{\mathsf{std}}$ over $\mathbb{F}_q$ with wires $(h_k; \mathbf{w}^k_{\mathsf{H}})$. After that, append the public wires $\widetilde{\mathbf{s}}_k \leftarrow \widetilde{\mathbf{s}}_k \cup \{h_k\}$, $\mathbf{s} \leftarrow \mathbf{s} \cup \{h_k\}$ and private wires $\mathbf{w} \leftarrow \mathbf{w} \cup \mathbf{w}^k_{\mathsf{H}}$.[10]

[8] We denote the circuit as $\mathcal{R}(\mathbf{s};\mathbf{w})$ where the wires $(\mathbf{s};\mathbf{w})$ are unassigned variables.

[9] If $\hat{P}_k$ does not contain wires for arithmetic gates, then remove it from the new circuit. We assume that the wires constituting $\hat{P}_k$ are either all connected to some arithmetic gates or none of them are, which is always the case in cryptographic algorithms.

[10] Typically, for the compiled circuit $\mathsf{H}^k_{\mathsf{std}}$ over $\mathbb{F}_q$, the private wires include not only the original input variables $(\widetilde{\mathbf{w}}_k, r_k)$ but also some intermediate wires. Additionally, the union operation ensures the witness for original circuit does not appear twice.

4. Output an arithmetic with hash circuit $\mathcal{R}_{\mathsf{std}}(\overline{\mathbf{s}};\overline{\mathbf{w}})$[11], and k exponentiation with hash relations $\mathcal{R}^k_{\mathsf{eh}}(\widetilde{\mathbf{s}}_k;(\widetilde{\mathbf{w}}_k, r_k))$, which are defined as follows:

- *Arithmetic with hash relation:* An arithmetic with hash circuit $\mathcal{R}_{\mathsf{std}}$ over $\mathbb{F}_q$ on wires $(\overline{\mathbf{s}};\overline{\mathbf{w}})$ is a standard arithmetic circuit that leaves the ADD and MUL gates in $\mathcal{R}_{\mathsf{orig}}$ and replaces the EXP gates with corresponding hash relations as follows:

$$\mathcal{R}_{\mathsf{std}} := \left\{ (\overline{\mathbf{s}};\overline{\mathbf{w}}) : \begin{array}{rl} \forall i \in I_{\mathsf{add}}, & \omega_{\mathsf{out}(i)} = \omega_{\mathsf{in}_1(i)} + \omega_{\mathsf{in}_2(i)} \pmod q \wedge \\ \forall j \in I_{\mathsf{mul}}, & \omega_{\mathsf{out}(j)} = \omega_{\mathsf{in}_1(j)} \cdot \omega_{\mathsf{in}_2(j)} \pmod q \wedge \\ \forall k \in I_{\mathsf{exp}}, & \mathsf{H}^k_{\mathsf{std}}(h_k; \mathbf{w}^k_{\mathsf{H}}) = 1 \\ \text{where:} & h_k \in \overline{\mathbf{s}}, \mathbf{w}^k_{\mathsf{H}} \subseteq \overline{\mathbf{w}} \end{array} \right\}$$

- *Exponentiation with hash relation:* For all $k \in I_{\mathsf{exp}}$, an exponentiation with hash relation $\mathcal{R}^k_{\mathsf{eh}}$ on public wires $\widetilde{\mathbf{s}}_k$ and private wires $(\widetilde{\mathbf{w}}_k, r_k)$ is as follows:

$$\mathcal{R}^k_{\mathsf{eh}} := \left\{ (\widetilde{\mathbf{s}}_k;(\widetilde{\mathbf{w}}_k, r_k)) : \begin{array}{l} \mathsf{H}(\widetilde{\mathbf{w}}_k, r_k) = h_k \wedge P_k^{x_k} = Q_k \\ \text{where: } h_k \in \widetilde{\mathbf{s}}_k, \\ P_k := \mathcal{L}(\hat{P}_k), Q_k := \mathcal{L}(\hat{Q}_k), x_k := \mathcal{L}(\hat{x}_k), \\ \text{and } \hat{P}_k, \hat{Q}_k, \hat{x}_k \text{ are in either } \widetilde{\mathbf{s}}_k \text{ or } \widetilde{\mathbf{w}}_k \end{array} \right\}$$

The input circuit $\mathcal{R}_{\mathsf{orig}}$ and output circuits $\mathcal{R}_{\mathsf{std}}$ and $\{\mathcal{R}^k_{\mathsf{eh}}\}_{I_{\mathsf{exp}}}$ naturally imply a mapping of wires following executions of the Compil:

$$\mathsf{wire} : \big(\mathbf{s}, \{h_k\}_{I_{\mathsf{exp}}}\big) \mapsto \big(\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}_{I_{\mathsf{exp}}}\big)$$

where $\overline{\mathbf{s}}$ and each $\widetilde{\mathbf{s}}_k$ contain an identical variable h_k. Therefore, for a given public wire assignments $\mathbf{s}$ in $\mathcal{R}_{\mathsf{orig}}$ and each hash value h_k, the public wire assignments of $\mathcal{R}_{\mathsf{std}}$ and $\{\mathcal{R}^k_{\mathsf{eh}}\}_{I_{\mathsf{exp}}}$ can be naturally computed by the mapping $\mathsf{wire}(\cdot)$. In the following text, we rewrite $\{a_k\}_{I_{\mathsf{exp}}}$ to $\{a_k\}$.

Lemma 1 *Assume* H *is a collision-resistant hash function. The compiler* Compil *given an input relation* $\mathcal{R}_{\mathsf{orig}}$ *outputs separate relations,* $\mathcal{R}_{\mathsf{std}}$ *and* $\{\mathcal{R}^k_{\mathsf{eh}}\}$*. Then there exist two efficient deterministic algorithms* S *and* E *such that, for any polynomial time algorithm* P *which outputs the public values* $\mathbf{s}$ *and* $\{h_k\}$ *and let* $(\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}) = \mathsf{wire}(\mathbf{s}, \{h_k\})$*, it satisfies:*

1. *If* P *outputs private values* $(\mathbf{w}, \{r_k\})$ *satisfying* $\mathcal{R}_{\mathsf{orig}}(\mathbf{s};\mathbf{w}) = 1$*, then the algorithm* S*, on input* $(\mathbf{w}, \{r_k\})$*, outputs* $\overline{\mathbf{w}}$ *and* $\{\widetilde{\mathbf{w}}_k, r_k\}$ *satisfying* $\mathcal{R}_{\mathsf{std}}$ *and* $\{\mathcal{R}^k_{\mathsf{eh}}\}$*, respectively.*
2. *If* P *outputs private values* $\overline{\mathbf{w}}$ *and* $\{\widetilde{\mathbf{w}}_k, r_k\}$ *satisfying* $\mathcal{R}_{\mathsf{std}}$ *and* $\{\mathcal{R}^k_{\mathsf{eh}}\}$*, then the algorithm* E*, on input* $\overline{\mathbf{w}}$ *and* $\{\widetilde{\mathbf{w}}_k, r_k\}$*, outputs* $\mathbf{w}$ *satisfying* $\mathcal{R}_{\mathsf{orig}}$*.*

Proof sketch. On one hand, for acceptable values $(\mathbf{s};\mathbf{w})$ for $\mathcal{R}_{\mathsf{orig}}$, algorithm S follows the executions of the compiler to obtain acceptable values for $\mathcal{R}_{\mathsf{std}}$ and $\{\mathcal{R}^k_{\mathsf{eh}}\}$. On

[11] We rename the output wires as $(\overline{\mathbf{s}};\overline{\mathbf{w}})$ to distinguish it from the original wires $(\mathbf{s};\mathbf{w})$. For simplicity, we write the circuit $\mathcal{R}(\mathbf{s};\mathbf{w})$ as $\mathcal{R}$ in the context.

the other hand, the acceptable values $(\overline{\mathbf{s}}; \overline{\mathbf{w}})$ and $(\widetilde{\mathbf{s}}_k; (\widetilde{\mathbf{w}}_k, r_k))$ with the same h_k in both $\overline{\mathbf{s}}$ and $\widetilde{\mathbf{s}}_k$ implies $\mathsf{H}^k_{\mathsf{std}}(h_k, \overline{\mathbf{w}}) = 1$ and $\mathsf{H}(\widetilde{\mathbf{w}}_k, r_k) = h_k$. That implies the same values $\widetilde{\mathbf{w}}_k, r_k \in \overline{\mathbf{w}}$. Otherwise, the polynomial time algorithm P breaks the collision resistance of H. Then, algorithm E follows the executions of the compiler to obtain acceptable values $\mathbf{w}$ for $\mathcal{R}_{\mathsf{orig}}$. □

3.3 NIZKs for Arithmetic with Exponentiation Circuit Satisfiability

During the setup of the HCP protocol, the circuit compiler decomposes arithmetic with exponentiation circuit $\mathcal{R}_{\mathsf{orig}}$ into two parts: $\mathcal{R}_{\mathsf{std}}$ and $\{\mathcal{R}^k_{\mathsf{eh}}\}$. The execution of the protocol consists of two parallel phases: a SNARK for $\mathcal{R}_{\mathsf{std}}$ and a *Sigma Argument of Knowledge (Sigma AoK)* for each $\mathcal{R}^k_{\mathsf{eh}}$. We propose a series of generalized Sigma AoKs defined in Section 2.1, consisting of a Sigma protocol accompanied by an internal SNARK proof. The construction of Sigma AoKs is detailed in Section 4. For the exponentiation with hash relation, let Π^k_{aok} denote the corresponding Sigma AoK for $\mathcal{R}^k_{\mathsf{eh}}$.

Protocol Π_{hcp} *(Hash-committed Commit-and-Prove protocol).* Consider a hash function H and an arbitrary zkSNARK scheme Π for C-SAT. For a circuit $\mathcal{R}_{\mathsf{orig}}$ with gates $\{\mathsf{EXP}_k\}$, let Π^k_{aok} be the customized Sigma AoK for the EXP_k. A HCP protocol is a triple of polynomial time algorithms $\Pi_{\mathsf{hcp}} := (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ behave as shown in Figure 2. After the protocol setup, the execution is described in four phases. In the input phase, P and V transform the input assignment into the forms of SNARK and Sigma statements. The subsequent two phases execute the SNARK and Sigma AoK protocols.

Theorem 2 *Assume* H *is a hiding hash function, then* Π_{hcp} *in Figure 2 is a 3-move public-coin SHVZK argument of knowledge (Sigma AoK) for NP relation* $\mathcal{R}_{\mathsf{orig}}$.

Proof Completeness is obvious.

Knowledge soundness. First, we analyze the soundness of Π_{hcp}, which is derived from the SNARK Π (with negligible soundness error δ_0) and Sigma AoKs Π^k_{aok} (with soundness error δ_k). Specifically, if a polynomial time prover P^* output a false instance $(\mathbf{s}^*, \{h_k\})$ with an tolerable transcript $(\pi_{\mathsf{std}}, \{\pi^k_{\mathsf{eh}}\})$, then at least one of the instance $(\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\})$ is a false instance, due to Lemma 1. Consequently, P^* breaks the soundness of either Π or at least one of Π^k_{aok}. Therefore, we rewrite the $I_{\mathsf{exp}} := \{1, \ldots, \hat{k}\}$, and the soundness error is given by $\delta_{\mathsf{hcp}} = \max\{\delta_0, \ldots, \delta_{\hat{k}}\}$.

Then, for any polynomial time prover P^* and any $\{\mathsf{crs}_k\} := (\mathsf{crs}_0, \ldots, \mathsf{crs}_{\hat{k}})$, the extractor Ext behaves as follows:

1. Call P^* on $\{\mathsf{crs}_k\}$ to obtain $(\mathbf{s}, \{h_k\})$, then compute $(\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}) = \mathsf{wire}(\mathbf{s}, \{h_k\})$.
2. Call the SNARK extractor $\Pi.\mathsf{Ext}^{\mathsf{P}^*}$ for crs_0 and instance $\overline{\mathbf{s}}$ to obtain the witness $\overline{\mathbf{w}}$.
3. For each $k \in I_{\mathsf{exp}}$, call the Sigma AoK extractor $\Pi^k_{\mathsf{aok}}.\mathsf{Ext}^{\mathsf{P}^*}$ for instance $\widetilde{\mathbf{s}}_k$ until a witness $(\widetilde{\mathbf{w}}_k, r_k)$ is obtained.

Hash-Committed Commit-and-Prove Protocol Π_{hcp}

Setup$(1^\lambda, \mathcal{R}_{\mathsf{orig}})$**.**

1. Sample a random hash function $\mathsf{H} \leftarrow \mathsf{Gen}(1^\lambda)$.
2. $\mathcal{R}_{\mathsf{std}}, \{\mathcal{R}^k_{\mathsf{eh}}\} \leftarrow \mathsf{Compil}(\mathsf{H}, \mathcal{R}_{\mathsf{orig}})$.
3. $\mathsf{crs}_0 \leftarrow \Pi.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathsf{std}})$.
4. $\mathsf{crs}_k \leftarrow \Pi^k_{\mathsf{aok}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^k_{\mathsf{eh}})$ for each $k \in I_{\mathsf{exp}}$.

• **Input**$(\mathsf{crs}_0, \{\mathsf{crs}_k\}, \mathbf{s}; \mathbf{w})$**.**

1. P and V are given the public input $\mathbf{s}$, and P is given the private input $\mathbf{w}$.
2. For each $k \in I_{\mathsf{exp}}$, P chooses random $r_k \in \{0,1\}^\lambda$. Then P calls the efficient algorithm S (detailed in Lemma 1) to obtain $\overline{\mathbf{w}}$ and $\{\widetilde{\mathbf{w}}_k\}$.
3. P computes $h_k = \mathsf{H}(\widetilde{\mathbf{w}}_k, r_k)$, and sends $\{h_k\}$ to V.
4. V computes $(\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}) = \mathsf{wire}(\mathbf{s}, \{h_k\})$.

• **SNARK phase.** P and V follow the work of the SNARK $\Pi.\langle\mathsf{P}(\overline{\mathbf{w}}), \mathsf{V}\rangle(\mathsf{crs}_0, \overline{\mathbf{s}})$ for relation $\mathcal{R}_{\mathsf{std}}$.

• **Sigma phase.** For each $k \in I_{\mathsf{exp}}$, P and V follow the work of Sigma AoK $\Pi^k_{\mathsf{aok}}.\langle\mathsf{P}(\widetilde{\mathbf{w}}_k, r_k), \mathsf{V}\rangle(\mathsf{crs}_k, \widetilde{\mathbf{s}}_k)$ for relation $\mathcal{R}^k_{\mathsf{eh}}$, in parallel.

• **Output.** Accept if and only if the verifications in both SNARK phase and Sigma phase are accepted. Otherwise, reject.

Fig. 2: Π_{hcp} for $\mathcal{R}_{\mathsf{orig}}(\mathbf{s}; \mathbf{w}) = 1$

4. Call the efficient algorithm E detailed in Lemma 1 to get $\mathbf{w}$ and output it.

Assume that P^* can output an acceptable transcript for instance $\mathbf{s}$ with probability $p \geq \delta_{\mathsf{hcp}} + \epsilon$ for a non-negligible ϵ, which includes the acceptable transcript π_{std} and $\{\pi^k_{\mathsf{eh}}\}$. According to the knowledge soundness of SNARK Π, the extractor successfully outputs a witness with a probability at least $1 - \mathsf{negl}_0$. For each $k \in I_{\mathsf{exp}}$, according to the knowledge soundness of Π^k_{aok}, the extractor call $\Pi^k_{\mathsf{aok}}.\mathsf{Ext}^{\mathsf{P}^*}$ in expected polynomial time and can successfully outputs the witness with probability $1-\mathsf{negl}_k$, since $p \geq \delta_k + \epsilon$. Following the Lemma 1, the efficient algorithm E can output a correct witness except for a negligible probability of finding a collision of the hash function. Therefore, the Ext successfully extracts a witness, except with a negligible error.

Computational SHVZK. There exists a polynomial time simulator Sim that given a random challenge $\mathbf{c} = \{c_k\}$, for any polynomial time algorithm A (which is only allowed to adaptively choose the instance), outputs a transcript $(\pi^*_{\mathsf{std}}, \{\pi^{k*}_{\mathsf{eh}}\})$, which is indistinguishable from the honest one $(\pi_{\mathsf{std}}, \{\pi^k_{\mathsf{eh}}\})$. The Sim behaves as follows:

1. Call the SNARK simulator $\Pi.\mathsf{Sim}$ and Sigma AoK simulator $\Pi^k_{\mathsf{aok}}.\mathsf{Sim}$ to obtain (crs_0, τ_0) and $\{\mathsf{crs}_k, \tau_k\}$. Then call A on $\{\mathsf{crs}_k\}$ to obtain the instance $\mathbf{s}$.

2. For each $k \in I_{\mathsf{exp}}$, choose a random $r_k \in \{0,1\}^\lambda$ and random elements $w^*(P_k, Q_k, x_k)$ acting witnesses in gate EXP_k, and then compute $h_k^* = \mathsf{H}(w^*(\hat{P}_k, \hat{Q}_k, \hat{x}_k), r_k)$.[12]
3. Compute $(\overline{\mathbf{s}}^*, \{\widetilde{\mathbf{s}}_k^*\}) = \mathsf{wire}(\mathbf{s}, \{h_k^*\})$, where $\overline{\mathbf{s}}^*$ and $\widetilde{\mathbf{s}}_k^*$ include the same h_k^*.
4. Call the SNARK simulator $\Pi.\mathsf{Sim}$ on $(\mathsf{crs}_0, \overline{\mathbf{s}}^*, \tau_0)$ to obtain the output π_{std}^*.
5. For each $k \in I_{\mathsf{exp}}$, call the SHVZK simulator $\Pi_{\mathsf{aok}}^k.\mathsf{Sim}$ on $(\mathsf{crs}_k, \widetilde{\mathbf{s}}_k^*, \tau_k)$ with the honest-verifier challenge c_k to obtain the output π_{eh}^{k*}.
6. Output $(\overline{\mathbf{s}}^*, \{\widetilde{\mathbf{s}}_k^*\}, \pi_{\mathsf{std}}^*, \{\pi_{\mathsf{eh}}^{k*}\})$.

For the transcript $(\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}, \pi_{\mathsf{std}}, \{\pi_{\mathsf{eh}}^k\})$ generated by the honest interaction between P and V on public input $\mathbf{s}$, we can show that Sim's output is computationally indistinguishable from the honest transcript via a hybrid argument:

$$\mathcal{H}_0 : \left\{ \begin{pmatrix} \{\mathsf{crs}_k\}, \\ \mathbf{s}, \overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}, \\ \pi_{\mathsf{std}}, \{\pi_{\mathsf{eh}}^k\} \end{pmatrix} \middle| \begin{array}{c} \{\mathsf{crs}_k\} \leftarrow \mathsf{Setup}(1^\lambda); (\mathbf{s}, \mathbf{w}) \leftarrow \mathsf{A}\{\mathsf{crs}_k\}; \\ (\{h_k\}, \overline{\mathbf{w}}, \{\widetilde{\mathbf{w}}_k, r_k\}) \leftarrow \mathsf{P}(\{\mathsf{crs}_k\}, \mathbf{s}, \mathbf{w}); \\ (\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}) = \mathsf{wire}(\mathbf{s}, \{h_k\}); \pi_{\mathsf{std}} \leftarrow \Pi.\mathsf{P}(\mathsf{crs}_0, \overline{\mathbf{s}}; \overline{\mathbf{w}}); \\ \{\pi_{\mathsf{eh}}^k \leftarrow \Pi_{\mathsf{aok}}^k.\langle \mathsf{P}(\widetilde{\mathbf{w}}_k, r_k), \mathsf{V}\rangle(\mathsf{crs}_k, \widetilde{\mathbf{s}}_k)\} \end{array} \right\}$$

$$\mathcal{H}_1 : \left\{ \begin{pmatrix} \{\mathsf{crs}_k\}, \\ \mathbf{s}, \overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}, \\ \pi_{\mathsf{std}}^*, \{\pi_{\mathsf{eh}}^k\} \end{pmatrix} \middle| \begin{array}{c} (\mathsf{crs}_0, \tau_0) \leftarrow \Pi.\mathsf{Sim}(1^\lambda); \{\mathsf{crs}_k \leftarrow \Pi_{\mathsf{aok}}^k.\mathsf{Setup}(1^\lambda)\}; \\ (\mathbf{s}, \mathbf{w}) \leftarrow \mathsf{A}(\{\mathsf{crs}_k\}); \{h_k, \widetilde{\mathbf{w}}_k, r_k\} \leftarrow \mathsf{P}(\{\mathsf{crs}_k\}, \mathbf{s}, \mathbf{w}); \\ (\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}) = \mathsf{wire}(\mathbf{s}, \{h_k\}); \pi_{\mathsf{std}}^* \leftarrow \Pi.\mathsf{Sim}(\mathsf{crs}_0, \overline{\mathbf{s}}, \tau_0); \\ \{\pi_{\mathsf{eh}}^k \leftarrow \Pi_{\mathsf{aok}}^k.\langle \mathsf{P}(\widetilde{\mathbf{w}}_k, r_k), \mathsf{V}\rangle(\mathsf{crs}_k, \widetilde{\mathbf{s}}_k)\} \end{array} \right\}$$

$\mathcal{H}_0$ represents the honest transcript, while $\mathcal{H}_1$ only substitutes the SNARK prover's proof with π_{std}^*, which is generated by the simulator. Therefore, the indistinguishability is derived from the zero-knowledge property of Π.

$$\mathcal{H}_{\hat{k}+1} : \left\{ \begin{pmatrix} \{\mathsf{crs}_k\}, \\ \mathbf{s}, \overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}, \\ \pi_{\mathsf{std}}^*, \{\pi_{\mathsf{eh}}^{k*}\} \end{pmatrix} \middle| \begin{array}{c} (\mathsf{crs}_0, \tau_0) \leftarrow \Pi.\mathsf{Sim}(1^\lambda); \{(\mathsf{crs}_k, \tau_k) \leftarrow \Pi_{\mathsf{aok}}^k.\mathsf{Sim}(1^\lambda)\}; \\ (\mathbf{s}, \mathbf{w}) \leftarrow \mathsf{A}(\{\mathsf{crs}_k\}); \{h_k\} \leftarrow \mathsf{P}(\{\mathsf{crs}_k\}, \mathbf{s}, \mathbf{w}); \\ (\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}) = \mathsf{wire}(\mathbf{s}, \{h_k\}); \pi_{\mathsf{std}}^* \leftarrow \Pi.\mathsf{Sim}(\mathsf{crs}_0, \overline{\mathbf{s}}, \tau_0); \\ \{\pi_{\mathsf{eh}}^{k*} \leftarrow \Pi_{\mathsf{aok}}^k.\mathsf{Sim}(\mathsf{crs}_k, \widetilde{\mathbf{s}}_k, c_k, \tau_k)\} \end{array} \right\}$$

$\mathcal{H}_1$ to $\mathcal{H}_{\hat{k}+1}$ sequentially replaces Π_{aok}^k prover's proof π_{eh}^k with the transcript π_{eh}^{k*} generated by the simulator with the challenge c_k. Therefore, the indistinguishability is derived from the SHVZK property of Π_{aok}^k.

$$\mathcal{H}_{\hat{k}+2} : \left\{ \begin{pmatrix} \{\mathsf{crs}_k\}, \\ \mathbf{s}, \overline{\mathbf{s}}^*, \{\widetilde{\mathbf{s}}_k^*\}, \\ \pi_{\mathsf{std}}^*, \{\pi_{\mathsf{eh}}^{k*}\} \end{pmatrix} \middle| \begin{array}{c} (\mathsf{crs}_0, \tau_0) \leftarrow \Pi.\mathsf{Sim}(1^\lambda); \{(\mathsf{crs}_k, \tau_k) \leftarrow \Pi_{\mathsf{aok}}^k.\mathsf{Sim}(1^\lambda)\}; \\ \mathbf{s} \leftarrow \mathsf{A}(\{\mathsf{crs}_k\}); w^*(P_k, Q_k, x_k) \xleftarrow{\$} (\mathbb{G}_p, I); \\ r_k \xleftarrow{\$} \{0,1\}^\lambda; \{h_k^* = \mathsf{H}(w^*(\hat{P}_k, \hat{Q}_k, \hat{x}_k), r_k)\}; \\ (\overline{\mathbf{s}}^*, \{\widetilde{\mathbf{s}}_k^*\}) = \mathsf{wire}(\mathbf{s}, \{h_k^*\}); \pi_{\mathsf{std}}^* \leftarrow \Pi.\mathsf{Sim}(\mathsf{crs}_0, \overline{\mathbf{s}}^*, \tau_0); \\ \{\pi_{\mathsf{eh}}^{k*} \leftarrow \Pi_{\mathsf{aok}}^k.\mathsf{Sim}(\mathsf{crs}_k, \widetilde{\mathbf{s}}_k^*, c_k, \tau_k)\} \end{array} \right\}$$

$\mathcal{H}_{\hat{k}+2}$ represents the transcript generated by simulator Sim. The change from $(\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\})$ to $(\overline{\mathbf{s}}^*, \{\widetilde{\mathbf{s}}_k^*\})$ means that the public value $\{h_k = \mathsf{H}(w(\hat{P}_k, \hat{Q}_k, \hat{x}_k), r_k)\}$ is replaced by

[12]Denote $w(P_k, Q_k, x_k)$ be the set of witnesses for the gate EXP_k (which may contain one to three elements from P_k, Q_k, x_k), and $w(\hat{P}_k, \hat{Q}_k, \hat{x}_k) := \mathcal{L}^{-1}(w(P_k, Q_k, x_k))$. And asterisk (*) indicates that the value is randomly chosen by the simulator.

$\{h_k^* = \mathsf{H}(w^*(\hat{P}_k, \hat{Q}_k, \hat{x}_k), r_k)\}$, which makes $(\bar{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\})$ and $(\bar{\mathbf{s}}^*, \{\widetilde{\mathbf{s}}_k^*\})$ computationally indistinguishable due to the hiding property of H. Additionally, all the polynomial time simulators output indistinguishable distributions since their input distributions are computational indistinguishable. □

Non-interactive via Fiat-Shamir heuristic [44]**.** For a 3-move public-coin interactive protocol, the Fiat-Shamir transformation can convert the SHVZK property into zero-knowledge while preserving the knowledge soundness in the random oracle model (ROM). We highlight the critical distinction that the construction of our interactive protocol assumes the existence of a hiding hash function, separate and independent from the random oracle relied upon by the Fiat-Shamir transformation.

4 Sigma for Exponentiation with Hash Relation

In this section, we instantiate the Sigma AoKs as $\{\Pi_{\mathsf{dl}}^k\}_{k\in[4]}$ and $\{\Pi_{\mathsf{rsa}}^k\}_{k\in[2]}$, for the discrete logarithm relation $\{\mathcal{R}_{\mathsf{dl}}^k\}_{k\in[4]}$ and the RSA relation $\{\mathcal{R}_{\mathsf{rsa}}^k\}_{k\in[2]}$.

"Ping-pong" alternating of SNARK and Sigma. Our Sigma AoK protocol introduces an additional internal SNARK to provide (1) enhanced functionality: supporting a great variety of algebraic relations; and (2) linkage: ensuring the consistency of the witnesses between the (external) SNARK and the Sigma protocol. Overall, in our HCP framework, the proving process begins with a SNARK, transitions to a Sigma protocol, and then reverts to an internal SNARK (see Figure 3). Readers may find this design somewhat confusing, but from a high-level perspective, our HCP framework operates on an inspiring "ping-pong" paradigm, which enables the circuit that ultimately reverts to the internal SNARK to be significantly smaller than the complete group exponentiation circuit.

We begin by constructing Sigma AoKs with 1/2 (knowledge) soundness error, and then reduce this error through parallelization. We also present some optimization strategies for Sigma AoKs to further reduce the parallel overhead. Furthermore, we propose effective tricks to improve the proof efficiency in different scenarios based on whether the original SNARK and the internal SNARK are in the same field.

4.1 Sigma Argument of Knowledge

We present Π_{dl}^1 and Π_{rsa}^1 to show how to use internal SNARKs to construct Sigma AoKs for the exponentiation with hash relation. These two protocols are used in Section 5, while the other four protocols are provided in Appendix A.

4.1.1 Sigma AoKs for Discrete Logarithm Hard Groups

Consider a simple example of $Q = P^x$ over a discrete logarithm hard group, where (P, Q) are public inputs and x is a private input. This can be efficiently proven using the Schnorr protocol [39]. To combine the algebraic statement $Q = P^x$ with the hash statement $h = \mathsf{H}(x, r)$ to a conjunction (AND) statement, inspired by the work of [36],

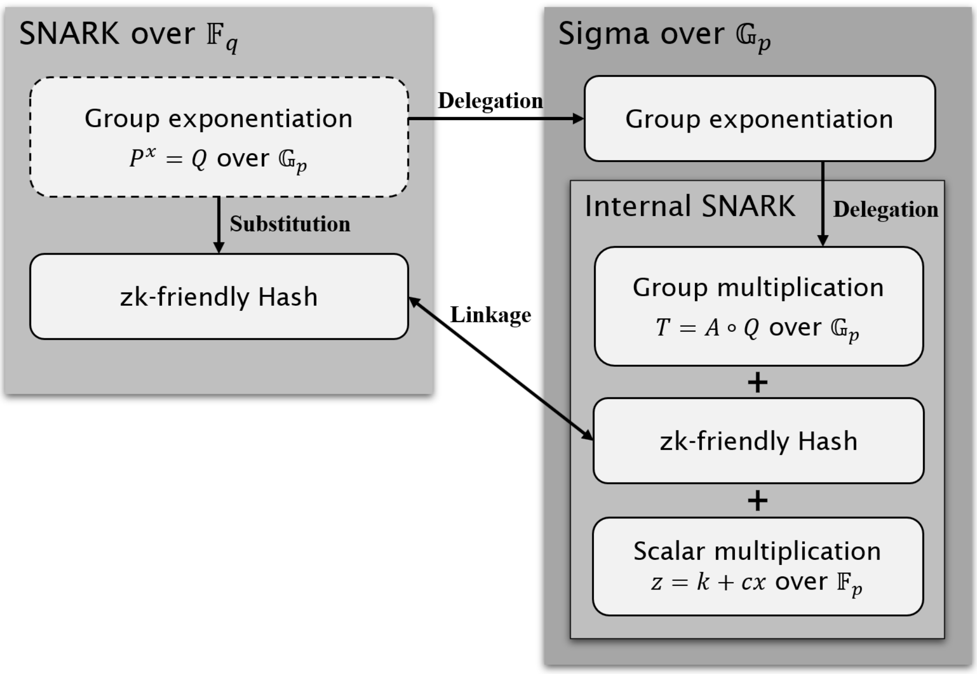

Fig. 3: An example HCP protocol for Π^1_{dl}

we found through empirical testing that it is remarkably efficient to verify that the third message $z = k + cx$ is actually computed using the hashed value x as detailed in Section 5. We outline the informal version of Π^0_{dl} for $\mathcal{R}^0_{\mathsf{dl}}$ below, with details provided in Appendix A.

1. P chooses $k \xleftarrow{\$} \mathbb{F}_p, r_k \xleftarrow{\$} \{0,1\}^\lambda$ and sends $A = P^k$, $h_k = \mathsf{H}(k, r_k)$ to V.
2. V chooses $c \xleftarrow{\$} \mathbb{F}_p$ and sends it to P.
3. P proves $\{z = k + cx \pmod p) \wedge h = \mathsf{H}(x, r) \wedge h_k = \mathsf{H}(k, r_k)\}$, sends z and the proof π_{int} to V.

However, the above method fails to construct proofs for statements with multiple witnesses, such as $\{(P, h; (Q, x, r)) : Q = P^x \wedge h = \mathsf{H}(Q, x, r)\}$ where both Q and x are private. To understand why the Schnorr protocol encounters difficulties when proving statements with multiple witnesses, we revisit the relation $\{(P; (Q, x)) : Q = P^x\}$. After receiving the third round message z, the verifier needs to check that $T = A \circ Q^c$ where $T = P^z$. However, since Q is private, the verifier cannot compute $A \circ Q^c$ on its own. One straightforward approach would be to require the prover to prove $T = A \circ Q^c$ using SNARKs. However, this necessitates proving a group exponentiation, which brings us back to the initial obstacle.

An observation here is that if we reduce the size of the challenge space, for example to 1-bit, then the response proof needs to imply only a single group operation $T = A \circ Q$

when $c = 1$, while no additional proof is needed when $c = 0$. When we execute the protocols in parallel to reduce the knowledge soundness error, the computation is significantly reduced compared to directly proving the exponentiation. Now, the prover can prove $T = A \circ Q$ via SNARKs to assist the verifier in completing the verification. Furthermore, directly sending A in the first round may lead to the leakage of Q through $T = A \circ Q$ when $c = 1$. Therefore, A must also be hidden in the hiding hash function.

Protocol Π^1_{dl} *(protocol with x and Q as witness).* Let $\mathbb{G}_p$ be a cyclic group with prime order p, H be a hash function. Since Q is also a witness, the statement includes the hash $h = \mathsf{H}(Q, x, r)$. The response includes an internal proof π_{int} from a zkSNARK Π_{int}, proving the following relationship:

$$\mathcal{R}^1_{\mathsf{int}} := \left\{ \begin{pmatrix} h, h_k, z, T; \\ (Q, x, k, A, r, r_k) \end{pmatrix} : \begin{array}{c} h = \mathsf{H}(Q, x, r) \wedge h_k = \mathsf{H}(A, k, r_k) \wedge \\ z = k + x \pmod{p} \wedge T = A \circ Q \end{array} \right\}$$

where $r, r_k \in \{0,1\}^\lambda$. The Sigma AoK for the relation $\mathcal{R}^1_{\mathsf{dl}}$ is shown in Figure 4.

Sigma Argument of Knowledge Π^1_{dl}

Setup. Run internal SNARK setup $\mathsf{crs} \leftarrow \Pi_{\mathsf{int}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^1_{\mathsf{int}})$

Public input(crs, P, h); **Private input**(Q, x, r);

Prover: $k \xleftarrow{\$} \mathbb{F}_p$; $r_k \xleftarrow{\$} \{0,1\}^\lambda$; $A = P^k$; $h_k = \mathsf{H}(A, k, r_k)$; send h_k.

Verifier: $c \xleftarrow{\$} \{0,1\}$; send c.

Prover:

Case $c = 0 : z = k \pmod{p}$; send z, r_k.

Case $c = 1 : z = k + x \pmod{p}; T = A \circ Q$;

Run Π_{int} for $\mathcal{R}^1_{\mathsf{int}}$: $\pi_{\mathsf{int}} \leftarrow \Pi_{\mathsf{int}}.\mathsf{P}(\mathsf{crs}, h, h_k, z, T; (Q, x, k, A, r, r_k))$; send z, π_{int}.

Verifier: compute $T = P^z$, and

Case $c = 0$: check $h_k \overset{?}{=} \mathsf{H}(T, z, r_k)$

Case $c = 1$: check $\Pi_{\mathsf{int}}.\mathsf{V}(\mathsf{crs}, h, h_k, z, T, \pi_{\mathsf{int}}) \overset{?}{=} 1$

Fig. 4: Π^1_{dl} for $\mathcal{R}^1_{\mathsf{dl}} := \{(P, h; (Q, x, r)) : Q = P^x \wedge h = \mathsf{H}(Q, x, r)\}$

Theorem 3 *Assume H is a hiding hash function, Π_{int} is a zkSNARK, then Π^1_{dl} in Figure 4 is a Sigma AoK for the relation:*

$$\mathcal{R}^1_{\mathsf{dl}} := \{(P, h; (Q, x, r)) : Q = P^x \wedge h = \mathsf{H}(Q, x, r)\}$$

where $P, Q \in \mathbb{G}_p$, $x \in \mathbb{F}_p$ and $r \in \{0,1\}^\lambda$.

We defer the detailed proof to Appendix B. We provide no details for the proofs of the remaining Sigma AoKs, as they employ a completely similar analytical approach to the theorem above.

4.1.2 Sigma AoKs for RSA Groups

In the discrete logarithm setting, the order of the cyclic group $\mathbb{G}_p$ is included in the public parameters. However, this is not the case for groups of unknown order. For example, in RSA encryption and signature schemes defined over the RSA group, variables are selected from $\mathbb{Z}_n^*$, and the order $\varphi(n)$ is unknown to anyone except the key holder.

In the RSA setting, we present two useful Sigma AoKs. Protocol Π^0_{rsa} constructs an argument of knowledge for RSA encryption: given a public key e and ciphertext c, the prover knows the corresponding plaintext m, as detailed in Appendix A. Protocol Π^1_{rsa} constructs an argument of knowledge for RSA signature: given a public key e, the prover knows a message-signature pair (m, Σ).

We now detail the protocol Π^1_{rsa}. To represent RSA signature verification, we consistently use the notation "$P^x = Q \mod n$", which is equivalent to "$\Sigma^e = m \mod n$". The construction of Sigma AoKs in the RSA setting continues the above pattern. If we restrict the challenge space to $\{0,1\}$, the extractor can apply the extended Euclidean algorithm on two different transcripts to compute the witness. Furthermore, for these Sigma AoKs with 1/2 knowledge soundness error, we employ parallel optimization techniques to reduce the error while saving costs. In particular, as the challenge space increases, the soundness is ensured by Sigma, whereas the knowledge soundness is directly provided by the SNARK extractor.

Protocol Π^1_{rsa} *(protocol with P and Q as witness).* Let $\mathbb{Z}_n^*$ be a cyclic group with RSA modulo n, H be a hash function. Protocol Π^1_{rsa} includes an internal proof π_{int} from a zkSNARK Π_{int}, proving the following relationship:

$$\mathcal{R}^5_{\mathsf{int}} := \left\{ \begin{pmatrix} h, h_k, Z, T; \\ (P, Q, K, A, r, r_k) \end{pmatrix} : \begin{array}{c} h = \mathsf{H}(P, Q, r) \wedge h_k = \mathsf{H}(K, A, r_k) \\ \wedge Z = K \circ P \pmod n \\ \wedge T = A \circ Q \pmod n \end{array} \right\}$$

where $r, r_k \in \{0,1\}^\lambda$. The Sigma AoK for the relation $\mathcal{R}^1_{\mathsf{rsa}}$ is shown in Figure 5.

Theorem 4 *Let n be an RSA modulo, assume H is a hiding hash function, Π_{int} is a zkSNARK, then Π^1_{rsa} in Figure 5 is a Sigma AoK for the relation:*

$$\mathcal{R}^1_{\mathsf{rsa}} := \{(x, h; (P, Q, r)) : Q = P^x \wedge h = \mathsf{H}(P, Q, r)\}$$

where $P, Q, x \in \mathbb{Z}_n^*$ and $r \in \{0,1\}^\lambda$.

The proof is similar to Theorem 3 (see Appendix B). Similarly, we present the Sigma AoK Π^0_{rsa} in Appendix A.

Sigma Argument of Knowledge Π^1_{rsa}

Setup. Run internal SNARK setup $\mathsf{crs} \leftarrow \Pi_{\mathsf{int}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^5_{\mathsf{int}})$

Public input(crs, x, h); **Private input**(P, Q, r);

<u>Prover:</u> $K \overset{\$}{\leftarrow} \mathbb{Z}^*_n; r_k \overset{\$}{\leftarrow} \{0,1\}^\lambda; A = K^x \pmod n; h_k = \mathsf{H}(K, A, r_k)$; send h_k.

<u>Verifier:</u> $c \overset{\$}{\leftarrow} \{0,1\}$; send c.

<u>Prover:</u>

Case $c = 0 : Z = K \pmod n$; send Z, r_k.

Case $c = 1 : Z = K \circ P \pmod n; T = A \circ Q \pmod n$;

Run Π_{int} for $\mathcal{R}^5_{\mathsf{int}} : \pi_{\mathsf{int}} \leftarrow \Pi_{\mathsf{int}}.\mathsf{P}\big(\mathsf{crs}, h, h_k, Z, T; (P, Q, K, A, r, r_k)\big)$; send Z, π_{int}.

<u>Verifier:</u> compute $T = Z^x \pmod n$, and

Case $c = 0$: check $h_k \overset{?}{=} \mathsf{H}(Z, T, r_k)$

Case $c = 1$: check $\Pi_{\mathsf{int}}.\mathsf{V}(\mathsf{crs}, h, h_k, Z, T, \pi_{\mathsf{int}}) \overset{?}{=} 1$

Fig. 5: Π^1_{rsa} for $\mathcal{R}^1_{\mathsf{rsa}} := \{(x, h; (P, Q, r)) : Q = P^x \wedge h = \mathsf{H}(P, Q, r)\}$

4.2 Parallel Optimization for Sigma AoKs

For Sigma AoKs with a (knowledge) soundness error of $1/2$, we can reduce the error through parallel repetition, similar to how the Sigma protocol reduces soundness error. To construct an extractor for the λ-times parallel Sigma AoK protocol, the extractor invokes the SNARK extractor to extract a witness of SNARK statement, and at the same time, it recovers the full statement of Sigma protocol and subsequently uses the Sigma extractor to extract a witness. The consistency of witnesses is guaranteed from the knowledge soundness of SNARKs and the collision resistance of hash. Since the negligible knowledge error of SNARK and negligible soundness error of parallel Sigma AoKs, this extractor can ensure knowledge soundness with negligible error.

To further enhance the efficiency of our protocols, we propose two different optimizations for the parallel of Sigma AoKs as following.

I. **Precompute and Reuse.** The prover cost in our proposed Sigma AoKs is dominated by the cost for generating the internal SNARK proof. Furthermore, the cost of computing this internal proof is almost entirely determined by the number of group multiplications as detailed in Section 5. Based on this analysis, we propose an optimization that can be directly applied to Π^1_{dl}, Π^2_{dl}, and Π^3_{dl}.[13]

Using protocol Π^1_{dl} as an example, we present an effective trick to optimize the number of parallel repetitions and achieve minimum circuit size. We slightly increase the size of challenge space and denote it by $[m]$. Observe that the prover needs to prove the same Q^c in parallel repetitions when the same $c \in [m]$ is chosen, so the

[13]The optimization cannot be directly used for $\{\Pi^k_{\mathsf{rsa}}\}_{k\in[2]}$ because in the unknown order group, the extractor of sigma may not be able to extract the witness from two transcripts.

prover can reuse the proof for $Q^2, \dots, Q^{m-1}$. We give the protocol of the version of Π^1_{dl} where the challenge space is increased to m, as follows, denoted by Π^{1*}_{dl}.

PROTOCOLΠ^{1*}_{dl} *(protocol with x and Q as witness).* An internal proof π_{int} from a zkSNARK Π_{int}, proving the following relationship:

$$\mathcal{R}^{1*}_{\mathsf{int}} := \left\{ \begin{pmatrix} h, h_k, c, z, T; \\ (Q, x, k, A, r, r_k) \end{pmatrix} : \begin{array}{c} h = \mathsf{H}(Q, x, r) \wedge h_k = \mathsf{H}(A, k, r_k) \wedge \\ z = k + cx \pmod{p} \wedge T = A \circ Q^c \end{array} \right\}$$

where $r, r_k \in \{0,1\}^\lambda$. The Sigma AoK for the relation $\mathcal{R}^{1*}_{\mathsf{dl}}$ is shown in Figure 6.

Sigma Argument of Knowledge Π^{1*}_{dl}

Setup. Run internal SNARK setup $\mathsf{crs} \leftarrow \Pi_{\mathsf{int}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^1_{\mathsf{int}}*)$

Public input(crs, P, h); **Private input**(Q, x, r);

Prover: $k \overset{\$}{\leftarrow} \mathbb{F}_p$; $r_k \overset{\$}{\leftarrow} \{0,1\}^\lambda$; $A = P^k$; $h_k = \mathsf{H}(A, k, r_k)$; send h_k.

Verifier: $c \overset{\$}{\leftarrow} \{0, 1, ..., m-1\}$; send c.

Prover:

Case $c = 0 : z = k \pmod{p}$; send z, r_k.

Case $c \neq 0 : z = k + cx \pmod{p}; T = A \circ Q^c$;

Run Π_{int} for $\mathcal{R}^{1*}_{\mathsf{int}} : \pi_{\mathsf{int}} \leftarrow \Pi_{\mathsf{int}}.\mathsf{P}(\mathsf{crs}, h, h_k, c, z, T; (Q, x, k, A, r, r_k))$; send z, π_{int}.

Verifier: compute $T = P^z$, and

Case $c = 0$: check $h_k \overset{?}{=} \mathsf{H}(T, z, r_k)$

Case $c \neq 0$: check $\Pi_{\mathsf{int}}.\mathsf{V}(\mathsf{crs}, h, h_k, c, z, T, \pi_{\mathsf{int}}) \overset{?}{=} 1$

Fig. 6: Π^{1*}_{dl} for $\mathcal{R}^{1*}_{\mathsf{dl}} := \{(P, h; (Q, x, r)) : Q = P^x \wedge h = \mathsf{H}(Q, x, r)\}$

For example, let the challenge $c \in \{0, 1, 2, 3\}$ and soundness error be 2^{-60}. Then, the number of parallel repetitions is $\ell = 60 \log_4 2 = 30$. The precomputation of Q^2, Q^3 requires 2 group operations. For a random $c \in \{0, 1, 2, 3\}$, the average number of group operations per parallel repeated circuit is $3/4$. This is because there is nothing to be proven when $c = 0$. In other cases, only one operation, $A \circ Q'$ (where $Q' = Q^c$), needs to be proven. Finally, the average number of group operations is 24.5, whereas the trivial parallel way requires 30. We present an strategy for selecting the optimal challenge space for Π^1_{dl} with a soundness error of $2^{-\lambda}$. This strategy involves determining the minimum point of the function $f(m) = \frac{\lambda(m-1)}{m} \log_m 2 + (m-2)$.

In practice, when executing these protocols in parallel, we can integrate the internal SNARKs into a single proof, thereby proving all instances of internal relations with one SNARK proof. Similarly, the prover can hash all information from the first round of the Sigma AoK protocol into a single output.

II. **Proof batching for uniform circuits.** When simply parallelizing the uniform version of Sigma AoKs, the internal SNARK are initialized for the same circuit with different assignments. We can employ various methods for this proof such as folding schemes, or proof batching techniques.

Although a binary challenge space requires λ repetitions (e.g., 128), our method maintains efficiency through two mechanisms: (1) Enlarging the challenge space (as per Optimization I) drastically reduces the repetition count (e.g., to 32 repetitions for 4-bit challenges); (2) The computational cost of each repetition involves lightweight hash verifications and NNA, which are significantly cheaper than group exponentiations. Even with repetitions, the aggregated cost remains an order of magnitude lower than direct circuit embedding.

4.3 Select hash functions and SNARKs

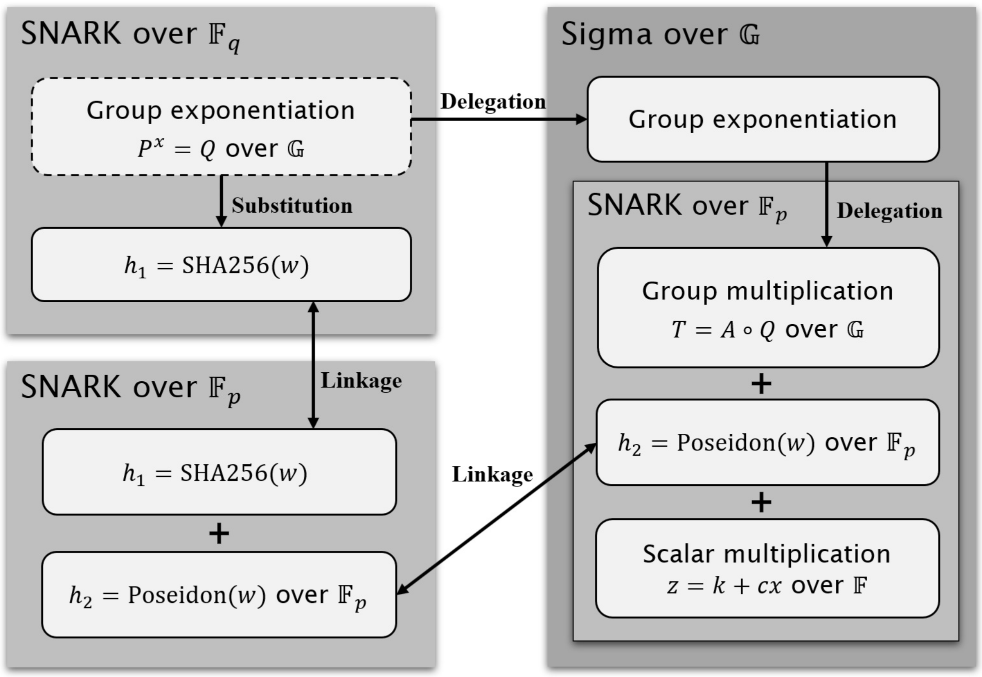
SNARK over $\mathbb{F}_q$
Group exponentiation $P^x = Q$ over $\mathbb{G}$
Substitution
$h_1 = \mathrm{SHA256}(w)$
Delegation
Sigma over $\mathbb{G}$
Group exponentiation
SNARK over $\mathbb{F}_p$
Delegation
Group multiplication $T = A \circ Q$ over $\mathbb{G}$
+
$h_2 = \mathrm{Poseidon}(w)$ over $\mathbb{F}_p$
+
Scalar multiplication $z = k + cx$ over $\mathbb{F}$
Linkage
SNARK over $\mathbb{F}_p$
$h_1 = \mathrm{SHA256}(w)$
+
$h_2 = \mathrm{Poseidon}(w)$ over $\mathbb{F}_p$
Linkage

Fig. 7: SNARKs in different field.

Another potential optimization involves choosing a different internal SNARK within the Sigma AoKs than the original SNARK. Higher efficiency is achieved when the operations of group $\mathbb{G}_p$ fall in the internal SNARK field $\mathbb{F}_q$, as this avoids non-native arithmetic. Due to recent works on SNARKs supporting arbitrary sufficiently large field [9, 11, 15, 16]. However, if the original SNARK and internal SNARK are over the same field, we can employ a zk-friendly hash function within the native field. In the case of different fields, we can only use cryptographic hash functions like SHA256, since proving zk-friendly hash function in non-native field would result in higher overhead. As shown in Figure 7, the prover additionally sends an internal SNARK proof, which implies that SHA256 and Poseidon contain the same witness. Then the internal SNARK can utilize the Poseidon hash over native field for multiple parallels.

5 Performance

In this section, we first demonstrate the improvement in prover efficiency by utilizing Sigma AoKs under the proposed Hash-committed commit-and-prove framework. Subsequently, we present some specific experimental tests to further support our analysis. All experiments were conducted on a machine equipped with a 12th Gen Intel i5-12500 (3.30 GHz) and 32 GB RAM, with results averaged over five runs. All experiments were implemented in single-threaded mode.

5.1 Asymptotic Analysis

For scalar multiplication of bit length N, existing schemes maintain an asymptotic complexity that scales with N. Our work is the first to replace an $O(N)$ problem with an $O(\lambda)$ problem, where λ is the security parameter (128-bit or 256-bit security). This is the key conceptual leap that enables an order-of-magnitude performance gain.

This $O(N)$ versus $O(\lambda)$ distinction has profound implications for scalability and generality. Consider a future application requiring 512-bit or 1024-bit scalars. For in-circuit methods, such as the ω-windowed approach [22] or the optimizations by Eagen et al. [23], the prover's cost scales linearly with the bit-length of the scalar. For our HCP framework, however, as long as the security parameter λ remains constant, the in-circuit prover cost will be largely unaffected. This demonstrates superior scalability for problems involving large algebraic structures. Furthermore, our framework is not limited to elliptic curves; it is equally applicable to RSA groups. The principle of offloading the expensive exponentiation and using a SNARK to prove a small part of a Sigma protocol is general. Thus, our contribution is not merely a faster way to prove elliptic curve scalar multiplication; it is a general framework for efficiently proving any statement dominated by expensive group exponentiations in small-field SNARKs.

5.2 Circuit-level cost breakdown

We report circuit-level cost statistics for representative algebraic relations. These results quantify how Hash-CP shifts expensive non-native algebraic operations out of the SNARK circuit and into lightweight Sigma-AoKs, while retaining only hash-friendly computations inside the SNARK.

In Table 1, we assume 256-bit elliptic curve group for $\{\mathcal{R}^i_{\mathsf{dl}}\}_{i\in[4]}$ and 2048-bit RSA group (where the $\mathsf{Integer\text{-}MUL}$ lies in) for $\{\mathcal{R}^i_{\mathsf{rsa}}\}_{i\in[2]}$. The first row represents the method of directly embedding the entire exponentiation operation into the SNARK circuit, then proving an elliptic curve exponentiation at most needs 256 point additions and 256 point doublings in the circuit and proving an integer modular exponentiation in RSA group needs at most 4096 integer modular multiplications in the circuit. The

Table 1: Comparison of circuit costs for $\{\mathcal{R}_{\mathsf{dl}}^{i}\}$ (assuming 256-bit EC group) and $\{\mathcal{R}_{\mathsf{rsa}}^{i}\}$ (assuming 2048-bit RSA group). The costs are measured in the number of group or integer operations required within the SNARK circuit.

Scheme	PoK for $\{\mathcal{R}_{\mathsf{dl}}^{i}\}_{i\in[4]}$		PoK for $\{\mathcal{R}_{\mathsf{rsa}}^{i}\}_{i\in[2]}$
	Group-ADD	Group-DBL	Integer-MUL
SNARK [18, 19, 38]	256	256	4096
ω-windowed [22]	144	272	−−

Table 2: Circuit embedding costs (number of operations) for our Sigma AoKs $\{\mathcal{R}_{\mathsf{dl}}^{i}\}_{i\in[4]}$ at 128-bit security. NNC denotes non-native arithmetic computations, and HASH refers to the count of elements hashed.

Scheme	Group-ADD	Group-DBL	NNC	HASH
Π_{dl}^{0}: PoK for $\mathcal{R}_{\mathsf{dl}}^{0}$	0	0	1	2
Π_{dl}^{1}: PoK for $\mathcal{R}_{\mathsf{dl}}^{1}$	41	3	32	66
Π_{dl}^{2}: PoK for $\mathcal{R}_{\mathsf{dl}}^{2}$	82	6	0	66
Π_{dl}^{3}: PoK for $\mathcal{R}_{\mathsf{dl}}^{3}$	82	6	65	135

Table 3: Circuit operations for Sigma AoKs $\{\mathcal{R}_{\mathsf{rsa}}^{i}\}_{i\in[2]}$ (128-bit security). These computations are parallelizable via optimizations in Section 4.2

Scheme	Integer-MUL	HASH
Π_{rsa}^{0}: PoK for $\mathcal{R}_{\mathsf{rsa}}^{0}$	64	192
Π_{rsa}^{1}: PoK for $\mathcal{R}_{\mathsf{rsa}}^{1}$	128	256

cost calculation for the ω-windowed includes computing lookup tables, as different exponentiations involve different bases and the required values (computed according to the equation (6) in [22]) contribute to this cost. In Table 2, the non-native computation refers to "$z = k + cx$" or "$k = tx$" ("$k = tx$" appears once in Π_{dl}^{3}). In Π_{dl}^{0}, Π_{dl}^{1} and Π_{dl}^{3}, we need to prove this constant multiplication and addition in the SNARK circuit. HASH refers to the total number of elements (group or field elements) that we need to hash in the SNARK circuit.

Comparing Tables 1, 2, and 3, it is shown that using our Sigma AoKs for exponentiation proofs very effectively reduces the necessary algebraic operations (the numbers are proportional to the security parameter, in this section we use 128-bit) in SNARK circuits. For example, for proofs in RSA groups, our method only requires proving 64 or 128 integer modular multiplications in parallel. This achieves a tens-of-times improvement compared to previous methods. The effect is also very significant in discrete-log hard groups; for instance, for Π^1_{dl}, the overhead is reduced by nearly a factor of 10.

5.3 Sigma AoK instantiation details

In this subsection, we explain how the Sigma-AoKs are concretely instantiated in our framework, with particular emphasis on security parameter selection and cost accounting. We further clarify how the quantitative results reported in Tables 2 and 3 are derived from these instantiations.

5.3.1 Concrete Parameter Settings

We illustrate how the numbers in Tables 2 and 3 are calculated. For $\mathcal{R}^1_{\mathsf{dl}}$, the soundness error of the Sigma AoK Π^1_{dl} is $1/2$. By employing the first optimization method from Section 4.2, we achieve a challenge space size of 16, resulting in a soundness error of 2^{-4} for a single execution. By running in parallel for 32 times, we obtain 2^{-128} knowledge soundness error. One instance of $\mathcal{R}^1_{\mathsf{dl}}$ requires proving 41 elliptic curve point additions, 3 point doublings, 32 non-native multiplications and a Poseidon hash of 66 elements in the SNARK. Sigma AoKs Π^2_{dl} and Π^3_{dl} follow a similar mechanism by employing the first optimization, and the challenge space is set to $[0, 1, ..., 15]$. For $\mathcal{R}^1_{\mathsf{rsa}}$, by running Π^1_{rsa} 128 times in parallel to ensure 128-bit security. On average, for 64 times $c = 0$, prover sends the random number used in hash function and "z" to verifier; and for 64 times $c = 1$, thus we need to prove $\mathcal{R}^5_{\mathsf{int}}$ with 64 different instances. In practical implementation, this can be accelerated by using the second optimization in Section 4.2.

We attain 128-bit security by setting the parallel repetition parameter k and challenge space size C such that the knowledge error $1/|C|^k \leq 2^{-128}$. We utilize standard, cryptanalyzed parameters for zk-friendly hashes (e.g., Poseidon) that provide 128-bit collision resistance, ensuring the scheme's practical security matches its theoretical bounds.

5.3.2 Overhead Analysis of Auxiliary Operations

According to the preceding protocols $\{\Pi^i_{\mathsf{rsa}}\}_{i\in[2]}$ and $\{\Pi^i_{\mathsf{dl}}\}_{i\in[4]}$, besides proving the group operations (Group-ADD, Group-DBL and Integer-MUL) shown in these Tables, the extra cost in Sigma AoKs mainly refers to proving hash preimages and the NNC. Below, we address that these two extra costs, compared to group operations, result in significantly smaller prover overhead in SNARKs.

First, let us consider the group operations. Taking elliptic curve groups as an example, the definition of group addition involves finding the intersection point of the line passing through the two points and the elliptic curve itself. This typically requires at least five or six non-native multiplications. For example, curves given by $y^2 = x^3 + ax + b$, $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ include the following non-native operations:

$$x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - (x_2 + x_1)$$
$$y_3 = \frac{y_2 - y_1}{x_2 - x_1}(x_1 - x_3) - y_1$$

In contrast, the NNC operations "$z = k + cx$" or "$k = tx$" each involve only one non-native multiplication. Note that "$k = tx$" appears only once in Π^3_{dl} (Figure A4), with all other instances being "$z = k + cx$", where c is a very small constant. Multiplying a constant by a non-native element requires significantly fewer multiplication gates. For illustration, a non-native multiplication between two elements from a non-native field, where each element is represented by l elements from the native field, may require approximately $O(l^2)$ multiplication gates, whereas multiplying a constant by a non-native element requires only $O(l)$ multiplication gates. Therefore, the prover overhead in SNARKs introduced by an NNC operation is substantially lower compared to that of $\mathsf{Group\text{-}ADD}$ or $\mathsf{Group\text{-}DBL}$. Second, let us consider proving hash preimages. The efficiency of proving hash preimages has witnessed rapid advancements in recent years. To demonstrate that the prover overhead introduced by the number of hash operations specified in the aforementioned Tables 2 and 3 is marginal, we present two illustrative examples. For instance, the SNARK construction over binary tower fields, as presented in [12], achieves a proving time of only 5.9 seconds for 8192 instances of Keccak-f (Table 6 in [12]). Another illustration comes from Plonky3 [45], developed the proof system over 31-bit fields, is capable of proving 1.7 million instances of the Poseidon2 hash function preimages within 1 second.

In conclusion, for proof systems capable of efficiently proving hash (zk-friendly hash functions or cryptographic hash functions), we present a technique that substantially reduces the size of SNARK circuits when proving group exponentiations. Our approach fundamentally replaces computationally expensive group operations with more efficiently provable hash computations within the circuit. In this context, "efficiently proving hash" specifically refers to the property of a SNARK scheme where the cost associated with proving hash computations is sufficiently low such that the overall efficiency gain derived from this replacement outweighs the cost incurred by directly embedding the group operations. Thus, relative efficiency is the key criterion. We proceed to detail some implementation tests that confirm the validity of our approach in the following sections.

5.4 Benchmarks across Proof systems

We next present end-to-end performance results over multiple proving backends. Each table reports prover time, verifier time, and proof size under a benchmark methodology described above.

5.4.1 Proof of knowledge for $\mathcal{R}^1_{\mathsf{dl}}$

We compare the prover efficiency of proving knowledge of $\mathcal{R}^1_{\mathsf{dl}}$ with and without the use of Sigma AoK. The relation $\mathcal{R}^1_{\mathsf{dl}}$ is prevalent in various applications, including but not limited to proof of solvency in blockchain applications and cryptographic protocols like ECDSA signature verification. For instance, proving knowledge of an x such that

SHA256(G^x) $= y$ in proof of solvency includes proving knowledge of x and y such that $G^x = y$, as an instance of $\mathcal{R}^1_{\mathsf{dl}}$. In the context of verifying ECDSA signatures, operations of the form $(x_1, y_1) = G^{u_1}P^{u_2}$ involve two instances of $\mathcal{R}^1_{\mathsf{dl}}$, for which one can apply our Sigma AoK Π^1_{dl} twice: once for G^{u_1} and once for P^{u_2}.

Table 4: We compare the prover efficiency for proving knowledge of $\mathcal{R}^1_{\mathsf{dl}}$ using different methods. For this evaluation, we instantiate $\mathcal{R}^1_{\mathsf{dl}}$ over the P-256 elliptic curve. The results in the third row represents the SNARK component's cost within the Π^1_{dl} proof system. The security parameter is set to be 128-bit.

	Prover time	Proof size	Verifier time
Plonky2 (Worst Case)	28.14s	156.92kb	2.23ms
Plonky2 (Average Case)	7.23s	144.47kb	1.96ms
This work (Plonky2+Sigma AoK)	0.71s	136.94kb	2.38ms

Table 5: Comparison of different methods for proving $\mathcal{R}^1_{\mathsf{dl}}$ using Groth16 as the underlying proof system.

	Prover time	Proof size	Verifier time
Groth16 (Worst Case)	0.76s	0.19kb	1.17ms
Groth16 (Average Case)	0.34s	0.19kb	1.27ms
This work (Groth16+Sigma AoK)	0.17s	0.75kb	1.56ms

Table 6: Comparison of different methods for proving $\mathcal{R}^1_{\mathsf{dl}}$ using Plonk as the underlying proof system.

	Prover time	Proof size	Verifier time
Plonk (Worst Case)	10.59s	0.57kb	1.73ms
Plonk (Average Case)	5.23s	0.57kb	1.33ms
This work (Plonk +Sigma AoK)	2.18s	1.13kb	1.62ms

In Table 4, we employ Plonky2 [14] as the underlying proof system for the test, which is a representative example of SNARKs over small fields (based on the Goldilocks field $\mathbb{F}_p$, $p = 2^{64} - 2^{32} + 1$). Directly proving $\mathcal{R}^1_{\mathsf{dl}}$ using Plonky2 in the worst case requires proving 256 point additions and 256 point doublings. On average, it still necessitates proving 256 point additions. In contrast, our proposed method requires proving significantly fewer computations, as detailed in Table 2. The proving time of our method is dominated by the SNARK component, with other parts contributing marginal to the total time. In our solution, the overall proof consists of 32 field elements, 2 random numbers within hash function, a single Poseidon hash output, and a SNARK proof. The length of random numbers within hash function is 128 bits.

In Tables 5 and 6, we instantiate the Sigma AoK with Groth16 and Plonk to further demonstrate that our method improves proof systems over large fields. For these experiments, we focus on the $\mathcal{R}^1_{\mathsf{dl}}$ relation due to its prevalence in practice. We implemented these benchmarks using the gnark library[14]. The number of constraints in R1CS for Groth16 is 38270. The number of constraints for Plonk is 134429. It is observed that Groth16 is significantly more efficient than Plonk in this context. This is because Groth16 employs a circuit-specific trusted setup, rendering its complexity independent of the number of addition gates. In contrast, while Plonk utilizes a universal setup, its construction complexity remains dependent on the number of addition gates. For SNARKs over large fields, our construction can still improves prover efficiency by 2-5×.

5.4.2 Proof of knowledge for $\mathcal{R}^2_{\mathsf{dl}}$

To further assess the efficacy of our approach, we conducted experiments on proving $\mathcal{R}^2_{\mathsf{dl}}$ utilizing Plonky2 in conjunction with Sigma AoK. The obtained metrics for the SNARK component are: a prover time of 1.53s, a proof size of 134.53kb, and a verifier time of 2.54ms. A comparison with the results presented in the first two rows in Table 4 reveals that our method similarly yields excellent performance for $\mathcal{R}^2_{\mathsf{dl}}$. The overall proof includes 32 group elements, two random numbers, a single Poseidon hash output, and a SNARK proof. When we use Groth16 and Plonk with Sigma AoK to prove $\mathcal{R}^2_{\mathsf{dl}}$, the obtained metrics for the SNARK component with Groth16 are: a prover time of 0.22s, a proof size of 0.81kb, and a verifier time of 1.74ms, the obtained metrics for the SNARK component with Plonk are: a prover time of 2.31s, a proof size of 1.22kb, and a verifier time of 1.69ms.

5.5 Comparison with Pedersen-committed CP-SNARKs

Finally, we compare Hash-committed CP against Pedersen-committed CP-SNARKs, focusing on the modulus-mismatch setting that motivates our framework. Table 7 summarizes asymptotic metrics and representative concrete costs. Existing CP-SNARKs rely on Pedersen-committed sigma protocols to verify algebraic statements. However, these sigma protocols lack flexibility regarding group choices and cannot support arbitrary algebraic statements effectively. They face two main limitations:

[14] https://github.com/Consensys/gnark

- **Group Order Rigidity:** These protocols require the order of the discrete logarithm group to match the field (or ring) of the corresponding algebraic statements. Finding a secure elliptic curve with a prescribed order is often not guaranteed. Furthermore, the Complex Multiplication (CM) method required to construct such curves is computationally intractable for many parameters. As noted in prior studies such as the ddlog protocol in [26], if a matching group cannot be found, the system suffers from a *modulus mismatch*. This forces the prover to simulate field arithmetic over a non-native field, incurring severe overhead in both computation and proof size.
- **Necessity of "Glue" Proofs:** Even when a suitable group is available, a "glue" proof is required to ensure consistency between the Pedersen commitments and the SNARK witnesses. While this overhead is minimal in specific constructions (e.g., LegoSNARK [28] or specific VSS-based protocols [31]), it becomes substantial in general cases, particularly when proving equality between commitments over groups of different orders, which necessitates expensive range proofs. Moreover, for state-of-the-art SNARKs based on Merkle tree commitments (e.g., FRI-based systems), efficient "glue" mechanisms do not currently exist.

To demonstrate the efficiency gap under the modulus mismatch setting, we conduct an experiment focusing on a statement common in Proof of Solvency (PoS) applications:

$$\text{Proof of Knowledge of } x \text{ such that } \mathsf{SHA256}(G^x) = y$$

Here, G is a generator of a group (e.g., Secp256k1) and y is a public hash output. This relation corresponds to $\mathcal{R}^1_{\mathsf{dl}}$. We assume a realistic scenario where the order of the Pedersen commitment group does not match the algebraic structure of the statement (i.e., the scalar field of Secp256k1), inducing a modulus mismatch. The experimental configuration is as following:

- **Our Construction:** We utilize the optimization from Section 4.2 to achieve a challenge space of size 8 (soundness error 2^{-3}). By executing the protocol 20 times in parallel, we achieve a knowledge soundness error of 2^{-60}, consistent with previous work [26].
- **Internal SNARK Costs:** One instance of $\mathcal{R}^1_{\mathsf{dl}}$ requires proving 26 elliptic curve point additions, 20 non-native multiplications, and a Poseidon hash of 42 elements. Using Groth16, this takes less than 0.11s.
- **Verification:** The generated proof consists of 20 field elements, a single Poseidon hash output, and the internal SNARK proof. The verifier performs 20 group exponentiations plus the standard SNARK verification.

Table 7 presents the comparison results. The efficiency gap is primarily driven by the overhead required to handle the modulus mismatch. It is important to acknowledge that in the ideal scenario where the order of the Pedersen commitment group perfectly matches the algebraic structure of the statement (modulus matching), Pedersen-CP methods are highly efficient. However, such matching groups are often

Table 7: Comparison of NIZK performance for PoK of "$\mathsf{SHA256}(G^x) = y$" (assuming $G \in$ Secp256k1). n denotes the anonymity set size. $T_{\mathsf{ps}}, T_{\mathsf{vs}}, |\pi_{\mathsf{s}}|$ refer to the SNARK prover time, verifier time, and proof size. $T_{\mathsf{pg}}, T_{\mathsf{vg}}, |\pi_{\mathsf{g}}|$ refer to the "glue" protocol metrics. c_1, c_2 are small constants.

	Proof Size	Prover Time	Verifier time
SNARK e.g. [2]	$\|\pi_{\mathsf{s}}\|$	$nT_{\mathsf{ps}}(\text{SHA256})$ $+nT_{\mathsf{ps}}(G^x)$	T_{vs}
Pedersen-CP NIZKs$^{\mathsf{pos}}$ [24–32]	$2370n$ elements $+c_1+\|\pi_{\mathsf{g}}\|+\|\pi_{\mathsf{s}}\|$	$nT_{\mathsf{ps}}(\text{SHA256})+T_{\mathsf{pg}}$ $+9480n$ exp	$2560n$ exp $+T_{\mathsf{vg}}+T_{\mathsf{vs}}$
This work (Hash-CP NIZKs$^{\mathsf{pos}}$)	$20n$ elements $+c_2+\|\pi_{\mathsf{s}}\|$	$nT_{\mathsf{ps}}(\text{SHA256})$ $+nT_{\mathsf{ps}}(\mathcal{R}^1_{\mathsf{int}})$	$20n$ exp $+T_{\mathsf{vs}}$

unavailable in practice. For instance, finding a curve to match the base field of another elliptic curve is restrictive, and finding a suitable elliptic curve group to serve as the Pedersen commitment group for an RSA group is computationally challenging. In these common modulus-mismatch settings, our framework offers significantly greater versatility and efficiency.

Prover Efficiency. In the Pedersen-CP approach, the modulus mismatch forces the prover to perform bit-decomposition, resulting in 9480 group exponentiations. We benchmarked these operations on the NIST P-521 curve, yielding a computation time of approximately 5.21s. In contrast, our Hash-CP method replaces these expensive operations with an internal SNARK proof, which takes only ≈ 0.11s using Groth16. This demonstrates an approximately $50\times$ reduction in prover time.

Communication and Verification. The Pedersen-CP approach requires transmitting $2370n$ group elements and performing $2560n$ exponentiations for verification. Our method requires only $20n$ elements and $20n$ exponentiations. This represents a reduction in both proof size and verification complexity by a factor of over $100\times$.

Acknowledgements

We are supported by the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDB0690200), and the National Key Research and Development Program of China (Grant No. 2023YFB4503203).

List of Abbreviations

AoK	Argument of Knowledge
CP	Commit-and-Prove
CRS	Common Reference String

EXP	Exponentiation
HCP	Hash-committed Commit-and-Prove
NIZK	Non-interactive Zero-knowledge
NNA	Non-Native Arithmetic
PoK	Proof of Knowledge
ROM	Random Oracle Model
SHVZK	Special Honest-Verifier Zero-Knowledge
zkSNARK	Zero-Knowledge Succinct Non-interactive Argument of Knowledge

Availability of data and materials

The source code used during the current study are available from https://github.com/Hash-CP-NIZKs.

References

[1] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: 17th Annual ACM Symposium on Theory of Computing, pp. 291–304. ACM Press, Providence, RI, USA (1985). https://doi.org/10.1145/22145.22178

[2] Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part II. Lecture Notes in Computer Science, vol. 9666, pp. 305–326. Springer, Vienna, Austria (2016). https://doi.org/10.1007/978-3-662-49896-5_11

[3] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018). https://eprint.iacr.org/2018/046

[4] Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019: 26th Conference on Computer and Communications Security, pp. 2111–2128. ACM Press, London, UK (2019). https://doi.org/10.1145/3319535.3339817

[5] Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019, Part III. Lecture Notes in Computer Science, vol. 11694, pp. 733–764. Springer, Santa Barbara, CA, USA (2019). https://doi.org/10.1007/978-3-030-26954-8_24

[6] Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Report 2019/953 (2019). https://eprint.iacr.org/2019/953

[7] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019, Part I. Lecture Notes in Computer Science, vol. 11476, pp. 103–128. Springer, Darmstadt, Germany (2019). https://doi.org/10.1007/978-3-030-17653-2_4

[8] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020, Part I. Lecture Notes in Computer Science, vol. 12105, pp. 738–768. Springer, Zagreb, Croatia (2020). https://doi.org/10.1007/978-3-030-45721-1_26

[9] Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part II. Lecture Notes in Computer Science, vol. 14082, pp. 193–226. Springer, Santa Barbara, CA, USA (2023). https://doi.org/10.1007/978-3-031-38545-2_7

[10] Chen, B., Bünz, B., Boneh, D., Zhang, Z.: HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part II. Lecture Notes in Computer Science, vol. 14005, pp. 499–530. Springer, Lyon, France (2023). https://doi.org/10.1007/978-3-031-30617-4_17

[11] Zeilberger, H., Chen, B., Fisch, B.: Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. Springer, ??? (2024). https://doi.org/10.1007/978-3-031-68403-6_5

[12] Diamond, B.E., Posen, J.: Succinct Arguments over Towers of Binary Fields. Cryptology ePrint Archive, Report 2023/1784 (2023). https://eprint.iacr.org/2023/1784

[13] Bruestle, J., Gafni, P.: RISC Zero zkVM: scalable, transparent arguments of RISCV integrity. https://www.risczero.com/proof-system-in-detail.pdf. Accessed on 2025-05-08 (2023)

[14] 0xPolygonZero: Plonky2: Fast, Recursive zkSNARKs (2023). https://github.com/0xPolygonZero/plonky2/blob/main/plonky2/plonky2.pdf

[15] Ben-Sasson, E., Carmon, D., Kopparty, S., Levit, D.: Scalable and transparent proofs over all large fields, via elliptic curves - (ECFFT part II). In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022: 20th Theory of Cryptography Conference, Part I. Lecture Notes in Computer Science, vol. 13747, pp. 467–496. Springer, Chicago, IL, USA (2022). https://doi.org/10.1007/978-3-031-22318-1_17

[16] Haböck, U., Levit, D., Papini, S.: Circle STARKs. Cryptology ePrint Archive, Report 2024/278 (2024). https://eprint.iacr.org/2024/278

[17] Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for zero-knowledge proof systems. In: Bailey, M., Greenstadt, R. (eds.) USENIX Security 2021: 30th USENIX Security Symposium, pp. 519–535. USENIX Association, ??? (2021)

[18] Kosba, A.E., Papamanthou, C., Shi, E.: xJsnark: A framework for efficient verifiable computation. In: 2018 IEEE Symposium on Security and Privacy, pp. 944–961. IEEE Computer Society Press, San Francisco, CA, USA (2018). https://doi.org/10.1109/SP.2018.00018

[19] JkY: Non-Native Field Arithmetic. https://hackmd.io/@JkY-zACaSqerTtn_UwFjKg/SJZw6x75o (2023)

[20] Lubarov, D., Melé, J.B.: Casting out Primes: Bignum Arithmetic for Zero-Knowledge Proofs. Cryptology ePrint Archive, Paper 2022/1470 (2022). https://eprint.iacr.org/2022/1470

[21] Ambrona, M., Firsov, D., Querejeta-Azurmendi, I.: Efficient Foreign-Field Arithmetic in PLONK. Cryptology ePrint Archive, Paper 2025/695 (2025). https://eprint.iacr.org/2025/695

[22] Zakharov, D., Kurbatov, O., Bista, M., Bist, B.: Optimizing Big Integer Multiplication on Bitcoin: Introducing w-windowed Approach. Cryptology ePrint Archive, Report 2024/1236 (2024). https://eprint.iacr.org/2024/1236

[23] Eagen, L., Housni, Y.E., Masson, S., Piellard, T.: Fast elliptic curve scalar multiplications in SN(T)ARK circuits. IACR Cryptol. ePrint Arch., 933 (2025)

[24] Chase, M., Ganesh, C., Mohassel, P.: Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016, Part III. Lecture Notes in Computer Science, vol. 9816, pp. 499–530. Springer, Santa Barbara, CA, USA (2016). https://doi.org/10.1007/978-3-662-53015-3_18

[25] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, pp. 315–334. IEEE Computer Society Press, San Francisco, CA, USA (2018). https://doi.org/10.1109/SP.2018.00020

[26] Agrawal, S., Ganesh, C., Mohassel, P.: Non-interactive zero-knowledge proofs for composite statements. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part III. Lecture Notes in Computer Science, vol. 10993, pp. 643–673. Springer, Santa Barbara, CA, USA (2018). https://doi.org/10.1007/978-3-319-96878-0_22

[27] Backes, M., Hanzlik, L., Herzberg, A., Kate, A., Pryvalov, I.: Efficient non-interactive zero-knowledge proofs in cross-domains without trusted setup. In: Lin, D., Sako, K. (eds.) PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 11442, pp. 286–313. Springer, Beijing, China (2019). https://doi.org/10.1007/978-3-030-17253-4_10

[28] Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019: 26th Conference on Computer and Communications Security, pp. 2075–2092. ACM Press, London, UK (2019). https://doi.org/10.1145/3319535.3339820

[29] Campanelli, M., Faonio, A., Fiore, D., Querol, A., Rodríguez, H.: Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021, Part III. Lecture Notes in Computer Science, vol. 13092, pp. 3–33. Springer, Singapore (2021). https://doi.org/10.1007/978-3-030-92078-4_1

[30] Aranha, D.F., Bennedsen, E.M., Campanelli, M., Ganesh, C., Orlandi, C., Takahashi, A.: ECLIPSE: Enhanced compiling method for pedersen-committed zkSNARK engines. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part I. Lecture Notes in Computer Science, vol. 13177, pp. 584–614. Springer, Virtual Event (2022). https://doi.org/10.1007/978-3-030-97121-2_21

[31] Zhang, M., Chen, Y., Yao, C., Wang, Z.: Sigma protocols from verifiable secret sharing and their applications. In: Guo, J., Steinfeld, R. (eds.) Advances in Cryptology – ASIACRYPT 2023, Part II. Lecture Notes in Computer Science, vol. 14439, pp. 208–242. Springer, Guangzhou, China (2023). https://doi.org/10.1007/978-981-99-8724-5_7

[32] Orrù, M., Kadianakis, G., Maller, M., Zaverucha, G.: Beyond the circuit: How to minimize foreign arithmetic in ZKP circuits. IACR Commun. Cryptol. **2**(1), 23 (2025) https://doi.org/10.62056/AN-4C3C2H

[33] Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) Advances in Cryptology – EUROCRYPT 2008. Lecture Notes in Computer Science, vol. 4965, pp. 415–432. Springer, Istanbul, Turkey (2008). https://doi.org/10.1007/978-3-540-78967-3_24

[34] Bowe, S., Grigg, J., Hopwood, D.: Halo: Recursive Proof Composition without a Trusted Setup. Cryptology ePrint Archive, Report 2019/1021 (2019). https://eprint.iacr.org/2019/1021

[35] Kohrita, T., Towa, P., Williamson, Z.J.: One-Shot Native Proofs of Non-Native Operations in Incrementally Verifiable Computations. IACR Cryptol. ePrint

Arch. (2024). https://eprint.iacr.org/2024/1651

[36] Kadianakis, G., Maller, M., Novakovic, A.: Sigmabus: Binding Sigmas in Circuits for Fast Curve Operations. Cryptology ePrint Archive, Report 2023/1406 (2023). https://eprint.iacr.org/2023/1406

[37] Garg, S., Goel, A., Wang, M.: How to prove statements obliviously? In: Annual International Cryptology Conference, pp. 449–487 (2024). Springer

[38] Jeong, G., Moon, M., Yoon, G., Oh, H., Kim, J.: Tangram: Encryption-friendly SNARK framework under Pedersen committed engines. Cryptology ePrint Archive, Paper 2025/540 (2025). https://eprint.iacr.org/2025/540

[39] Schnorr, C.-P.: Efficient signature generation by smart cards. Journal of Cryptology **4**(3), 161–174 (1991) https://doi.org/10.1007/BF00196725

[40] Guillou, L.C., Quisquater, J.-J.: A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory. In: Günther, C.G. (ed.) Advances in Cryptology – EUROCRYPT'88. Lecture Notes in Computer Science, vol. 330, pp. 123–128. Springer, Davos, Switzerland (1988). https://doi.org/10.1007/3-540-45961-8_11

[41] Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive zero-knowledge arguments from folding schemes. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022, Part IV. Lecture Notes in Computer Science, vol. 13510, pp. 359–388. Springer, Santa Barbara, CA, USA (2022). https://doi.org/10.1007/978-3-031-15985-5_13

[42] Damgård, I.: Efficient and probabilistic proofs. PhD thesis, Aarhus University, Denmark, Aarhus, Denmark (1991)

[43] Dao, Q., Grubbs, P.: Spartan and bulletproofs are simulation-extractable (for free!). In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part II. Lecture Notes in Computer Science, vol. 14005, pp. 531–562. Springer, Lyon, France (2023). https://doi.org/10.1007/978-3-031-30617-4_18

[44] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology – CRYPTO'86. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer, Santa Barbara, CA, USA (1987). https://doi.org/10.1007/3-540-47721-7_12

[45] Developers, P.: Plonky3 Repository: Benchmarks. https://github.com/Plonky3/Plonky3/tree/main?tab=readme-ov-file#benchmarks. Accessed on 2025-05-11 (2024)

[46] Guillou, L.C., Quisquater, J.-J.: A "paradoxical" indentity-based signature

scheme resulting from zero-knowledge. In: Goldwasser, S. (ed.) Advances in Cryptology – CRYPTO'88. Lecture Notes in Computer Science, vol. 403, pp. 216–231. Springer, Santa Barbara, CA, USA (1990). https://doi.org/10.1007/0-387-34799-2_16

Appendix A Other Sigma AoKs

In discrete logarithm setting, let $\mathbb{G}_p$ be a cyclic group with prime order p. The statement $Q = P^x$ has two trivial cases, when only P or Q is the witness, the verifier can directly compute it. Additionally, in the case where P and x are witness, the statement can be converted to $P = Q^{x^{-1} \bmod p}$, thus using Π^1_{dl} to prove it.

Protocol Π^0_{dl} *(protocol with x as witness).* Let H be a hash function, and define the relation $\mathcal{R}^0_{\mathsf{int}}$ as follows:

$$\mathcal{R}^0_{\mathsf{int}} := \left\{ (h, h_k, c, z; (x, k, r, r_k)) : \begin{matrix} h = \mathsf{H}(x, r) \wedge h_k = \mathsf{H}(k, r_k) \\ \wedge z = k + cx \pmod p \end{matrix} \right\}$$

where $x, k, c, z \in \mathbb{F}_p$, $r, r_k \in \{0,1\}^\lambda$. The Sigma AoK for the relation $\mathcal{R}^0_{\mathsf{dl}}$ is shown in Figure A1, and the proof is similar to Theorem 3 (see Appendix B).

Sigma Argument of Knowledge Π^0_{dl}

Setup. Run internal SNARK setup $\mathsf{crs} \leftarrow \Pi_{\mathsf{int}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^0_{\mathsf{int}})$

Public input(crs, P, Q, h); **Private input**(x, r);

Prover: $k \xleftarrow{\$} \mathbb{F}_p; r_k \xleftarrow{\$} \{0,1\}^\lambda; A = P^k; h_k = \mathsf{H}(k, r_k)$; send h_k, A.

Verifier: $c \xleftarrow{\$} \mathbb{F}_p$; send c.

Prover: $z = k + cx \pmod p$
Run Π_{int} for $\mathcal{R}^0_{\mathsf{int}}$: $\pi_{\mathsf{int}} \leftarrow \Pi_{\mathsf{int}}.\mathsf{P}(\mathsf{crs}, h, h_k, c, z; (x, k, r, r_k))$; send z, π_{int}.

Verifier:
check $P^z \overset{?}{=} A \circ Q^c \wedge \Pi_{\mathsf{int}}.\mathsf{V}(\mathsf{crs}, h, h_k, c, z, \pi_{\mathsf{int}}) \overset{?}{=} 1$

Fig. A1: Π^0_{dl} for $\mathcal{R}^0_{\mathsf{dl}} := \{(P, Q, h; (x, r)) : Q = P^x \wedge h = \mathsf{H}(x, r)\}$

Theorem 5 *Assume H is a hiding hash function, Π_{int} is a zkSNARK, then Π^0_{dl} in Figure A1 is a Sigma AoK for the relation:*

$$\mathcal{R}^0_{\mathsf{dl}} := \{(P, Q, h; (x, r)) : Q = P^x \wedge h = \mathsf{H}(x, r)\}$$

where $P, Q \in \mathbb{G}_p$ and $x \in \mathbb{F}_p, r \in \{0,1\}^\lambda$.

Protocol Π^2_{dl} *(protocol with P and Q as witness).* Protocol Π^2_{dl} is quite similar to Π^1_{rsa}. Let H be a hash function, and define the relation $\mathcal{R}^2_{\mathsf{int}}$ as follows:

$$\mathcal{R}^2_{\mathsf{int}} := \left\{ \begin{pmatrix} h, h_k, Z, T; \\ (P, Q, K, A, r, r_k) \end{pmatrix} : \begin{matrix} h = \mathsf{H}(P, Q, r) \wedge h_k = \mathsf{H}(K, A, r_k) \\ \wedge Z = K \circ P \wedge T = A \circ Q \end{matrix} \right\}$$

where $r, r_k \in \{0,1\}^\lambda$. The Sigma AoK for relation $\mathcal{R}^2_{\mathsf{dl}}$ is shown in Figure A2, and the proof is similar to Theorem 3 (see Appendix B).

Theorem 6 *Assume H is a hiding hash function, Π_{int} is a zkSNARK, then Π^2_{dl} in Figure A2 is a Sigma AoK for the relation:*

$$\mathcal{R}^2_{\mathsf{dl}} := \{(x, h; (P, Q, r)) : Q = P^x \wedge h = \mathsf{H}(P, Q, r)\}$$

where $P, Q \in \mathbb{G}_p$ and $x \in \mathbb{F}_p, r \in \{0,1\}^\lambda$.

Sigma Argument of Knowledge Π^2_{dl}

Setup. Run internal SNARK setup $\mathsf{crs} \leftarrow \Pi_{\mathsf{int}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^2_{\mathsf{int}})$

Public input(crs, x, h); **Private input**(P, Q, r);

<u>Prover:</u> $K \xleftarrow{\$} \mathbb{G}_p; r_k \xleftarrow{\$} \{0,1\}^\lambda; A = K^x; h_k = \mathsf{H}(K, A, r_k)$; send h_k.

<u>Verifier:</u> $c \xleftarrow{\$} \{0,1\}$; send c.

<u>Prover:</u>

Case $c = 0 : Z = K$; send Z, r_k.

Case $c = 1 : Z = K \circ P; T = A \circ Q$;

Run Π_{int} for $\mathcal{R}^2_{\mathsf{int}} : \pi_{\mathsf{int}} \leftarrow \Pi_{\mathsf{int}}.\mathsf{P}\big(\mathsf{crs}, h, h_k, Z, T; (P, Q, K, A, r, r_k)\big)$; send Z, π_{int}.

<u>Verifier:</u> compute $T = Z^x$, and

Case $c = 0$: check $h_k \overset{?}{=} \mathsf{H}(Z, T, r_k)$

Case $c = 1$: check $\Pi_{\mathsf{int}}.\mathsf{V}(\mathsf{crs}, h, h_k, Z, T, \pi_{\mathsf{int}}) \overset{?}{=} 1$

Fig. A2: Π^2_{dl} for $\mathcal{R}^2_{\mathsf{dl}} := \{(x, h; (P, Q, r)) : Q = P^x \wedge h = \mathsf{H}(P, Q, r)\}$

Similar to Π^{1*}_{dl}, we present the protocol for Π^2_{dl} when the challenge space is increased to m, as follows, and denote it by Π^{2*}_{dl}.

Protocol Π^{2*}_{dl} *(protocol with P and Q as witness).* Let H be a hash function, and define the relation $\mathcal{R}^{2*}_{\mathsf{int}}$ as follows:

$$\mathcal{R}^{2*}_{\mathsf{int}} := \left\{ \begin{pmatrix} h, h_k, c, Z, T; \\ (P, Q, K, A, r, r_k) \end{pmatrix} : \begin{matrix} h = \mathsf{H}(P, Q, r) \wedge h_k = \mathsf{H}(K, A, r_k) \\ \wedge Z = K \circ P^c \wedge T = A \circ Q^c \end{matrix} \right\}$$

where $c \in \{0,1\}$, $r, r_k \in \{0,1\}^\lambda$, and H is a hash function.The Sigma AoK for relation $\mathcal{R}_{\mathsf{dl}}^{2*}$ is shown in Figure A3.

Sigma Argument of Knowledge Π_{dl}^{2*}

Setup. Run internal SNARK setup $\mathsf{crs} \leftarrow \Pi_{\mathsf{int}}.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathsf{int}}^{2*})$

Public input(crs, x, h); **Private input**(P, Q, r);

Prover: $K \xleftarrow{\$} \mathbb{G}_p; r_k \xleftarrow{\$} \{0,1\}^\lambda; A = K^x; h_k = \mathsf{H}(K, A, r_k)$; send h_k.

Verifier: $c \xleftarrow{\$} \{0, 1, ..., m-1\}$; send c.

Prover:

Case $c = 0 : Z = K$; send Z, r_k.

Case $c \neq 0 : Z = K \circ P^c; T = A \circ Q^c$;

Run Π_{int} for $\mathcal{R}_{\mathsf{int}}^{2*} : \pi_{\mathsf{int}} \leftarrow \Pi_{\mathsf{int}}.\mathsf{P}\big(\mathsf{crs}, h, h_k, c, Z, T; (P, Q, K, A, r, r_k)\big)$; send Z, π_{int}.

Verifier: compute $T = Z^x$, and

Case $c = 0$: check $h_k \overset{?}{=} \mathsf{H}(Z, T, r_k)$

Case $c \neq 0$: check $\Pi_{\mathsf{int}}.\mathsf{V}(\mathsf{crs}, h, h_k, c, Z, T, \pi_{\mathsf{int}}) \overset{?}{=} 1$

Fig. A3: Π_{dl}^{2*} for $\mathcal{R}_{\mathsf{dl}}^{2*} := \{(x, h; (P, Q, r)) : Q = P^x \wedge h = \mathsf{H}(P, Q, r)\}$

Protocol Π_{dl}^3 *(protocol with P, Q and x as witness).* Protocol Π_{dl}^3 employs the "intermediate value", which involves selecting a pair of inverse exponents s and t, computing a public intermediate value $T = P^s$, and then deriving $P = T^t$ and $Q = T^{tx}$ as two statements of $\mathcal{R}_{\mathsf{dl}}^1$, which can be proven using Π_{dl}^1 in parallel. And it also includes a proof for the relation:

$$\mathcal{R}_{\mathsf{int}}^3 := \left\{ \begin{pmatrix} h_x, h_t, h_k; \\ (x, t, k, r_x, r_t, r_k) \end{pmatrix} : \begin{matrix} k = tx (\mathrm{mod}\ p) \wedge h_x = \mathsf{H}(x, r_x) \\ \wedge h_t = \mathsf{H}(t, r_t) \wedge h_k = \mathsf{H}(k, r_k) \end{matrix} \right\}$$

and the Sigma AoK for the relation $\mathcal{R}_{\mathsf{dl}}^3$ is shown in Figure A4.

Theorem 7 *Assume H is a hiding hash function, Π_{int} is a zkSNARK, and Π_{dl}^1 is the Sigma AoK for $\mathcal{R}_{\mathsf{dl}}^1$ as described in Theorem 3, then Π_{dl}^3 in Figure A4 is a Sigma AoK for the relation:*

$$\mathcal{R}_{\mathsf{dl}}^3 := \left\{ \begin{pmatrix} h_p, h_q, h_x; \\ (P, Q, x, r_p, r_q, r_x) \end{pmatrix} : \begin{matrix} Q = P^x \wedge h_p = \mathsf{H}(P, r_p) \wedge \\ h_q = \mathsf{H}(Q, r_q) \wedge h_x = \mathsf{H}(x, r_x) \end{matrix} \right\}$$

where $P, Q \in \mathbb{G}_p$, $x \in \mathbb{F}_p$ and $r_p, r_q, r_x \in \{0,1\}^\lambda$.

As for Π_{dl}^3, its version with the challenge space increased to m can be realized by calling Π_{dl}^{1*}, which is quite trivial and will not be repeated here.

Sigma Argument of Knowledge Π^3_{dl}

Setup. Run internal SNARK setup $\mathsf{crs} \leftarrow \Pi_{\mathsf{int}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^3_{\mathsf{int}})$
and independently $\mathsf{crs}_1, \mathsf{crs}_2 \leftarrow \Pi^2_{\mathsf{dl}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^2_{\mathsf{dl}})$

Public input$(\{\mathsf{crs}_i\}, h_p, h_q, h_x)$; **Private input**$(P, Q, x, r_p, r_q, r_x)$;

Prover: $s \xleftarrow{\$} \mathbb{F}_p; r_t, r_k \xleftarrow{\$} \{0,1\}^\lambda; T = P^s; t = s^{-1} \pmod p$;
$k = tx \pmod p; h_t = \mathsf{H}(t, r_t); h_k = \mathsf{H}(k, r_k)$;
Run Π_{int} for $\mathcal{R}^3_{\mathsf{int}}$: $\pi_{\mathsf{int}} \leftarrow \Pi_{\mathsf{int}}.\mathsf{P}(\mathsf{crs}_0, h_x, h_t, h_k; (x, t, k, r_x, r_t, r_k))$;
send $T, h_t, h_k, \pi_{\mathsf{int}}$.

Prover and Verifier: run Π^1_{dl} :
$(\mathsf{crs}_1, T, h_p, h_t; (P, t, r_p, r_t)) \in \mathcal{R}^1_{\mathsf{dl}}$
$(\mathsf{crs}_2, T, h_q, h_k; (Q, k, r_q, r_k)) \in \mathcal{R}^1_{\mathsf{dl}}$

Verifier:
check $\Pi_{\mathsf{int}}.\mathsf{V}(\mathsf{crs}_0, h_x, \pi_{\mathsf{int}}) \stackrel{?}{=} 1$ and both Π^1_{dl} are acceptable.

Fig. A4: Π^3_{dl} for $\mathcal{R}^3_{\mathsf{dl}}$

Protocol Π^0_{rsa} *(protocol with P as witness).* Protocol Π^0_{rsa} is a variant of the Guillou-Quisquater protocol [46]. Let n be an RSA modulo, H be a hash function, and define the relation $\mathcal{R}^4_{\mathsf{int}}$ as follows:

$$\mathcal{R}^4_{\mathsf{int}} := \left\{ (h, h_k, Z; (P, K, r, r_k)) : \begin{array}{c} h = \mathsf{H}(P, r) \wedge h_k = \mathsf{H}(K, A, r_k) \\ \wedge Z = K \circ P \pmod n \end{array} \right\}$$

where $r, r_k \in \{0,1\}^\lambda$. The Sigma AoK for relation $\mathcal{R}^0_{\mathsf{rsa}}$ is shown in Figure A5, and the proof is similar to Theorem 3 (see Appendix B).

Theorem 8 *Assume H is a hiding hash function, Π_{int} is a zkSNARK, then Π^0_{rsa} in Figure A5 is a Sigma AoK for the relation:*

$$\mathcal{R}^0_{\mathsf{rsa}} := \{(Q, x, h; (P, r)) : Q = P^x \wedge h = \mathsf{H}(P, r)\}$$

where $P, Q, x \in \mathbb{Z}^*_n$ and $r \in \{0,1\}^\lambda$.

Sigma Argument of Knowledge Π^0_{rsa}

Setup. Run internal SNARK setup $\mathsf{crs} \leftarrow \Pi_{\mathsf{int}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^4_{\mathsf{int}})$

Public input(crs, Q, x, h); **Private input**(P, r);

Prover: $K \xleftarrow{\$} \mathbb{Z}_n^*; r_k \xleftarrow{\$} \{0,1\}^\lambda$; $h_k = \mathsf{H}(K, r_k); A = K^x \pmod n$; send h_k, A.

Verifier: $c \xleftarrow{\$} \{0,1\}$; send c.

Prover:

Case $c = 0 : Z = K \pmod n$; send Z, r_k.

Case $c = 1 : Z = K \circ P \pmod n$;

Run Π_{int} for $\mathcal{R}^4_{\mathsf{int}} : \pi_{\mathsf{int}} \leftarrow \Pi_{\mathsf{int}}.\mathsf{P}(\mathsf{crs}, h, h_k, Z; (P, K, r, r_k))$; send Z, π_{int}.

Verifier:

Case $c = 0$: check $Z^x \overset{?}{=} A \pmod n) \wedge h_k \overset{?}{=} \mathsf{H}(K, r_k)$

Case $c = 1$: check $Z^x \overset{?}{=} A \circ Q \pmod n) \wedge \Pi_{\mathsf{int}}.\mathsf{V}(\mathsf{crs}, \pi_{\mathsf{int}}) \overset{?}{=} 1$

Fig. A5: Π^0_{rsa} for $\mathcal{R}^0_{\mathsf{rsa}} := \{(Q, x, h; (P, r)) : Q = P^x \wedge h = \mathsf{H}(P, r)\}$

Appendix B Proof of Theorem 3

Proof Completeness is obvious.

Knowledge soundness. The knowledge soundness of Π^1_{dl} is derived from the SNARK proof π_{int}, which ensures that the prover who can produce acceptable transcripts authentically knows the witness x consistently in both h and z, and also knows the witness Q consistently in both h and T.

For any polynomial time prover P^* and any crs, P^* choose an instance (P, h), and the extractor Ext behaves as follows:

1. Call P^* to obtain the first message h_k.
2. Query P^* with two challenges $c = 0$ and $c = 1$, if P^* output two acceptable transcripts $(h_k, 0, z_0, r^0_k)$ and $(h_k, 1, z_1, \pi_{\mathsf{int}})$, then continue; otherwise, return to the step 1.
3. Call the SNARK extractor $\Pi_{\mathsf{int}}.\mathsf{Ext}$ for the instance $(h, h_k, 1, z_1, P^{z_1})$ and acceptable transcript π_{int}, then $\Pi_{\mathsf{int}}.\mathsf{Ext}$ output a witness (Q, x, k, A, r, r^1_k).[15]
4. Check if $(z_0, P^{z_0}, r^0_k) = (k, A, r^1_k)$, then output (Q, x, r). Otherwise, abort.

Assume that P^* can choose a first message for instance (P, h) with probability $p \geq \frac{1}{2} + \epsilon$ for a non-negligible ϵ, enabling it to respond correctly to both $c = 0$ and $c = 1$. Then, P^* can succeed in step 2 within expected polynomial time.

According to the knowledge soundness of SNARK Π_{int}, the extractor successfully output a witness (Q, x, k, A, r, r^1_k) in step 3 with a probability at least $1 - \mathsf{negl}_0$. And (Q, x, k, A, r, r^1_k) satisfies $h = \mathsf{H}(Q, x, r)$, $h_k = \mathsf{H}(A, k, r^1_k)$, $z_1 = k + x \pmod p$ and $P^{z_1} = A \circ Q$.

In steps 4, the extractor accepts the consistency of witness with probability $1 - \mathsf{negl}_1$. This is due to the collision resistance of H, which ensures that a fixed h_k has the same preimage

[15]Although the non-native field element $x \in \mathbb{F}_p$ is represented as a native field element (or vector) in SNARK, residing in a different field, it uniquely corresponds to $x \in \mathbb{F}_p$.

in different cases. Then, we have $P^{k+x} = A \circ Q$, $A = P^k$ and $h = \mathsf{H}(Q, x, r)$, which implies that (Q, x, r) is the witness for (P, h). Consequently, the Ext successfully extracts the witness (Q, x, r), except with a negligible error.

Computational SHVZK. There exists a polynomial time simulator Sim that for any instance (P, h) in $\mathcal{R}^1_{\mathsf{dl}}$ and given a random challenge $c \in \{0,1\}$ outputs a transcript (h^*_k, c, z^*, π^*), indistinguishable from the honest transcript (h_k, c, z, π). The Sim behaves as follows:[16]

1. Randomly choose $k^* \xleftarrow{\$} \mathbb{F}_p$ and $r_k \xleftarrow{\$} \{0,1\}^\lambda$.
2. Compute $A^* = P^{k^*}$ and the first message $h^*_k = \mathsf{H}(A^*, k^*, r_k)$.
3. Depending on the challenge c, there are two cases:

   Case 1. If $c = 0$, let $z^* = k^*$ and $\pi^* = r_k$.

   Case 2. If $c = 1$, randomly choose $z^* \xleftarrow{\$} \mathbb{F}_p$ and let $T^* = P^{z^*}$. Then, call $\Pi_{\mathsf{int}}.\mathsf{Sim}$ with (h, h^*_k, z^*, T^*) to obtain π^*.
4. Output (h^*_k, c, z^*, π^*).

Obviously, when $c = 0$, the transcript produced by Sim is identically distributed to that of an honest party. Now we consider the case when $c = 1$ via a hybrid argument:

$$\mathcal{H}_0 : \left\{ \begin{pmatrix} P, h, h_k, \\ c, z, \pi \end{pmatrix} \middle| (h_k, c, z, \pi) \leftarrow \langle \mathsf{P}(Q, x, r), \mathsf{V} \rangle (P, h) \right\}$$

$$\mathcal{H}_1 : \left\{ \begin{pmatrix} P, h, h_k, \\ c, z, \pi' \end{pmatrix} \middle| \begin{array}{c} k \xleftarrow{\$} \mathbb{F}_p; r_k \xleftarrow{\$} \{0,1\}^\lambda; A = P^k; \\ h_k = \mathsf{H}(A, k, r_k); z = k + x (\text{mod } p); T = A \circ Q; \\ \pi' \leftarrow \Pi_{\mathsf{int}}.\mathsf{Sim}(h, h_k, z, T) \end{array} \right\}$$

$\mathcal{H}_0$ represents the honest transcript, while $\mathcal{H}_1$ only substitutes the honest prover's proof with π', which is generated by $\Pi_{\mathsf{int}}.\mathsf{Sim}$. Therefore, the indistinguishability is derived from the zero-knowledge property of Π_{int}.

$$\mathcal{H}_2 : \left\{ \begin{pmatrix} P, h, h^*_k, \\ c, z, \pi'' \end{pmatrix} \middle| \begin{array}{c} k, k^* \xleftarrow{\$} \mathbb{F}_p; r_k \xleftarrow{\$} \{0,1\}^\lambda; A^* = P^{k^*}; \\ h^*_k = \mathsf{H}(A^*, k^*, r_k); z = k + x (\text{mod } p); T = P^z; \\ \pi'' \leftarrow \Pi_{\mathsf{int}}.\mathsf{Sim}(h, h^*_k, z, T) \end{array} \right\}$$

$\mathcal{H}_2$ uses a new random k^* to compute h^*_k, and $\Pi_{\mathsf{int}}.\mathsf{Sim}$ uses h^*_k to simulate. At first, the SNARK instance (h, h^*_k, z, T) is indistinguishable from (h, h_k, z, T) in $\mathcal{H}_1$, due to the computational hiding property of H. Therefore, the output π' and π'' produced by the polynomial time algorithm $\Pi_{\mathsf{int}}.\mathsf{Sim}$ with these instances is also indistinguishable.

$$\mathcal{H}_3 : \left\{ \begin{pmatrix} P, h, h^*_k, \\ c, z^*, \pi^* \end{pmatrix} \middle| \begin{array}{c} k^*, z^* \xleftarrow{\$} \mathbb{F}_p; r_k \xleftarrow{\$} \{0,1\}^\lambda; \\ A^* = P^{k^*}; h^*_k = \mathsf{H}(A^*, k^*, r_k); T^* = P^{z^*}; \\ \pi^* \leftarrow \Pi_{\mathsf{int}}.\mathsf{Sim}(h, h^*_k, z^*, T^*) \end{array} \right\}$$

$\mathcal{H}_3$ represents the transcript computed by the simulator Sim. Similar to Sigma protocol, due to that the distributions of $\{(P, c, z)\}$ in $\mathcal{H}_2$ and $\mathcal{H}_3$ are identical, the distributions $\mathcal{H}_2$ and $\mathcal{H}_3$ are actually the same, which concludes the proof. □

[16]We omit the process of generating CRS during the SNARK setup. In the CRS model, the simulator is allowed to modify the CRS.