Generic, Fast and Short Proofs for Composite Statements[⋆]

Zhuo Wu[1,2], Shi Qi[1,2], Xinxuan Zhang[1,2] and Yi Deng[1,2]

[1] 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
`{wuzhuo, qishi, zhangxinxuan, deng}@iie.ac.cn`

Abstract. This work introduces a novel technique to enhance the efficiency of proving composite statements. We present the *Hash-and-Prove* framework to construct zkSNARKs for proving satisfiability of arithmetic circuits with additional *Algebraic Gate*. These algebraic gates serve as building blocks for forming more generalized relations in algebra. Unlike Pedersen-committed *Commit-and-Prove* SNARKs, which suffer from increased proof size and verification overhead when proving composite statements, our solution significantly improves both proof size and verification time while maintaining competitive and practical prover efficiency.

In the application of proof of solvency where we need to prove knowledge of x such that SHA256$(g^x) = y$, our approach achieves a $100\times$ reduction in proof size and a $500\times$ reduction in verification time, along with a $2\times$ speedup in proving time compared to the work of Agrawal et al.(CRYPTO 2018). For proving ECDSA signatures verification, we achieve a proof time of 2.1 seconds, which is a $70\times$ speedup compared to using Groth16, and a proof size of 4.81 kb, which is a $160\times$ reduction compared to Field Agnostic SNARKs(Block et al., CRYPTO 2024).

1 Introduction

Zero-Knowledge Succinct Non-interactive Arguments of Knowledge (zkSNARKs) are extremly short and non-interactive zero-knowledge proofs [GMR85]. In recent years, SNARKs have garnered significant interest, leading to the development of efficient constructions (e.g., [PHGR13,Gro16,WTS$^+$18,BBHR18,MBKM19,XZZ$^+$19], [GWC19,BCR$^+$19,CHM$^+$20,Set20,GLS$^+$23,CBBZ23]). Various SNARKs designed to prove arithmetic circuit satisfiability have made great strides in efficiency and scalability. They are now capable of efficiently proving non-algebraic statements, such as hash functions like SHA256. However, a significant challenge remains in proving algebraic statements, including but not limited to elliptic curve operations and exponentiation of large integers.

Algebraic operations cannot be accommodated by native field arithmetic representations. Thus, when employing SNARKs to prove these statements, we

[⋆] Submitted on October 2, 2024

must emulate non-native arithmetic operations using additions and multiplications within the native field arithmetic circuits, known as the "Non-Native Arithmetic Problem". Practical solutions like xJsnark and approaches based on the Chinese Remainder Theorem [KPS18,JkY23] exist. However, current SNARKs still struggle to efficiently prove algebraic operations, particularly those involving group exponentiation.

An alternative approach for efficiently proving algebraic operations is provided by Sigma protocols, which are well-suited for algebraic statements within a group structure. However, the main limitations of Sigma protocols include increased proof size and verification time. In contrast, SNARKs generally offer much lower overhead in both aspects.

Commit-and-Prove SNARKs. In many real-world scenarios, we need to prove composite statements involving both algebraic and non-algebraic components, such as ECDSA signatures verification, Elgamal encryption/decryption, and proof of solvency [AGM18]. To tackle the challenge of efficiently proving these statements, research has focused on developing *Commit-and-Prove*(CP) SNARKs [Kil89,CLOS02,CGM16,BBB+18,AGM18,BHH+19,CFQ19,CFF+21], [ABC+22,ZCYW23,OKMZ24]. CP-SNARKs prove statements about committed values and have a wide range of applications across various domains, such as verifying the integrity of Machine Learning models, enabling privacy preserving voting systems, and more [BCF+21]. When employing CP-SNARKs to prove composite statements, the algebraic operations are addressed by Sigma protocols under Pedersen commitment, while the arithmetic circuit operations, such as hash functions, are executed within SNARKs. Pedersen commitment can be utilized to prove composite statements for two reasons. Firstly, the additive homomorphism and the capacity to prove modular multiplications allow effective handling of operations within group structures, enhancing prover efficiency in algebraic statements. Secondly, many SNARKs use commitment schemes based on elliptic curves, which can be connected to Pedersen commitment through "glue" proofs.

Efficiency Bottlenecks. Existing solutions employ Sigma protocols under Pedersen commitment, resulting in increased proof size and verification overhead when proving algebraic statements of the form: knowledge of x and g_2 such that $g_1^x = g_2$ (or knowledge of g_1 and g_2), where additional witnesses are included [CGM16,AGM18]. Their proof size and verification time could be thousands of times larger than SNARKs. Another drawback of Pedersen commitment is the necessity of additional "glue" protocols to prove the the consistency of witnesses.[3] These "glue" proofs restrict SNARK flexibility and increase proof size.

The discussion above highlights that SNARKs suitable for proving hash functions are inefficient for proving algebraic statements. While existing Pedersen-

[3] One exception is that in MPC-in-the-head proof systems, Sigma protocols from VSS can be employed to bypass the need for "glue" proofs. However, these SNARKs result in relatively large proof size and linear verification [ZCYW23].

committed CP-SNARKs enable fast proving for algebraic statements, they result in large proof size and increased verification overhead. This motivates the central question we address in this work:

Can we obtain both fast and short proofs for composite statements?

1.1 Our Results

In this work, we provide a positive answer to the above question by completely discarding the Pedersen commitment. Unlike embedding all algebraic operations within arithmetic circuits for SNARK proofs, which significantly reduces prover efficiency, or delegating these operations to Sigma protocols under Pedersen commitment to achieve faster proving at the cost of increased proof size and verification overhead, we introduce a new method for building proofs for composite statements that strikes a better trade-off. Our construction achieves faster proving than SNARKs, along with smaller proof size and faster verification compared to Sigma protocols under Pedersen commitment. Moreover, for elliptic curve operations, our approach outperforms all these methods in every aspect.

Hash-and-Prove Framework and Sigma Protocols. We start with the definition of *Algebraic Gate*, which performs group exponentiation over a given cyclic group. Algebraic gates serve as fundamental components for constructing various algebraic statements. Then we develop the *Hash-and-Prove*(HP) framework for proving the satisfiability of arithmetic circuits with algebraic gates. This framework eliminates the need for additional "glue" proofs.

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

Implementation and Applications. Our proposed HP framework can be utilized to prove a wide range of exponentiation operations in cyclic groups, including elliptic curve groups, large prime-order groups $\mathbb{Z}_p$ and RSA groups $\mathbb{Z}_n^*$ where $n = pq$.

We provide the experimental results corresponding to each of these different cases. In the application of proof of solvency within blockchains, our approach achieves a 100$\times$ reduction in proof size, a 500 $\times$ reduction in verification time, and a 2$\times$ speedup in proving time compared to the work of Agrawal et al. [AGM18]. In the application of proving knowledge of signatures verification: for ECDSA, we achieve a 160$\times$ reduction in proof size compared to the work of Block et al. [BFK$^+$24], and a 70$\times$ speedup in proving time compared to solutions using Groth16 as the underlying prove system [Sun24]; for DSA, we achieve a 10$\times$ reduction in proof size and a 20$\times$ reduction in verification time compared to the work of Chase et al. [CGM16]; for RSA signature, we provide a more generic solution without sacrificing efficiency compared to the work of Agrawal et al. [AGM18].

1.2 Techniques

We explore how to develop a general approach to solve the problem of proving algebraic statements and integrating with existing SNARK systems to efficiently prove composite statements. While specific cases exist, such as utilizing cycle of elliptic curves [T+23], and straightforward solutions in bilinear groups by leveraging pairings [GS08], these specialized systems cannot address general scenarios. In this work, we propose a more general approach that covers a wide range of algebraic statements while achieving high efficiency.

In public key cryptography, exponentiations within a group are core operations. While a single group multiplication can be efficiently verified within SNARKs, proving the correctness of group exponentiation involves several hundred to a few thousand multiplications. We introduce algebraic gates, 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 algebraic 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 finite cyclic groups. Many cryptographic systems rely on the hardness of certain computational problems, most notably the Discrete Logarithm Problem (DLP). 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 a prime-order group derived from a finite field modulo a prime integer, or a prime-order elliptic curve group. We describe the following relations $\{\mathcal{R}_{\mathsf{dl}}^i\}_{i\in[4]}$ to target any valuable algebraic operations within the DLP setting, where $g_1, g_2 \in \mathbb{G}_p$ and $x \in \mathbb{F}_p$.

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

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

These four relations are sufficient to encompass valuable algebraic statements within the DLP setting. Note that 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 $\mathcal{R}_{\mathsf{dl}}^1$. We construct specially designed Sigma protocols for $\{\mathcal{R}_{\mathsf{dl}}^i\}_{i\in[4]}$, denoted as $\{\Pi_{\mathsf{dl}}^i\}_{i\in[4]}$. These protocols are then used to build proofs for other cryptographic schemes based on DLP, 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 algebraic gates targeting the RSA setting: $\mathbb{G} = \mathbb{Z}_n^*$, $I = \{e, d\}$. In this case, we consider relations $\{\mathcal{R}_{\mathsf{rsa}}^i\}_{i\in[2]}$, where $g_1, g_2 \in \mathbb{Z}_n^*$ and $x \in \{e, d\}$.

$$\mathcal{R}_{\mathsf{rsa}}^i := \{\mathbf{st}_i : g_1^x = g_2\}$$

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

Similarly, We construct specially designed Sigma protocols 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 and Algebraic Circuit. Our starting point is extending arithmetic circuits with addition and multiplication gates over a field $\mathbb{F}_q$ by incorporating algebraic gates.[4] To prove the satisfiability of such a circuit $\mathcal{C}_{\mathsf{alg}}$ with three types of gates, we first need to decompose it using a circuit compiler. An example is shown in Figure 1.

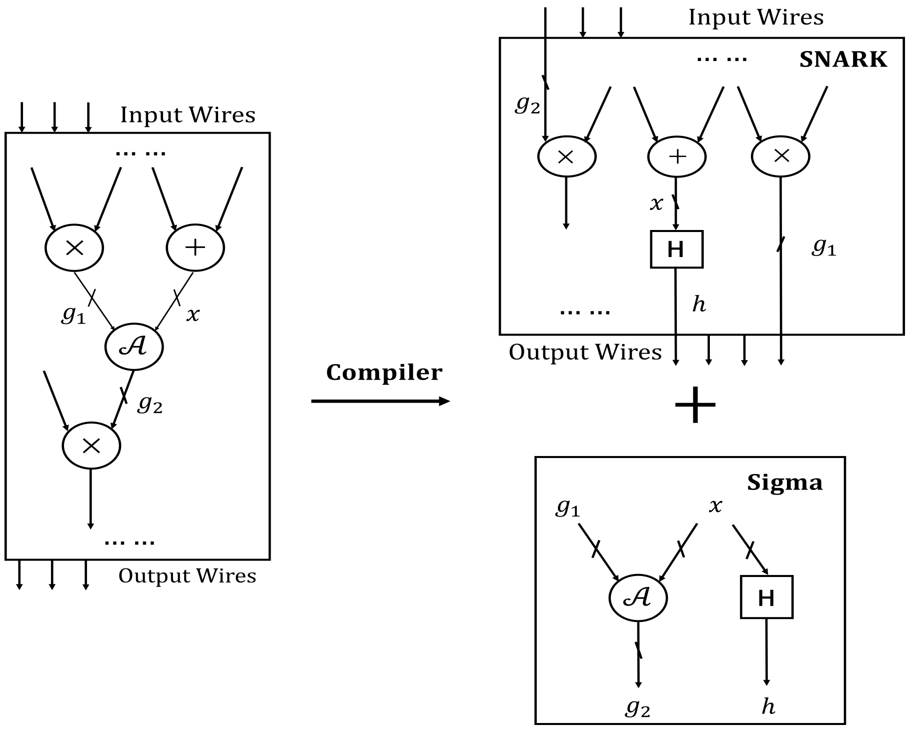


Fig. 1: An example of algebraic gate $\mathcal{A} : g_1^x = g_2$ with private x. H denote the hash function circuit over $\mathbb{F}_q$. Similarly, if g_1 or g_2 is also private, it can be handled in the same manner as x.

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

On input $\mathcal{C}_{\mathsf{alg}}$ and a hash function H that is randomly chosen from a hash family, the compiler $\mathsf{Compil}(\mathsf{H}, \mathcal{C}_{\mathsf{alg}})$ outputs a standard arithmetic circuit $\mathcal{C}_{\mathsf{std}}$ comprising addition and multiplication gates over $\mathbb{F}_q$, along with a "Algebraic with Hash" relation $\mathcal{C}_{\mathsf{ah}}$: $g_1^x = g_2 \wedge \mathsf{H}(\mathbf{w}, r)$, where r is a fixed length random number, $\mathbf{w}$ are the witnesses consisting of one to three variables among x, g_1 and g_2. Any of these three that are public variables and not used as inputs or outputs in other addition or multiplication gates are removed from $\mathcal{C}_{\mathsf{std}}$. By utilizing a common SNARK for $\mathcal{C}_{\mathsf{std}}$ and our propsed Sigma protocols for $\mathcal{C}_{\mathsf{ah}}$, we can successfully prove the satisfiability of $\mathcal{C}_{\mathsf{alg}}$. In most cases, H refers to a zk-friendly hash function like Poseidon [GKR+21]. This provides us with the following properties that we will exploit in the subsequent protocol construction:

– It is very cheap to prove in SNARKs. Thus unlike the Pedersen commitment, we can achieve linkage without an additional "glue" proof.
– It offers collision resistance and computational hiding properties, as detailed in Sections 2.

Proving Algebraic with Hash Relation. Our Sigma protocol 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.

We first address how to prove $g_1^x = g_2$, where x is private and g_1, g_2 are public within the DLP setting. One solution is to directly use SNARKs to prove both $z(x)$ and $\mathsf{H}(x, r)$ during the generation of the response in the Schnorr protocol [KMN23,OKMZ24]. However, if either g_1 or g_2 serves as a witness, it is difficult to generate such proofs. Previous approaches have leveraged the additive homomorphic properties of Pedersen commitment, along with its capacity to prove modular multiplications, to establish proofs involving additional witnesses like g_1 or g_2. As a substitute, we use hash functions instead of Pedersen commitment.

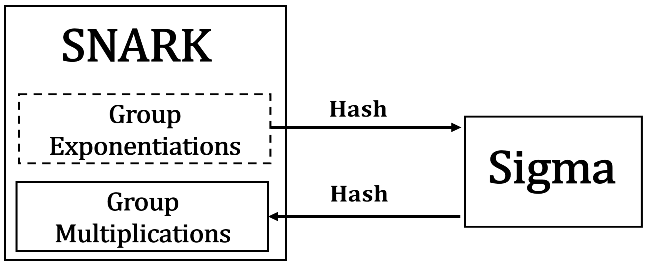


Fig. 2: "Ping-Pong" Alternating of SNARK and Sigma.

The difficulty resides in the inherent lack of homomorphic properties within hash functions, which makes the Sigma verification difficult to execute. To address this, we propose a "Ping-Pong" alternating approach, as illustrated in Figure 2. From a high-level perspective, the proving process begins with group exponentiations in a SNARK, transitions to a Sigma protocol, and then returns to a SNARK for group multiplications and several lightweight computations, such as zk-friendly hash. This process significantly reduces the circuit size, specifically the number of group multiplications within the circuit.

An important observation is that if the challenge space is small enough (e.g., $c \in \{0,1\}$), proving the relation $\{\mathsf{V}(\mathbf{s}, a, c, z) = 1 \wedge z(\mathbf{w}) \wedge \mathsf{H}(\mathbf{w}, r)\}$ directly using a SNARK has become efficient, with minimal proof size owing to the succinctness of SNARKs. For example, in protocols like Schnorr [Sch91] and Guillou-Quisquater [GQ88], which prove the knowledge of 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 such as $\{\mathcal{R}^i_{\mathsf{dl}}\}_{i\in[4]}$ and $\{\mathcal{R}^i_{\mathsf{rsa}}\}_{i\in[2]}$. By repeatedly proving the same statement for multiple challenges c, we can reduce the soundness error (and knowledge error).

On the one hand, this approach enhances prover efficiency by significantly more than tenfold compared to embedding the entire group exponentiation operation into the SNARK circuit. On the other hand, 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:

– **Minimizing the Circuit Size.** We can adjust the size of challenge space and the number of parallel repetitions to reduce circuit size that needs to be proved by SNARKs, ensuring the soundness error remains sufficiently small while minimizing the circuit size.
– **Proof Aggregation for Uniform Circuits.** Since the parallel repetitions involve proving different inputs of the same circuit, we can leverage techniques such as incrementally verifiable computation (IVC), folding schemes, or proof batching to significantly accelerate the prover efficiency for uniform circuits.

1.3 Related Work

Constructing SNARKs for composite statements has garnered significant attention, beginning with the works of Chase et al. [CGM16] and Agrawal et al. [AGM18]. These works respectively introduced proof systems based on garbled circuits and proof systems based on linear-PCP combined with Sigma protocols to prove statements such as proof of solvency, DSA, ECDSA and RSA signatures. While these approaches significantly improved prover efficiency and their innovative designs opened new avenues for proving composite statements, the corresponding solutions incurred substantial overhead in proof size and verification time when proving algebraic statements using sigma protocols under Pedersen commitment, due to involving numerous group elements.

Backes et al. [BHH+19] constructed CP-SNARKs that combine MPC-in-the-head techniques with Sigma protocols, offering a non-interactive approach without requiring a trusted setup. Campanelli et al. [CFQ19] proposed a framework for constructing CP-SNARKs that prove composite statements, where various proof systems can be integrated with Pedersen commitment. They also introduced several proving gadgets to enhance efficiency, providing a versatile and impactful solution for integrating diverse proof systems.

Aranha et al. [ABC+22] further developed a framework that extends universal setup SNARKs to support *Commit-and-Prove* mechanisms, leveraging compressed-Σ techniques to reduce the size of "glue" proofs to logarithmic levels. Zhang et al. [ZCYW23] constructed a novel Sigma protocol based on verifiable secret sharing (VSS), connecting it with MPC-in-the-head proof systems, and eliminating the need for a "glue" protocol to maintain witness consistency between the SNARKs and Sigma protocols. More recently, Orrù et al. [OKMZ24] utilized hiding-compatible functions to enable the integration of Pedersen commitment into any SNARKs, thus enabling SNARKs not utilizing elliptic curves such as code-based SNARKs which utilize Merkle Hash, to be integrated with Pedersen commitment.

Most works in this line of research focus on expanding SNARKs that can be connected to commitments like Pedersen commitment, Merkle Hash Tree, and others. Fewer works concentrate on providing concrete constructions and implementations for proving algebraic statements using Sigma protocols, leaving significant room for further research in this area.

Existing approaches for proving algebraic statements using Sigma protocols rely on Pedersen commitments and are connected to specific SNARKs. These Sigma protocols significantly enhance prover efficiency, enabling proofs for algebraic operations in composite statements. However, compared to standard SNARKs, they suffer from increased proof size and slower verifier performance. Specifically, when the group order in Pedersen commitments is large, the prover efficiency can become a limiting factor. In this work, we eliminate the reliance on Pedersen commitments and construct Sigma protocols using hash functions, resulting in improved practicality and efficiency for real-world applications.

2 Preliminaries

In this section, we first provide the definitions essential for understanding this paper. Next, we propose the hiding assumption that the hash function needs, which plays a crucial role in our constructions. Finally, we discuss the recursion of proofs constructed in the random oracle model, and offer a different perspective that enhances confidence in the security of recursion instantiated with concrete hash functions.

2.1 Zero-Knowledge Proofs

Definition 1 (Sigma protocol [Dam]). *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 probability 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 [DG23]. 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 with knowledge error* κ *for relation* $\mathcal{R}$*, if it satisfies completeness, SHVZK, and the following property:*

- **Knowledge soundness:** *An interactive protocol is knowledge sound with knowledge error* κ*, if for any polynomial time (potentially malicious) prover* P^**, there exists a polynomial time extractor* Ext *such that for any instance* x*, given full access to* P^**'s state and random coins, it can output a witness with probability:*

$$\Pr[(x, w) \in \mathcal{R} | w \leftarrow \mathsf{Ext}^{\mathsf{P}^*}(x)] \geq \Pr[\langle \mathsf{P}^*, \mathsf{V} \rangle(x) = 1] - \kappa(|x|) - \mathsf{negl}(|x|)$$

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:*

– $\mathsf{Setup}(1^\lambda)$ *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 argument* 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 (with negligible knowledge error), if for any polynomial time (potentially malicious) prover* P^*, *there exists a polynomial time extractor* Ext *such that, given full access to* P^**'s state and random coins, it can output a witness with probability:*

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

- **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 following probability is negligible:*

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

- **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,

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

- **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. In this context, we use H to represent the instantiated hiding hash function H_η. Under the random oracle model, the hiding property is evident because $\mathsf{H}(x, r)$ is a uniform distribution. Therefore, if a collision-resistant hash is believed to be secure to instantiate a random oracle in the real world, then it should be believed as a hiding hash function.

Furthermore, we note that we cannot directly analysis the security of our constructions in the random oracle model due to the necessity of proving such a hash function. Therefore, we need to assume the existence of hiding hash functions. In fact, it is a common issue appearing in the recursion of SNARKs. We will discuss such issue further in the next subsection and provide confidence in the security of corresponding constructions.

2.3 On Recursive Proof in Random Oracle Model

Here we primarily discuss recursive SNARKs, as they are more common. The obtained observation also applies similarly to above case of hiding function, which is more simple.

To be detailed, the constructions of recursive SNARKs [COS20] (furthermore, IVC [Val08] and proof-carrying data (PCD) [BCMS20,BCL+21]), might face the significant theoretical challenge that some times the hash function is treated as a random oracle and sometimes it is described as a small circuit. It is somewhat unreasonable, in fact, Chiesa and Liu [CL20] demonstrated that the PCP theorem does not hold for random oracle, and Hall-Andersen and Nielsen [HAN23] proved that constructing zero-knowledge IVC from random oracle model is impossible.

To address this issue, several studies [CT10,CCS22,CCG+23] have attempted to endow the random oracle with additional structure, enabling the correctness of queries to be proven using these structures. This approach often leads to more complex constructions or security proofs. Furthermore, when instantiated with concrete hash functions, the security of the resulting schemes still remain heuristic.

Here, we observe a different perspective on this problem. Suppose that recursive schemes instantiated with concrete hash functions are flawed or vulnerable to certain attacks. It is highly likely that the original schemes, when instantiated with the same concrete hash functions, are already vulnerable to the same attacks.

This observation has appeared implicitly in many constructions [COS20], [BCMS20,BCL+21] which assume the existence of concrete hash functions, where replacing the random oracle with the hash function leads to a secure zkSNARK or zk-accumulation scheme in the standard model. Now, without the random oracle, they show that the zero-knowledge and knowledge soundness properties are still preserved in the recursive setting.

We formalize our observation and extend existing results. Recall that, when instantiated with concrete hash functions, it may be somewhat unreasonable to assume that the resulting schemes maintain the same level of security (such as standard zero-knowledge and knowledge soundness), due to the absence of programmable and observable properties. We demonstrate that, as long as one assumes the underlying scheme still satisfies some weaker security properties, e.g., knowledge soundness with extractor (might not be efficient) and T-time (T might not be a polynomial) simulatable zero-knowledge, then the recursive schemes should also be considered to satisfy these properties.

In this context, we focus on a simple situation, $\Pi^O := \mathit{\Pi}_1^O(\mathit{\Pi}_2^O(\mathit{\Pi}_3^O(\cdots(\mathit{\Pi}_n^O))))$ (e.g., IVC and path PCD).

Theorem 1. *(inherent from [COS20,BCMS20,BCL+21]) Let $\{\mathit{\Pi}_i^O\}_{i\in[n]}$ be a series of proof systems, and $\Pi^O := \mathit{\Pi}_1^O(\mathit{\Pi}_2^O(\mathit{\Pi}_3^O(\cdots(\mathit{\Pi}_n^O))))$ be the recursion of these proof systems. Specifically, for each $\mathit{\Pi}_i$, it proves $x_i \in L_i$ defined as that "$(x_i; x_{i+1}, w_i) \in R_{L_i'} \wedge$ there exists a valid proof π_{i+1} for $x_{i+1} \in L_{i+1}$" for some L_i' (with the exception of $\mathit{\Pi}_n$, which only proves that $(x_n; w_n) \in R_{L_n'}$). Then, when instantiated with concrete hash functions h, we have that:*

- *Recursion of Knowledge Soundness. Suppose that for each i, $\mathit{\Pi}_i^h$ satisfies knowledge soundness with witness extractors of size f_i (i.e., for any T-size P^*, the size of witness extractor is $f_i(T)$, which may not be efficient). Then, Π^h satisfies knowledge soundness with f-size extractor, where $f = f_1 \circ f_2 \circ \cdots \circ f_n$.*
- *Recursion of Zero-Knowledge. Suppose that for each i, $\mathit{\Pi}_i^h$ satisfies T-time simulatable zero-knowledge, where T may not be a polynomial. Then, Π^h also satisfies T-time simulatable zero-knowledge.*

Remark 1. For the recursion of knowledge soundness, if the sizes of both adversary and the extractor are polynomial, the result corresponds to the cases discussed in [Val08,BCCT13].

Remark 2. For the recursion of T-time simulatable zero-knowledge, it is sufficient for only $\mathit{\Pi}_n^h$ to satisfy this property. Additionally, other weak variants of zero-knowledge can also be considered, such as distributional distinguisher-dependent zero-knowledge (for general PCD, one additionally needs to assume that the local inputs of all nodes are independent), witness indistinguishable or witness hiding. All of these properties can be preserved in recursion.

3 Hash-and-Prove SNARKs

In this section, we first define a *Arithmetic with Algebraic Circuit Satisfiability* relation $\mathcal{R}_{\mathsf{alg}}$ for the arithmetic C-SAT that includes algebraic gates. Then, we provide a circuit compiler that decomposes the relation circuit $\mathcal{R}_{\mathsf{alg}}$ into a standard arithmetic C-SAT relation $\mathcal{R}_{\mathsf{std}}$ and some algebraic relations $\mathcal{R}_{\mathsf{ah}}$. Finally, we give a generalized description of Hash-and-Prove SNARKs.

3.1 Arithmetic with Algebraic Circuit Satisfiability

Notation. Let $(\mathbb{G}_p, \circ)$ be a cyclic group, for a group element $P \in \mathbb{G}_p$, and an integer $x \in \mathbb{F}_p$, denote the scalar multiplication over $\mathbb{G}_p$ as P^x, which means multiplying x group elements P.

We extend the arithmetic C-SAT problem over a field $\mathbb{F}_q$ by allowing some non-native operations over a cyclic group $\mathbb{G}_p$ without representing them as arithmetic over $\mathbb{F}_q$. In this circuit, there are three types of gates: addition and multiplication over $\mathbb{F}_q$, and algebraic gates over $\mathbb{G}_p$ which are defined as follows:

Definition 5 (Algebraic gate). *Let $(\mathbb{G}_p, \circ)$ be a cyclic group, and $I \subseteq \mathbb{Z}$ be a finite set.*[5] *An algebraic gate $\mathcal{A}$ over $\mathbb{G}_p$ is parameterized by two inputs, a group element $P \in \mathbb{G}_p$ and a scalar $x \in I$, and an output $Q \in \mathbb{G}_p$, satisfying $P^x = Q$.*

Algebraic gates can be used to model exponentiation operations in various cryptographic protocols, forming a core component of circuits that involve group operations. Since cryptographic algorithms are typically defined over a single cyclic group, we will provide an arithmetic circuit with operations over a single cyclic group. However, this can naturally be extended to circuits that include operations over multiple cyclic groups.

Definition 6 (Arithmetic with algebraic circuit satisfiability). *Let $\mathbb{F}_q$ be an arithmetic field and $\mathbb{G}_p$ be a cyclic group, an arithmetic circuit with algebraic gates $\{\mathcal{A}_k\}$ over $\mathbb{G}_p$, denoted as $\mathcal{R}_{\mathsf{alg}}$, operates on wires $\omega := (\mathbf{s}; \mathbf{w})$ and satisfies a relation of the following form:*

$$\mathcal{R}_{\mathsf{alg}} := \left\{ (\mathbf{s}; \mathbf{w}) : \begin{array}{l} \forall i \in I_{\mathsf{add}}, \quad \omega_{\mathsf{add}_0(i)} = \omega_{\mathsf{add}_1(i)} + \omega_{\mathsf{add}_2(i)} \pmod q \wedge \\ \forall j \in I_{\mathsf{mul}}, \quad \omega_{\mathsf{mul}_0(j)} = \omega_{\mathsf{mul}_1(j)} \cdot \omega_{\mathsf{mul}_2(j)} \pmod q \wedge \\ \forall k \in I_{\mathsf{alg}}, \quad P_k^{x_k} = Q_k \quad \textit{where:} \\ \quad \hat{Q}_k := \left(\omega_{\mathsf{ah}_{0,1}(k)}, \omega_{\mathsf{ah}_{0,2}(k)}, \dots, \omega_{\mathsf{ah}_{0,\xi}(k)}\right), \\ \quad \hat{P}_k := \left(\omega_{\mathsf{ah}_{1,1}(k)}, \omega_{\mathsf{ah}_{1,2}(k)}, \dots, \omega_{\mathsf{ah}_{1,\xi}(k)}\right), \\ \quad \hat{x}_k := \left(\omega_{\mathsf{ah}_{2,1}(k)}, \omega_{\mathsf{ah}_{2,2}(k)}, \dots, \omega_{\mathsf{ah}_{2,\zeta}(k)}\right), \\ \quad Q_k := \mathcal{L}(\hat{Q}_k), \; P_k := \mathcal{L}(\hat{P}_k), \; x_k := \mathcal{L}'(\hat{x}_k) \end{array} \right\}$$

These symbols mean:

- $\omega := (\mathbf{s}; \mathbf{w})$ *denotes the vector of all wires in the circuit, where* $\mathbf{s}$ *represents public wires,* $\mathbf{w}$ *represents private wires and* ω_i *denotes the wire at index* i.
- $I_{\mathsf{add}}, I_{\mathsf{mul}}, I_{\mathsf{alg}} \subseteq \mathbb{Z}^+$ *are index sets for addition gates, multiplication gates, and algebraic gates, respectively.*
- $\mathsf{add}_0(i), \mathsf{add}_1(i), \mathsf{add}_2(i)$ *denote indices for the output and input wires of the* i*-th addition gate. Similarly for multiplication gates.*
- $P_k^{x_k} = Q_k$ *is the algebraic relation for* $\mathcal{A}_k$*, where* $P_k, Q_k \in \mathbb{G}_p$ *and* $x_k \in I$.
- ξ *and* ζ *are the numbers of* $\mathbb{F}_q$ *variables required to represent each group or scalar variable, respectively.*

[5] The scalar variable used in operations over cyclic group $\mathbb{G}_p$ is typically an $\mathbb{F}_p$ element. However, there are exceptions, such as RSA accumulator schemes, where the scalar belongs to a larger set of integers.

– *$\hat{Q}_k, \hat{P}_k \in \mathbb{F}_q^{\xi}$ and $\hat{x}_k \in \mathbb{F}_q^{\zeta}$ are vectors over $\mathbb{F}_q$ that uniquely represent the original variables Q_k, P_k and x_k, respectively.*
– *$\mathcal{L}, \mathcal{L}'$ are the linear mapping that reconstructs original variables from their representation of $\mathbb{F}_q$ elements.*
– $\mathsf{ah}_{0,1\sim\xi}(k), \mathsf{ah}_{1,1\sim\xi}(k), \mathsf{ah}_{2,1\sim\zeta}(k)$ *denote indices for the output and input wires of $\mathcal{A}_k$, which correspond to $\hat{Q}_k, \hat{P}_k$ and $\hat{x}_k$, respectively.*

For $\hat{Q}_k$ in the circuit $\mathcal{R}_{\mathsf{alg}}$, we restrict the wires in $\hat{Q}_k$ to be either entirely composed of the public wires in $\mathbf{s}$ or entirely composed of the private wires in $\mathbf{w}$. This distinction indicates whether the group element Q_k is public or private in $\mathcal{A}_k$. The same applies to $\hat{P}_k$ and $\hat{x}_k$. Furthermore, to ensure that $\mathcal{L}$ is reversible, we assume that the value of each wire in $\mathcal{A}_k$ lies within a specific range.[6]

The standard arithmetic C-SAT relation and the relation $\mathcal{R}_{\mathsf{alg}}$ can be mutually reduced to each other, in particular, $\mathcal{R}_{\mathsf{alg}}$ is equivalent to the standard arithmetic C-SAT when there are no algebraic gates in the circuit, thus $\mathcal{R}_{\mathsf{alg}}$ is also an NP-complete relation. We can encompass nearly all mainstream constraint systems such as R1CS, PLONK, AIR, QAP, and layered arithmetic circuits into relation $\mathcal{R}_{\mathsf{alg}}$ by incorporating these constraint systems along with algebraic gates. Consequently, our methods can be applied to any proof system based on C-SAT.

3.2 Decompose Circuit

Given a arithmetic with algebraic circuit $\mathcal{R}_{\mathsf{alg}}(\mathbf{s};\mathbf{w})$[7], we demonstrate how to apply a circuit compiler to decompose it into two relations. If both relations are satisfied, this establishes that $\mathcal{R}_{\mathsf{alg}}(\mathbf{s};\mathbf{w})$ is satisfied.

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

1. Leave the addition gates and multiplication gates over $\mathbb{F}_q$ unchanged.
2. For every $k \in I_{\mathsf{alg}}$, 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 $\widetilde{\mathbf{s}}_k \leftarrow \widetilde{\mathbf{s}}_k \cup \{P_k\}$, and if $\hat{P}_k \not\subseteq \{\omega_{\mathsf{add}(I_{\mathsf{add}})} \cup \omega_{\mathsf{mul}(I_{\mathsf{mul}})}\}$ then remove $\omega \leftarrow \omega \setminus \hat{P}_k$.[8]
   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 $\widetilde{\mathbf{s}}_k \leftarrow \widetilde{\mathbf{s}}_k \cup \{Q_k\}$, and if $\hat{Q}_k \not\subseteq \{\omega_{\mathsf{add}(I_{\mathsf{add}})} \cup \omega_{\mathsf{mul}(I_{\mathsf{mul}})}\}$ then remove $\omega \leftarrow \omega \setminus \hat{Q}_k$.

[6] An example in practice is that we represent a Secp256k1 element of 256-bits using 16 variables of 16-bits each in the Goldilocks field $\mathbb{F}_q$ where $q = 2^{64} - 2^{32} + 1$. Therefore, we need a range proof to ensure that these $\mathbb{F}_q$ variables are less than $2^{16} - 1$.

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

[8] 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.

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 $\widetilde{\mathbf{s}}_k \leftarrow \widetilde{\mathbf{s}}_k \cup \{x_k\}$, and if $\hat{x}_k \not\subseteq \{\omega_{\mathsf{add}(I_{\mathsf{add}})} \cup \omega_{\mathsf{mul}(I_{\mathsf{mul}})}\}$ then remove $\omega \leftarrow \omega \setminus \hat{x}_k$.

3. For every $k \in I_{\mathsf{alg}}$, 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 a arithmetic circuit $\mathsf{H}^k_{\mathsf{std}}$ over $\mathbb{F}_q$ with wires $(h_k; \mathbf{w}^k_{\mathsf{H}})$.[9] 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 $\mathbf{w} \leftarrow \mathbf{w} \cup \mathbf{w}^k_{\mathsf{H}}$.
4. Output an arithmetic with hash circuit $\mathcal{R}_{\mathsf{std}}(\overline{\mathbf{s}}; \overline{\mathbf{w}})$[10], and k algebraic with hash relations $\mathcal{R}^k_{\mathsf{ah}}(\widetilde{\mathbf{s}}_k; (\widetilde{\mathbf{w}}_k, r_k))$, 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 addition and multiplication relationship in $\mathcal{R}_{\mathsf{alg}}$ and replaces the $\{\mathcal{A}_k\}_{I_{\mathsf{alg}}}$ relations with the relations of arithmetic circuit $\{\mathsf{H}^k_{\mathsf{std}}\}_{I_{\mathsf{alg}}}$ over $\mathbb{F}_q$ as follows:

$$\mathcal{R}_{\mathsf{std}} := \left\{ (\overline{\mathbf{s}}; \overline{\mathbf{w}}) : \begin{array}{rl} \forall i \in I_{\mathsf{add}}, & \omega_{\mathsf{add}_0(i)} = \omega_{\mathsf{add}_1(i)} + \omega_{\mathsf{add}_2(i)} \pmod q \wedge \\ \forall j \in I_{\mathsf{mul}}, & \omega_{\mathsf{mul}_0(j)} = \omega_{\mathsf{mul}_1(j)} \cdot \omega_{\mathsf{mul}_2(j)} \pmod q \wedge \\ \forall k \in I_{\mathsf{alg}}, & \mathsf{H}^k_{\mathsf{std}}(h_k; \mathbf{w}^k_{\mathsf{H}}) = 1 \\ \text{where:} & h_k \in \overline{\mathbf{s}}, \mathbf{w}^k_{\mathsf{H}} \subseteq \overline{\mathbf{w}} \end{array} \right\}$$

– **Algebraic with hash relation:** For all $k \in I_{\mathsf{alg}}$, an algebraic relation $\mathcal{R}^k_{\mathsf{ah}}$ on public wires $\widetilde{\mathbf{s}}_k$ and private wires $(\widetilde{\mathbf{w}}_k, r_k)$ is as follows:

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

Remark 3. The the input circuit $\mathcal{R}_{\mathsf{alg}}$ and output relation circuits $\mathcal{R}_{\mathsf{std}}$ and $\{\mathcal{R}^k_{\mathsf{ah}}\}_{I_{\mathsf{alg}}}$ naturally imply a mapping of wires following executions of the Compil:

$$\mathsf{wire} : \left(\mathbf{s}, \{h_k\}_{I_{\mathsf{alg}}}\right) \mapsto \left(\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}_{I_{\mathsf{alg}}}\right)$$

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

Lemma 1. *Assume* H *is a collision-resistant hash function. The compiler* Compil *given an input relation $\mathcal{R}_{\mathsf{alg}}$ outputs separate relations, $\mathcal{R}_{\mathsf{std}}$ and $\{\mathcal{R}^k_{\mathsf{ah}}\}$. 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:*

[9] 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.

[10] 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.

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

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

3.3 Hash-and-Prove Protocols

We restate the HP protocol, which links a SNARK to some Sigma protocols via hash function, and an internal SNARK as a component of the Sigma protocol. Both the SNARKs can be arbitrary, and we confirm that almost all algebraic statements, as long as a Sigma protocol exists, can be efficiently utilized in the HP protocol.

Specifically, during the HP protocol setup, the above compiler decomposes arithmetic with algebraic circuit $\mathcal{R}_{\mathsf{alg}}$ into two parts: $\mathcal{R}_{\mathsf{std}}$ and $\{\mathcal{R}^k_{\mathsf{ah}}\}$. The protocol execution consists of two parallelizable phases: a SNARK for $\mathcal{R}_{\mathsf{std}}$ and a Sigma AoK for each $\mathcal{R}^k_{\mathsf{ah}}$. The internal SNARK operates as a sub-protocol within the Sigma AoKs, which provides a linking function for the hash relation and the algebraic relation in $\mathcal{R}^k_{\mathsf{ah}} := \{\mathsf{H}(\widetilde{\mathbf{w}}_k, r_k) = h_k \wedge P_k^{x_k} = Q_k\}$.

Sigma AoKs with internal SNARK. There is almost no efficient Sigma protocol for the relation $\{\mathsf{H}(\widetilde{\mathbf{w}}_k, r_k) = h_k \wedge P_k^{x_k} = Q_k\}$ with a hash function, but a zk-friendly hash can be efficiently proven by SNARKs. A series of generalized Sigma AoK (defined in Section 2.1) consisting of a Sigma protocol accompanied by a SNARK proof are designed for these relations. The construction of Sigma AoKs is detailed in Section 4. Let Π_{lnk} be the internal zkSNARK. For an algebraic with hash relation $\mathcal{R}^k_{\mathsf{ah}}$ with algebraic gate $\mathcal{A}_k$, let Π^k_{aok} be the corresponding Sigma AoK for $\mathcal{R}^k_{\mathsf{ah}}$.

Hash-and-Prove protocol Π_{hp}. Let λ be the security parameter, H be a hash function. For a circuit $\mathcal{R}_{\mathsf{alg}}$ with algebraic gates $\{\mathcal{A}_k\}$, let Π be a zkSNARK for relation $\mathcal{R}_{\mathsf{std}}$ and Π^k_{aok} be the customized Sigma AoK with negligible knowledge error κ_k. A Hash-and-Prove protocol is a triple of polynomial time algorithms $\Pi_{\mathsf{hp}} := (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$ behave as shown in Figure 3.

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

Hash-and-Prove Protocol Π_{hp}

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

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

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

1. P, V are given the public input $\mathbf{s}$, and P is given the private input $\mathbf{w}$.
2. For each $k \in I_{\mathsf{alg}}$, 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, 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 $k \in I_{\mathsf{alg}}$, P, 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 $\mathcal{R}^k_{\mathsf{ah}}$, in parallel.

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

Fig. 3: Π_{hp} for $\mathcal{R}_{\mathsf{alg}}(\mathbf{s}; \mathbf{w}) = 1$

Due to space limitation, we defer the detailed proof to Appendix B.

Non-interactive via Fiat-Shamir heuristic [FS87]**.** For a 3-move public-coin interactive protocol, the Fiat-Shamir transformation can convert the HVZK property into zero-knowledge while preserving the knowledge soundness in the ROM. In the following, we would apply the recursion proof technique (includes IVC,PCD) for further optimization. However, we cannot analysis the security of the whole protocol in the random oracle model due to the issue of proving hash function treated as random oracle. Therefore, we need to assume the existence of concrete hash function such that when instantiated with this function, the resulting proofs are secure in the real world. For the same reason, we need to assume the existence of hiding hash functions rather than directly treat it as a random oracle. A further discussion of this issue is provided in Section 2.3.

4 Sigma Protocols for Algebraic with Hash Relation

In this section, we instantiate the Sigma AoKs as $\{\Pi^k_{\mathsf{dl}}\}_{k\in[4]}$ and $\{\Pi^k_{\mathsf{rsa}}\}_{k\in[2]}$, for the discrete logarithm relation $\{\mathcal{R}^k_{\mathsf{dl}}\}_{k\in[4]}$ and the RSA relation $\{\mathcal{R}^k_{\mathsf{rsa}}\}_{k\in[2]}$, where $[i]$ denotes the set of natural numbers $\{0, 1, \ldots, i-1\}$. We first introduce a protocol as an example for describing the main idea behind our technique. Next, we discuss the variations of the protocol in different settings, such as the RSA setting. Subsequently, we present some optimization strategies for Sigma AoKs to reduce the parallel overhead further.

"Ping-pong" alternating of SNARK and Sigma. From a high-level perspective, our scheme applies a "ping-pong" mode: the proving starts with a SNARK, transitions to a Sigma protocol, and then returns to a SNARK with a reduced circuit size to be proven (see Figure 4). On the one hand, a trivial Sigma protocol can only prove the knowledge of one witness in the statement, whereas our Sigma AoK is supplemented by a SNARK to complete the proof for the statement containing multiple witnesses. On the other hand, the circuit size for this supplementary proof is significantly smaller than that of the complete group exponentiation circuit in the original SNARK.

Some Sigma protocols require parallelism to achieve sufficient soundness. We present two optimization strategies to enhance the parallel efficiency in Section 4.2. The first strategy involves applying computation reuse techniques and adjusting the size of challenge space to minimize the prover's computation in parallel repeated circuits. Another strategy is that since the internal SNARK circuit is a uniform circuit, which allows for different assignments to the same circuit when executed in parallel. We can leverage techniques such as proof-carrying data, folding schemes, or proof batching to significantly reduce prover time.

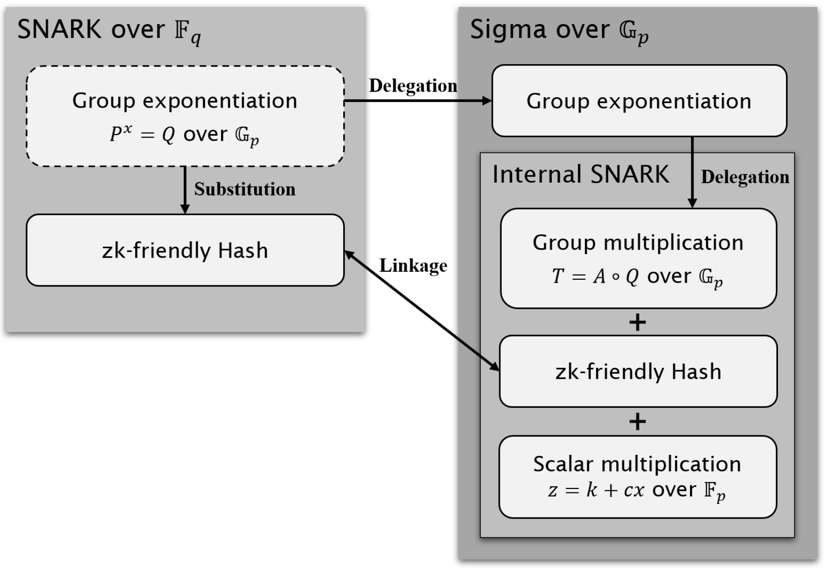


Fig. 4: An example HP protocol for Π^1_{dl}

Compared to using general zkSNARKs to prove everything within an arithmetic circuit, these Sigma AoKs reduce more than 90% of the group operations in the original circuit and delegate it to an internal SNARK. Further through our parallel optimization techniques, we improve the prover efficiency by more that $100\times$. On the other hand, compared to the Commit-and-Prove schemes, these Sigma AoKs (1) can connect to a generalized SNARK rather than a specific SNARK, (2) do not require sending Pedersen commitments, which are typically over a large group and result in intolerable proof size, and (3) do not require an additional "glue" proof to connect to the circuit witness. So it significantly reduces the proof size.

4.1 Sigma Argument of Knowledge

To facilitate understanding, we start with simple examples of several Sigma AoKs for cyclic groups in discrete logarithm setting, including Schnorr protocol [Sch91] and its variants. Additionally, to achieve the completeness of algebraic gates over cyclic groups, we present the constructions of all remaining Sigma AoK protocols in Appendix A.

Notation. Let $(\mathbb{G}_p, \circ)$ be a cyclic group with a prime order p and a generator G. We denote scalar operations as G^x, where x belongs to the scalar field $\mathbb{F}_p$. Consider a zkSNARK Π_{lnk} over $\mathbb{F}_q$. And denote $h := \mathsf{H}(x, r)$, where $r \xleftarrow{\$} \{0,1\}^\lambda$ and H is the hiding hash function defined by Definition 4.

I. Linking Hash into Sigma Protocols. The goal here is to achieve the linkage of any SNARK and Sigma without an additional "glue" proof, reducing the proof size. Here we build on the idea of Orrù et al. [KMN23,OKMZ24] and extend it to general cases in general cyclic groups (including RSA groups).

Consider a simple example of proving the statement $Q = P^x$, where (P, Q) are public input and x is a private input. This can be efficiently proven using Schnorr's Sigma protocol. To combine the Schnorr statement $Q = P^x$ with the SNARK statement $h = \mathsf{H}(x)$ to a conjunction(AND) statement, the prover can be required to prove that the third message $z = k + cx$ is actually computed using the hashed value x, which can be efficiently achieved using SNARKs.

Above idea is enough for the construction of Sigma Aok for relations single witness, like $\mathcal{R}^0_{\mathsf{dl}} := \{(P, Q, h; (x, r)) : Q = P^x \wedge h = \mathsf{H}(x, r)\}$. And we provide the constructions in Appendix A. However, as shown in Introduction, above method fails to build proofs for statements with more witnesses, like the statement $\{(P, h; (Q, x, r)) : Q = P^x \wedge h = \mathsf{H}(Q, x, r)\}$.

II. Expanding Linking to Support More Witnesses. First, we revisit Schnorr's Sigma protocol to understand why it fails when proving statements with multiple witnesses. Consider 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. A straightforward approach is to require the prover to demonstrate that $T = A \circ Q^c$ using SNARKs. However, this necessitates proving a group exponentiation, which brings us back to the initial challenge.

An observation here is that, if we reduce the size of challenge space, and consider the corresponding Schnorr's protocol as follows:

1. P chooses $k \xleftarrow{\$} \mathbb{F}_p$ and sends $A = P^k$ to V.
2. V chooses $c \xleftarrow{\$} \{0,1\}$ and sends it to P.
3. If $c = 0$, P sends $z = k$ to V. Otherwise, P computes $z = k + x \pmod p), T = P^z$ and sends z to V.
4. If $c = 0$, V checks $A = P^z$. Otherwise, V computes $T = P^z$ and checks $T = A \circ Q$.

Then, the response proof is required to imply only a single group operation $T = A \circ Q$ when $c = 1$ for checking. Even consider parallelization for reducing soundness error, the computation is still much less than compute the exponential directly. Now, we can require the prover to prove $T = A \circ Q$ via SNARK (some optimization technique will be discuss in the next subsections), to help the verifier to conclude 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 also needs to be hidden in the hash function.

Protocol Π^1_{dl} with x and Q as witness. Since Q is also a witness, the statement includes the hash function $h = \mathsf{H}(Q, x, r)$. The response includes a linking proof π_{lnk} from the SNARK Π_{lnk}, proving the following relationship:[11]

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

where $c \in \{0,1\}$ and $r, r_k \in \{0,1\}^\lambda$ and H is a hash function.

Theorem 3. *Let $\mathbb{G}_p$ be a cyclic group with prime order p, if H is a hiding hash function, Π_{lnk} is a zkSNARK, then Π^1_{dl} in Figure 5 is a Sigma AoK with $1/2$ knowledge error 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 $r \in \{0,1\}^\lambda$.

III. Sigma AoKs in Other Settings . 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

[11] We present the protocol of uniform version to support the proof batching (see Section 4.2). The protocol also has a non-uniform version, where the SNARK circuit does not include group operations in $c = 0$, which supports the parallel composition.

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

Prover$(\mathsf{crs}, P, h; (Q, x, r))$ **Verifier**(crs, P, h)

$k \xleftarrow{\$} \mathbb{F}_p; r_k \xleftarrow{\$} \{0,1\}^\lambda;$

$A = P^k; h_k = \mathsf{H}(A, k, r_k);$

$$\xrightarrow{\quad h_k \quad}$$

$c \xleftarrow{\$} \{0,1\};$

$$\xleftarrow{\quad c \quad}$$

$z = k + cx \pmod p; T = A \circ Q^c;$

Run Π_{lnk} for $\mathcal{R}^1_{\mathsf{lnk}}$:

$\pi_{\mathsf{lnk}} \leftarrow \Pi_{\mathsf{lnk}}.\mathsf{P}\big(\mathsf{crs}, h, h_k, c, z, T;$
$(Q, x, k, A, r, r_k)\big);$

$$\xrightarrow{\quad z, \pi_{\mathsf{lnk}} \quad}$$

Compute $T = P^z;$

$\Pi_{\mathsf{lnk}}.\mathsf{V}(\mathsf{crs}, \pi_{\mathsf{lnk}}) \overset{?}{=} 1$

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

and signature schemes defined over the RSA ring $\mathbb{Z}_n$, 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, as detailed in Appendix A. 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. 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). The latter can be widely used in anonymous credential scenarios.

The construction of Sigma AoKs in the RSA setting is essentially the same as before. The key point to note is that, due to the unknown order of the RSA group, the challenge space must be set to $\{0, 1\}$ to provide the knowledge soundness. In this case, the extractor can use the extended Euclidean algorithm on two different transcripts to compute the witness. Then, for these Sigma AoKs with 1/2 knowledge error, we use the parallel optimization techniques introduced below to reduce the error while saving costs.

4.2 Parallel Optimization for Sigma AoKs

For Sigma AoKs with a knowledge error of 1/2, we can reduce the knowledge error through parallel repetition, similar to how the Sigma protocol reduces soundness error. We can construct an extractor for the ℓ-times parallel Sigma AoK protocol, where the extractor rewinds the prover to obtain two acceptable Sigma proofs and extracts the witness, and at the same time, calls the SNARK extractor to extract witness of SNARK statement. The consistency of witnesses is guaranteed from the soundness of SNARKs. Since the negligible knowledge error of SNARK and the exponential-sized challenge space, this extractor can ensure knowledge soundness with negligible error.

Using protocol Π^1_{dl} as an example, it has a knowledge error of 1/2, thus requiring parallel. However, trivial parallelization leads to linear overhead. We propose two different optimization strategies for the parallel of Sigma AoKs. The first strategy for the non-uniform version is general and results in optimized prover time and significant reducing proof size. The second strategy focuses on proof batching for uniform circuits, which can significantly improve prover time when using specific SNARKs.

Optimization I. Parallel Composition of Sigma AoKs. 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 prover can reuse the proof for $Q^2, \dots, Q^m$.

For example, let the challenge $c \in \{0, 1, 2, 3\}$ and knowledge 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 total number of group operations is 24.5, whereas the trivial parallel way requires 30. We propose an optimization strategy for Π^1_{dl} when the knowledge error is $2^{-\lambda}$, which involves computing the minimum point of $f(m) = \frac{\lambda(m-1)}{m} \log_m 2 + (m-1)$.

Next, we will introduce another optimization strategy for uniform circuits in some special SNARK schemes. However, applying optimization I, which is generally applicable, will cause part of the circuit to become non-uniform, thereby affecting the efficiency of the optimization II.

Optimization 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 proof-carrying data, folding schemes, or proof batching. Specifically, we instantiate the internal SNARK using the Varuna scheme, which leverage the proof batching technique [GMN22,ASS+22] to significantly reduce prover time. This technique allows us to add more assignments of the same circuit with only an additional prover time cost of approximately 1/8 for each assignment.

For example, the uniform circuit $\mathcal{R}^1_{\mathsf{lnk}}$ includes two zk-friendly hash, a non-native scalar multiplication $z = k + cx$, and a group operation $T = A \circ Q^c$. Then we discuss how to select appropriate zk-friendly hash functions and internal SNARKs.

Select hash functions. We first consider two scenarios based on whether the original SNARK and internal SNARK in the same field. In the same field scenario, we can use a zk-friendly hash function within the native field. In different field scenarios, we can only use binary field hash functions like SHA256, since proving zk-friendly hash function in non-native field would result in higher overhead. Despite employing proof batching techniques for SHA256 proofs, the prover time remains intolerable.

We propose an effective trick to replace the SHA256 in the internal SNARK with a zk-friendly hash like Poseidon over native field. As shown in Figure 6, the prover additionally sends a "linking" 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.

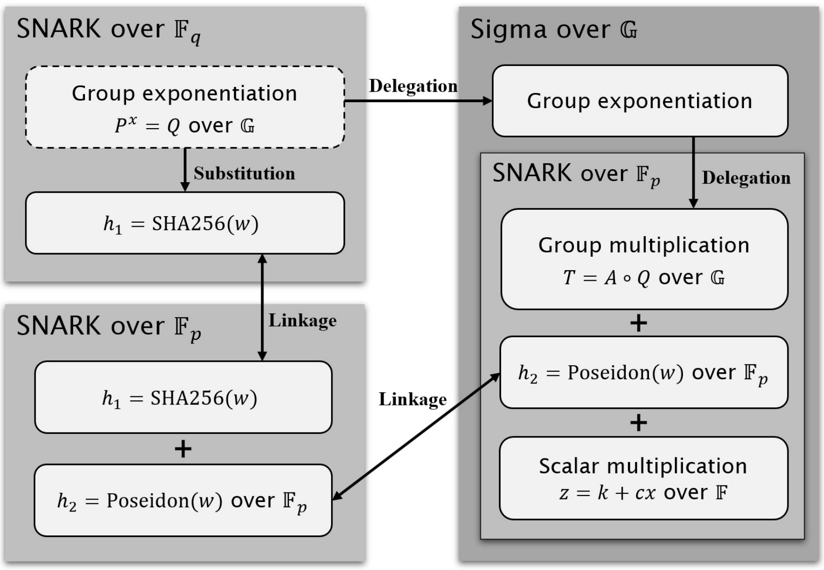

SNARK over $\mathbb{F}_q$
Group exponentiation $P^x = Q$ over $\mathbb{G}$
Substitution
$h_1 = \mathsf{SHA256}(w)$
Delegation
Sigma over $\mathbb{G}$
Group exponentiation
SNARK over $\mathbb{F}_p$
Delegation
Group multiplication $T = A \circ Q$ over $\mathbb{G}$
+
$h_2 = \mathsf{Poseidon}(w)$ over $\mathbb{F}_p$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

Fig. 6: Optimization II in different field.

Select internal SNARKs. 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 [GLS+23,ZCF24,BCKL22,HLP24], the HP protocol can support this optimization.

5 Performance

The configuration of the testing machine used in the experiments is as follows: Intel(R) Core(TM) i7-10870H CPU @ 2.20 GHz, 16.0 GB RAM. The reported numbers are averages over five runs. To facilitate a clearer comparison with previous work, our scheme adopts a knowledge error of 2^{-60}, the same as in prior study [AGM18]. Our framework is compatible with any SNARK, allowing for faster implementations in practice. To showcase its efficiency, this section primarily utilizes two proof systems: Plonky2 [0xP23] and Varuna [Ale23]. For implementation details please refer to Appendix D.

In the comparative experiments of this section, we primarily aim to highlight the advantages of our Sigma AoKs for algebraic statements compared to Sigma protocols under Pedersen commitment. Specifically in terms of proving elliptic curve operations, our approach demonstrates significant advantages in all aspects: proof time, proof size and verification time. For non-algebraic part, we assume all the following solutions need to pay the same cost for some hash functions like SHA256. One SHA256 hash can be proved within one second for most SNARKs, which is relatively minimal overhead. But in high-throughput signature verification scenarios, we need to prove large amount of hash functions. Our scheme offers the advantage of combining SNARKs that can efficiently proves hash functions with those that are efficient in proving algebraic operations, as illustrated in Figure 6.

5.1 Proof for Solvency

In the context of Proof of Solvency, we are required to prove a statement of the form "$\text{SHA256}(G^x) = y$", where the algebraic component "G^x" corresponds precisely to the relation $\mathcal{R}_{\mathsf{dl}}^1$ as defined in our work. Accordingly, we retain all other aspects of the proof unchanged, except for employing our proposed sigma AoK Π_{dl}^1 to prove the statement involving "G^x". The knowledge error of the sigma AoK Π_{dl}^1 is 1/2. By employing the first optimization method from Section 4.2, we achieve a challenge space size of 8, resulting in a knowledge error of 2^{-3} for a single execution. By conducting this process in parallel 20 times, we obtain a protocol with knowledge error of 2^{-60}.

For $n = 1$, the computations we need to prove in the internal SNARK include, on average: 25 elliptic curve point additions, 20 non-native multiplications and a Poseidon hash of 42 elements. The proof consists of 20 field elements, 1 output of Poseidon Hash and a proof of an internal SNARK. And the verifier work consists of 20 group exponentiation and the verification of internal SNARK.

The internal SNARK circuit along with one SHA256 function can be proved in 2.5 second within the Plonky2 system. In Plonky2, $|\pi| \approx 144.8$ kb and $T_v \approx 7$ ms; however, in plonky2 we can shrink to the recursion threshold which results in $|\pi| \approx 50$kb, $T_v \approx 4$ms, this depends on the specific parameter settings of the proof system. Previous works typically either encode an entire algebraic statement in SNARKs or utilize SNARKs and Sigma protocols under Pedersen

Table 1: Comparison for Proof of Solvency using different methods. T_p, T_v and $|\pi_{\mathsf{s}}|$ represent the prover time, verifier time and proof size of the internal SNARK. We can always generate one SNARK proof to include all necessary computations in the third-round of Sigma AoKs for multiple statements. T_{sha} denotes the cost for proving a SHA256 permutation. Let $|\mathsf{H}| = 256$ denotes the fixed bit-length output of the hash function H. n is the size of the anonymity set, m is $\lceil \text{logMax} \rceil = 51$, and p denotes the size of the field over which the curve is defined).

	Proof size	Prover time	Verifier time
QAP+Sigma [AGM18]	$(2396n + \log p + \log m)$ elements	$(T_{\mathsf{sha}} + 30p + 1800)n$ exp	$(10p + 4)n$ exp +30 pairings
Our work HP(Plonky2+Sigma)	$20n$ elements $+ \ |\mathsf{H}| + |\pi_{\mathsf{s}}|$	nT_{sha} exp $+T_p$	$20n$ exp $+T_v$

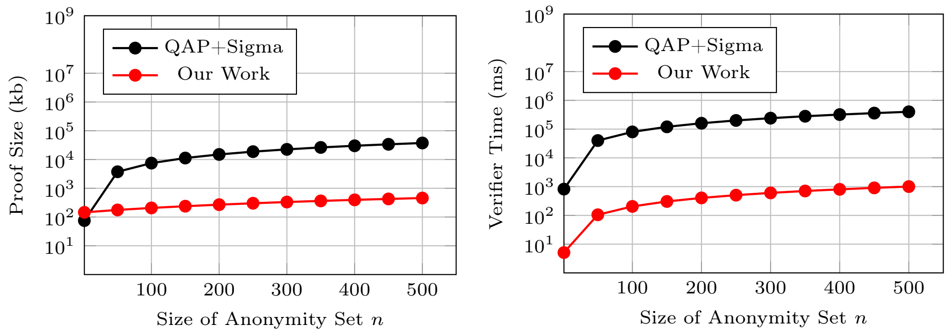


Fig. 7: Comparison for Proof Size and Verifier Time of QAP+Sigma [AGM18] and our work (Plonky2+Sigma)

commitments. In the former case, we need to prove 256 elliptic curve point additions on average in SNARKs. For this, we tested in Plonky2 and obtained a proof time of approximately 20 seconds. In the latter case, the prover's work involves more than 20000 elliptic curve exponentiation, the order of corresponding elliptic group is at least 2^{768} [AGM18]. We tested BLS12-377 for 24840 scalar multiplications result in 6.5s. This is far shorter than the time it is supposed to be. Therefore, this causes the prover to spend at least 2 times greater than the solution in this work. In the proof size analysis, we account for all group elements and field elements together. To clearly and directly demonstrate the advantages of our scheme, we assume that these group and field elements are 256-bit in size, although in practice, other schemes may involve elements with larger bit lengths(Exp denotes exponentiation in corresponding groups).

5.2 Proof for ECDSA Signature Verification

In the middle of ECDSA signature verification, for group operations of the form "$(x_1, y_1) = G^{u_1} P^{u_2}$", we apply our Sigma AoK Π^1_{dl} twice: once for G^{u_1} and once for P^{u_2}. These precisely correspond to the relation $\mathcal{R}^1_{\mathsf{dl}}$ that we have defined.

When employing Plonky2 as underlying proof system, the optimization method is exactly the same as in Section 5.1. The only difference here is that we need to use the Sigma AoK to prove two algebraic gates instead of one. When employing Varuna as underlying proof system, we use the second optimization in section 4.2. We simply parallel run 60 times Π^1_{dl}, on average, for 30 times c=0, prover sends the random number used in hash function and "z" to verifier; and for 30 times c=1, thus we need to prove $\mathcal{R}^1_{\mathsf{lnk}}$ with 30 different instances by utilizing batch strategy in Varuna. This batch strategy fundamentally involves linear combinations to address batch sum-check problem instances. For one ECDSA signature verification, the proof consists of 60 field elements, 30 outputs of Poseidon Hash, 30 random numbers within hash function and a proof of an internal SNARK. We fix the random number length to 256 bits.

Table 2: Comparison for ECDSA signature verification using different methods. The proof size of Varuna is 1.06 kb, while the proof size for Plonky2 is 145.1 kb.

	Proof size	Prover time	Verifier time
Groth16 [Gro16,Sun24]	128bytes	149s	2ms
Field-Agnostic[BFK+24]	780kb	0.23s	67ms
Our work HP(Plonky2+Sigma)	146.42kb	4.1s	7.3ms
Our work HP(Varuna+Sigma)	4.81kb	2.1s	7.9ms

The test machine used for the experiments in the first two rows of table 2 is: AWS c5a.16xlarge, Ubuntu 22.04, with 64 cores and 124 GB memory [BFK+24].

5.3 Proof for RSA Signature Verification

In the middle of RSA signature verification, the group operations of the form "$\sigma^e = m$" precisely correspond to the relation $\mathcal{R}^1_{\mathsf{rsa}}$ that we have defined. We utilize the second optimization method from Section 4.2, running it in parallel 60 times, thus we need to prove $\mathcal{R}^4_{\mathsf{lnk}}$ for 30 different instances.

For one RSA signature verification, the circuit for internal SNARK consists of 2 non-native multiplications over RSA modulus and a Poseidon hash for 4 elements, with batching number of 30. This can be proved in the Varuna proof system in less than 3.5 seconds in practice and proof size of 16.5kb. Our scheme allows the public key e to take any value less than $\varphi(n)$, where n is the RSA modulus, while maintaining constant proof time and proof size.

Table 3: Comparison for RSA signature verification using different methods. $|n_e|$ denotes the bit length of public key e, $|N|$ denotes the length of corresponding RSA mudulo which is normally 2000. $|\pi_{\mathsf{g}}|$ denotes the proof size of "glue" proof. T_{g} denotes the cost of "glue" proof's verifier. Let $|r|$=256 denotes the fixed length of random number in hash function.

	Proof size	Verifier time
QAP+Sigma[AGM18]	$(3\|n_e\|+1)\|N\|+\|\pi_{\mathsf{s}}\|+\|\pi_{\mathsf{g}}\|$	$10\|n_e\|\exp + T_{\mathsf{v}} + T_{\mathsf{g}}$
Our work HP(Varuna+Sigma)	$60\|N\| + 30(\|\mathsf{H}\| + \|r\|) + \|\pi_{\mathsf{s}}\|$	60 exp $+T_{\mathsf{v}}$

5.4 Proof for DSA Signature Verification

In DSA signature verification, the group operations are the same as ECDSA except the discrete-log is derived from large prime number modular exponentiation. But we utilize the second optimization method from Section 4.2, running it in parallel 60 times, thus we need to prove $\mathcal{R}^1_{\mathsf{lnk}}$ for 30 different instances.

For one DSA signature verification, the circuit for internal SNARK consists of 1 non-native multiplication over an L-bit prime modulus, 1 non-native multiplication over an N-bit prime modulus and a Poseidon hash for 4 elements, with batching number of 30. This can be proved in the Varuna proof system in less than 3 seconds in practice and proof size of 3.75kb(assuming $L = 2048$ and $N = 256$).

Table 4: Comparison for DSA signature verification using different methods. For DSA parameters generation, we assume an N-bit prime number q and L-bit prime number p such that $q|p-1$.

	Proof size	Verifier time
GC+Sigma[CGM16]	$120(L + 2N) + \|\pi_{\mathsf{s}}\| + \|\pi_{\mathsf{g}}\|$	1200 exp$+T_{\mathsf{v}}+T_{\mathsf{g}}$
Our work HP(Varuna+Sigma)	$60N + 30(\|\mathsf{H}\| + \|r\|) + \|\pi_{\mathsf{s}}\|$	60 exp $+ T_{\mathsf{v}}$

References

0xP23. 0xPolygonZero. Plonky2: Fast, recursive zksnarks, 2023.

ABC$^+$22. Diego F. Aranha, Emil Madsen Bennedsen, Matteo Campanelli, Chaya Ganesh, Claudio Orlandi, and Akira Takahashi. ECLIPSE: Enhanced compiling method for pedersen-committed zkSNARK engines. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13177 of *Lecture Notes in Computer Science*, pages 584–614, Virtual Event, March 8–11, 2022. Springer, Cham, Switzerland.

AGM18. Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. Non-interactive zero-knowledge proofs for composite statements. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 643–673, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

Ale23. AleoNet. Varuna: Readme, 2023.

ASS$^+$22. Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. Varuna: scalable, low-cost training of massive deep learning models. In Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, editors, *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 472–487. ACM, 2022.

BBB$^+$18. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

BBHR18. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018.

BCCT13. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 111–120, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

BCF$^+$21. Daniel Benarroch, Matteo Campanelli, Dario Fiore, Jihye Kim, Jiwon Lee, Hyunok Oh, and Anaïs Querol. Proposal: commit-and-prove zero-knowledge proof systems and extensions. In *4th ZKProof Workshop*, 2021.

BCKL22. Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Scalable and transparent proofs over all large fields, via elliptic curves - (ECFFT part II). In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part I*, volume 13747 of *Lecture Notes in Computer Science*, pages 467–496, Chicago, IL, USA, November 7–10, 2022. Springer, Cham, Switzerland.

BCL$^+$21. Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 681–710, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

BCMS20. Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. Cryptology ePrint Archive, Report 2020/499, 2020.

BCR$^+$19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

BFK$^+$24. Alexander R. Block, Zhiyong Fang, Jonathan Katz, Justin Thaler, Hendrik Waldner, and Yupeng Zhang. Field-agnostic snarks from expand-accumulate codes. Springer-Verlag, 2024.

BHH$^+$19. Michael Backes, Lucjan Hanzlik, Amir Herzberg, Aniket Kate, and Ivan Pryvalov. Efficient non-interactive zero-knowledge proofs in cross-domains without trusted setup. In Dongdai Lin and Kazue Sako, editors, *PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 11442 of *Lecture Notes in Computer Science*, pages 286–313, Beijing, China, April 14–17, 2019. Springer, Cham, Switzerland.

CBBZ23. Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

CCG$^+$23. Megan Chen, Alessandro Chiesa, Tom Gur, Jack O'Connor, and Nicholas Spooner. Proof-carrying data from arithmetized random oracles. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 379–404, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

CCS22. Megan Chen, Alessandro Chiesa, and Nicholas Spooner. On succinct non-interactive arguments in relativized worlds. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 336–366, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

CFF$^+$21. Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 3–33, Singapore, December 6–10, 2021. Springer, Cham, Switzerland.

CFQ19. Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2075–2092, London, UK, November 11–15, 2019. ACM Press.

CGM16. Melissa Chase, Chaya Ganesh, and Payman Mohassel. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In Matthew Robshaw and Jonathan

Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 499–530, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Berlin, Heidelberg, Germany.

CHM$^+$20. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

CL20. Alessandro Chiesa and Siqi Liu. On the impossibility of probabilistic proofs in relativized worlds. In Thomas Vidick, editor, *ITCS 2020: 11th Innovations in Theoretical Computer Science Conference*, volume 151, pages 57:1–57:30, Seattle, WA, USA, January 12–14, 2020. LIPIcs.

CLOS02. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.

COS20. Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 769–793, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

CT10. Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, *ICS 2010: 1st Innovations in Computer Science*, pages 310–331, Tsinghua University, Beijing, China, January 5–7, 2010. Tsinghua University Press.

Dam. Ivan Damgard. *On Sigma Protocols.*

DG23. Quang Dao and Paul Grubbs. Spartan and bulletproofs are simulation-extractable (for free!). In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 531–562, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

EFG22. Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Report 2022/1763, 2022.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Berlin, Heidelberg, Germany.

GKR$^+$21. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 519–535. USENIX Association, August 11–13, 2021.

GLS$^+$23. Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 193–226, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

GMN22. Nicolas Gailly, Mary Maller, and Anca Nitulescu. SnarkPack: Practical SNARK aggregation. In Ittay Eyal and Juan A. Garay, editors, *FC 2022: 26th International Conference on Financial Cryptography and Data Security*, volume 13411 of *Lecture Notes in Computer Science*, pages 203–229, Grenada, May 2–6, 2022. Springer, Cham, Switzerland.

GMR85. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th Annual ACM Symposium on Theory of Computing*, pages 291–304, Providence, RI, USA, May 6–8, 1985. ACM Press.

GQ88. Louis C. Guillou and Jean-Jacques Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory. In C. G. Günther, editor, *Advances in Cryptology – EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128, Davos, Switzerland, May 25–27, 1988. Springer, Berlin, Heidelberg, Germany.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer, Berlin, Heidelberg, Germany.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Heidelberg, Germany.

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

HAN23. Mathias Hall-Andersen and Jesper Buus Nielsen. On valiant's conjecture: Impossibility of incrementally verifiable computation from random oracles. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 438–469, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

HLP24. Ulrich Haböck, David Levit, and Shahar Papini. Circle STARKs. Cryptology ePrint Archive, Report 2024/278, 2024.

JkY23. JkY. Non-native field arithmetic. https://hackmd.io/@JkY-zACaSqerTtn_UwFjKg/SJZw6x75o, 2023.

Kil89. Joe Kilian. *Uses of Randomness in Algorithms and Protocols.* Doctoral dissertation, Massachusetts Institute of Technology, 1989.

KMN23. George Kadianakis, Mary Maller, and Andrija Novakovic. Sigmabus: Binding sigmas in circuits for fast curve operations. Cryptology ePrint Archive, Report 2023/1406, 2023.

KPS18. Ahmed E. Kosba, Charalampos Papamanthou, and Elaine Shi. xJsnark: A framework for efficient verifiable computation. In *2018 IEEE Symposium on Security and Privacy*, pages 944–961, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

MBKM19. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng

Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2111–2128, London, UK, November 11–15, 2019. ACM Press.

OKMZ24. Michele Orrù, George Kadianakis, Mary Maller, and Greg Zaverucha. Beyond the circuit: How to minimize foreign arithmetic in ZKP circuits. Cryptology ePrint Archive, Report 2024/265, 2024.

PHGR13. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

Sch91. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

Sun24. Y. Sun. circom-ecdsa. `https://github.com/0xPARC/circom-ecdsa`, 2024.

T$^+$23. Amir Tehrani et al. Spartan-ecdsa. `https://github.com/personaelabs/spartan-ecdsa`, 2023.

Val08. Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, March 19–21, 2008. Springer, Berlin, Heidelberg, Germany.

WTS$^+$18. Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 926–943. IEEE Computer Society, 2018.

XZZ$^+$19. Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 733–764, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

ZCF24. Hadas Zeilberger, Binyi Chen, and Ben Fisch. Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. Springer-Verlag, 2024.

ZCYW23. Min Zhang, Yu Chen, Chuanzhou Yao, and Zhichao Wang. Sigma protocols from verifiable secret sharing and their applications. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part II*, volume 14439 of *Lecture Notes in Computer Science*, pages 208–242, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore.

A Other Sigma AoKs

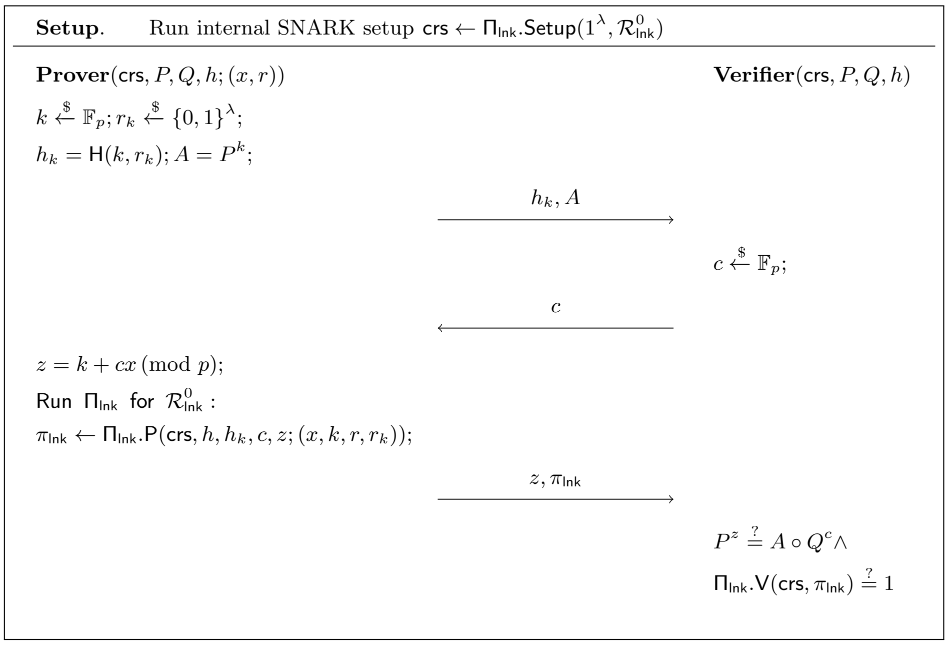
In discrete logarithm setting, there are two trivial cases, when only P or Q is the witness, the verifier can directly compute it. Additionally, in the case where $Q = P^x$ with P and x as witness, the statement can be converted to $P = Q^{x^{-1} \bmod p}$, thus using Π^1_{dl} to prove it.

A.1 Protocol Π^0_{dl} with x as witness.

Define relation $\mathcal{R}$ as follows:

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

where $x, k, c, z \in \mathbb{F}_p$, $r, r_k \in \{0,1\}^\lambda$, and H is a hash function. The Sigma AoK protocol with negligible knowledge error for the relation $\mathcal{R}^0_{\mathsf{dl}} := \{(P, Q, h; (x, r)) : Q = P^x \wedge h = \mathsf{H}(x, r)\}$ is shown in Figure 8. And a complete proof for it security is shown in Appendix C


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

Prover$(\mathsf{crs}, P, Q, h; (x, r))$ | **Verifier**(crs, P, Q, h)

$k \xleftarrow{\$} \mathbb{F}_p; r_k \xleftarrow{\$} \{0,1\}^\lambda;$

$h_k = \mathsf{H}(k, r_k); A = P^k;$

$\xrightarrow{h_k, A}$

$c \xleftarrow{\$} \mathbb{F}_p;$

$\xleftarrow{c}$

$z = k + cx \,(\mathrm{mod}\ p);$

Run Π_{lnk} for $\mathcal{R}^0_{\mathsf{lnk}}$:

$\pi_{\mathsf{lnk}} \leftarrow \Pi_{\mathsf{lnk}}.\mathsf{P}(\mathsf{crs}, h, h_k, c, z; (x, k, r, r_k));$

$\xrightarrow{z, \pi_{\mathsf{lnk}}}$

$P^z \stackrel{?}{=} A \circ Q^c \wedge$

$\Pi_{\mathsf{lnk}}.\mathsf{V}(\mathsf{crs}, \pi_{\mathsf{lnk}}) \stackrel{?}{=} 1$

Fig. 8: Π^0_{dl} for $\mathcal{R}^0_{\mathsf{dl}} := \{(P, Q, h; (x, r)) : Q = P^x \wedge h = \mathsf{H}(x, r)\}$ [KMN23,OKMZ24]

Theorem 4. [KMN23,OKMZ24] *Assume* H *is a hiding hash function, then* Π^0_{dl} *in Figure 8 is a Sigma AoK with negligible knowledge error 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* $x \in \mathbb{F}_p, r \in \{0,1\}^\lambda$.

A.2 Protocol Π^2_{dl} with P and Q as witness.

Protocol Π^2_{dl} is a variant of the Guillou-Quisquater protocol [GQ88] for $Q = P^x$ where P and Q are hashed. It includes a extending linking proof π_{lnk} from the SNARK Π_{lnk}, proving the following relationship:

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

where $c \in \{0, 1\}$, $r, r_k \in \{0, 1\}^\lambda$ and H is a hash function.

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

Prover$(\mathsf{crs}, x, h; (P, Q, r))$		**Verifier**(crs, x, h)
$K \xleftarrow{\$} \mathbb{G}_p; A = K^x;$		
$r_k \xleftarrow{\$} \{0,1\}^\lambda; h_k = \mathsf{H}(K, A, r_k);$		
	$\xrightarrow{h_k}$	
		$c \xleftarrow{\$} \{0, 1\};$
	$\xleftarrow{c}$	
$Z = K \circ P^c; T = A \circ Q^c;$		
Run Π_{lnk} for $\mathcal{R}^2_{\mathsf{lnk}}$:		
$\pi_{\mathsf{lnk}} \leftarrow \Pi_{\mathsf{lnk}}.\mathsf{P}(\mathsf{crs}, h, h_k, c, Z, T; (P, Q, K, A, r, r_k));$		
	$\xrightarrow{Z, \pi_{\mathsf{lnk}}}$	
		Compute $T = Z^x$;
		$\Pi_{\mathsf{lnk}}.\mathsf{V}(\mathsf{crs}, \pi_{\mathsf{lnk}}) \overset{?}{=} 1$

Fig. 9: Π^2_{dl} for $\mathcal{R}^2_{\mathsf{dl}} := \{(x, h; (P, Q, r)) : Q = P^x \wedge h = \mathsf{H}(P, Q, r)\}$

Theorem 5. *Let $\mathbb{G}_p$ be a cyclic group with prime order p, if H is a hiding hash function, Π_{lnk} is a zkSNARK, then Π^2_{dl} in Figure 9 is a Sigma AoK with $1/2$ knowledge error 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 $x \in \mathbb{F}_p, r \in \{0, 1\}^\lambda$.

A.3 Protocol Π^3_{dl} with P, Q and x as witness.

Protocol Π^3_{dl} employs the "intermediate value" technique, 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 $\mathcal{R}^1_{\mathsf{dl}}$, which can be proven using Π^1_{dl} in parallel. And it also includes a linking proof for $\mathcal{R}^3_{\mathsf{lnk}} := \{(h_x; (x, r_x)) : h_x = \mathsf{H}(x, r_x)\}$.

Setup.Run $\mathsf{crs}_0 \leftarrow \Pi_{\mathsf{lnk}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^3_{\mathsf{lnk}})$ and $\mathsf{crs}_1, \mathsf{crs}_2 \leftarrow \Pi^2_{\mathsf{dl}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^2_{\mathsf{dl}})$

Prover$(\{\mathsf{crs}\}, h_p, h_q, h_x; (P, Q, x, r_p, r_q, r_x))$ **Verifier**$(\{\mathsf{crs}\}, h_p, h_q, h_x)$

$s \xleftarrow{\$} \mathbb{F}_p; r_t, r_k \xleftarrow{\$} \{0,1\}^\lambda; T = P^s;$

$t = s^{-1} \pmod p; k = tx \pmod p;$

$h_t = \mathsf{H}(t, r_t); h_k = \mathsf{H}(k, r_k);$

Run Π_{lnk} for $\mathcal{R}^3_{\mathsf{lnk}}$:

$\pi_{\mathsf{lnk}} \leftarrow \Pi_{\mathsf{lnk}}.\mathsf{P}(\mathsf{crs}_0, h_x; (x, r_x))$

$$\xrightarrow{T, h_t, h_k, \pi_{\mathsf{lnk}}}$$

$\Pi_{\mathsf{lnk}}.\mathsf{V}(\mathsf{crs}_0, \pi_{\mathsf{lnk}}) \overset{?}{=} 1$

Prover and Verifier run Π^1_{dl} :

$(\mathsf{crs}_1, T, h_p, h_t; (P, t, r_p, r_t)) \in \mathcal{R}^2_{\mathsf{dl}}$

$(\mathsf{crs}_2, T, h_q, h_k; (Q, k, r_q, r_k)) \in \mathcal{R}^2_{\mathsf{dl}}$

$$\longleftrightarrow$$

Fig. 10: Π^3_{dl} for $\mathcal{R}^3_{\mathsf{dl}} := \left\{ \begin{pmatrix} h_p, h_q, h_x; \\ (P, Q, x, r_p, r_q, r_x) \end{pmatrix} : \begin{matrix} Q = P^x \wedge h_p = \mathsf{H}(P, r_p) \wedge \\ h_q = \mathsf{H}(Q, r_q) \wedge h_x = \mathsf{H}(x, r_x) \end{matrix} \right\}$

Theorem 6. *Let $\mathbb{G}_p$ be a cyclic group with prime order p, if H is a hiding hash function, Π_{lnk} is a zkSNARK, Π^1_{dl} is the parallel version Sigma AoK with negligible knowledge error for $\mathcal{R}^1_{\mathsf{dl}}$ as described in Theorem 3, then Π^3_{dl} in Figure 10 is a Sigma AoK with negligible knowledge error for the relation:*

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

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

A.4 Protocol Π^0_{rsa} with P as witness.

Protocol Π^0_{rsa} is a variant of the Guillou-Quisquater protocol for $Q = P^x$ where $P, Q, x \in \mathbb{Z}_n^*$ and P is hashed. It includes a linking proof π_{lnk} from the SNARK Π_{lnk}, proving the following relationship:

$$\mathcal{R}^3_{\mathsf{lnk}} := \left\{ (h, h_k, c, Z; (P, K, r, r_k)) : \begin{array}{c} h = \mathsf{H}(P, r) \wedge h_k = \mathsf{H}(K, A, r_k) \\ \wedge Z = K \circ P^c \pmod n \end{array} \right\}$$

where $c \in \{0, 1\}$ and $r, r_k \in \{0, 1\}^\lambda$ and H is a hash function.

Setup. Run internal SNARK setup $\mathsf{crs} \leftarrow \Pi_{\mathsf{lnk}}.\mathsf{Setup}(1^\lambda, \mathcal{R}^3_{\mathsf{lnk}})$

Prover$(\mathsf{crs}, Q, x, h; (P, r))$ **Verifier**(crs, Q, x, h)

$K \xleftarrow{\$} \mathbb{Z}_n^*; r_k \xleftarrow{\$} \{0, 1\}^\lambda;$

$h_k = \mathsf{H}(K, r_k); A = K^x \pmod n;$

$\xrightarrow{h_k, A}$

$c \xleftarrow{\$} \{0, 1\};$

$\xleftarrow{c}$

$Z = K \circ P^c \pmod n;$

Run Π_{lnk} for $\mathcal{R}^3_{\mathsf{lnk}}$:

$\pi_{\mathsf{lnk}} \leftarrow \Pi_{\mathsf{lnk}}.\mathsf{P}(\mathsf{crs}, h, h_k, c, Z; (P, K, r, r_k));$

$\xrightarrow{Z, \pi_{\mathsf{lnk}}}$

$Z^x \overset{?}{=} A \circ Q^c \pmod n;$

$\wedge\ \Pi_{\mathsf{lnk}}.\mathsf{V}(\mathsf{crs}, \pi_{\mathsf{lnk}}) \overset{?}{=} 1$

Fig. 11: Π^0_{rsa} for $\mathcal{R}^0_{\mathsf{rsa}} := \{(Q, x, h; (P, r)) : Q = P^x \wedge h = \mathsf{H}(P, r)\}$

Theorem 7. *Let $\mathbb{Z}_n$ be a RSA ring, if H is a hiding hash function, Π_{lnk} is a zkSNARK, then Π^0_{rsa} in Figure 11 is a Sigma AoK with $1/2$ knowledge error 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 $r \in \{0, 1\}^\lambda$.

A.5 Protocol Π^1_{rsa} with P and Q as witness.

Protocol Π^1_{rsa} is quite similar to Π^2_{dl}. It includes a linking proof π_{lnk} from the SNARK Π_{lnk}, proving the following relationship:

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

where $c \in \{0, 1\}$ and $r, r_k \in \{0, 1\}^\lambda$ and H is a hash function.

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

Prover$(\mathsf{crs}, x, h; (P, Q, r))$ **Verifier**(crs, x, h)

$K \xleftarrow{\$} \mathbb{Z}_n^*; A = K^x \pmod{n};$

$r_k \xleftarrow{\$} \{0, 1\}^\lambda; h_k = \mathsf{H}(K, A, r_k);$

$\xrightarrow{\quad h_k \quad}$

$c \xleftarrow{\$} \{0, 1\};$

$\xleftarrow{\quad c \quad}$

$Z = K \circ P^c \pmod{n};$

$T = A \circ Q^c \pmod{n};$

Run Π_{lnk} for $\mathcal{R}^4_{\mathsf{lnk}}$:

$\pi_{\mathsf{lnk}} \leftarrow \Pi_{\mathsf{lnk}}.\mathsf{P}\big(\mathsf{crs}, h, h_k, c, Z, T;$
$\quad (P, Q, K, A, r, r_k)\big);$

$\xrightarrow{\quad Z, \pi_{\mathsf{lnk}} \quad}$

Compute $T = Z^x$;

$\Pi_{\mathsf{lnk}}.\mathsf{V}(\mathsf{crs}, \pi_{\mathsf{lnk}}) \overset{?}{=} 1$

Fig. 12: Π^1_{rsa} for $\mathcal{R}^1_{\mathsf{rsa}} := \{(x, h; (P, Q, r)) : Q = P^x \wedge h = \mathsf{H}(P, Q, r)\}$

Theorem 8. *Let $\mathbb{Z}_n$ be a RSA ring, if H is a hiding hash function, Π_{lnk} is a zkSNARK, then Π^1_{rsa} in Figure 12 is a Sigma AoK with $1/2$ knowledge error 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 $r \in \{0, 1\}^\lambda$.

B Proof of Theorem 2

Proof. **Completeness** is obvious.

Knowledge soundness. For any polynomial time prover P^*, the extractor Ext is constructed as follows:

1. Call P^* on $(\{\mathsf{crs}_k\}, \mathbf{s})$ to obtain $\{h_k\}$, 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 $k \in I_{\mathsf{alg}}$, 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.
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 ϵ, which includes the acceptable transcript π_{std} and $\{\pi^k_{\mathsf{ah}}\}$. According to the knowledge soundness of SNARK Π, the extractor successfully outputs a witness with a probability at least $\epsilon - \mathsf{negl}_0$. For each $k \in I_{\mathsf{alg}}$, according to the knowledge soundness of Π^k_{aok}, the extractor call $\Pi^k_{\mathsf{aok}}.\mathsf{Ext}^{\mathsf{P}^*}$ in expected polynomial times and can successfully outputs the witness with probability $1 - \mathsf{negl}_1$. 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 with a negligible knowledge 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 $(\pi^*_{\mathsf{std}}, \{\pi^{k*}_{\mathsf{ah}}\})$, which is indistinguishable from the honest transcript $(\pi_{\mathsf{std}}, \{\pi^k_{\mathsf{ah}}\})$. 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 $(\mathsf{crs}_0, \{\mathsf{crs}_k\})$ to obtain the instance $\mathbf{s}$.
2. For each $k \in I_{\mathsf{alg}}$, choose a random $r_k \in \{0,1\}^\lambda$ and random elements $w^*(P_k, Q_k, x_k)$ acting witnesses in $\{P_k, Q_k, x_k\}$, and then compute $h^*_k = \mathsf{H}(\mathcal{L}^{-1}(w^*(P_k, Q_k, x_k)), r_k)$.
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 $k \in I_{\mathsf{alg}}$, call the SHVZK simulator $\Pi^k_{\mathsf{aok}}.\mathsf{Sim}$ on $(\mathsf{crs}_k, \widetilde{\mathbf{s}}_k, \tau_k)$ with the honest-verifier challenge c_k to obtain the output π^{k*}_{ah}.
6. Output $(\overline{\mathbf{s}}^*, \{\widetilde{\mathbf{s}}^*_k\}, \pi^*_{\mathsf{std}}, \{\pi^{k*}_{\mathsf{ah}}\})$.

For the transcript $(\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\}, \pi_{\mathsf{std}}, \{\pi^k_{\mathsf{ah}}\})$ 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:

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

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

$\mathcal{H}_0$ 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.

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

$\mathcal{H}_1$ to $\mathcal{H}_{|I_{\mathsf{alg}}|+1}$ sequentially replaces Π^k_{aok} prover's proof π^k_{ah} with the transcript π^{k*}_{ah} generated by the simulator with the challenge c_k. Therefore, the indistinguishability is derived from the SHVZK property of Π^k_{aok}.

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

$\mathcal{H}_{|I_{\mathsf{alg}}|+2}$ represents the transcript generated by the 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 $\{h_k = \mathsf{H}(\hat{w}(P_k, Q_k, x_k), r_k)\}$ is replaced by $\{h^*_k = \mathsf{H}(\hat{w}^*(P_k, Q_k, x_k), r_k)\}$, which makes $(\overline{\mathbf{s}}, \{\widetilde{\mathbf{s}}_k\})$ and $(\overline{\mathbf{s}}^*, \{\widetilde{\mathbf{s}}^*_k\})$ 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. $\square$

C Proof of Theorem 4

Proof. **Completeness** is obvious.

Knowledge soundness. The knowledge soundness of Π^0_{dl} is derived from the SNARK proof π_{lnk}, which ensures that the prover who can produce acceptable transcripts authentically knows the witness x consistently in both h and z.

For any polynomial time prover P^*, there exists an expected polynomial time extractor Ext which behaves as follows:

1. Call P^* for instance (P, Q, h) and a random challenge $c_1 \xleftarrow{\$} \mathbb{F}_p$ to obtain a transcript $(h_k, A, c_1, z_1, \pi_1)$. If it is acceptable, continue, otherwise abort.
2. Call the SNARK extractor $\Pi_{\mathsf{lnk}}.\mathsf{Ext}$ for the instance (h, h_k, c_1, z_1) and acceptable transcript π_1, then $\Pi_{\mathsf{lnk}}.\mathsf{Ext}$ output a witness (x_1, k_1, r_1, r_k^1).[12]
3. Then, rewind P^* with fresh random challenge until obtain another acceptable transcript $(h_k, A, c_2, z_2, \pi_2)$. Similarly , call $\Pi_{\mathsf{lnk}}.\mathsf{Ext}$ for the instance (h, h_k, c_2, z_2) and acceptable transcript π_2 to obtain the second witness (x_2, k_2, r_2, r_k^2).
4. Check if $(x_1, k_1, r_1, r_k^1) \neq (x_2, k_2, r_2, r_k^2)$, then reject.
5. Output (x_1, r_1).

Assume that P^* can output an acceptable transcript for instance (P, Q, h) with probability ϵ, which implies $P^{z_1} = A \circ Q^{c_1}$ and π_1 is acceptable in step 1. Therefore, P^* as the SNARK prover, generates an acceptable proof with at least a probability of ϵ.

According to the knowledge soundness of SNARK Π_{lnk}, the probability that $\Pi_{\mathsf{lnk}}.\mathsf{Ext}$ successfully output a witness (x_1, k_1, r_1, r_k^1) in step 2 is $\epsilon - \kappa_{\mathsf{lnk}}$. And (x_1, k_1, r_1, r_k^1) satisfies $h = \mathsf{H}(x_1, r_1)$, $h_k = \mathsf{H}(k_1, r_k^1)$ and $z_1 = k_1 + c_1 x_1 \pmod p$.

In step 3, the extractor can rewind P^* in expected $1/\epsilon$ times to obtain the second transcripts with $c_2 \neq c_1$ with probability $1 - \mathsf{negl}$. And the probability that $\Pi_{\mathsf{lnk}}.\mathsf{Ext}$ fails to output a valid witness (x_2, k_2, r_2, r_k^2) is negligiblr.

In steps 4, the extractor accepts the consistency of witness with probability $1 - \mathsf{negl}$, since the collision resistance of H. Then, we denote (x_1, k_1) as the witness extracted above, and $x, k \in \mathbb{F}_p$ are the discrete logarithm of $Q, A \in \mathbb{G}_p$ with base P. According to the standard Sigma protocol extraction, we have:

$$k_1 + c_1 x_1 = k + c_1 x \pmod p \qquad k_1 + c_2 x_1 = k + c_2 x \pmod p$$

Since $c_1 \neq c_2$, it follows that $x = x_1$ and $k_1 = k$ consistently. Finally, the probability of Ext successfully extracting the witness (x, r) is $\epsilon - \kappa_{\mathsf{lnk}} - \mathsf{negl}$. Therefore, the knowledge error is negligible.

Computational SHVZK. There exists a polynomial time simulator Sim that for any instance (P, Q, h) in $\mathcal{R}^0_{\mathsf{dl}}$ and for random $c \in \mathbb{F}_p$ outputs a transcript

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

$(h_k^*, A^*, c, z^*, \pi^*)$, indistinguishable from the honest transcript (h_k, A, c, z, π). The Sim behaves as follows:[13]

1. Randomly choose $k^*, z^* \xleftarrow{\$} \mathbb{F}_p$ and $r_k \xleftarrow{\$} \{0,1\}^\lambda$.
2. Compute the first message $h_k^* = \mathsf{H}(k^*, r_k)$ and $A^* = P^{z^*}/Q^c$.
3. Call Π_{lnk}'s simulator $\Pi_{\mathsf{lnk}}.\mathsf{Sim}$ with (h, h_k^*, c, z^*) to obtain the output π^*.
4. Output $(h_k^*, A^*, c, z^*, \pi^*)$.

For the transcript (h_k, A, c, z, π) generated by the honest interaction between P and V on public input (P, Q, h), where P has witness (x, r). We can show that Sim's output is computationally indistinguishable from the honest transcript via a hybrid argument:

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

$$\mathcal{H}_1 : \left\{ \begin{pmatrix} P, Q, h, h_k, \\ A, c, z, \pi' \end{pmatrix} \middle| \begin{array}{c} k, c \xleftarrow{\$} \mathbb{F}_p; r_k \xleftarrow{\$} \{0,1\}^\lambda; \\ A = P^k; h_k = \mathsf{H}(k, r_k); \\ z = k + cx; \pi' \leftarrow \Pi_{\mathsf{lnk}}.\mathsf{Sim}(h, h_k, c, z) \end{array} \right\}$$

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

$$\mathcal{H}_2 : \left\{ \begin{pmatrix} P, Q, h, h_k^*, \\ A, c, z, \pi'' \end{pmatrix} \middle| \begin{array}{c} k^*, k, c \xleftarrow{\$} \mathbb{F}_p; r_k \xleftarrow{\$} \{0,1\}^\lambda; \\ A = P^k; h_k^* = \mathsf{H}(k^*, r_k); \\ z = k + cx; \pi'' \leftarrow \Pi_{\mathsf{lnk}}.\mathsf{Sim}(h, h_k^*, c, z) \end{array} \right\}$$

$\mathcal{H}_2$ uses a new random k^* to compute h_k^*, and $\Pi_{\mathsf{lnk}}.\mathsf{Sim}$ uses h_k^* to simulate. At first, the SNARK instance (h, h_k^*, c, z) is indistinguishable from (h, h_k, c, z) in $\mathcal{H}_1$, due to the computational hiding property of H.Therefore, the output π' and π'' produced by the polynomial time algorithm $\Pi_{\mathsf{lnk}}.\mathsf{Sim}$ with these instances is also indistinguishable.

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

$\mathcal{H}_3$ represents the transcript computed by the simulator Sim. Similar to Sigma protocol, due to that the distributions of $\{(P, Q, c, z)\}$ 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. $\square$

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

D Circuits in Internal SNARKs

Prove $\mathcal{R}^1_{\mathsf{lnk}}$ in elliptic curve discrete-log problem. Elliptic curve point addition for $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ include the following non-native operations:

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

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

So in total, we need to prove 4×25+20=120 non-native multiplications over the base field of curve P256, several non-native additions and poseidon hash functions.

Prove $\mathcal{R}^4_{\mathsf{lnk}}$ and $\mathcal{R}^1_{\mathsf{lnk}}$ in modular exponentiation discrete-log problem. In $\mathcal{R}^4_{\mathsf{lnk}}$, the most resource-intensive part is proving the modular multiplication of elements in $\mathbb{Z}^*_n$, where n is the RSA modulus of approximately 2000-bit length. In $\mathcal{R}^1_{\mathsf{lnk}}$, we primarily need to prove modular multiplications over prime modulus p and prime modulus q, where $p|q-1$.

In modular multiplication, we need to prove equations of the form "$A \times B = Q \times N + R$," which represents $(A \times B) \bmod N = R$. This process primarily involves proving two large integer multiplications: $A \times B$ and $Q \times N$, along with corresponding range checks for the integers. These range checks can be efficiently handled using lookup arguments [EFG22]. Additionally, further constraints arise from non-native additions and the use of Poseidon hash functions.