Extending Groth16 for Disjunctive Statements

Xudong Zhu[1,2], Xinxuan Zhang[1,2], Xuyang Song[3], Yi Deng[1,2], Yuanju Wei[1,2], and Liuyu Yang[1,2]

[1] Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[3] Anoma
zhuxudong@iie.ac.cn
zhangxinxuan@iie.ac.cn
xuyangsong1012@gmail.com
deng@iie.ac.cn
weiyuanju@iie.ac.cn
yangliuyu@iie.ac.cn

Abstract. Two most common ways to design non-interactive zero knowledge (NIZK) proofs are based on Sigma (Σ)-protocols (an efficient way to prove algebraic statements) and zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) protocols (an efficient way to prove arithmetic statements). However, in the applications of cryptocurrencies such as privacy-preserving credentials, privacy-preserving audits, and blockchain-based voting systems, the zk-SNARKs for general statements are usually implemented with encryption, commitment, or other algebraic cryptographic schemes. Moreover, zk-SNARKs for many different arithmetic statements may also be required to be implemented together. Clearly, a typical solution is to extend the zk-SNARK circuit to include the code for algebraic part. However, complex cryptographic operations in the algebraic algorithms will significantly increase the circuit size, which leads to impractically large proving time and CRS size. Thus, we need a flexible enough proof system for composite statements including both algebraic and arithmetic statements. Unfortunately, while the conjunction of zk-SNARKs is relatively natural and numerous effective solutions are currently available (e.g. by utilizing the commit-and-prove technique), the disjunction of zk-SNARKs is rarely discussed in detail.

In this paper, we mainly focus on the disjunctive statements of Groth16, and we propose a Groth16 variant—CompGroth16, which provides a framework for Groth16 to prove the disjunctive statements that consist of a mix of algebraic and arithmetic components. Specifically, we could directly combine CompGroth16 with Σ-protocol or even CompGroth16 with CompGroth16 just like the logical composition of Σ-protocols. From this, we can gain many good properties, such as broader expression, better prover's efficiency and shorter CRS. In addition, for the combination of CompGroth16 and Σ-protocol, we also present two representative application scenarios to demonstrate the practicality of our construction.

Keywords: Zk-SNARK · Sigma protocol · Disjunctive statement · Logical composition.

1 Introduction

Zero-knowledge proof (ZKP) allows a prover to convince a verifier that a statement is true without revealing any other information. The introduction of zero-knowledge argument systems [42], particularly non-interactive zero-knowledge (NIZK) systems [19], has greatly impacted cryptography research and applications. Over the last decade, significant advancements have been made in zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs). The pairing-based zk-SNARK—Groth16 [45], stands out for its verification efficiency and small proof size. This scheme is widely used in privacy-preserving applications such as verifiable database outsourcing [72], verifiable machine learning [69], privacy-preserving cryptocurrencies [13, 16, 64, 20], electronic voting [73], online auctions [36], and anonymous credentials [33].

Various zk-SNARKs have been developed to prove the non-algebraic statements (e.g. knowledge of preimage of SHA256), which are expressed by arithmetic/boolean circuits supporting general computations in NP. Despite their appeal, these general-purpose schemes can introduce overhead. Separately, there are also algebraic statements which are defined by relations over algebraic structures like prime-order groups. And there are specialized ZKPs for efficiently proving these statements (e.g. knowledge of discrete logarithm). However, the composite statements combining algebraic and non-algebraic statements are common, such as proving a committed value w satisfies both an arithmetic/boolean circuit C (e.g. $\exists w$ s.t. $cm = Com(w) \wedge y = C(w)$). Below we briefly survey ZKPs for different types of statements.

ZKPs for non-algebraic statements. zk-SNARKs are general-purpose, efficient ZKPs used for proving non-algebraic statements due to their compactness and efficient verification. Recent advancements have included works [4, 12, 15, 70, 26, 66, 43] based on polynomial interactive oracle proofs (PIOPs), which do not rely on public-key cryptography, require no trusted setup, and offer conjectured post-quantum security. Other works [58, 35, 25] utilize constant-round PIOPs with KZG [52] polynomial commitment, needing a universal and updatable CRS with constant-size proofs and fast verification. Additionally, works [44, 54, 38, 45] based on linear probabilistic checkable proofs (LPCPs) feature very small proofs and fast verification but are slow on the prover side and require long and "toxic" CRS. By leveraging the KZG polynomial commitment, the recent work [56] combined the ideas of Groth16 and IOPs to trade longer CRS and slower prover's efficiency for concrete smaller proof size and faster verification.

zk-SNARKs can also prove algebraic statements, like the knowledge of discrete-log in a cyclic group, by, for example, representing the exponentiation circuit as Quadratic Arithmetic Programs (QAP). However, proving single exponentiations involves thousands to millions of gates, making zk-SNARKs based on QAP inefficient due to the quasi-linear prover cost and growing CRS size. In contrast, Σ-protocols can prove knowledge of discrete-log with a constant number of exponentiations.

ZKPs for algebraic statements. A Σ-protocol is a 3-move public-coin interactive proof system introduced by Cramer [31]. They are commonly used and ef-

ficient for proving algebraic statements. For example, Alice can use a Σ-protocol to convince Bob that she knows an a such that $a \cdot G = Y$ for publicly known values $G, Y \in \mathbb{G}$. Σ-protocol-based ZKPs are efficient for these statements, yielding short proofs, requiring a constant number of public-key operations, and not needing trusted CRS generation [47, 65, 32, 60, 37, 46]. They can also be made non-interactive using the Fiat-Shamir transformation [34]. For more complex statements, Σ-protocols can be combined in parallel to prove compound statements efficiently. Additionally, there are also efficient ZKP compilers [59, 3, 57] having bridged the gap between Σ-protocol design and working implementations.

While Σ-protocols are efficient for algebraic statements, they are much slower for non-algebraic ones. Consider a cryptographic hash function or block cipher represented by a boolean or arithmetic circuit C. Suppose Alice wants to show that she knows an input w such that $C(w) = y$ for some public y. Alice can treat each gate in C as an algebraic function and prove that the input and output wires of each gate satisfy the related algebraic relation, to show that she indeed knows w. However, this would be prohibitively expensive. The proving and verification time, as well as the proof size, would grow linearly with the circuit size, which for hash functions and block ciphers can be tens of thousands of exponentiations and group elements.

ZKPs for composite statements. Composite statements involve both algebraic and non-algebraic components, such as x being a Pedersen commitment to w with SHA256(w)=y. ZKPs for these statements have various applications [24, 2, 10], including proof of solvency for Bitcoin exchanges, anonymous credentials based on RSA and ECDSA signatures, and 2PC with authenticated inputs.

One approach is transforming composite statements into either algebraic or non-algebraic form, using only Σ-protocols or zk-SNARKs. However, this increases proof size and computation. Instead, a better way is to use Σ-protocols for the algebraic part and efficient zk-SNARKs for the non-algebraic part, linking them with customized "glue" protocols. Many works [2, 22, 21, 5, 55, 62] have constructed such "glue" proofs, featuring low communication costs and efficient verification. Their core idea is to use additional commitments or vector commitments as a bridge to commit the common witness w of two parts, and then use "glue" protocols to prove that the committed values are indeed witness in both two parts. Alternatively, [10] developed transparent ZKPs with a fast prover and linear proof size by linking Σ-protocols with ZKBoo [39]/ZKB++ [23]. Recent work [71] proposed a generic framework of Σ-protocols for algebraic statements from verifiable secret sharing schemes, designing ZKPs for composite statements without "glue" proofs.

ZKPs for disjunctive statements. Zero-knowledge techniques for disjunctive statements have a long history [32, 1, 37]. Disjunctive statements are NP statements composed of a logical "OR" of clauses. For example, Alice can prove to Bob that $x_1 \in \mathcal{L}_1 \vee \cdots \vee x_l \in \mathcal{L}_l$. The witness includes one clause's witness (also called the active clause) and its index. These statements are common in practice, making them crucial for proof optimizations. Disjunctive proofs provide privacy as the verifier cannot determine which clause is satisfied. Applications include

membership proofs like ring signatures [63], proving the existence of bugs in a large codebase [50], and proving the correct execution of a processor [14]. Recent interest in optimizing protocols for disjunctions spans zero-knowledge [46, 28, 53, 50, 40, 7] and secure multiparty computation [11, 49, 51].

Since a disjunction of NP statements is an NP statement, it can be proved using proof systems for NP-completeness. but this may lead to a significant increase in the complexity. Alternatively, works [46, 7, 50] have manually modified specific ZKPs to support disjunctive statements, though these rely on individual protocol structures and may not generalize. A more flexible approach is building disjunctive compilers [32, 1, 40, 41], which transform zero-knowledge protocols into disjunctive protocols with sub-linear communication complexity relative to the number of clauses. Recent work [11, 67, 48] has applied these techniques to the VOLE-based ZK setting.

1.1 Our Contributions

In this paper, we deal with the "OR" composition of the efficient zk-SNARK Groth16 for arithmetic statements (non-algebraic statements) and other Σ-protocols for algebraic statements, or even another Groth16 for arithmetic statements. As we have discussed in introduction section, while the logic composition of Σ-protocols is well-known, most of existing research on zk-SNARKs focuses on "AND" logic composite statements. Using Groth16 to prove disjunctive statements by encoding the entire "OR" statement into the circuit is possible but causes circuit expansion. For the pairing-based Groth16, the CRS length is linear with the circuit size, and the prover's computation is quasi-linear, creating a need for a composition-friendly variant. Our solution separates "OR" logic from the circuit and allows combining two independent proofs for R_1 and R_2 into a proof for $R_1 \vee R_2$ at minimal cost. Additionally, recall that the prover in Σ-OR need to run the entire prover algorithm for the active clause and simulate a transcript for the non-active clause. Therefore, our efficient simulator enables the prover to simulate non-active arithmetic clauses efficiently, improving prover efficiency. The main contributions of this paper are summarized as follows:

Build a bridge between zk-SNARKs and Σ-protocols. By using a Σ-protocol, we create a new zk-SNARK called CompGroth16 in the RO model, which can combine with Σ-protocols to prove disjunctive statements. Specifically, CompGroth16 handle non-algebraic statements and Σ-protocols handle algebraic ones. These can be composed using the classic Σ-OR. Our modular design can work with any Σ-protocol. The main advantage is avoiding the need to encode algebraic parts into the circuit for "OR" statements, increasing efficiency and reducing the size of CRS. For disjunctive statements with an active algebraic clause, a Σ-protocol in Σ-OR can be used. For the arithmetic part, our efficient simulator in Σ-OR eliminates the need to run Groth16's prover algorithm, significantly improving prover's efficiency. We also present two representative applications of our technique for this type of statements.

Construct a new composition friendly zk-SNARK. Following the above contribution, we offer a new framework for proving disjunctions of non-algebraic

statements. This modular framework allows combining two zk-SNARK implementations for R_1 and R_2 into a zk-SNARK scheme for $R_1 \vee R_2$ without rewriting circuits or regenerating the specific CRS. Using our efficient simulator in Σ-OR, only one Groth16 prover algorithm is needed, with the other simulated efficiently. This means the time to prove $R_1 \vee R_2$ is nearly the same as proving R_1 or R_2, depending on the active clause. Additionally, our solution can be extended to efficiently prove more general disjunctions such as $x_1 \in \mathcal{L}_1 \vee \cdots \vee x_l \in \mathcal{L}_l$ or even conjunctive normal form (CNF) relations by using existing optimizations.

2 Preliminaries

2.1 Notation

If S is a finite set, then $s \leftarrow S$ denotes picking an element uniformly from S and assigning it to s. We use $\lambda \in \mathbb{N}$ to denote a security parameter and 1^λ for its unary representation. "Probabilistic polynomial time" is abbreviated as "PPT". By $y \leftarrow A(x_1, \ldots)$ we mean running algorithm A on inputs $x_1, \ldots$ to output y. $\overline{\mathrm{P}}$ and $\overline{\mathrm{V}}$ represent the malicious prover and verifier, respectively. A function $negl(n)$ is negligible if it vanishes faster than any inverse polynomial. A disjunctive statement consists of a logical "OR" of clauses. To distinguish between active clauses, we use a horizontal line above $\mathcal{L}$, e.g., $x_1 \in \bar{\mathcal{L}}_1 \vee x_2 \in \mathcal{L}_2$ means the witness of $x_1 \in \mathcal{L}_1$ is the witness for the entire disjunctive statement.

2.2 Bilinear Groups

Following the notation of [45], we work on bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ with the following properties:
- $\mathbb{G}_1, \mathbb{G}_2$ *and* $\mathbb{G}_T$ are elliptic curve groups of prime order p
- Pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map
- G is a generator for $\mathbb{G}_1$, H is a generator for $\mathbb{G}_2$, $e(G, H)$ is a generator for $\mathbb{G}_T$
- There are efficient algorithms for computing the generic group operations.

For $a, b, c \in \mathbb{Z}_p$, we write $[a]_1$ for $a \cdot G$, $[b]_2$ for $b \cdot H$, and $[c]_T$ for $c \cdot e(G, H)$. For notation $G = [1]_1, H = [1]_2$ and $e(G, H) = [1]_T$, whereas the neutral elements are $[0]_1, [0]_2$ and $[0]_T$. Then, we have $[a]_i + [b]_i = [a + b]_i$ for $i \in \{1, 2, T\}$. Given two group elements $[a]_1$ and $[b]_2$, we define their dot product as $[a]_1 \cdot [b]_2 = [ab]_T$, which can be computed efficiently by pairing e. Sometimes we abbreviate $[a]_{i \in \{1,2,T\}}, [b]_{i \in \{1,2,T\}}, \ldots$ as $[a, b, \ldots]_{i \in \{1,2,T\}}$.

2.3 Quadratic Arithmetic Programs

Quadratic Arithmetic Programs (QAP), which was first introduced by Gennaro *et al.* [38], is a language in which the instances can be verified by using a parallel quadratic check. The efficient reduction between QAP and CIRCUIT-SAT means that when we need an efficient zk-SNARK for CIRCUIT-SAT, we can construct an efficient zk-SNARK for QAP instead. More discussions on encoding an arithmetic circuit to a QAP instance can be found in [38].

Let $\mathbb{F} = \mathbb{Z}_p$. We denote the number of multiplication gates by n and the number of wires by m. QAP instance $\mathcal{Q}_p$ can be specified by ($\mathbb{Z}_p$, l, $\{u_i(X)$, $v_i(X)$, $w_i(X)\}_{i=0}^m$), where $1 \leq l \leq m$ is the length of the statement (e.g., public inputs and outputs in an arithmetic circuit); u_i, v_i *and* w_i are the three sets of polynomials that encode the wires in the target arithmetic circuit. All the polynomials have strictly lower degrees than n and the degree of $t(X)$. QAP instance $\mathcal{Q}_p$ defines the following relation, where we assume that $a_0 = 1$,

$$R = \left\{ (x, w) \middle| \begin{array}{l} x = (a_1, \ldots, a_l) \in \mathbb{Z}_p^l \\ w = (a_{l+1}, \ldots, a_m) \in \mathbb{Z}_p^{m-l} \\ \sum_{i=0}^m a_i u_i(X) \cdot \sum_{i=0}^m a_i v_i(X) \equiv \sum_{i=0}^m a_i w_i(X) \bmod t(X) \end{array} \right\}.$$

Alternatively, $(x, w) \in R$ if there exists (degree $\leq n-2$) polynomial $h(X)$, such that

$$\left(\sum_{i=0}^m a_i u_i(X)\right) \cdot \left(\sum_{i=0}^m a_i v_i(X)\right) - \sum_{i=0}^m a_i w_i(X) \equiv h(X)t(X),$$

In general, the goal of the prover of a zk-SNARK for QAP is to prove that for public $(a_1, \ldots, a_l)$ and $a_0 = 1$, the prover knows $(a_{l+1}, \ldots, a_m)$ and degree $\leq n-2$ polynomial $h(X)$, such that the above equation holds.

2.4 zk-SNARKs

We define $\mathcal{R}_\lambda$ as the set of possible NP relations R the relation generator $\mathcal{R}$ may output given 1^λ. $\mathcal{R}$ may also output some side information, an auxiliary input aux, which is given to the adversary. crs, x, w, τ, and π denote the common reference string, statement, witness, simulation trapdoor, and proof, respectively. Language is a set composed of statements. Given the NP relation R, we can define the language with respect to R as $\mathcal{L} = \{x | \exists\ w\ s.t.\ (x, w) \in R\}$.

Definition 1. (SNARG). $\Pi = (\text{Setup}, \text{P}, \text{V})$ is a succinct non-interactive argument (SNARG) for $\mathcal{R}_\lambda$ if it satisfies the following three properties:

Completeness: *For all* $\lambda \in \mathbb{N}, R \in \mathcal{R}_\lambda, (x, w) \in R$,

$$\Pr\left[\text{V}(R, crs, x, \pi) = 1 \,|(crs, \tau) \leftarrow \text{Setup}\,(R)\,; \pi \leftarrow \text{P}(R, crs, x, w)\right] = 1.$$

Computational Soundness: *For all* $\lambda \in \mathbb{N}$ *and efficient* $\overline{\text{P}}$,

$$\Pr\left[\begin{array}{c} \text{V}(R, crs, x, \pi) = 1 \\ \wedge x \notin \mathcal{L} \end{array} \middle| \begin{array}{c} (R, aux) \leftarrow \mathcal{R}\left(1^\lambda\right); (crs, \tau) \leftarrow \text{Setup}\,(R) \\ (x, \pi) \leftarrow \overline{\text{P}}\,(R, aux, crs) \end{array}\right] = negl(\lambda).$$

Succinctness: *The length of a proof is given by*

$$|\pi| = poly(\lambda) polylog\left(|x| + |w|\right).$$

Definition 2. (SNARK). A succinct non-interactive argument of knowledge (SNARK) is a SNARG that comes together with an extractor χ. Formally, sound-

ness is replaced by knowledge soundness as follows:

Computational Knowledge Soundness: *For all* $\lambda \in \mathbb{N}$ *and PPT* $\overline{\mathrm{P}}$, *there exists a PPT extractor* $\chi_{\overline{P}}$,

$$\Pr\left[\begin{array}{c}\mathrm{V}(R, crs, x, \pi) = 1 \\ \wedge (x, w) \notin R\end{array}\middle|\begin{array}{c}(R, aux) \leftarrow \mathcal{R}\left(1^{\lambda}\right); (crs, \tau) \leftarrow \mathrm{Setup}\left(R\right) \\ ((x, \pi); w) \leftarrow (\overline{\mathrm{P}}||\chi_{\overline{P}})\left(R, aux, crs\right)\end{array}\right] = negl(\lambda).$$

Definition 3. (Zero-knowledge SNARK). A SNARK for an NP language $\mathcal{L}$ with a corresponding NP relation R is computationally zero knowledge, if there exists a simulator Sim for all $\lambda \in \mathbb{N}$,$(R, z) \leftarrow \mathcal{R}(1^{\lambda})$, $(x, w) \in R$ and every PPT distinguisher D

$$\begin{aligned}&\Pr[(crs, \tau) \leftarrow \mathrm{Setup}(R); \pi \leftarrow \mathrm{P}(R, crs, x, w) : \mathrm{D}(R, aux, crs, \tau, \pi) = 1] \\ &\approx \Pr[(crs, \tau) \leftarrow \mathrm{Setup}(R); \pi \leftarrow \mathrm{Sim}(R, \tau, x) : \mathrm{D}(R, aux, crs, \tau, \pi) = 1]\end{aligned}$$

2.5 Σ-protocols

Our constructions utilize a special subclass of interactive zero-knowledge proof systems, called Σ-protocols [65]. Therefore, we recall its definition here [40] .

Definition 4. (Σ-protocols). A Σ-Protocol Π for R is a 3 move protocol between a prover P and a verifier V consisting of a tuple of PPT algorithms $\Pi = (A, Z, \phi)$ with the following interfaces:

- $a \leftarrow A(x, w; r_p)$: On input the statement x, corresponding witness w, such that $R(x, w) = 1$, and prover randomness r_p (we usually omit the randomness r_p as a default input), output the first message a that P sends to V in the first round.
- $c \leftarrow \{0,1\}^{\kappa}$: Sample a random challenge c that V sends to P in the second round. Note that we use a new flexible parameter κ to describe the distribution. The soundness error directly depends on the selection of κ, the larger κ, the smaller soundness error.
- $z \leftarrow Z(x, w, a, c; r_p)$: On input the statement x, the witness w, the challenge c, and prover randomness r_p, output the message z that P sends to V in the third round.
- $b \leftarrow \phi(x, a, c, z)$: On input the statement x, prover's message a, z, and the challenge c, this algorithm run by V, outputs a bit $b \in \{0,1\}$.
- $(a, c, z) \leftarrow S(x)$: On input the statement x, this simulator outputs the simulated prover's message a, c, z, where c is randomly sampled from a distribution.

A Σ-protocol has completeness, special soundness, and special honest verifier zero-knowledge properties (HVZK). We recommend referring to [40] for more formal description of these properties. Notably, every Σ-protocol can be transformed into a non-interactive, fully secure zero knowledge proof in the random oracle (RO, denoted by $\mathcal{O}$) model using the Fiat-Shamir heuristic [34], in which the challenge is generated as $c = \mathcal{O}(x, a)$. The extractor $\mathcal{E}$ and simulator S_{NIZK} can make use of rewinding the other party and programming the random oracle.

2.6 Disjunction of Σ-protocols

The set of relations with Σ-protocols is closed under conjunction and disjunction [32]. The classic protocol for disjunction of Σ-protocols, which we denote Σ-OR, is proposed by Cramer et al. [32]. Then, Ciampi et al. [27] introduced a different Σ-OR protocol with certain advantages over the Cramer et al. construction. We denote Σ_{R_1} and Σ_{R_2} as Σ-protocols for relations R_1 and R_2 respectively. W.l.o.g., we assume that the prover wants to prove the disjunction of the statement $x = (x_1, x_2)$ and knows a witness w_1 showing that $(x_1, w_1) \in R_1$. The proof for relation $R_1 \vee R_2$ is constructed as in **Fig. 1**.

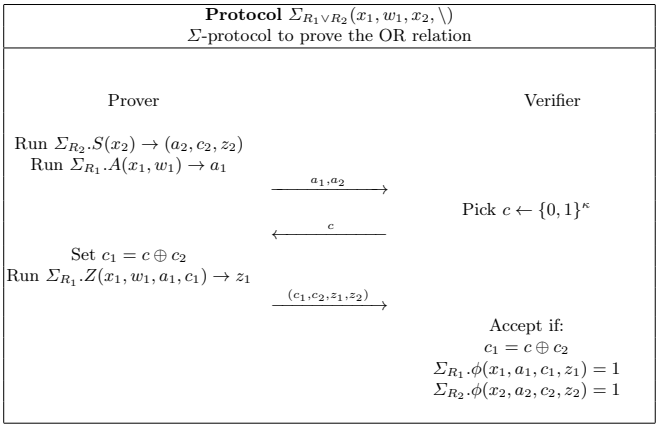

Fig. 1: The Σ-protocol for Relation $R_1 \vee R_2$

The essence of this protocol is that the prover can freely decide one of the challenges from two Σ-protocols, while the other challenge becomes a random challenge due to the influence of the random challenge sent by the verifier. This means that the prover can only simulate the proof for one statement and must then honestly prove the another statement.

3 Framework for Disjunction of Groth16

In this section, we are going to introduce a main construction and a framework. Specifically, in **Subsection 3.1**, we review the well-known zk-SNARK scheme—Groth16. In **Subsection 3.2**, we introduce a new Groth16 variant scheme—CompGroth16 to prove the disjunction of arithmetic and algebraic statements. And we provide two representative applications for this type of statement. Moreover, we also provide a framework for Groth16 to prove more general disjunctive arithmetic statements. In **Subsection 3.3**, we prove the security and analyze the performance of our main construction.

3.1 Groth's Near-Optimal SNARK

The Groth16 [45] is a state-of-the-art scheme for pairing-based zk-SNARKs. Groth16 requires to express the computation as an arithmetic circuit and relies on some trusted setup to prove the circuit satisfiability. Due to its short proof size (3 group elements) and verifier's efficiency (within several milliseconds), Groth16 has become a de facto standard in blockchain projects. This results in a great number of available implementations, code auditing and multiple trusted setup ceremonies run by independent institutions. Therefore, this article takes the construction Groth16 as the starting point. Since our construction is closely related to Groth16, for the convenience of our future discussion, we will briefly review Groth16 in this subsection:

Groth16. Groth16 [45] is a non-universal zk-SNARK for QAP widely used due to its verifier-efficiency. In this paper, we denote the Groth16 scheme by $\Pi_{Groth} = (\text{Groth.Setup}, \text{Groth.P}, \text{Groth.V}, \text{Groth.Sim})$. By m, n and l, we denote the number of wires, multiplication gates and the length of public input of the circuit for $x \in \mathcal{L}$. The statement x and the witness w are expressed as $(a_1, \ldots, a_l)$ and $(a_{l+1}, \ldots, a_m)$ respectively. The Groth16 zk-SNARK scheme, illustrated in **Fig. 2**, involves a prover generating three group elements: $([A, C]_1, [B]_2)$. The verifier executes a single verification equation that requires the computation of three pairings. Here, $[A]_1$ and $[B]_2$ serve as the commitments to the witness, while $[C]_1$ encodes auxiliary information to prove compliance with the specified public input. The elements in CRS are carefully designed to satisfy the soundness. The Groth16 scheme can be proved to satisfy the computational knowledge soundness in the generic group model (GGM) and perfect zero knowledge. Reference to [45] is recommended for more details.

$\text{Groth.Setup}(R)$:

Sample $\alpha, \beta, \gamma, \delta, \zeta \leftarrow \mathbb{F}^*$ such that $\zeta^n \neq 1$. Let

$$crs_p : \left(\left[\alpha, \beta, \delta, \left\{ \zeta^i \right\}_{i=0}^{n-1}, \left\{ \frac{\zeta^i t(\zeta)}{\delta} \right\}_{i=0}^{n-2}, \left\{ \frac{\beta u_i(\zeta) + \alpha v_i(\zeta) + w_i(\zeta)}{\delta} \right\}_{i=l+1}^{m} \right]_1, \left[\beta, \delta, \left\{ \zeta^i \right\}_{i=0}^{n-1} \right]_2 \right)$$

$$crs_v : \left(\left[\left\{ \frac{\beta u_i(\zeta) + \alpha v_i(\zeta) + w_i(\zeta)}{\gamma} \right\}_{i=0}^{l} \right]_1, [\gamma, \delta]_2, [\alpha\beta]_T \right)$$

$crs = (crs_p, crs_v); td = (\alpha, \beta, \gamma, \delta, \zeta)$

Return (crs, td)

$\text{Groth.P}(R, crs_p, a_1, \ldots, a_m)$:

$u(X) \leftarrow \sum_{i=0}^{m} a_i u_i(X); v(X) \leftarrow \sum_{i=0}^{m} a_i v_i(X); w(X) \leftarrow \sum_{i=0}^{m} a_i w_i(X);$

$h(X) \leftarrow (u(X)v(X) - w(X))/t(X)$

$(r, s) \leftarrow \mathbb{F}^2$; $[A]_1 \leftarrow [\alpha]_1 + [u(\zeta)]_1 + r[\delta]_1$; $[B]_2 \leftarrow [\beta]_2 + [v(\zeta)]_2 + s[\delta]_2$

$[C]_1 = \sum_{i=l+1}^{m} a_i \left[\frac{\beta u_i(\zeta) + \alpha v_i(\zeta) + w_i(\zeta)}{\delta} \right]_1 + \left[\frac{h(\zeta)t(\zeta)}{\delta} \right]_1 + s[A]_1 + r[B]_1 - rs[\delta]_1$

Return $\pi \leftarrow ([A, C]_1, [B]_2)$

$\text{Groth.V}(R, crs_v, a_1, \ldots, a_l, \pi = ([A, C]_1, [B]_2))$:

$[D]_1 = \sum_{i=0}^{l} a_i \left[\frac{\beta u_i(\zeta) + \alpha v_i(\zeta) + w_i(\zeta)}{\gamma} \right]_1$

Check that $[A]_1 \cdot [B]_2 = [\alpha\beta]_T + [D]_1 \cdot [\gamma]_2 + [C]_1 \cdot [\delta]_2$

$\text{Groth.Sim}(R, crs, td, a_1, \ldots, a_l)$:

$A \leftarrow \mathbb{F}$; $B \leftarrow \mathbb{F}$; $[D]_1 = \sum_{i=0}^{l} a_i \left[\frac{\beta u_i(\zeta) + \alpha v_i(\zeta) + w_i(\zeta)}{\gamma} \right]_1$

$[C]_1 = \frac{AB[1]_1 - [\alpha\beta]_1 - \gamma[D]_1}{\delta}$

Return $\pi \leftarrow ([A, C]_1, [B]_2)$

Fig. 2: Groth16 construction.

3.2 New Groth16 Variant—CompGroth16

Based on logic "AND" and logic "OR", we can divide all the composite statements into two basic categories. Notably, the "AND" composition between statements with unrelated witnesses is straightforward, e.g. proving that two independent statements x_1, x_2 without a common witness satisfy $(x_1, w_1) \in R_1 \wedge (x_2, w_2) \in R_2$ is trivial. This type of statements can be directly proved by proving that $(x_1, w_1) \in R_1$ and $(x_2, w_2) \in R_2$ respectively.

However, the "AND" composition between statements with related witnesses is more involved, e.g. prove that two statements x_1, x_2 with a common witness satisfy $(x_1, w) \in R_1 \wedge (x_2, w) \in R_2$. Fortunately, numerous studies have shown that introducing "glue" protocols can effectively address such problems without expanding the circuit [2, 22, 21, 5, 55, 62]. On the contrary, the "OR" composition of statements (that is the disjunctive statements) seems to be trickier and has only been well studied for algebraic statements (and for Σ-protocols). For more details, refer to **Subsection 2.6**.

Notably, it is very succinct and efficient to prove the algebraic statements with Σ-protocols. As for arithmetic statements, from the **Subsection 3.1**, we could learn that Groth16 could also be used to prove them with very fast verification and very succinct proof size (constant verification time and proof size). However, when considering disjunctive statements involving both algebraic and non-algebraic components, we face two challenges. Firstly, we cannot simply use the Σ-protocol and Groth16 separately. Secondly, because computing a single exponentiation in secp256k1 will incur 95444 constraints [29] (depending on the group, the number of constraints may be greater), and Groth16 has quasi-linear prover time, directly incorporating "OR" logic into the circuit will result in a significant increase in proof time. Moreover, the CRS size of Groth16 is very large, which includes $m + 2n$ elements in $\mathbb{G}_1$ and n elements in $\mathbb{G}_2$, this also implies rapid expansion as the circuit size increases. Additionally, integrating the "OR" logic directly into the circuit essentially requires a complete proof of the entire "OR" statement, and cannot optimize the non-active clauses.

Motivated by these challenges, we formally propose an efficient solution for proving the disjunctive statements involving both algebraic and non-algebraic components. We complete these compositions by establishing a connection between Groth16 and Σ-protocols, and introducing a new scheme—CompGroth16. This scheme then allows us to perform "OR" composition of CompGroth16 and Σ-protocols in the same way as "OR" composition of Σ-protocols, to prove disjunction of arithmetic and algebraic statements. The main difficulty lies in the construction differences between Groth16 and Σ-protocols, which initially prevents us from establishing such a connection. Notably, Groth16 has a large-sized CRS while Σ-protocol does not. Σ-protocol is a 3 move protocol while Groth16 is non-interactive.

Our construction, CompGroth16, stems from the observation that the essential goal of the Groth16 prover is to demonstrate the validity of the pairing check equation: $[A]_1 \cdot [B]_2 = [\alpha\beta]_T + [D]_1 \cdot [\gamma]_2 + [C]_1 \cdot [\delta]_2$.

Notably, all these group elements are public in Groth16. However, we observe that if we consider the element $[B]_2$ as a witness, we can then use a simple Σ-protocol to prove the following relation R_1 (where $pp = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$):

$$R_1 = \left\{ \begin{array}{c} (pp, x = ([A, D, C]_1 \in \mathbb{G}_1, [\gamma, \delta]_2 \in \mathbb{G}_2, [\alpha\beta]_T \in \mathbb{G}_T); w = [B]_2 \in \mathbb{G}_2) : \\ [A]_1 \cdot [B]_2 = [\alpha\beta]_T + [D]_1 \cdot [\gamma]_2 + [C]_1 \cdot [\delta]_2 \end{array} \right\}$$

The protocol denoted by Σ_{R_1} is a Σ-protocol for relation R_1. As shown in **Fig.3**, this protocol is a public-coin protocol and can be transformed into a non-interactive version using the Fiat-Shamir transformation [34]. We denote the proof of this Σ-protocol (i.e., the transcript $[a]_T, c, [z]_2$) as π_1.

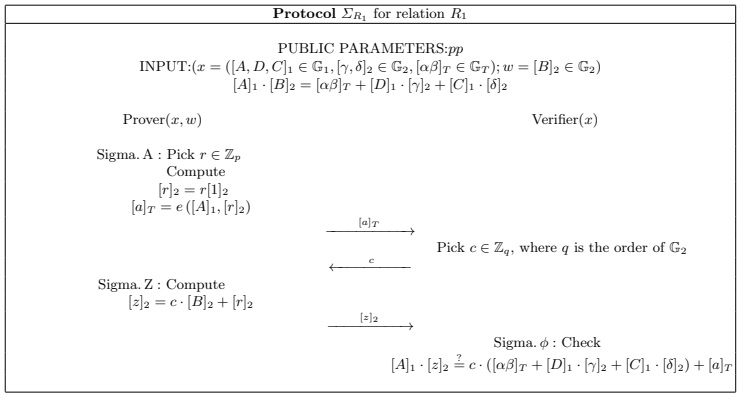

Fig. 3: The Σ-protocol Σ_{R_1} for Relation R_1

Theorem 1. *Σ_{R_1} is a three-move public-coin protocol for relation R_1. It is perfectly complete, unconditionally special sound, and special honest-verifier zero-knowledge (HVZK).*

The proof of **Theorem 1** follows the standard proof method naturally. Specifically, the HVZK simulator Sigma. S picks the challenges $c \leftarrow \mathbb{Z}_p$, generates $[z]_2 \leftarrow \mathbb{G}_2$ and computes $[a]_T = [A]_1 \cdot [z]_2 - c \cdot ([\alpha\beta]_T + [D]_1 \cdot [\gamma]_2 + [C]_1 \cdot [\delta]_2)$. Notably, Σ-protocols can be transformed to be non-interactive in the RO model using the Fiat-Shamir transformation [34]. Knowledge-soundness of the transformed NIZK relies on the special-soundness of the Σ-protocol and therefore requires an extractor to rewind the malicious prover to obtain two transcripts with a shared prefix by programming the RO after rewinding. Zero-knowledge of the NIZK follows from HVZK of the Σ-protocol and programming the RO.

CompGroth16 construction. Now, we formally present a new zk-SNARK $\Pi_{CompGroth}$, which is based on Groth16 and compatible with Σ-protocols. We

implicitly assume the bilinear group parameter is included in R. In addition, we assume that each algorithm checks whether their inputs belong to the correct groups.

Finally, we will describe the main protocol as a non-interactive protocol by using the Fiat-Shamir transformation [34]. Therefore, we denote the concatenation of the statement, elements in the CRS, public input, and proof elements written by the prover up to a certain point in time by *transcript* (this is a common method of simplifying descriptions, just like in [35]). In order to prevent security issues as shown in [17], this definition is necessary.

By the tuple $\Pi_{Groth} = (\text{Groth. Setup}, \text{Groth. P}, \text{Groth. V}, \text{Groth. Sim})$, we have denoted the well-known Groth16 scheme. Now, for the convenience of discussion, we also denote the Σ-protocol for relation R_1 we have mentioned before in **Fig.3** by $\Sigma_{R_1} = (\text{Sigma. A}, \text{Sigma. Z}, \text{Sigma.}\ \phi, \text{Sigma. S})$. On input QAP relation R, we denote our new construction by $\Pi_{CompGroth} = (\text{CompGroth. Setup}, \text{CompGroth. P}, \text{CompGroth. V}, \text{CompGroth. Sim})$. For simpler expression, we denote the hash function by H. This construction is described in detail in **Fig. 4**. The core idea of CompGroth16 is that during the process of running Groth16, we can have the prover to prove the pairing check equation with Σ-protocols, and the proof generated by prover includes $([A, C]_1, \pi_1)$ rather than $([A, C]_1, [B]_2)$.

$\text{CompGroth. Setup}(R)$:
  Run $(crs, td) \leftarrow \text{Groth. Setup}(R)$
Return (crs, td)

$\text{CompGroth. P}(R, crs_p, a_1, \ldots, a_m)$:
  Run $([A, C]_1, [B]_2) \leftarrow \text{Groth. P}(R, crs_p, a_1, \ldots, a_m)$
  Run $[a]_T \leftarrow \text{Sigma. A}([A, C]_1, [\gamma, \delta]_2, [\alpha\beta]_T; [B]_2)$
  Run $c \leftarrow H(transcript)$
  Run $[z]_2 \leftarrow \text{Sigma. Z}([A, C]_1, [\gamma, \delta]_2, [\alpha\beta]_T, [a]_T, c)$
Return $\pi \leftarrow ([A, C]_1, [z]_2, [a]_T)$

$\text{CompGroth. V}(R, crs_v, a_1, \ldots, a_l, \pi = ([A, C]_1, [z]_2, [a]_T))$:
  Compute $[D]_1 = \sum_{i=0}^{l} a_i \left[\frac{\beta u_i(\zeta) + \alpha v_i(\zeta) + w_i(\zeta)}{\gamma}\right]_1$
  Run $c \leftarrow H(transcript)$
  Run $b \leftarrow \text{Sigma.}\ \phi([A, D, C]_1, [\gamma, \delta]_2, [\alpha\beta]_T, [a]_T, c, [z]_2)$
Return b

$\text{CompGroth. Sim}(R, crs, a_1, \ldots, a_l)$:
  Pick random group elements $[A]_1, [C]_1 \leftarrow \mathbb{G}_1$.
  Compute $[D]_1 = \sum_{i=0}^{l} a_i \left[\frac{\beta u_i(\zeta) + \alpha v_i(\zeta) + w_i(\zeta)}{\gamma}\right]_1$
  Run $([a]_T, c, [z]_2) \leftarrow \text{Sigma. S}([A, D, C]_1, [\gamma, \delta]_2, [\alpha\beta]_T)$ (where $c \leftarrow \mathbb{Z}_p$)
  Let $c = \mathcal{O}(transcript)$ (To explain non-interactive security, the programmable $\mathcal{O}$ is required in this non-interactive simulation. In Σ-OR, the HVZK simulator Sigma. S is enough to simulate the non-active arithmetic part, so this step can be omitted)
  Return $\pi \leftarrow ([A, C]_1, [z]_2, [a]_T)$

Fig. 4: CompGroth16 construction $\Pi_{CompGroth}$.

Disjunction of arithmetic and algebraic statements. From the **Fig. 4**, we can see that the current CompGroth. P algorithm in $\Pi_{CompGroth}$ is actually a Σ-protocol. Therefore, it seems easy to perform "OR" composition of CompGroth16 and other Σ-protocols just as "OR" composition of Σ-protocols to prove disjunction of arithmetic and algebraic statements (just as in **Fig. 1**). However, there is a detail that needs our attention here. Since we need to consider two protocols now, namely a CompGroth16 protocol and a Σ-protocol, the distribution of challenge spaces for these two protocols may naturally be inconsistent. W.l.o.g., for two prime p, q, we set the challenge from CompGroth16 to be $c_1 \in \mathbb{Z}_p$, and the challenge from Σ-protocol to be $c_2 \in \mathbb{Z}_q$. When we construct the Σ-protocol for "OR" relation just as in **Fig. 1**, we can just set the c to be a random ele-

ment in $\mathbb{Z}_{pq}$. Then, with the simulated challenge $c_2 \in \mathbb{Z}_q$, we can compute the $c_1 = (c - c_2) \mod p$. Obviously, given that c is a random element in $\mathbb{Z}_{pq}$, c_1 is also a random element in $\mathbb{Z}_p$. The advantage of OR composition of CompGroth16 and Σ-protocol is to prevent the expansion of the circuit by separating the algebraic part and "OR" logic from the circuit, thereby avoiding CRS regeneration, improving the efficiency of the prover and reducing the size of CRS. Moreover, as we will discussed in **Section 3.3**, our technique can also greatly improve the efficiency of provers when the algebraic clause of the disjunctive statement is an active clause.

Two Representative Applications. As we have discussed in **Section 1**, the general disjunctive statements occur commonly in practice, making them an important target for proof optimizations. For example, disjunctive proofs are often used to give the prover some degree of privacy, as a verifier cannot determine which clause is being satisfied. For the specific disjunction of arithmetic and algebraic statements, we also give two examples of use cases here.

The first application is to directly transform Groth16 into a solution that supports designated-verifier property. Note that in scenarios where proof transfer is not desired, only a specific verifier should know if the proof passes verification, such as anonymous transactions in cryptocurrencies. Of course, we can just obtain the designated-verifier schemes from the well-known and efficient "LIPs to designated-verifier zk-SNARK" transformation mentioned in [18]. In [18], Bitansky et al. proposed that a two-move linear-interactive proof can be combined with pairing-based techniques (or additively homomorphic encryption techniques) to obtain a publicly verifiable (or designated-verifier) zk-SNARK. However, this means that compared to the publicly verifiable scheme, we need a completely different set of new CRS and prove/verify algorithms. In fact, we can let the verifier sample a secret a and publish the $Y = a \cdot G$. Then, the prover only need to prove that $x \in \mathcal{L} \vee \exists\ a\ s.t.\ Y = a \cdot G$. This is clearly a designated-verifier scheme, and this solution can be directly achieved by applying our technique on existing Groth16 proof system anytime. Therefore, our technique provides a plug and play interface for the designated-verifier property of Groth16.

The second application is to construct short-lived proofs. Notably, Arun et al. [6] has provided us a framework to construct short-lived proofs from a standard Σ-protocol and a Σ-protocol for the zero knowledge verifiable delay functions (zkVDF): given a Σ-protocol for R_{zkVDF} and any relation R for which we have a Σ-protocol Σ_R, we can use the standard Σ-OR construction to create a disjunction protocol $\Sigma_{R \vee R_{zkVDF}}$ (See theorem 3 in section 7 of [6] for more details). Now, because that our technique has built a bridge between Groth16 and Σ-protocols, we can generalize the conclusions in [6] to any NP relation with generic zero knowledge proofs instead of just the relation for which we have a Σ-protocol.

Disjunction of arithmetic and arithmetic statements. Inspired by OR composition of CompGroth16 and Σ-protocol, we can also provide a framework to perform OR composition of CompGroth16 and CompGroth16. Suppose that there are two Groth16 proof systems $\Pi_{Groth_1}, \Pi_{Groth_2}$ with crs_1 and crs_2 for

R_1 and R_2 respectively. We can transform $\Pi_{Groth_1}, \Pi_{Groth_2}$ into $\Pi_{CompGroth_1}$, $\Pi_{CompGroth_2}$. And then we can obtain the proof for $R_1 \vee R_2$ directly by using Σ-OR. The main advantage of OR composition of CompGroth16 protocols is to obtain the solution for proving $R_1 \vee R_2$ using existing schemes for proving R_1 and R_2 directly, without rewriting the circuit and regenerating the relation specific CRS. This actually broadens the expression range of Groth16 and to some extent alleviates the deficiency of Groth16 in CRS. Moreover, as we will discussed in **Section 3.3**, our technique can also greatly improve the efficiency of provers for proving this type of statements.

More general extensions. Notably, for the reason that we have built a bridge between Groth16 and Σ-protocols, it is natural that we could directly utilize a series of optimizations for disjunction of Σ-protocols such as [40, 41] to prove the more general disjunction such as $x_1 \in \mathcal{L}_1 \vee \cdots \vee x_l \in \mathcal{L}_l$ efficiently. Recall that [40, 41] have built disjunctive compilers, generic approaches that automatically transform large classes of zero-knowledge protocols into disjunctive zero-knowledge protocols with communication complexity sub-linear in the number of clauses. However, the LPCP based zkSNARK Groth16 is not included in these classes. Therefore, from another perspective, our work has expanded the types of protocols that can be applied by these compilers. Moreover, we could also handle statements that contain multiple conjunction and disjunction statements simultaneously. In fact, there are already some optimizations on composition of Σ-protocol for CNF [40, 9, 68, 8]. By building a bridge between Groth16 and Σ-protocols, our technique could also be utilized with these optimizations for handling the general arithmetic CNF. e.g. $(x_1^1 \in \mathcal{L}_1 \vee \cdots \vee x_l^1 \in \mathcal{L}_l) \wedge \cdots \wedge (x_1^k \in \mathcal{L}_1 \vee \cdots \vee x_l^k \in \mathcal{L}_l)$ for arithmetic $\{R_i\}_{i \in [1,l]}$.

Notably, our technique may also be extended to other pairing-based non-interactive arguments such as [35, 25]. However, compared to these two schemes, Groth16 requires our technique more urgently. This is because [35, 25] do not depend on circuit-dependent CRS, and the lookup table technique can be applied to these two schemes to alleviate circuit expansion.

3.3 Security and Performance Analysis

Theorem 2. *Protocol $\Pi_{CompGroth}$ is a non-interactive argument with perfect completeness and perfect zero-knowledge in the RO model. It has computational knowledge soundness in the RO model against adversaries that use only the polynomial number of generic bilinear group operations.*

Proof. **Completeness**: The completeness of this construction comes straightly from the completeness of Groth16 and Σ-protocol.

Knowledge Soundness: Notably, the Σ-proof generated by prover can be transformed to be non-interactive in the RO model. Using the forking lemma discussed in [61], if the prover of the Σ-protocol can find, with non-negligible probability, a valid transcript (a, c, z), the prover of the Σ-protocol can also find another transcript (a, c', z'). This yields an extractor with the expected polynomial time to extract $[B]_2$ which satisfies the Groth16's pairing check verification

equation $[A]_1 \cdot [B]_2 = [\alpha\beta]_T + [D]_1 \cdot [\gamma]_2 + [C]_1 \cdot [\delta]_2$ with probability 1. Now, we get all the statements and proofs appearing in the original Groth16, and the our CRS is the same as the CRS of Groth16. Then, we can extract the witness by running the extractor of original Groth16 in GGM.

Zero-Knowledge: The simulator CompGroth. Sim is constructed in **Fig.4**. On the one hand, our scheme run the Groth. P to obtain the $[A, C]_1$, and from **Fig. 2** we can see that the elements $[A]_1$ and $[C]_1$ in real transcript are blinded by factors r and s respectively. Therefore, the elements $[A, C]_1$ in both real and simulated transcript are independent and random. On the other hand, from the ZK property of the Σ_{R_1}, we can directly know that the distributions of the tuple $([z]_2, [a]_T)$ in both real and simulated transcript are the same. Thus, taking into account all the above discussions, the distributions of $([A, C]_1, [z]_2, [a]_T)$ in both real and simulated transcript are the same, and zero knowledge property holds.

Performance comparison. Notably, our scheme is a specially designed solution for disjunctive statements. Therefore, we give a performance comparison for disjunction of arithmetic and algebraic statements (e.g. $x_1 \in \mathcal{L}_{ari} \vee x_2 \in \mathcal{L}_{alg}$, where $x_2 \in \mathcal{L}_{alg}$ means $\exists\ a\ s.t.\ Y = a \cdot G$) in **Table 1**. By m, n and l, we denote the number of wires, multiplication gates and the length of public input of the circuit for $x \in \mathcal{L}_{ari}$, respectively. By adding a tilde to m, n and l, we want to express the circuit parameters for $x \in \mathcal{L}_{alg}$. In comparison, the number of wires m $(\widetilde{m})$ exceeds the number of multiplication gates n $(\widetilde{n})$, since each gate has an output wire. The statement size l $(\widetilde{l})$ is typical smaller compared to m $(\widetilde{m})$ and n $(\widetilde{n})$. Notably, the exact size of m $(\widetilde{m})$ and n $(\widetilde{n})$ depends on the specific "OR" statement to be proven. Therefore, compared with the traditional method, although sacrificing a bit of proof size, our solution does not require the heavy generation of a new CRS, has smaller CRS size, and has faster prover efficiency. Additionally, it is worth noting that our approach exhibits significant differences in the efficiency of prover based on which clause is the active clause. In the traditional way, in order to prove the "OR" statement, we have to run Groth16's prover algorithm no matter what. However, in our CompGroth16, if the algebraic clause of the disjunctive statement is an active clause, we can use a very efficient simulator in Σ-OR to simulate the proof transcript of the arithmetic part (just as the CompGroth. Sim shown in **Fig. 4**). Therefore, by utilizing our technique, we can completely avoid running the expensive Groth16's prover algorithm for such statements. This is a significant improvement in the efficiency of prover.

We also give a performance comparison for disjunction of arithmetic and arithmetic statements (e.g. $x_1 \in \mathcal{L}_1 \vee x_2 \in \mathcal{L}_2$) in **Table 2**. We use n, m, l and $\bar{n}, \bar{m}, \bar{l}$ to represent the circuit parameters of R_1 and R_2 respectively. Then, we can see from the **Table 2** that compared with the traditional method, although sacrificing a bit of proof size and verifier efficiency, our solution has a significantly faster prover efficiency. The main advantage is that our solution does not require the heavy generation of a new CRS, and we directly use the existing CRS for R_1 and R_2 to prove that $R_1 \vee R_2$. Moreover, by utilizing the efficient simulator we have constructed, we can obtain significant efficiency improvements in prover's efficiency from the non-active clause. Suppose that the circuit scales of R_1 and

R_2 are the same, anyway, our scheme can save more than half of the proving time when proving the disjunction of arithmetic and arithmetic statements.

Table 1: Theoretical comparison for $x_1 \in \mathcal{L}_{ari} \vee x_2 \in \mathcal{L}_{alg}$

	CRS renew	CRS size	Proof size	Prover comp.	Verifier comp.	PPE
Groth16 for $x_1 \in \mathcal{L}_{ari} \vee x_2 \in \mathcal{L}_{alg}$	Yes	$M+2N\lvert\mathbb{G}_1\rvert$, $N\lvert\mathbb{G}_2\rvert$	$\mathbf{2}\lvert\mathbb{G}_1\rvert$, $\mathbf{1}\lvert\mathbb{G}_2\rvert$	$M+3N-LE_1$, NE_2, $\Theta(N)\mathbb{F}$	LE_1, $3P$	1
This work for $x_1 \in \mathcal{L}_{ari} \vee x_2 \in \mathcal{L}_{alg}$	**No**	$\boldsymbol{m+2n}\lvert\mathbb{G}_1\rvert$, $\boldsymbol{n}\lvert\mathbb{G}_2\rvert$	$2\lvert\mathbb{G}_1\rvert$, $1\lvert\mathbb{G}_2\rvert$, $1\lvert\mathbb{G}_T\rvert$, $1\lvert\mathbb{G}_0\rvert$, $3\lvert\mathbb{F}\rvert$	$\boldsymbol{m+3n-lE_1}$, $\boldsymbol{nE_2}$, $\boldsymbol{\Theta(n)}\mathbb{F}$	$\boldsymbol{lE_1}$, $\boldsymbol{3P}$	1
This work for $x_1 \in \mathcal{L}_{ari} \vee x_2 \in \tilde{\mathcal{L}}_{alg}$	**No**	$\boldsymbol{m+2n}\lvert\mathbb{G}_1\rvert$, $\boldsymbol{n}\lvert\mathbb{G}_2\rvert$	$2\lvert\mathbb{G}_1\rvert$, $1\lvert\mathbb{G}_2\rvert$, $1\lvert\mathbb{G}_T\rvert$, $1\lvert\mathbb{G}_0\rvert$, $3\lvert\mathbb{F}\rvert$	$\boldsymbol{lE_1}$, $\boldsymbol{3P}$	$\boldsymbol{lE_1}$, $\boldsymbol{3P}$	1

This table show us the theoretical performance comparison. By $m(\tilde{m}), n(\tilde{n})$ and $l(\tilde{l})$, we denote the number of wires, multiplication gates and the length of public input, respectively. These parameters satisfy $M = m + \tilde{m}, N = n + \tilde{n}, L = l + \tilde{L}$ We use $\mathbb{G}_0$ to represent a standard elliptic curve group (for algebraic part) and $\mathbb{G}_i$ for $i \in \{1, 2, T\}$ to form the bilinear groups (for non-algebraic part)."E_i" represents exponential computations in group $\mathbb{G}_i$ for $i \in \{1, 2, T\}$. "P" means pairing operations. "PPE" represents the number of pairing product equations used to verify a proof. In "Prover comp." column, "$\mathbb{F}$" represents field operations. When we discuss size, we use $|\cdot|$ notation on groups or field operations to represent corresponding elements.And we have omitted some very small constants in this table. The notation Θ represents quasi-linear.

Table 2: Theoretical comparison for $x_1 \in \mathcal{L}_1 \vee x_2 \in \mathcal{L}_2$

	CRS renew	CRS size	Proof size	Prover comp.	Verifier comp.	PPE
Groth16 for $x_1 \in \mathcal{L}_1 \vee x_2 \in \mathcal{L}_2$	Yes	$M+2N\lvert\mathbb{G}_1\rvert$, $N\lvert\mathbb{G}_2\rvert$	$\mathbf{2}\lvert\mathbb{G}_1\rvert$, $\mathbf{1}\lvert\mathbb{G}_2\rvert$	$M+3N-LE_1$, NE_2, $\Theta(N)\mathbb{F}$	$\boldsymbol{LE_1}$, $\boldsymbol{3P}$	**1**
This work for $x_1 \in \tilde{\mathcal{L}}_1 \vee x_2 \in \mathcal{L}_2$	**No**	$M+2N\lvert\mathbb{G}_1\rvert$, $N\lvert\mathbb{G}_2\rvert$	$4\lvert\mathbb{G}_1\rvert$, $2\lvert\mathbb{G}_2\rvert$, $2\lvert\mathbb{G}_T\rvert$	$\boldsymbol{m+3nE_1}$, $\boldsymbol{nE_2}$, $\boldsymbol{\Theta(n)}\mathbb{F}$	LE_1, $6P$	2
This work for $x_1 \in \tilde{\mathcal{L}}_1 \vee x_2 \in \tilde{\mathcal{L}}_2$	**No**	$M+2N\lvert\mathbb{G}_1\rvert$, $N\lvert\mathbb{G}_2\rvert$	$4\lvert\mathbb{G}_1\rvert$, $2\lvert\mathbb{G}_2\rvert$, $2\lvert\mathbb{G}_T\rvert$	$\boldsymbol{\tilde{m}+3\tilde{n}E_1}$, $\boldsymbol{\tilde{n}E_2}$, $\boldsymbol{\Theta(\tilde{n})}\mathbb{F}$	LE_1, $6P$	2

This table show us the theoretical performance comparison. By $m(\tilde{m}), n(\tilde{n})$ and $l(\tilde{l})$, we denote the number of wires, multiplication gates and the length of public input, respectively. These parameters satisfy $M = m + \tilde{m}, N = n + \tilde{n}, L = l + \tilde{l}$. And other explanations are similar to the explanations of **Table 1**.

4 Implementation

We also compare the concrete efficiency of Groth's near-optimal zk-SNARK—Groth16 and our zk-SNARK construction (CompGroth16) during implementation in **Table 3** and **Table 4**. In particular, we distinguish between active clauses, and compare their CRS size (denoted by $|CRS|$), CRS generation time (denoted by CG), proof time (denoted by P), and verification time (denoted by V) across different circuit scales. Similar to the pre-existing implementation of Groth's zk-SNARK in the `arkworks` [30] library, we implemented our zk-SNARKs in the `Rust` library using low-level subroutines of `arkworks`. The specific results were measured in a 64-bit Windows 10 Operating System, which was installed on a standard laptop (Victus by HP Laptop 16-d0xxx), with an Intel `core i5-11400H 2.70 GHz` CPU and 16GB RAM. Each of our data is obtained by running the corresponding algorithm ten times and taking the average.

Table 3: Performance of the implementations for $x_1 \in \mathcal{L}_{ari} \vee x_2 \in \mathcal{L}_{alg}$.

	Groth16				CompGroth16 ($\tilde{\mathcal{L}}_{ari}$)				CompGroth16 ($\tilde{\mathcal{L}}_{alg}$)			
Scales	$\lvert CRS\rvert$	CG	P	V	$\lvert CRS\rvert$	CG	P	V	$\lvert CRS\rvert$	CG	P	V
$n = 2^{14}, \tilde{n} = 2^{12}$	2MB	276ms	273ms	3ms	985KB	120ms	112ms	4ms	985KB	120ms	5ms	4ms
$n = 2^{14}, \tilde{n} = 2^{17}$	12MB	1.2s	1.2s	3ms	985KB	120ms	106ms	4ms	985KB	120ms	5ms	4ms
$n = 2^{16}, \tilde{n} = 2^{12}$	7MB	738ms	691ms	3ms	3MB	425ms	414ms	4ms	3MB	425ms	5ms	4ms
$n = 2^{16}, \tilde{n} = 2^{17}$	12MB	1.3s	1.4s	3ms	3MB	425ms	412ms	4ms	3MB	425ms	5ms	4ms
$n = 2^{18}, \tilde{n} = 2^{12}$	25MB	2.3s	2.5s	3ms	12MB	1.5s	1.5s	4ms	12MB	1.5s	5ms	4ms
$n = 2^{18}, \tilde{n} = 2^{17}$	24MB	2.5s	2.8s	3ms	12MB	1.5s	1.5s	4ms	12MB	1.5s	5ms	4ms

This table show us the specific performance comparison for $x_1 \in \mathcal{L}_{ari} \vee x_2 \in \mathcal{L}_{alg}$ across different circuit scales (we fix the number of witness to be 2^{10}). For the algebraic statements, we prove that $\exists\ a\ s.t.\ Y = a \cdot G$, where G is selected from a native elliptic curve group $\mathbb{G}$ ($\tilde{n} = 2^{12}$, meaning that the base field of $\mathbb{G}$ is the same as the scalar field of BLS12-381 curve) and a non-native ECDSA group ($\tilde{n} = 2^{17}$, which is the secp256k1 curve) respectively.

Table 4: Performance of the implementations for $x_1 \in \mathcal{L}_1 \vee x_2 \in \mathcal{L}_2$.

	Groth16				CompGroth16 ($\tilde{\mathcal{L}}_1$)				CompGroth16 ($\tilde{\mathcal{L}}_2$)			
Scales	$\lvert CRS\rvert$	CG	P	V	$\lvert CRS\rvert$	CG	P	V	$\lvert CRS\rvert$	CG	P	V
$n = 2^{16}, \tilde{n} = 2^{16}$	6MB	797ms	780ms	3ms	6MB	871ms	404ms	8ms	6MB	871ms	400ms	8ms
$n = 2^{16}, \tilde{n} = 2^{18}$	24MB	2.4s	2.5s	3ms	15MB	2s	413ms	8ms	15MB	2s	1.5s	8ms
$n = 2^{16}, \tilde{n} = 2^{20}$	96MB	8.5s	10.2s	3ms	51MB	6.2s	414ms	8ms	51MB	6.2s	6.5s	8ms
$n = 2^{18}, \tilde{n} = 2^{18}$	24MB	3s	3.1s	3ms	24MB	2.9s	1.5s	8ms	24MB	2.9s	1.5s	8ms
$n = 2^{18}, \tilde{n} = 2^{20}$	96MB	9s	11s	3ms	60MB	7.3s	1.6s	8ms	60MB	7.3s	6.4s	8ms
$n = 2^{20}, \tilde{n} = 2^{20}$	96MB	12s	15.8s	3ms	96MB	11.7s	6.7s	8ms	96MB	11.7s	6.7s	8ms

This table show us the specific performance comparison for $x_1 \in \mathcal{L}_1 \vee x_2 \in \mathcal{L}_2$ across different circuit scales (we fix the number of witness to be 2^{10}).

In summary, for arithmetic-algebraic disjunctions, our solution improves prover efficiency (especially with active algebraic clauses) without new CRS. For arithmetic-arithmetic disjunctions, it enables proving multiple statements using existing CRSs, extending Groth16's expressiveness while offering significantly better prover efficiency and acceptable communication.

Acknowledgments. We would like to thank anonymous reviewers from ESORICS 2025 for their valuable suggestions, which have helped us a lot to improve this paper. We are supported by the National Key Research and Development Program of China (Grant No. 2023YFB4503203), the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDB0690200), and the National Natural Science Foundation of China (Grant No. 62372447 and No. 61932019).

References

1. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n signatures from a variety of keys. In: ASIACRYPT'02. pp. 415–432. LNCS 2501, Springer (2002)
2. Agrawal, S., Ganesh, C., Mohassel, P.: Non-interactive zero-knowledge proofs for composite statements. In: CRYPTO'18. pp. 643–673. LNCS 10993, Springer (2018)
3. Akinyele, J.A., Garman, C., Miers, I., Pagano, M.W., Rushanan, M., Green, M., Rubin, A.D.: Charm: a framework for rapidly prototyping cryptosystems. J. Cryptogr. Eng. **3**(2), 111–128 (2013)
4. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: lightweight sublinear arguments without a trusted setup. DCC **91**(11), 3379–3424 (2023)
5. Aranha, D.F., Bennedsen, E.M., Campanelli, M., Ganesh, C., Orlandi, C., Takahashi, A.: ECLIPSE: enhanced compiling method for pedersen-committed zksnark engines. In: PKC'22. pp. 584–614. LNCS 13177, Springer (2022)
6. Arun, A., Bonneau, J., Clark, J.: Short-lived zero-knowledge proofs and signatures. In: ASIACRYPT'22. pp. 487–516. LNCS 13793, Springer (2022)
7. Attema, T., Cramer, R., Fehr, S.: Compressing proofs of k-out-of-n partial knowledge. In: CRYPTO'21. pp. 65–91. LNCS 12828, Springer (2021)
8. Avitabile, G., Botta, V., Friolo, D., Venturi, D., Visconti, I.: Compact proofs of partial knowledge for overlapping CNF formulae. J. Cryptol. **38**(1), 7 (2025)
9. Avitabile, G., Botta, V., Friolo, D., Visconti, I.: Efficient proofs of knowledge for threshold relations. In: ESORICS'22. pp. 42–62. LNCS 13556, Springer (2022)
10. Backes, M., Hanzlik, L., Herzberg, A., Kate, A., Pryvalov, I.: Efficient non-interactive zero-knowledge proofs in cross-domains without trusted setup. In: PKC'19. pp. 286–313. LNCS 11442, Springer (2019)
11. Baum, C., Malozemoff, A.J., Rosen, M.B., Scholl, P.: Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions. In: CRYPTO'21. pp. 92–122. LNCS 12828, Springer (2021)
12. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: CRYPTO'19. pp. 701–732. LNCS 11694, Springer (2019)
13. Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474 (2014)
14. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: Snarks for C: verifying program executions succinctly and in zero knowledge. In: CRYPTO'13. pp. 90–108. LNCS 8043, Springer (2013)
15. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: EUROCRYPT'19. LNCS, vol. 11476, pp. 103–128. Springer (2019)
16. Benet, J., Greco, N.: Filecoin: A decentralized storage network. Protocol Labs pp. 1–36 (2017)

17. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In: ASIACRYPT'12. pp. 626–643. LNCS 7658, Springer (2012)
18. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct non-interactive arguments via linear interactive proofs. In: TCC'13. pp. 315–333. LNCS 7785, Springer (2013)
19. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: STOC'88. pp. 103–112. ACM (1988)
20. Bonneau, J., Meckler, I., Rao, V., Shapiro, E.: Coda: Decentralized cryptocurrency at scale. IACR Cryptol. ePrint Arch. p. 352 (2020)
21. Campanelli, M., Faonio, A., Fiore, D., Querol, A., Rodríguez, H.: Lunar: A toolbox for more efficient universal and updatable zksnarks and commit-and-prove extensions. In: ASIACRYPT'21. pp. 3–33. LNCS 13092, Springer (2021)
22. Campanelli, M., Fiore, D., Querol, A.: Legosnark: Modular design and composition of succinct zero-knowledge proofs. In: CCS'19. pp. 2075–2092. ACM (2019)
23. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: CCS'17. pp. 1825–1842. ACM (2017)
24. Chase, M., Ganesh, C., Mohassel, P.: Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In: CRYPTO'16. pp. 499–530. LNCS 9816, Springer (2016)
25. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., Ward, N.P.: Marlin: Preprocessing zksnarks with universal and updatable SRS. In: EUROCRYPT'20. pp. 738–768. LNCS 12105, Springer (2020)
26. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: EUROCRYPT'20. pp. 769–793. LNCS 12105, Springer (2020)
27. Ciampi, M., Persiano, G., Scafuro, A., Siniscalchi, L., Visconti, I.: Improved or-composition of sigma-protocols. In: TCC 2016-A. pp. 112–141. LNCS 9563, Springer (2016)
28. Ciampi, M., Persiano, G., Scafuro, A., Siniscalchi, L., Visconti, I.: Online/offline OR composition of sigma protocols. In: EUROCRYPT'16. pp. 63–92. LNCS 9666, Springer (2016)
29. circom-ecdsa contributors: Implementation of ecdsa operations in circom (2022), https://github.com/0xPARC/circom-ecdsa
30. arkworks contributors: `arkworks` zksnark ecosystem (2022), https://arkworks.rs
31. Cramer, R.: Modular design of secure yet practical cryptographic protocols (1997), https://api.semanticscholar.org/CorpusID:60892379
32. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: CRYPTO '94. pp. 174–187. LNCS 839, Springer (1994)
33. Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Parno, B.: Cinderella: Turning shabby x.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In: 2016 IEEE Symposium on Security and Privacy (SP). pp. 235–254 (2016)
34. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO'86. pp. 186–194. LNCS 263, Springer (1986)
35. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch. p. 953 (2019)

36. Galal, H.S., Youssef, A.M.: Verifiable sealed-bid auction on the ethereum blockchain. In: Financial Cryptography and Data Security. pp. 265–278. Springer (2019)
37. Garay, J.A., MacKenzie, P.D., Yang, K.: Strengthening zero-knowledge protocols using signatures. J. Cryptol. **19**(2), 169–209 (2006), https://doi.org/10.1007/s00145-005-0307-3
38. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: EUROCRYPT'13. pp. 626–645. LNCS 7881, Springer (2013)
39. Giacomelli, I., Madsen, J., Orlandi, C.: Zkboo: Faster zero-knowledge for boolean circuits. In: 25th USENIX Security Symposium, USENIX Security 16. pp. 1069–1083. USENIX Association (2016)
40. Goel, A., Green, M., Hall-Andersen, M., Kaptchuk, G.: Stacking sigmas: A framework to compose $\varsigma $-protocols for disjunctions. In: EUROCRYPT'22. pp. 458–487. LNCS 13276, Springer (2022)
41. Goel, A., Hall-Andersen, M., Kaptchuk, G., Spooner, N.: Speed-stacking: Fast sublinear zero-knowledge proofs for disjunctions. In: EUROCRYPT'23. pp. 347–378. LNCS 14005, Springer (2023)
42. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1), 186–208 (1989)
43. Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic snarks for R1CS. In: CRYPTO'23. pp. 193–226. LNCS 14082, Springer (2023)
44. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: ASIACRYPT'10. pp. 321–340. LNCS 6477, Springer (2010)
45. Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT'16. pp. 305–326. LNCS 9666, Springer (2016)
46. Groth, J., Kohlweiss, M.: One-out-of-many proofs: Or how to leak a secret and spend a coin. In: EUROCRYPT'15. pp. 253–280. LNCS 9057, Springer (2015)
47. Guillou, L.C., Quisquater, J.: A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In: EUROCRYPT'88. pp. 123–128. LNCS 330, Springer (1988)
48. Hazay, C., Heath, D., Kolesnikov, V., Venkitasubramaniam, M., Yang, Y.: Logrobin++: Optimizing proofs of disjunctive statements in vole-based ZK. IACR Cryptol. ePrint Arch. p. 1427 (2024)
49. Heath, D., Kolesnikov, V.: Stacked garbling - garbled circuit proportional to longest execution path. In: CRYPTO'20. pp. 763–792. LNCS 12171, Springer (2020)
50. Heath, D., Kolesnikov, V.: Stacked garbling for disjunctive zero-knowledge proofs. In: EUROCRYPT'20. pp. 569–598. LNCS 12107, Springer (2020)
51. Heath, D., Kolesnikov, V.: sf logstack: Stacked garbling with o(b log b) computation. In: EUROCRYPT'21. pp. 3–32. LNCS 12698, Springer (2021)
52. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: ASIACRYPT'10. pp. 177–194. LNCS 6477, Springer (2010)
53. Kolesnikov, V.: $$\mathsf {Free\ }{} \mathtt{IF} $$: How to omit inactive branches and implement S -universal garbled circuit (almost) for free. In: ASIACRYPT'18. pp. 34–58. LNCS 11274, Springer (2018)
54. Lipmaa, H.: Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In: TCC'12. pp. 169–189. LNCS 7194, Springer (2012)
55. Lipmaa, H.: On black-box knowledge-sound commit-and-prove snarks. In: ASIACRYPT'23. pp. 41–76. LNCS 14439, Springer (2023)

56. Lipmaa, H.: Polymath: Groth16 is not the limit. In: CRYPTO'24. pp. 170–206. LNCS 14929, Springer (2024)
57. Lueks, W., Kulynych, B., Fasquelle, J., Bail-Collet, S.L., Troncoso, C.: zksk: A library for composable zero-knowledge proofs. In: Proceedings of WPES@CCS 2019. pp. 50–54. ACM (2019)
58. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In: Proceedings of the 2019 ACM SIGSAC Conference on CCS'19. pp. 2111–2128. ACM (2019)
59. Meiklejohn, S., Erway, C.C., Küpçü, A., Hinkle, T., Lysyanskaya, A.: ZKPDL: A language-based system for efficient zero-knowledge proofs and electronic cash. In: 19th USENIX Security Symposium. pp. 193–206. USENIX Association (2010)
60. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: EUROCRYPT'96. pp. 387–398. LNCS 1070, Springer (1996)
61. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Advances in Cryptology — EUROCRYPT '96. pp. 387–398. Springer (1996)
62. Raymond, M., Evers, G., Ponti, J., Krishnan, D., Fu, X.: Efficient zero knowledge for regular language. IACR Cryptol. ePrint Arch. p. 907 (2023)
63. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: ASIACRYPT'01. pp. 552–565. LNCS 2248, Springer (2001)
64. Rondelet, A., Zajac, M.: ZETH: on integrating zerocash on ethereum. CoRR **abs/1904.00905** (2019)
65. Schnorr, C.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991), https://doi.org/10.1007/BF00196725
66. Setty, S.T.V.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: CRYPTO'20. pp. 704–737. LNCS 12172, Springer (2020)
67. Yang, Y., Heath, D., Hazay, C., Kolesnikov, V., Venkitasubramaniam, M.: Batchman and robin: Batched and non-batched branching for interactive ZK. In: Proceedings of the 2023 ACM SIGSAC Conference on CCS'23. pp. 1452–1466. ACM (2023)
68. Zeng, G., Lai, J., Huang, Z., Wang, Y., Zheng, Z.: Dag-Σ: A dag-based sigma protocol for relations in CNF. In: ASIACRYPT'22. pp. 340–370. LNCS 13792, Springer (2022)
69. Zhang, J., Fang, Z., Zhang, Y., Song, D.: Zero knowledge proofs for decision tree predictions and accuracy. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. p. 2039–2053. CCS '20, ACM (2020)
70. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: 2020 IEEE Symposium on Security and Privacy, SP 2020. pp. 859–876. IEEE (2020)
71. Zhang, M., Chen, Y., Yao, C., Wang, Z.: Sigma protocols from verifiable secret sharing and their applications. In: ASIACRYPT'23. pp. 208–242. LNCS 14439, Springer (2023)
72. Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vsql: Verifying arbitrary sql queries over dynamic outsourced databases. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 863–880 (2017)
73. Zhao, Z., Chan, T.H.H.: How to vote privately using bitcoin. In: Information and Communications Security. pp. 82–96. Springer International Publishing (2016)