Fast and Designated-verifier Friendly zkSNARKs in the BPK Model

Xudong Zhu[1,2], Xuyang Song[3], and Yi Deng[1,2]

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

Abstract. In ASIACRYPT 2016, Bellare et al. first demonstrated that it is impossible to achieve subversion soundness and standard zero knowledge simultaneously. Subsequently, there have been lots of effort to construct zero-knowledge succinct non-interactive arguments of knowledge protocols (zk-SNARKs) that satisfy subversion zero knowledge (S-ZK) and standard soundness from the zk-SNARK in the common reference string (CRS) model. The various constructions could be regarded secure in the bare public key (BPK) model because of the equivalence between S-ZK in the CRS model, and uniform non-black-box zero knowledge in the BPK model has been proved by Abdolmaleki et al. in PKC 2020.

In this study, We proposed the first publicly verifiable non-uniform ZK zk-SNARK scheme in the BPK model maintaining comparable efficiency with its conventional counterpart, which can also be compatible with the well-known transformation proposed by Bitansky et al. in TCC 2013 to obtain an efficient designated-verifier zk-SNARK. We achieve this goal by only adding a constant number of elements into the CRS, and using an unconventional but natural method to transform Groth's zk-SNARK in EUROCRYPT 2016. In addition, we propose a new speed-up technique that provides a trade-off. Specifically, if a logarithmic number of elements are added into the CRS, according to different circuits, the CRS verification time in our construction could be approximately $9\% - 23\%$ shorter than that in the conventional counterpart.

Keywords: Subversion zero knowledge · SNARK · Common reference string · Bare public key · Random Oracle · Generic group model.

1 Introduction

The proposal of a zero-knowledge argument system [29], especially the non-interactive zero-knowledge argument system (NIZK) [14], has a significant effect on both cryptography theory research and the application field of cryptography. In the last decade, remarkable progress has been made in research on zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs). After various studies [35, 41, 30, 38, 27], Groth constructed an pairing-based zk-SNARK Groth16 [31] with the best verification efficiency and proof size. However, many zk-SNARK constructions in the common reference string (CRS) model, including Groth16, may encounter attacks during the setup process. That is, the trusted setup necessary for this type of zk-SNARK to be secure is difficult to achieve in reality. There are many real-life subversion cases, the most well-known of which have recently attracted increasing interest in constructing cryptographic primitives and protocols secure against active subversion. Although universal updatable zk-SNARKs [33, 40, 26, 20, 19] and transparent zk-SNARKs [10, 44, 11, 21, 43, 18, 17] have bypassed the setup subversion, there is still a certain gap in verification efficiency or proof length between these schemes and zk-SNARKs with trapdoor in CRS such as Groth16.

A direct research direction involves improving the existing efficient scheme that is secure in the CRS model to be secure against parameter-subversion adversaries. To study the type of security that needs to be maintained for parameter subversion, Bellare et al. first formalized the notion of subversion soundness (S-SND) and subversion zero knowledge (S-ZK) in [9]. They demonstrated that S-SND and S-ZK cannot be satisfied simultaneously and constructed a NIZK scheme to demonstrate that a protocol can satisfy soundness and S-ZK. To characterize parameter subversion better, Abdolmaleki et al. proposed some stronger security definitions in [1] than those proposed in [9]. By adding extra elements to the CRS of Groth16 and running an additional CRS verification algorithm, the scheme in [1] improved Groth16 to be knowledge sound and S-ZK under a knowledge assumption. Fuchsbauer [25] demonstrated that some quadratic arithmetic program (QAP)-based zk-SNARKs, including Groth16, can be adapted to achieve knowledge soundness and S-ZK. Specifically, under a new knowledge assumption and with a new reduction technique, Fuchsbauer improved Groth16 to become a knowledge sound and S-ZK without changing the original CRS in [25]. Then Abdolmaleki et al. proposed an improved version of [1] in [4], they optimized the original version in detail and utilized the same reduction technique as [25]. In addition, there are some studies about the improvement of zk-SNARKs to guarantee S-ZK and simulation extractability (SE) simultaneously, such as [34, 7, 39]. As pointed out in [8], the SE constructions given in [16, 8, 6] can also satisfy S-ZK, as they use the original CRS of Groth16 to achieve SE and ZK.

To associate the subversion model with the BPK model, Abdolmaleki et al. [3] proved that S-ZK in the CRS model is equal to the no-auxiliary-string non-black-box zero knowledge in the BPK model. Recently, Fauzi et al. [23] proposed a general theoretical framework to demonstrate that one could construct a knowledge sound S-ZK NIZK proof from any knowledge sound subversion

witness indistinguishable (S-WI) NIWI proof and keyless verifiably-extractable generalized extractable one-way functions (VEGOWFs). Abdolmaleki et al. [2] starts from LAMASSU construction [5], and for the first time, show how one can achieve UC-secure updatable circuit-succinct NIZK, by converting this framework to a black-box version. They achieve this by using the $\Sigma-$protocol and Fiat Shamir (FS) transformation paradigm to prove some relations related to their framework and then utilize the black box extraction feature of this paradigm.

1.1 Our Contributions

In this study, we extend the paradigm used in [2] to deal with some more complex relations and leverage the non-uniform resistant extraction feature of this paradigm to achieve our goals. Specifically, we carefully design an unconventional but natural method, Sigma-CRS Verification, to obtain a zk-SNARK that is secure in the BPK model and with some good properties. Additionally, we present a new speed-up technique, Compute-with-Help Mechanism, to optimize our construction. We believe that our work takes a step towards finding an acceptable technique for decentralization in terms of both theory and practice.

First, we note that the $\Sigma-$protocol and FS transformation paradigm has been used in [2] to achieve the black-box extraction and some results about the updatable zkSNARK with well structured CRS. To extend this paradigm in the BPK model with more complex CRS, we carefully design a new Sigma-CRS verification algorithm to achieve the non-uniform resistant extraction, and by which we construct a variant of Groth16 that satisfies auxiliary-string zero knowledge in the BPK model without using the newly proposed knowledge assumption. Moreover, by making a slight modification without adding any additional assumptions, our construction can achieve both non-uniform ZK in the BPK model and SE. Following our analysis, we point out that the scheme in [34] actually also satisfies these properties.

In addition, our approach is modular, and the core idea of our construction could be used to modify many other schemes that are secure in the CRS model to be secure in the BPK model, or simply to replace the pairing checks in other subversion resistant schemes to obtain schemes with better properties. In particular, if we combine our technique with a variant construction [16] of Groth16, we can directly achieve both non-uniform ZK in the BPK model and SE without adding extra assumptions.

Second, we propose a new technique, Compute-with-Help Mechanism, to speed up the computation of the prover. The essence of our technique is the division and recursion. This technique provides a trade-off between the size of the public file and the running time of the prover.

Although we only need to add a constant number of extra elements into the public file to ensure the correct verification of the elements in the public file, our modified subversion resistant scheme is still at the cost of efficiency of the prover, just like the previous schemes. In fact, Abdolmaleki et al. have raised how to design a method for minimizing the computational complexity of CRS verification as an interesting open question in [1]. Fortunately, we found that if

we are allowed to add a logarithmic number of elements into the original public file, we can use our speed-up technique to reduce some computation done by the prover from the linear level to the logarithmic level. Moreover, we found that in some cases our new technique could also be regarded as a speed-up technique to the multi-scalar multipication (MSM) which is a fundamental computational problem in the field of cryptography, especially zk-SNARK. So we proposed our new technique as an independent interest.

Third, we provide a solution to efficiently implement the designated-verifier subversion-resistant zk-SNARK. In fact, almost all known zk-SNARKs that are secure in the BPK model are only discussed in the publicly verifiable setting, and unfortunately both of them could not be implemented with the well-known and efficient "LIPs to designated-verifier zk-SNARK" transformation proposed by Bitansky et al. in [13] to obtain a very efficient designated-verifier zk-SNARK, which could be just implemented with the standard (generic) group rather than the bilinear group. Specifically, other subversion-resistant zk-SNARKs require pairing checks to ensure security. However, when we implement these subversion-resistant zk-SNARKs in the designated-verifier setting with the well-known and efficient "LIPs to designated-verifier zk-SNARK" transformation, the pairing-check technique never works again because the elements in the CRS are encoded by the homomorphic encryption scheme rather than the bilinear group scheme. Fortunately, our construction will not be affected by the application scenario, and the "LIPs to designated-verifier zk-SNARK" transformation could be used to obtain very efficient zk-SNARKs that are secure in the BPK model and designated-verifier setting directly.

1.2 Technique Overview

We aim to construct an efficient zk-SNARK that is secure in the BPK model. Therefore, we consider Groth16 as the starting point owing for its good efficiency. Because S-ZK in the CRS model has been proven equal to the no-auxiliary-string non-black-box zero knowledge in the BPK model [3], we will discuss our idea mainly in the BPK model for convenience.

Some research on how to modify Groth16 to be secure in the parameter subversion model have been conducted, such as those in [1, 25, 4]. These constructions have provided a good way to modify Groth16 to achieve S-ZK. Specifically, we can let the verifier generate CRS and then let the prover run pairing checks to verify the well-formness of CRS. However, the zk-SNARKs obtained by this method have some limitations, e.g., they do not satisfy non-uniform security and are not compatible with the well-known and efficient "LIPs to designated-verifier zk-SNARK" transformation. In addition, the additional CRS verification algorithm always includes many exponential computations, which are expensive.
A new construction with better application prospects. A natural question is how to solve the existing theoretical limitations of the zk-SNARKs that are secure in the BPK model. We observed that it is necessary for most subversion resistant schemes to be S-ZK; that is, they are non-auxiliary-string zero knowledge in the BPK model, which means that the ZK property of these schemes

cannot hold against non-uniform adversaries. In [28], Goldreich and Oren proved that an auxiliary string (even non-black box) zero-knowledge argument system for languages outside the BPP require at least three messages in the plain model. However, an auxiliary string (non-black-box) NIZK argument system in the BPK model can be interpreted as a two-message auxiliary string (non-black-box) zero-knowledge argument system in the plain model, where the verifier creates CRS and sends it as the first message. Therefore, the natural idea is to select a stronger model first, such as the knowledge assumption or the generic model. To prove the zero-knowledge property, [1, 25, 4] adopted knowledge assumptions to extract the trapdoor required by the simulator. However, these knowledge assumptions could not lead to auxiliary-string zero knowledge. This is because the assumptions do not hold with auxiliary-string. Thus, we require another strong model. In addition, almost all previous constructions in this field require pairing checks, but the bilinear group hinders the efficient implement of these schemes in the designated-verifier setting.

While the $\Sigma-$protocol plus FS transformation paradigm has been used in [2] to achieve the UC security of updatable zkSNARK (whose CRS has a simple structure), we observed that the scheme discussed in [2] only focus on the black box extraction feature of the paradigm on the one hand, and only deal with the CRS with simple structure such as some g powers on the other hand, to achieve their goals. Fortunately, by extending this paradigm to deal with more complex CRS and exploring the non-uniform resistant extraction feature of the paradigm, we could utilizing the power of RO to bypass this lower bound to prove that our construction satisfies the auxiliary-string zero knowledge in the BPK model while simultaneously eliminating the need for pairing. Specifically, by carefully $\Sigma-$protocols and strategy designing, we found that the pairing checks of the well-formness of CRS (with more complex algebraic structures) could all be proven by $\Sigma-$protocols, and $\Sigma-$protocols could be transformed to be non-interactive using the Fiat Shamir transformation [24] in the RO model. This means that we can eliminate constraints at the conceptual level. In the BPK model, instead of following the conventional way in which the verifier generates the highly structured CRS and then the prover computes pairing checks, we could try another way in which the verifier generates the highly structured CRS together with the Σ-proofs, and the prover verifies the Σ-proofs to ensure that the highly structured CRS is well formed. In this way, the trapdoor needed by the simulator could be extracted by the forking lemma [42] (instead of using the newly proposed knowledge assumption), which still holds with the auxiliary string. Moreover, we will discuss that the scheme constructed by this way could be very compatible with the "LIPs to designated-verifier zk-SNARK" transformation, so it will have a very efficient implementation in the designated-verifier setting.

A new speed-up technique to speed up the CRS verification. The extra cost of the transformation from security in the CRS model to security in the BPK model is mainly at the time of the prover. Fortunately, we observed that a change in conception can bring us more. That is now that we allow the verifier to

generate Σ-proofs in our construction, what happens if the verifier is not limited to proving only the well-formness of CRS? In fact, the verifier who generates the CRS naturally knows extra information about the CRS (e.g., knowledge of public key trapdoors that are used to generate the CRS). If we use the extra power of the verifier fully and let the verifier generate more proofs, it is desirable to speed up the computation of the prover further. For example, supposing c is a publicly known field element, n is equal to a power of 2 and the $\left\{g^{x^i}\right\}_{i\in[1,n]}$ are well-formed, when the prover need to compute $g^{cx+c^2x^2+\cdots+c^nx^n}$ by $O(n)$ exponential multiplication operations, the prover only needs to compute $g^{cx+c^2x^2+\cdots+c^{\frac{n}{2}}x^{\frac{n}{2}}}$ by $O\left(\frac{n}{2}\right)$ exponential multiplication operations, and get the

$$\left(g^{cx+c^2x^2+\cdots+c^{\frac{n}{2}}x^{\frac{n}{2}}}\right)^{c^{\frac{n}{2}}x^{\frac{n}{2}}}$$

together with the proof for the exponential equality relation

$$\left(P=\left(g^{cx+c^2x^2+\cdots+c^{\frac{n}{2}}x^{\frac{n}{2}}}\right)^{c^{\frac{n}{2}}x^{\frac{n}{2}}},Q=g^{c^{\frac{n}{2}}x^{\frac{n}{2}}}\right)$$

from the verifier. By ensuring that this proof is passed, the prover can directly compute $g^{cx+c^2x^2+\cdots+c^nx^n}$ using one multiplication operation. Evidently, $g^{cx+c^2x^2+\cdots+c^{\frac{n}{2}}x^{\frac{n}{2}}}$ can also be computed recursively with the verifier's help as above. Thus, with the help of the verifier, the prover can compute $g^{cx+c^2x^2+\cdots+c^nx^n}$ using only $O(\log n)$ exponential multiplication operations.

The essence of our speed-up technique is to delegate some of the zk-SNARK prover's computations in the CRS verification algorithm to the zk-SNARK verifier. One may doubt the significance of reducing some of the prover's computations to increase the verifier's computations. We need to emphasize that in the BPK model, the CRS generation algorithm run by the verifier can be computed offline and one-shot for a specific relation, while the CRS verification algorithm needs to be run by each potential prover. So we are more concerned about the computations of the prover in the BPK model.

In fact, in some application scenarios with a large number of provers and the specific relation, the advantages brought by our speed-up technique are significant for the entire system. For example, in an attribute-based anonymous credential system, there are many users with their credentials presented by the organization (or issuer), and many service providers with their different access control requirements. The access control requirement of each service provider is clearly defined, in other words, the relation that need to be proven to each service provider is specific. So each service provider only need to run the CRS generation algorithm offline and one-shot, then to obtain services from a certain service provider, a large number of users will verify the CRS and prove that they meet the requirement.

1.3 On Efficiency

Since our new zk-SNARK is closely related to the most efficient known zk-SNARK of Groth [31], both the Prove and Verify algorithms in our scheme have almost the same computational complexity as that of Groth16. As we have discussed in **Subsection 1.2**, the CRS generation algorithm can be run offline and one-shot for a specific relation. Thus, we focus on the computational complexity of the CRS verification algorithm run by prover. In [1], Abdolmaleki et al. proposed how to minimize the computational complexity of CRS verification as an interesting open question. Unfortunately, not so much optimization is possible here. The CRS verification algorithm has to have running time at least linear in the size of the CRS since the zk-SNARK prover has to read the whole CRS. Thus, the best we can hope is to improve the constant factor, which is exactly what we did. Combined with our new speed-up technique, the CRS verification time of our construction could be approximately $9\% - 23\%$ shorter than that of conventional pairing checks (Table 1). Specifically, the efficiency improvement of our construction is more significant for circuits with a large proportion of the multiplication gates. For asymptotic comparison, we transform some computations with $O(n)$ complexity into computations with $O(\log n)$ complexity (Table 2).

2 Preliminaries

2.1 Notation

We denote Φ as the empty string. If x is a binary string, $|x|$ represents its length. If S is a finite set, $|S|$ denotes its size and $s \leftarrow S$ denotes picking an element uniformly from S and assigning it to s. We used $\lambda \in \mathbb{N}$ to denote a security parameter and 1^λ to denote its unary representation.

All algorithms were non-uniform and randomized unless otherwise indicated. "PPT" stands for "probabilistic polynomial time." By $y \leftarrow A(x_1, \ldots)$ we denote the operation of running algorithm A on inputs $x_1, \ldots$ and omitted coins r to output y. We used $\overline{\mathrm{P}}$ and $\overline{\mathrm{V}}$ for the malicious prover and verifier, respectively. Generally, any algorithm with a horizontal bar above it in our study represents a malicious algorithm. A function $negl(n)$ is considered negligible if it vanishes faster than any inverse polynomial. Sometimes, we abuse the concept of CRS, and the concept of public key can be called CRS in the BPK model.

2.2 Bilinear Groups

Following the notation of [31], we work on bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ using the following properties:
- $\mathbb{G}_1, \mathbb{G}_2$ *and* $\mathbb{G}_T$ are groups of prime order p
- Pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear map
- g is a generator for $\mathbb{G}_1, h$ is a generator for $\mathbb{G}_2$, and $e(g, h)$ is a generator for $\mathbb{G}_T$

- There are efficient algorithms for computing group operations, evaluating the bilinear map, deciding membership of the groups, deciding equality of group elements, and sampling generators of the groups. We refer to these operations as the generic group operations.

It is useful and convenient to use a notation that represents group elements based on their discrete logarithms. We write $[a]_1$ for $g^a, [b]_2$ for h^b, and $[c]_T$ for $e(g,h)^c$. For notation $g = [1]_1, h = [1]_2$ and $e(g,h) = [1]_T$, whereas the neutral elements are $[0]_1, [0]_2$ and $[0]_T$. We can then use the additive notation in all groups, and 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.

2.3 Definition of zk-SNARKs in the BPK Model

We define $\mathcal{R}_\lambda$ as the set of possible relations R the relation generator $\mathcal{R}$ may output given 1^λ. $\mathcal{R}$ may also output some side information, an auxiliary input z, which is given to the adversary. crs, u, w, τ, and R denote the common reference string, statement, witness, simulation trapdoor, and relation, respectively.

zk-SNARKs

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

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

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

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

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

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

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

Definition 2. *(SNARK). A succinct non-interactive argument of knowledge (SNARK) is a SNARG that comes together with an extractor χ. Formally, soundness 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 $\chi_{\overline{P}}$,*

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

Definition 3. *(Zero-knowledge SNARK). A SNARK for an NP language L with a corresponding NP relation R is computationally zero knowledge, if there exists*

a simulator Sim *for all* $\lambda \in \mathbb{N}$,$(R, z) \leftarrow \mathcal{R}(1^\lambda)$, $(u, w) \in R$ *and every PPT distinguisher* D

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

If we consider the unbounded distinguisher D*, we obtain the definition of the statistic ZK. If we replace the notation* $\approx$ *with* $=$*, we obtain the definition of a perfect ZK.*

zk-SNARKs in the BPK model

The BPK model can be regarded to work in two phases, the key-registration and proof phases.

First, the verifier registers public key pk (the honest verifier is supposed to store the corresponding secret key sk) in public file F in the key-registration phase. Formally, we regard F as a set, and it is initialized as an empty set Φ. The following event will occur in this phase:

$$(R, z) \leftarrow \mathcal{R}\left(1^\lambda\right), (pk, sk) \leftarrow \overline{\mathrm{V}}\left(R\right), F = F \cup pk$$

We implicitly assumed that the bilinear group parameter is included in R. In this phase, $\overline{\mathrm{V}}$ outputs (pk, sk) rather than (pk, τ) because instead of the simulation trapdoor, which is denoted as τ and is used to simulate the transcripts, we regard sk as the public key trapdoor that is used to generate pk and τ. Notably, τ is not always equal to sk, (e.g., the Groth16 construction's CRS generation need $(x, \alpha, \beta, \gamma, \delta)$; however, as Fuchsbauer has proven in [25], it suffices to know (x, δ) for simulation).

Second, in the proof phase, by oracle accessing F at the prover's and verifier's will, the prover outputs the proof π on an input (R, u, w), then the verifier verifies the π on an input (R, u, sk, π).

Specifically, an efficient prover publicly verifiable non-interactive argument for R in the BPK model includes a set initiated to be $F = \Phi$ and a tuple of probabilistic polynomial algorithms (AddF, Check^F, Prove^F, Verify^F, Sim^F) such that

- $(pk, sk) \leftarrow \mathrm{AddF}(R)$: On input the relation R, this algorithm output the public key pk and the public key trapdoor sk. It is run by the verifier in the key-registration phase. Public file F is usually set to $F = F \cup pk$.
- $0/1 \leftarrow \mathrm{Check}^F(R)$: This algorithm is run by a prover who has access to public file F. It outputs 1 if and only all the specified elements in pk are well-formed.
- $\pi \leftarrow \mathrm{Prove}^F(R, u, w)$: On input relation R, statement u, witness w, and public file F, the algorithm output the argument π.
- $0/1 \leftarrow \mathrm{Verify}^F(R, u, sk, \pi)$: This algorithm has access to public file F and output 1 if and only the verification equation holds.

– $\pi \leftarrow \mathrm{Sim}^F(R, u, \tau)$: The simulator inputs statement u and simulation trapdoor τ (which could be computed by the public key trapdoor sk) and returns argument π.

Definition 4. *(Zero knowledge SNARK in the BPK model). With the set F, $\Pi = (\mathrm{AddF}, \mathrm{Check}^F, \mathrm{Prove}^F, \mathrm{Verify}^F)$ is a zero-knowledge SNARK in the BPK model for an NP language L with a corresponding NP relation R if Π satisfies the following five properties (the F in each definition is assumed to be initialized as Φ):*

Completeness: *for all* $\lambda \in \mathbb{N},\ (R, z) \leftarrow \mathcal{R}(1^\lambda),\ (u, w) \in R,$

$$\Pr\left[\mathrm{Verify}^F(R, u, \pi) = 1 \,\middle|\, \begin{array}{c}(pk, sk) \leftarrow \mathrm{AddF}(R); F = F \cup pk \\ \pi \leftarrow \mathrm{Prove}^F(R, u, w)\end{array}\right] = 1.$$

Public-key Verifiability: *for all* $\lambda \in \mathbb{N},\ (R, z) \leftarrow \mathcal{R}(1^\lambda),$

$$\Pr\left[\mathrm{Check}^F(R) = 1 \,\middle|\, (pk, sk) \leftarrow \mathrm{AddF}(R); F = F \cup pk\right] = 1.$$

Computational Knowledge Soundness: *for all* $\lambda \in \mathbb{N},\ (R, z) \leftarrow \mathcal{R}(1^\lambda)$ *and efficient* $\overline{\mathrm{Prove}}^F$*, there exists a PPT extractor* $\chi_{\overline{P}}$*,*

$$\Pr\left[\begin{array}{c}\mathrm{Verify}^F(R, u, \pi) = 1 \\ \wedge (u, w) \notin R\end{array} \,\middle|\, \begin{array}{l}(pk, sk) \leftarrow \mathrm{AddF}(R); F = F \cup pk \\ ((u, \pi); w) \leftarrow \left(\overline{\mathrm{Prove}}^F || \chi_{\overline{P}}^F\right)(R, z)\end{array}\right] = negl(\lambda).$$

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

$$|\pi| = poly(\lambda) polylog(|u| + |w|).$$

Computatinal Zero Knowledge: *If for all PPT adversaries* $\overline{\mathrm{AddF}}$*, there exists a PPT simulator* Sim *with access to F and a PPT extractor* $\chi_{\overline{AddF}}$*, for every PPT distinguisher* D^F *and for all* $\lambda \in \mathbb{N},\ (R, z) \leftarrow \mathcal{R}(1^\lambda),\ (u, w) \in R,$

$$\Pr\left[\begin{array}{c}(pk, sk) \leftarrow \overline{\mathrm{AddF}}(R) \\ F = F \cup pk \\ \pi \leftarrow \mathrm{Prove}^F(R, u, w)\end{array} : \begin{array}{c}\mathrm{D}^F(R, z, u, sk, \pi) = 1 \\ \wedge \mathrm{Check}^F(R) = 1\end{array}\right] \approx$$

$$\Pr\left[\begin{array}{c}(pk, sk; \tau) \leftarrow (\overline{\mathrm{AddF}} || \chi_{\overline{AddF}})(R) \\ F = F \cup pk \\ \pi \leftarrow \mathrm{Sim}^F(R, \tau, u)\end{array} : \begin{array}{c}\mathrm{D}^F(R, z, u, sk, \pi) = 1 \\ \wedge \mathrm{Check}^F(R) = 1\end{array}\right]$$

If we consider the unbounded distinguisher D^F *in the definition of ZK, we obtain the definition of the statistic ZK in the BPK model. If we replace the notation $\approx$ with $=$, we obtain the definition of a perfect ZK in the BPK model.*

3 Classical Transformation

In [9], Bellare et al. concluded (non-subversion) soundness and computational subversion zero-knowledge (ZK, even if the CRS is not trusted) can be

obtained. According to the results in [3], the notion of sub-ZK in the CRS model is equivalent to the notion of no auxiliary string (uniform) non-black-box zero knowledge in the BPK model. In [25], Fuchsbauer claimed that the four well-known SNARKs constructions [27, 12, 22, 31] in the CRS model can be transformed to satisfy the uniform (and non-black-box) zero-knowledge property in the BPK model. Moreover, [1, 4] studied the modification of Groth16 to satisfy the uniform (and non-black-box) zero-knowledge property in the BPK model.

To date, all related works have a common technical core: utilizing the CRS verification algorithm to ensure that the CRS is well formed and makes use of knowledge assumptions for trapdoor extractability. For most SNARKs constructions, there is no natural CRS verification algorithm for the original scheme; therefore, we need to add extra elements to the CRS while maintaining the soundness guaranteed. Here, we introduce how Groth16 is transformed into a zk-SNARK which is secure in the BPK model, similar to the method proposed in [25].

3.1 CRS in Groth's zk-SNARK

Here, we introduce the original CRS and trapdoor in Groth16 [31] in detail; however, we omit specific construction details. Recall that after the setup algorithm in Groth16, we obtain

$$sk = (\alpha, \beta, \gamma, \delta, x); \tau = (x, \delta)$$

$$\boldsymbol{\sigma_1} = \left(\begin{array}{c} \alpha, \beta, \delta, \left\{x^i\right\}_{i=0}^{n-1}, \left\{ \dfrac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma} \right\}_{i=0}^{l}, \\ \left\{ \dfrac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta} \right\}_{i=l+1}^{m}, \left\{ \dfrac{x^i t(x)}{\delta} \right\}_{i=0}^{n-2} \end{array} \right)$$

$$\boldsymbol{\sigma_2} = (\beta, \gamma, \delta, \left\{x^i\right\}_{i=0}^{n-1})$$

$$crs = \left([\boldsymbol{\sigma_1}]_1, [\boldsymbol{\sigma_2}]_2 \right)$$

The CRS above is classified according to the group from which each element comes, but it is convenient for us to classify the elements according to which party uses each element. The new classification is as follows:

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

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

Notably, Groth argued in [31] that using a new larger CRS that contains precomputed $[u_i(x)]_1, [v_i(x)]_1, [v_i(x)]_2$ elements for $i = 0, \ldots, m$ rather than

$\left[x^i\right]_1, \left[x^i\right]_2$ elements for $i = 0, \ldots, n-1$ can lead to a construction with faster prover's computation. However, in contrast to the CRS verification algorithm that was used in this study (it was first proposed by [25] with the original CRS) that requires no extra elements in CRS, $O(n)$ extra elements need to be added to the CRS to secure the protocol with the new larger CRS in the BPK model [1, 4]. Thus, to ensure fewer extra CRS elements and to describe our technique clearly, the technology in this study is based on [25]. We need to emphasize that our technique can be used to replace all existing pairing checks of the scheme which is secure in the BPK model, including [1, 4], to achieve better theoretical properties with constant extra cost in the size of the CRS (or even obtain more efficiency with $O(\log n)$ extra cost in the size of the CRS when our Compute-with-Help Mechanism can be used). This means that by implementing our technique in the scheme [25], we obtain a zk-SNARK that is secure in the BPK model with better prover efficiency than [25] and a shorter CRS than [1, 4].

3.2 CRS Verification Algorithm

Notably, the CRS can be efficiently verified to be well-formed without adding extra elements, similar to the method proposed by [25]. The CRS verification algorithm should be run by the prover before the proof phase begins, and we call the CRS verification algorithm CV for short. The CV presented below is slightly different from that presented in [25]. We describe the algorithm in detail below:

$$\mathrm{CV}(R, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h, crs)$$

1. Check whether
$$[\gamma]_2 \neq [0]_2; \text{ for } \zeta \in \{x, \alpha, \beta, \delta, \tfrac{t(x)}{\delta}\}, [\zeta]_1 \neq [0]_1$$
2. For $i = 1, 2, \ldots, n-1$ check whether
$$\left[x^i\right]_1 \cdot [1]_2 = \left[x^{i-1}\right]_1 \cdot [x]_2$$
3. For $i = 1, 2, \ldots, n-1$ check whether
$$[1]_1 \cdot \left[x^i\right]_2 = \left[x^i\right]_1 \cdot [1]_2$$
4. Check whether
$$[1]_1 \cdot [\beta]_2 = [\beta]_1 \cdot [1]_2$$
$$[1]_1 \cdot [\delta]_2 = [\delta]_1 \cdot [1]_2$$
5. For $i = 0, 1, \ldots, n-2$ check whether
$$\left[\tfrac{x^i t(x)}{\delta}\right]_1 \cdot [\delta]_2 = \left[x^i\right]_1 \cdot \left[\textstyle\sum_{j=0}^{n-1} t_j x^j\right]_2 + \left[x^{i+1}\right]_1 \cdot \left[t_n x^{n-1}\right]_2$$
6. For $i = l+1, \ldots, m$ check whether
$$\left[\tfrac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right]_1 \cdot [\delta]_2 =$$
$$\left[\textstyle\sum_{j=0}^{n-1} u_{i,j} x^j\right]_1 \cdot [\beta]_2 + [\alpha]_1 \cdot \left[\textstyle\sum_{j=0}^{n-1} v_{i,j} x^j\right]_2 + \left[\textstyle\sum_{j=0}^{n-1} w_{i,j} x^j\right]_1 \cdot [1]_2$$

7. For $i = 0, \ldots, l$ check whether

$$\left[\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right]_1 \cdot [\gamma]_2 =$$
$$\left[\sum_{j=0}^{n-1} u_{i,j} x^j\right]_1 \cdot [\beta]_2 + [\alpha]_1 \cdot \left[\sum_{j=0}^{n-1} v_{i,j} x^j\right]_2 + \left[\sum_{j=0}^{n-1} w_{i,j} x^j\right]_1 \cdot [1]_2$$

8. Check whether

$$[\alpha\beta]_T = [\alpha]_1 \cdot [\beta]_2$$

If all checks pass, the checking algorithm outputs 1; otherwise, it outputs 0.

Note that the steps 7 and 8 above aim to verify the elements in crs_v, which will be used only in the verification. Therefore, we can remove these two steps without compromising the zero-knowledge property in the BPK model. In addition, check 5 is different from the corresponding check in [25]. For better batching efficiency, we place both x^i and x^{i+1} in group $\mathbb{G}_1$ rather than $\mathbb{G}_2$. However, it seems that this leads to computation of $\left[\sum_{j=0}^{n-1} t_j x^j\right]_2$. In fact, to obtain greater efficiency, almost all zk-SNARKs are implemented with polynomials, whose interpolation points are nth primitive root of unity modulo p. Thus, we almost always have $t_n = 1, t_0 = -1$ while the other $t_j = 0, j \in [1, n-1]$.

While many pairing checks must be performed, one may ask whether the CV is efficient enough. As Abdolmaleki et al. show in [1], we can compress the number of pairing computations to a constant via batching techniques. Unfortunately, as the implementation results show in [1], even the running times of the batched CV are approximately the same as the running times of prover algorithm. Recall that the CV algorithm is run by the prover. Hence, considering this result, we attempt to find a new technique to optimize the running time of the prover.

4 Sigma-CRS Verification

The previous works that employed classical CRS verification had two flaws. First, old schemes relied on knowledge assumptions and could only satisfy uniform ZK in the BPK model. Second, these schemes were highly dependent on pairing, which hindered their efficient implementation using the 'LIPs to designated-verifier zk-SNARK' transformation in the designated-verifier setting. Motivated by these, we also use Groth16 and attempt another method to achieve the transformation from the CRS model security to the BPK model security. In the conventional CV transformation, the verifier first generates the CRS, and then the prover runs the CV (some pairing checks) to check whether the CRS is well-formed. Our attempt requires the verifier to generate the CRS and some proofs to prove that the CRS is well-formed. Thereafter, the prover only needs to compute some exponentiations in the group to check the proofs instead of performing pairing checks. By this way, we could directly achieve uniform ZK in the BPK model and intentionally avoid the need for pairing to support our discussion in **Section 7**. We define the new verification algorithm run by the prover the sigma-CRS verification algorithm (SCV). From now on, the prover

means the prover of the SNARK protocol(that is, the verifier of Σ-protocol), and the Σ-prover means the prover of Σ-protocol (that is, the verifier of the SNARK protocol).

Next, we introduce how the verifier proves that the CRS is well formed by the Σ-protocol. Clearly, check 1 in the CV is trivial and can be checked by the prover as before. As for the other checks, only two types of Σ protocols must be utilized: one is the proof of exponential equality and the other one is the proof of exponential multiplication.

4.1 Proof for Exponential Equality Relation

Recall that the check 3 in CV is:
For $i = 1, 2, \ldots, n-1$ check whether

$$[1]_1 \cdot \left[x^i\right]_2 = \left[x^i\right]_1 \cdot [1]_2$$

From the respective exponent, if we define the public parameter $pp = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbb{Z}_p, p, g, h)$, we only need to prove that for $i \in [1, n-1]$ the relation R_3 shown below holds

$$R_3 = \left\{ \left(pp, P_i \in \mathbb{G}_1, Q_i \in \mathbb{G}_2; x^i \in \mathbb{Z}_p\right) : P_i = g^{x^i} \wedge Q_i = h^{x^i} \right\}$$

The **Protocol 1** denoted by Π_3 is a basic Σ-protocol for relation R_3. As shown in **Fig.1**, this protocol is a public-coin protocol; thus, it can be transformed to be non-interactive by the Fiat-Shamir transformation [24].

Theorem 1. *Π_3 is a three-move public-coin protocol for relation R_3. It is perfectly complete, unconditionally special sound, and special honest-verifier zero-knowledge (SHVZK).*

As the proof of **Theorem 1** is natural, we provide the corresponding proof in **Appendix C.1**.

Remark that the check 4 in the CV can also be proven using this type of Σ-protocol. However, rather than proving that there exist β and δ such that $P = [\beta]_1 \wedge Q = [\beta]_2$ and $P' = [\delta]_1 \wedge Q' = [\delta]_2$, we prove that there exist β and δ such that $P = [\beta]_1 \wedge Q = [\beta]_2 \wedge T = [\beta]_T$ and $P' = [\delta]_1 \wedge Q' = [\delta]_2 \wedge T' = [\delta]_T$. As explained in **Subsection 4.3**, we must ensure that the exponential of new elements $[\beta]_T$ and $[\delta]_T$ added to the CRS is consistent with the exponential of $[\beta]_1, [\delta]_1$. We denote this protocol by Π_4. One may think that the Σ-verifier could just compute the $[\beta]_T$ and $[\delta]_T$ by pairing. Note that we intentionally avoided the need for pairing to support our discussion in **Section 7**.

4.2 Proof for Exponential Multiplication Relation

Recall that check 2 in the CV is
For $i = 1, 2, \ldots, n-1$ check whether

$$\left[x^i\right]_1 \cdot [1]_2 = \left[x^{i-1}\right]_1 \cdot [x]_2$$

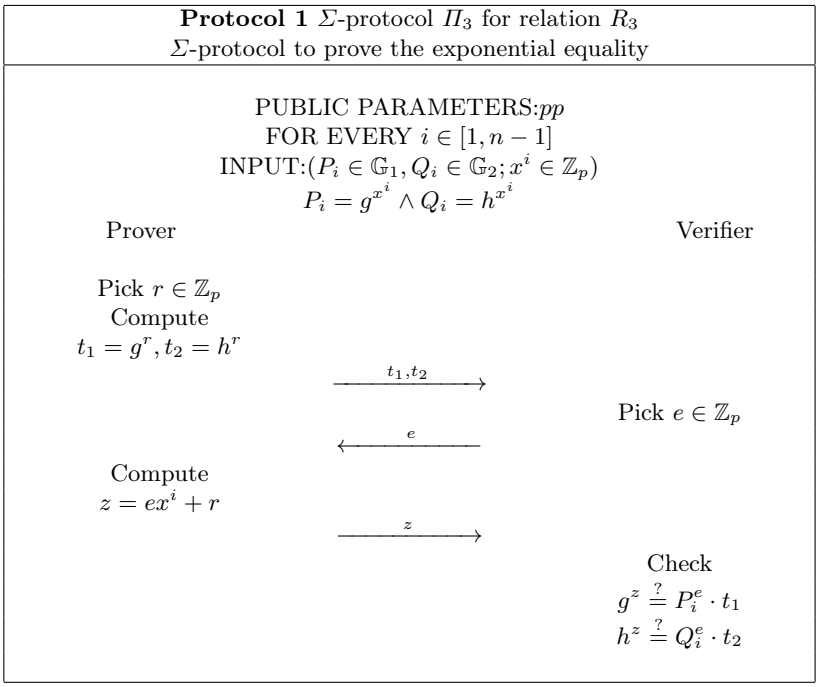

Fig. 1: The Σ-protocol for Relation R_3

From the respective exponents, we just hope to prove for $i \in [1, n-1]$ the relation $R_2^{'}$ as follows:

$$R_2^{'} = \left\{ \begin{array}{c} (pp, P_i \in \mathbb{G}_1, Q_i \in \mathbb{G}_1, T' \in \mathbb{G}_2; x \in \mathbb{Z}_p) : \\ P_i = Q_i^x \wedge T^{'} = h^x \end{array} \right\}$$

However, it is not convenient to design the Σ-protocol with this relation. Therefore, we chose to prove this relation in a more twisty but convenient manner. Our strategy involves proving that there exists x such that $T = [x]_1 \wedge T^{'} = [x]_2$ firstly. Recall that this relation is included in relation R_3, which has been proven. Thereafter, for $i \in [1, n-1]$, we can prove that the relation R_2 holds.

$$R_2 = \left\{ \begin{array}{c} \left(pp, P_i \in \mathbb{G}_1, Q_i \in \mathbb{G}_1, T \in \mathbb{G}_1; x^{i-1} \in \mathbb{Z}_p\right) : \\ P_i = T^{x^{i-1}} \wedge Q_i = g^{x^{i-1}} \end{array} \right\}$$

The **Protocol 2** denoted by Π_2 is a Σ-protocol for relation R_2. As shown in **Fig.2**, this protocol is a public-coin protocol; thus, it can be transformed to be non-interactive by the Fiat-Shamir transformation [24].

Theorem 2. *Π_2 is a three-move public-coin protocol for relation R_2. It is perfectly complete, unconditionally special sound, and SHVZK.*

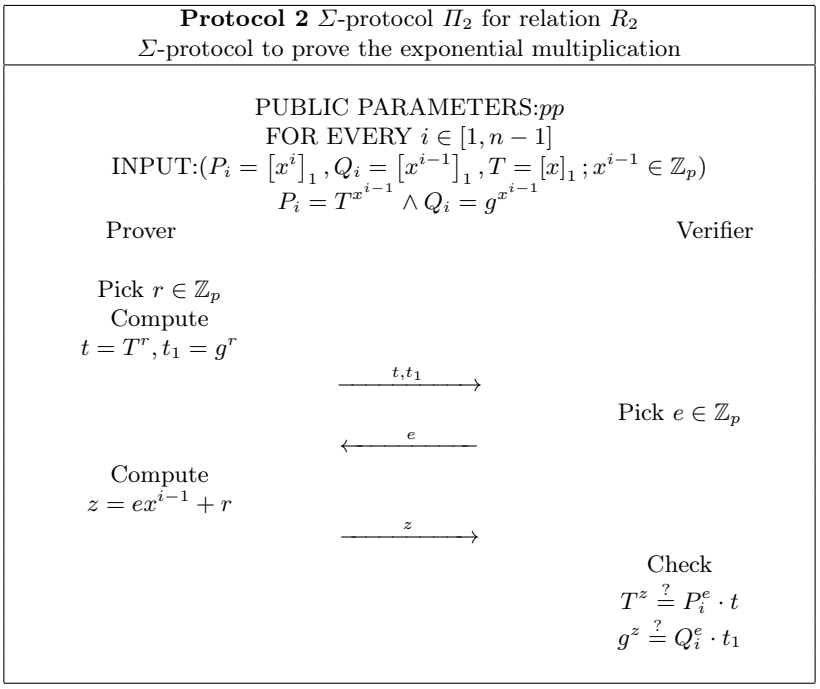

Fig. 2: The Σ-protocol for Relation R_2

The proof of **Theorem 2** is provided in **Appendix C.2**.

4.3 Proof for More Complex Pairing Equation

Here, we introduce the method to prove checks 5 and 6 in the CV. Our strategy is to prove that each pairing in checks 5 and 6 is well-formed by using proofs which are very similar to the protocol for exponential multiplication relation. For example, we present the protocol Π_5^1 which is very similar to Π_2 in **Appendix D**: **Fig.5**, to prove the pairing $\left[\frac{x^i t(x)}{\delta}\right]_1 \cdot [\delta]_2$ is well-formed.

For $i \in [0, n-2]$, we prove R_5^1 as follows:

$$R_5^1 = \left\{ \left(pp, P_i \in \mathbb{G}_T, Q_i \in \mathbb{G}_1, T \in \mathbb{G}_T; \frac{x^i t(x)}{\delta} \in \mathbb{Z}_p \right) : P_i = T^{\frac{x^i t(x)}{\delta}} \wedge Q_i = g^{\frac{x^i t(x)}{\delta}} \right\}$$

Theorem 3. *Π_5^1 is a three-move public-coin protocol for relation R_5^1. It is perfectly complete, unconditionally special sound, and SHVZK.*

The proof of **Theorem 3** is very similar to the proof of **Theorem 2** which is provided in **Appendix C.2**, so we omit it here.

Notably, in the relation R_5^1, $P_i = [x^i t(x)]_T$ are elements in the group $\mathbb{G}_T$, which are computed and provided by the Σ-prover. You may wonder why we did not use $P_i = [x^i t(x)]_1$ directly, that is because these elements are extra elements added by the verifier to the original CRS of the Groth16 scheme, which may compromise the soundness property. Further, $[x^i t(x)]_T$ can be computed by the prover directly, with the original CRS of the Groth16 scheme. This means that the extra element $[x^i t(x)]_T$ will not provide more capabilities to the generic group adversary, and thus will not compromise soundness. In addition, with $P = [x^i t(x)]_T$, T is natural to be $[\delta]_T$ rather than $[\delta]_1$; thus, the Σ-protocol works well.

Now, we introduce how to prove the entire check 5, recall that check 5 in the CV is

For $i = 0, 1, \ldots, n-2$ checks whether

$$\left[\frac{x^i t(x)}{\delta}\right]_1 \cdot [\delta]_2 = \left[x^i\right]_1 \cdot \left[\sum_{j=0}^{n-1} t_j x^j\right]_2 + \left[x^{i+1}\right]_1 \cdot \left[t_n x^{n-1}\right]_2$$

Evidently, this check is more complex than an exponential equality check or exponential multiplication check. Our proof strategy includes the following four steps:

1. For $i \in [0, n-2]$, the verifier adds $[\delta]_T, [x^i t(x)]_T$ into the CRS, and then proves that the tuple $\left([x^i t(x)]_T, \left[\frac{x^i t(x)}{\delta}\right]_1, [\delta]_T\right)$ satisfies the exponential multiplication relation with the Σ-protocol Π_5^1. Notably, the consistency between $[\delta]_T$ and $[\delta]_2$ can be proved in Π_4, as we have remarked in **Subsection 4.1**.
2. For $i \in [0, n-2]$, the verifier adds $\left[\sum_{j=0}^{n-1} t_j x^j\right]_T, \left[x^i \sum_{j=0}^{n-1} t_j x^j\right]_T$ into the CRS, and then proves the tuple $\left(\left[x^i \sum_{j=0}^{n-1} t_j x^j\right]_T, [x^i]_1, \left[\sum_{j=0}^{n-1} t_j x^j\right]_T\right)$ satisfies the exponential multiplication relation with the Σ-protocol similar to Π_5^1, we denote this protocol by Π_5^2. Notably, the consistency between $\left[\sum_{j=0}^{n-1} t_j x^j\right]_T$ and $\left[\sum_{j=0}^{n-1} t_j x^j\right]_2$ can also be proved with an exponential equality relation Σ-protocol similar to the protocol Π_4 (we denote this Σ-protocol by Π_0^1) while the $\left[\sum_{j=0}^{n-1} t_j x^j\right]_2$ can be computed via CRS by the prover who has checked the Π_2 and Π_3.
3. For $i \in [0, n-2]$, the verifier adds $\left[t_n x^{n-1}\right]_T, \left[x^{i+1} t_n x^{n-1}\right]_T$ into the CRS, and then proves that the tuple $\left(\left[x^{i+1} t_n x^{n-1}\right]_T, \left[x^{i+1}\right]_1, \left[t_n x^{n-1}\right]_T\right)$ satisfies the exponential multiplication relation with the Σ-protocol similar to Π_5^1, we denote this protocol by Π_5^3. Notably, the consistency between $\left[t_n x^{n-1}\right]_T$ and $\left[t_n x^{n-1}\right]_2$ can also be proved with an exponential equality relation Σ-protocol similar to the protocol Π_4 (we denote this Σ-protocol by Π_0^2) while the $\left[t_n x^{n-1}\right]_2$ can be computed via CRS by the prover who has checked Π_2 and Π_3.

4. For $i \in [0, n-2]$, the prover checks whether equation

$$[x^i t(x)]_T = [x^i \sum_{j=0}^{n-1} t_j x^j]_T \cdot [x^{i+1} t_n x^{n-1}]_T$$

holds.

Recall that check 6 in CV is that for $i = l+1, \ldots, m$, checking whether

$$\left[\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right]_1 \cdot [\delta]_2 = \left[\sum_{j=0}^{n-1} u_{i,j} x^j\right]_1 \cdot [\beta]_2 + [\alpha]_1 \cdot \left[\sum_{j=0}^{n-1} v_{i,j} x^j\right]_2 + \left[\sum_{j=0}^{n-1} w_{i,j} x^j\right]_1 \cdot [1]_2$$

Using the same strategy introduced above, check 6 in the CV can also be proved. Particularly,

1. For $i \in [l+1, m]$, by adding $[\delta]_T$, $[\beta u_i(x) + \alpha v_i(x) + w_i(x)]_T$ into CRS, we can use Π_6^1 to prove that the tuple

$$\left([\beta u_i(x) + \alpha v_i(x) + w_i(x)]_T, \left[\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right]_1, [\delta]_T\right)$$

satisfies the exponential multiplication relation while the consistency between $[\delta]_T$ and $[\delta]_2$ can be proved in Π_4.
2. For $i \in [l+1, m]$, by adding the extra elements $[\beta]_T$, $\left[\beta \sum_{j=0}^{n-1} u_{i,j} x^j\right]_T$, we can use Π_6^2 to prove that the tuple $\left(\left[\beta \sum_{j=0}^{n-1} u_{i,j} x^j\right]_T, \left[\sum_{j=0}^{n-1} u_{i,j} x^j\right]_1, [\beta]_T\right)$ satisfies the exponential multiplication relation, whereas the consistency between $[\beta]_T$ and $[\beta]_2$ can be proved in Π_4 and $\left[\sum_{j=0}^{n-1} u_{i,j} x^j\right]_1$ can be computed by the prover who has checked Π_2 from CRS.
3. For $i \in [l+1, m]$, by adding $[\alpha]_T$, $\left[\alpha \sum_{j=0}^{n-1} v_{i,j} x^j\right]_T$, we can use Π_6^3 to prove that the tuple $\left(\left[\alpha \sum_{j=0}^{n-1} v_{i,j} x^j\right]_T, \left[\sum_{j=0}^{n-1} v_{i,j} x^j\right]_2, [\alpha]_T\right)$ satisfies the exponential multiplication relation, whereas $\left[\sum_{j=0}^{n-1} v_{i,j} x^j\right]_2$ can be computed by the prover who has checked Π_2 and Π_3 from CRS. We denote the proof for consistency between $[\alpha]_T$ and $[\alpha]_1$ by Π_0^3.
4. For $i \in [l+1, m]$, we use Π_0^4 to prove the consistency between $\left[\sum_{j=0}^{n-1} w_{i,j} x^j\right]_T$ and $\left[\sum_{j=0}^{n-1} w_{i,j} x^j\right]_1$. In Π_0^4 $\left[\sum_{j=0}^{n-1} w_{i,j} x^j\right]_1$ for $i \in [l+1, m]$ can also be computed by a prover who has checked Π_2 from CRS.
5. For $i \in [l+1, m]$, the prover checks whether equation

$$[\beta u_i(x) + \alpha v_i(x) + w_i(x)]_T = [\beta \sum_{j=0}^{n-1} u_{i,j} x^j]_T \cdot [\alpha \sum_{j=0}^{n-1} v_{i,j} x^j]_T \cdot [\sum_{j=0}^{n-1} w_{i,j} x^j]_T$$

holds.

Recall that all Σ-protocols above are public-coin; thus, they can be transformed to be non-interactive by the Fiat-Shamir transformation [24]. Thus, we can replace all pairing checks that need to be computed by the prover with the verification of the Σ-protocol. In this study, the computation of $\left[\sum_{j=0}^{n-1} u_{i,j}x^j\right]_1$, $\left[\sum_{j=0}^{n-1} v_{i,j}x^j\right]_2$, $\left[\sum_{j=0}^{n-1} w_{i,j}x^j\right]_1$ for $i \in [l+1, m]$ can be regarded as being excluded from the prover's CV or SCV stage because these computations will also be performed in the proof stage.

4.4 Batching Techniques

The previous subsections show how to achieve SCV at the cost of adding a linear number of elements into the CRS. Here, we utilize batching techniques and present a method for reducing the number of extra elements in the CRS to a constant value.

As shown in the previous subsections, there are linear (about n or $m - l$) number of protocols that need to be run, which also leads to linear extra elements. Specifically, in addition to the constant number of simple protocols (e.g., $\Pi_0^1, \Pi_0^2, \Pi_0^3, \Pi_4$), we must run

- Π_2, Π_3 for every $i \in [1, n-1]$.
- $\Pi_5^1, \Pi_5^2, \Pi_5^3$ for every $i \in [0, n-2]$.
- $\Pi_6^1, \Pi_6^2, \Pi_6^3, \Pi_0^4$ for every $i \in [l+1, m]$.

Next, we will show in detail how to batch these protocols. We can batch Π_3 with batching randomness c to obtain $bat\Pi_3$, as shown in **Fig.3**. It is clear that the proof size of $bat\Pi_3$ is constant.

Theorem 4. *$bat\Pi_3$ is a three-move public-coin protocol. It is perfectly complete, unconditionally special sound, and SHVZK.*

The proof of **Theorem 4** is provided in **Appendix C.3**. By similar ways, we can obtain $bat\Pi_2$, $bat\Pi_5^1$, and $bat\Pi_0^4$ by batching Π_2, Π_5^1 and Π_0^4 with independent batching randomness, The corresponding protocols are shown in **Appendix D**: **Fig.6**, **Fig.7**, and **Fig.8**. It is clear that the proof sizes of these protocols are all constant. Additionally, because the security proofs of the these batching protocols are similar, we have omitted them for brevity.

In fact, we can treat Π_5^2, Π_5^3 as Π_5^1 to obtain $bat\Pi_5^2, bat\Pi_5^3$ and then run $bat\Pi_5^1, bat\Pi_5^2$, and $bat\Pi_5^3$ under the same batching randomness c. Similarly, $\Pi_6^1, \Pi_6^2, \Pi_6^3$ can be batched in the same way to get $bat\Pi_6^1, bat\Pi_6^2$, *and* $bat\Pi_6^3$. Additionally, Π_0^4 can be batched as $bat\Pi_0^4$ as described above. Protocols $bat\Pi_6^1$, $bat\Pi_6^2$, $bat\Pi_6^3$, and Π_0^4 should be run under the same batching randomness c. Until now, we only needed a constant number of checks, and we only added a constant number of extra elements into the CRS of the original scheme.

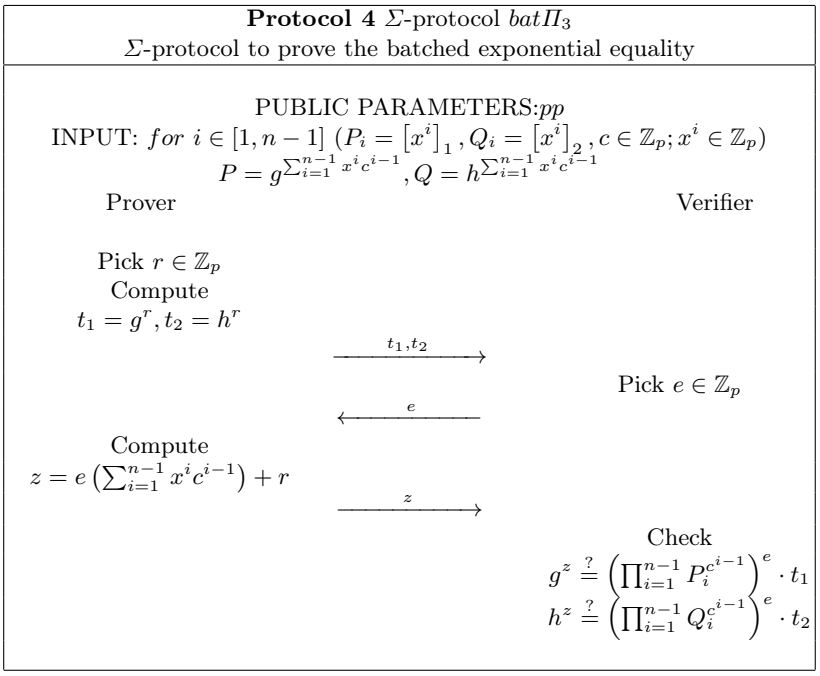

Fig. 3: The Batched Σ-protocol for Relation R_3

5 A New Publicly Verifiable zk-SNARK in the BPK Model

Up to now, we have introduced many sub-protocols in **Section 4** to prove different consistencies. Thus, for the convenience of reviewing them, we summarize them in **Appendix A**. Thereafter, we present the main zk-SNARK protocol with SCV, which is based on those sub-protocols, and show that it is secure in the BPK model.

5.1 Construction

Here, we present a new zk-SNARK $\Pi_{SCV-Groth}$, which is secure in the BPK model and based closely on Groth16. We implicitly assume that 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 [24]. Therefore, we denote the concatenation of the statement, elements in the public file, public input, and proof elements written by the prover up to a certain point in time by *transcript*.

On input QAP relation R, we denote our construction by $\Pi_{SCV-Groth} = (\mathrm{AddF}, \mathrm{Check}^F, \mathrm{Prove}^F, \mathrm{Verify}^F)$. We denote the Groth16 scheme by $\Pi_{Groth} =$

$(\text{Setup}, \text{Prove}^F_{Groth}, \text{Verify}^F_{Groth})$. For a simpler expression, we denote the hash function by H, regard the pk of our construction as a set, and divide pk into three parts. We denote the part including group elements in the CRS of Groth16 by pk_{Origin} and the other part including only proofs of the Σ-protocols by pk_{Sigma} while the rest is denoted by pk_{Rest}. This construction is described in detail as follows:

Key-registration Phase

1. $(pk, sk) \leftarrow \text{AddF}(R)$: We set the public file as $F = \Phi$ and $pk = pk_{Origin} \cup pk_{Sigma} \cup pk_{Rest} = \Phi$. Thereafter, the verifier runs $\text{AddF}(R)$ to get (pk, sk) and put the pk into the public file F. The algorithm AddF on input R is described as follows:
   - The verifier runs $\text{Setup}(R)$ of Groth16 to get $pk_{Origin} = CRS$ and $sk = (\alpha, \beta, \gamma, \delta, x)$.
   - The verifier computes $[\beta]_T, [\delta]_T$ and runs Π_4 non-interactively in the RO model to obtain proof π_4, which proves that there exist β and δ such that $P = [\beta]_1 \wedge Q = [\beta]_2 \wedge T = [\beta]_T$ and $P' = [\delta]_1 \wedge Q' = [\delta]_2 \wedge T' = [\delta]_T$, then set $pk_{Rest} = pk_{Rest} \cup [\beta]_T \cup [\delta]_T$, $pk_{Sigma} = pk_{Sigma} \cup \pi_4$.
   - Compute the challenges $c_1 = H(transcript, 0), c_2 = H(transcript, 1)$, the verifier then runs protocols $bat\Pi_2$ and $bat\Pi_3$ (with c_1, c_2 respectively) non-interactively in the RO model to generate proofs $bat\pi_2$ and $bat\pi_3$, respectively, which proves that for $i = 1, 2, \ldots, n-1$
   $$\left[x^i\right]_1 \cdot [1]_2 = \left[x^{i-1}\right]_1 \cdot [x]_2;\ [1]_1 \cdot \left[x^i\right]_2 = \left[x^i\right]_1 \cdot [1]_2$$
   Set $pk_{Sigma} = pk_{Sigma} \cup bat\pi_2 \cup bat\pi_3$.
   - Compute the challenge $c_3 = H(transcript, 2)$, the verifier then computes and puts the following elements in pk_{Rest}:
   $$\left[\sum_{i=0}^{n-2} x^i t(x) c_3^i\right]_T, [\delta]_T;\ \left[\sum_{j=0}^{n-1} t_j x^j \sum_{i=0}^{n-2} x^i c_3^i\right]_T,\ \left[\sum_{j=0}^{n-1} t_j x^j\right]_T;$$
   $$\left[t_n x^{n-1} \sum_{i=0}^{n-2} x^{i+1} c_3^i\right]_T,\ \left[t_n x^{n-1}\right]_T$$
   Under the same c_3, by running protocol $bat\Pi_5^1$ non-interactively in the RO model to obtain $bat\pi_5^1$, the verifier can conform the prover that $\left[\sum_{i=0}^{n-2} x^i t(x) c_3^i\right]_T$ is well-formed. Similarly, the verifier can run $\Pi_0^1, bat\Pi_5^2$ and $\Pi_0^2, bat\Pi_5^3$ non-interactively to obtain $\pi_0^1, bat\pi_5^2, \pi_0^2, bat\pi_5^3$ and conform the prover that $\left[\sum_{j=0}^{n-1} t_j x^j \sum_{i=0}^{n-2} x^i c_3^i\right]_T$ and $\left[t_n x^{n-1} \sum_{i=0}^{n-2} x^{i+1} c_3^i\right]_T$ are well-formed. Additionally, set $pk_{Sigma} = pk_{Sigma} \cup bat\pi_5^1 \cup \pi_0^1 \cup bat\pi_5^2 \cup \pi_0^2 \cup bat\pi_5^3$.
   - Compute the challenge $c_4 = H(transcript, 3)$, the verifier then computes and puts the following elements in pk_{Rest}:
   $$\left[\sum_{i=l+1}^{m} \left(\beta u_i(x) + \alpha v_i(x) + w_i(x)\right) c_4^{i-l-1}\right]_T, [\delta]_T;$$
   $$\left[\sum_{i=l+1}^{m} \left(\beta \sum_{j=0}^{n-1} u_{i,j} x^j\right) c_4^{i-l-1}\right]_T, [\beta]_T;$$
   $$\left[\sum_{i=l+1}^{m} \left(\alpha \sum_{j=0}^{n-1} v_{i,j} x^j\right) c_4^{i-l-1}\right]_T, [\alpha]_T;$$
   $$\left[\sum_{i=l+1}^{m} \left(\sum_{j=0}^{n-1} w_{i,j} x^j\right) c_4^{i-l-1}\right]_T$$

Under the same c_4, by running $bat\Pi_6^1, bat\Pi_6^2, \Pi_0^3, bat\Pi_6^3$, and $bat\Pi_0^4$ non-interactively in the RO model to obtain $bat\pi_6^1, bat\pi_6^2, \pi_0^3, bat\pi_6^3,$ *and* $bat\pi_0^4$, respectively, the verifier can mainly conform the prover that the following elements are well-formed:

$$\left[\sum_{i=l+1}^{m}\left(\beta u_i(x)+\alpha v_i(x)+w_i(x)\right)c_4^{i-l-1}\right]_T;$$
$$\left[\sum_{i=l+1}^{m}\left(\beta\sum_{j=0}^{n-1}u_{i,j}x^j\right)c_4^{i-l-1}\right]_T;\ \left[\sum_{i=l+1}^{m}\left(\alpha\sum_{j=0}^{n-1}v_{i,j}x^j\right)c_4^{i-l-1}\right]_T;$$
$$\left[\sum_{i=l+1}^{m}\left(\sum_{j=0}^{n-1}w_{i,j}x^j\right)c_4^{i-l-1}\right]_T$$

Additionally, set $pk_{Sigma} = pk_{Sigma} \cup bat\pi_6^1 \cup bat\pi_6^2 \cup \pi_0^3 \cup bat\pi_6^3 \cup bat\pi_0^4$.

- Algorithm AddF(R) outputs $pk = pk_{Origin} \cup pk_{Sigma} \cup pk_{Rest}$ and sk.

2. Set the public file $F = \Phi \cup pk$.

Proof Phase

1. $0/1 \leftarrow \text{Check}^F(R)$: The prover runs $\text{Check}^F(R)$ to ensure all elements in pk_{Origin} are well-formed. Algorithm Check^F on input R includes the following steps:
   - Check all the proofs in pk_{Sigma} with elements in pk_{Origin} and pk_{Rest}. If all checks pass, the prover continues to run the following steps, otherwise the prover outputs 0.
   - The prover checks whether the following equations hold:

$$\left[\sum_{i=0}^{n-2}x^i t(x)c_3^i\right]_T = \left[\sum_{j=0}^{n-1}t_j x^j \sum_{i=0}^{n-2}x^i c_3^i\right]_T \cdot \left[t_n x^{n-1}\sum_{i=0}^{n-2}x^{i+1}c_3^i\right]_T$$
$$\left[\sum_{i=l+1}^{m}\left(\beta u_i(x)+\alpha v_i(x)+w_i(x)\right)c_4^{i-l-1}\right]_T =$$
$$\left[\sum_{i=l+1}^{m}\left(\beta\sum_{j=0}^{n-1}u_{i,j}x^j\right)c_4^{i-l-1}\right]_T \cdot \left[\sum_{i=l+1}^{m}\left(\alpha\sum_{j=0}^{n-1}v_{i,j}x^j\right)c_4^{i-l-1}\right]_T \cdot$$
$$\left[\sum_{i=l+1}^{m}\left(\sum_{j=0}^{n-1}w_{i,j}x^j\right)c_4^{i-l-1}\right]_T$$

   If the two equations hold, then the elements in pk_{Origin} are ensured to be well-formed, algorithm $\text{Check}^F(R)$ outputs 1, and the prover continues to run the following steps. Otherwise, the algorithm outputs 0 and the prover outputs 0.
2. $\pi \leftarrow \text{Prove}^F(R, u, w)$: The prover runs $\text{Prove}^F_{Groth}(R, u, w)$ of Groth16 and outputs its output π.
3. $0/1 \leftarrow \text{Verify}^F(R, u, sk, \pi)$: The verifier runs $\text{Verify}^F_{Groth}(R, u, \pi)$ of Groth16 and outputs its output.

5.2 Security Analysis

Theorem 5. *Protocol $\Pi_{SCV-Groth}$ is a non-interactive argument with perfect completeness and computational zero-knowledge in the BPK model. It has computational knowledge soundness in the BPK 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.

Public-key Verifiability: The public-key verifiability property comes straightly from the completeness of the Σ-protocols.

Knowledge Soundness: Our construction is similar to that of Groth16 except that we include an efficient algorithm, SCV, and there are some extra elements in the CRS that may compromise the soundness. In fact, the extra elements in the CRS include only two parts. One part comprises the elements in group $\mathbb{G}_T$ (which are included in pk_{Rest}), and the other part is the proof of Σ-protocol (which are included in pk_{Sigma}). Because the elements in pk_{Rest} can be computed by pairing the elements from pk_{Origin}, the generic adversary cannot gain any ability improvement. The zero knowledge property of the Σ-protocol can also ensure that no extra power is given to the adversary. Altogether, we can complete the SCV without compromising the knowledge soundness of Groth16. Thus, the proof of knowledge soundness is the same as the proof of knowledge soundness in [31].

Zero-Knowledge: Notably, the verifier generate the CRS together with the Σ-proofs and these proofs can be transformed to be non-interactive in the RO model. Using the forking lemma discussed in [42], if the prover of the Σ-protocol can find, with non-negligible probability, a valid transcript (a, e, z), the prover of the Σ-protocol can also find another transcript (a, e', z'). This yields an extractor with the expected polynomial time to extract $\tau = (x, \delta)$ with probability 1 (note that x is one of the witness in $bat\Pi_3$, and δ is the witness in Π_4). Simulator $\mathrm{Sim}^F(R, u, \tau)$ can run $\mathrm{Check}^F(R)$ and output 0 if algorithm $\mathrm{Check}^F(R)$ outputs 0. Thereafter, the simulator chooses $r, s \leftarrow \mathbb{F}$ and defines proof $\pi' = ([A']_1, [B']_2, [C']_1)$ output by $\mathrm{Sim}^F(R, \tau, u)$ as follows:

$$A' = \alpha + r \qquad B' = \beta + s$$
$$C' = \frac{rs + \alpha s + \beta r - \sum_{i=0}^{l} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\delta}$$

If we denote the honestly generated proofs by $\pi = ([A]_1, [B]_2, [C]_1)$, it is clear that $[A]_1, [B]_2, [A']_1, [B']_2$ are uniformly random. From the constant equation $C = \frac{AB - \alpha\beta - \sum_{i=0}^{l} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\delta}$, we know that $[C]_1$ and $[C']_1$ are uniquely determined by $[A]_1, [B]_2$ and $[A']_1, [B']_2$, respectively with fixed F. Thus, the zero knowledge property holds.

Notably, the knowledge assumptions of previous studies that are used to extract τ do not hold for non-uniform machines. However, our extraction comes from the forking lemma, which bypasses this limitation. Therefore, our construction satisfies the auxiliary-string black-box zero knowledge in the BPK model. In addition, if we combine our Sigma-CRS verification technique with a variant construction [16] of Groth16, we can easily achieve both ZK in the BPK model and SE without adding extra assumptions.

Following the above analysis, we point out that the scheme in [34] actually also satisfies (non-uniform) ZK in the BPK model, rather than just S-ZK. In fact, they include "proof of discrete logarithm" Σ-protocols for each of the simulation trapdoors, while the well-formedness of the public keys are still be checked with pairings as in other works. Based on the results presented in **Section 8**,

we can conclude that the main computational cost lies in batching. Therefore, the method to achieve ZK in the BPK model, as described in [34], is similar in efficiency to ours. However, as we will discuss in **Section 7**, achieving the auxiliary-string zero knowledge property in the BPK model is only one of our goals, and our construction has remove dependence on pairings at the same time.

6 New Speed-up Technique

Here, we first present a new computation mechanism, Compute-with-Help Mechanism, and then demonstrate how to use this new technique in our new construction. Interestingly, we found that in the scenario we are about to describe, our new speed-up technique could also be considered as a speed-up technique to the multi-scalar multipication (MSM) which is a fundamental computational problem in the field of cryptography, especially zk-SNARK. So we proposed our new technique as an independent interest.

6.1 Compute-with-Help Mechanism

For a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$, we define the public parameter $pp = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathbb{Z}_p, p, g, h)$. For a generalized consideration, we also denote g, h and $e(g, h)$ as g_1, g_2 and g_3, respectively. We assume that CRS $\left\{g_\tau^{x^i}\right\}_{i\in[0,d]}$ is generated by the honest setup, where $\tau \in \{1, 2\}$. Evidently, $\left\{g_3^{x^i}\right\}_{i\in[0,d]}$ can be computed through pairing. It may appear in cryptography that with the public parameters pp, $\left\{g_\tau^{x^i}\right\}_{i\in[0,d],\tau\in\{1,2\}}$ and any $n \leq d$ degree polynomial $f(X) \in \mathbb{F}[X]$, one party has to compute $O(n)$ times exponential multiplication to obtain $g_\tau^{f(x)}, \tau \in \{1, 2, 3\}$ (this is exactly a MSM computation). However, we found that for some special polynomials, and with the help of another party who is knowledgeable about x, element $g_\tau^{f(x)}$ can be obtained by computing only $O(m)$ times of exponential multiplication, where $m \ll n$. Thus, we call this speed up technique Computation-with-Help (CWH) mechanism. Note that in the field of zk-SNARK, the x^i in the CRS $\left\{g_\tau^{x^i}\right\}_{i\in[0,d]}$ are always regarded as toxic information and should be discarded. However, our CWH technique show that they are not just toxic information, but also computing accelerators that could be well utilized.

To demonstrate the technique to achieve CWH, we first need to define a type of polynomial.

Definition 5. *(Self-recursive polynomial (SRP)) We define polynomial $f(X)$ on field $\mathbb{F}$ as a self-recursive polynomial if there exists positive integer m and monomial set $\{q_i(X)\}_{i\in[1,m]}$ such that $f(X) = (1 + q_1(X)) \cdots (1 + q_m(X))$, where each monomial $q_i(X)$ is a non-zero degree monomial over the field $\mathbb{F}$. We denote the polynomial set including all the self-recursive polynomials by SRP.*

Notably, for a fixed polynomial $f(X) \in SRP$ and different choices of field $\mathbb{F}$, the decomposition of $f(X)$ may not be unique. There could be k correct but different monomial sets $\{q_i^j(X)\}_{i\in[1,m_j]}$, where $j \in [1,k]$, such that polynomial $f(X)$ can be represented as $f(X) = \left(1+q_1^j(X)\right)\cdots\left(1+q_{m_j}^j(X)\right)$. However, there is no need to require decomposition uniqueness, and the condition that $f(X) \in SRP$ is sufficient for us to utilize our technique.

In protocol 8 (**Fig.4**), we introduce how the prover could help the verifier to compute $g_\tau^{f(x)}$, where $\tau \in \{1,2,3\}$ when the $\left\{g_\tau^{x^i}\right\}_{i\in[0,d]}$ have already been well-formed. We assume that n degree polynomial $f(X) \in SRP$; thus, $f(X) = (1+q_1(X))\cdots(1+q_m(X)) = (1+q_1(X))\cdot h(X)$, where monomials $q_i(X)$ are defined in $\mathbb{Z}_p[X]$. Our idea comes from an observation where we found that if the verifier computes $g_\tau^{h(x)}$ by self, the verifier could just obtain the $g_\tau^{q_1(x)h(x)}$ and verify its well-formness with the prover's help. Finally, the verifier computes $g_\tau^{f(x)}$ by just multiplying $g_\tau^{h(x)}$ and $g_\tau^{q_1(x)h(x)}$.

Theorem 6. *Π_{CWH} is a three-move public-coin protocol. It is perfectly complete, unconditionally special sound, and SHVZK.*

The core component in protocol 8 is the proof of the exponential equality relation introduced in **Subsection 4.1**. Because $\left\{g_1^{x^i}\right\}, i \in [0,d]$ are assumed to be well-formed, the prover can utilize the exponential equality proof to prove that for base element g_1 and $g_\tau^{h(x)}$, the exponentials of $g_1^{q_1(x)}$ and T are equal. Thus, the security proof of this protocol is similar to that of protocol Π_3, and the specific proof of **Theorem 6** is given in **Appendix C.4**.

Notably, while many other computations only take constant time, at most one group exponential operation is needed to compute $g_1^{q_1(x)}$ because $q_1(x)$ is monomial. Thus, the primary expense of the verifier is the computation of $g_\tau^{h(x)}$. Because the polynomial $h(X) = (1+q_2(X))\cdots(1+q_m(X))$ also belongs to SRP, protocol Π_{CWH} can be called recursively m times to make the verifier obtain $g_\tau^{f(x)}$ with a computational complexity of $O(m)$ rather than $O(n)$. The size of m is closely related to the polynomial, and for some specific polynomials, the size of m can be significantly smaller than n, for example, $m = O(\log n)$. This means that we can utilize the Computation-with-Help mechanism to significantly improve efficiency in specific situations.

We denote the recursive protocol as Π_{RCWH}. For the security of Π_{RCWH}, we recommend going to **Appendix B** for some relevant definitions, and we present the following theorem.

Theorem 7. *Π_{RCWH} is a $(2m+1)$-move public-coin protocol. It is perfectly complete, unconditionally $(2,\ldots,2)$-special sound, and SHVZK.*

Proof. The completeness, $(2,\ldots,2)$-special soundness, and honest-verifier zero-knowledge of Π_{RCWH} follows directly from the completeness, special soundness, and honest-verifier zero-knowledge of Π_{CWH}, respectively.

Protocol 8 Compute-with-Help Mechanism Π_{CWH}
An abstract protocol to introduce the Compute-with-Help Mechanism

PUBLIC PARAMETERS: pp

INPUT: $\left(All\ \left\{g_\tau^{x^i}\right\}_{i\in[0,d],\tau\in\{1,2\}}, f(X), h(X), q_1(X) \in \mathbb{F}_{<d}[X]; q_1(x) \in \mathbb{Z}_p\right)$

Prover		Verifier
Compute $T = g_\tau^{q_1(x)h(x)}$, where $\tau \in \{1,2,3\}$		
Pick $r \in \mathbb{Z}_p$		
Compute $t_1 = g_1^r, t_2 = \left(g_\tau^{h(x)}\right)^r$		
	$\xrightarrow{T,t_1,t_2}$	
		Pick $e \in \mathbb{Z}_p$
	$\xleftarrow{e}$	
Compute $z = eq_1(x) + r$		
	$\xrightarrow{z}$	
		Compute $g_1^{q_1(x)}$ with one of the $\left\{g_1^{x^i}\right\}_{i\in[0,d]}$
		Compute $g_\tau^{h(x)}$ with $\left(\left\{g_\tau^{x^i}\right\}_{i\in[0,d]}\right)$
		Check $g_1^z \stackrel{?}{=} \left(g_1^{q_1(x)}\right)^e \cdot t_1$
		$\left(g_\tau^{h(x)}\right)^z \stackrel{?}{=} T^e \cdot t_2$
		Compute $g_\tau^{f(x)}$ by multiplying $g_\tau^{h(x)}$ and T

Fig. 4: The Compute-with-Help Protocol

In addition, interactive protocol Π_{RCWH} can be transformed into a non-interactive protocol using the Fiat-Shamir transformation [24], and the witness extended emulation is known to follow from $(2,\ldots,2)$-special soundness according to the generalized forking lemma (**Appendix B.2**). Recall that to prove the zero knowledge property of the zk-SNARK in **Subsection 5.2**, we used the forking lemma in [42], which yielded an extractor to obtain the simulation trapdoor. Notably, when we plug the CWH protocol in the SCV construction, both the forking lemma and generalized forking lemma could be used to extract the specific trapdoor.

6.2 Application in Our Construction

Here, we present a specific application about the Compute- with-Help Mechanism introduced in **Subsection 6.1**. We demonstrate how to use the CWH to speed up the prover algorithm of the zk-SNARK construction in **Subsection 5**.

As shown in **Subsection 4.4**, the computational complexity of the Σ-verifier (the prover of the main protocol) depends mainly on the final verification, which includes a linear number of group exponentiations. Therefore, we attempt to speed up this process and improve the computational complexity of the Σ-verifier. Notably, in the BPK model, it is crucial to speed up the prover's algorithm because the verifier's main computational cost is a one-time computation of generating the CRS.

As we will discuss in **Section 8**, the transformation from CV to SCV can only provide limited optimization. Fortunately, in addition to theoretical contributions, this transformation has inspired us with more ideas. Instead of the way that the verifier generates the CRS first and then the prover runs the CV (some pairing checks) to verify whether the CRS is well-formed, our attempt requires that the verifier generates the CRS and some proofs to prove that the CRS is well-formed. This means that we let verifier do more things (e.g., generating extra proofs). Now that we enable the verifier to generate CRS and proofs simultaneously in our construction, the verifier can further exploit the knowledge derived from the CRS generation to produce additional proofs and thus reduce the computation of the prover. In other words, the verifier could execute the Compute-with-Help protocol with the prover.

Given that $\left[x^i\right]_1$ for $i \in [1, n-1]$ is well-formed (we assume that $n-1$ is a power of 2 without loss of generality), the prover can compute $\left[\sum_{i=1}^{n-1} x^i c^{i-1}\right]_1$ in logarithmic time rather than linear time by utilizing the CWH protocol, where $f(X) = \sum_{i=1}^{n-1} X^i c^{i-1}$. The core of our technique is to utilize the verifier's knowledge of x. Specifically, when we want to compute

$$G = g^{\sum_{i=1}^{n-1} x^i c^{i-1}}$$

Equivalently,

$$G = g^{x+x^2c+\cdots+x^{\frac{n-1}{2}}c^{\frac{n-3}{2}}} \cdot \left(g^{x+x^2c+\cdots+x^{\frac{n-1}{2}}c^{\frac{n-3}{2}}}\right)^{x^{\frac{n-1}{2}}c^{\frac{n-1}{2}}}$$

We observed that the prover only needs to compute

$$K = \left(g^{x+x^2c+\cdots+x^{\frac{n-1}{2}}c^{\frac{n-3}{2}}}\right)^{c^{\frac{n-1}{2}}}$$

The verifier can help the prover compute and publish $P = K^{x^{\frac{n-1}{2}}}$ together with an non-interactive exponential equality relation proof that proves there exists $x^{\frac{n-1}{2}}$ such that $P = K^{x^{\frac{n-1}{2}}}$ and $Q = g^{x^{\frac{n-1}{2}}}$ in the key-registration phase.

Clearly, this protocol can be recursively called logarithmic time. Thus, the computations of $g^{\sum_{i=1}^{n-1} x^i c_2^{i-1}}$ and $g^{\sum_{i=1}^{n-1} x^{i-1} c_3^{i-1}}$ in $batΠ_3$ and $batΠ_5^2$, respectively, can be reduced to be logarithmic. The only cost is that we need to add logarithmic extra elements (e.g., $K^{x^{\frac{n-1}{2^i}}}$ for $i \in [1, \log(n-1)]$ and the logarithmic number of exponential equality relation proofs) to the CRS.

As our zk-SNARK in the BPK model has excluded the trusted third party, you may wonder why we could utilize the CWH protocol, which requires trusted generated reference strings. In fact, the above speed-up process could be regarded as being run after the related reference strings produced by the verifier (e.g., g_τ powers) in the zk-SNARK have been verified to be well-formed.

7 Efficient Implementation in the Designated-Verifier Setting

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. Therefore, we will focus on implementing our scheme in the designated-verifier setting in this section. To date, we have presented an efficient publicly verifiable zk-SNARK in the BPK model using bilinear maps. Although we may have many simple ways to turn publicly verifiable schemes into schemes with designated verifier. We hope that designated-verifier schemes have unique advantages in efficiency or size, as its application scenarios are more limited compared to publicly verifiable schemes. So, we aspire to the designated-verifier schemes that are compatible with the well-known and efficient "LIPs to designated-verifier zk-SNARK" transformation mentioned in [13]. In [13], 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. What we want to emphasize is that this efficient designated-verifier transformation is still valid for the linear non-interactive proofs (NILPs) introduced by Groth16. Unfortunately, this transformation has failed for all current zk-SNARKs in the BPK model based on Groth16. By comparison, our scheme has removed all the requirements for pairing and can still leverage additively homomorphic encryption techniques (i.e., Paillier encryption) to obtain a very efficient designated-verifier zk-SNARK in the BPK model.

Specifically, when we consider using the "LIPs to designated-verifier zk-SNARK" transformation on Groth16, all elements in the public file are ciphertexts outputed by an additively homomorphic encryption algorithm and are in a standard (generic) group rather than bilinear group. The prover then generates the proof by homomorphic operations on a standard group rather than operations on a bilinear group. Finally, the verifier verifies the proof by just decrypting the ciphertexts and directly checking whether the plaintexts satisfy the verification equation of NILP. It is evident that the conventional approach (adding pairing checks) to obtain security in the BPK model no longer works because there

is no bilinear group in the construction at all. In addition, with the ciphertexts in the public file, the prover cannot check the well-formness of the corresponding plaintexts directly because the decryption private key is only owned by the verifier. Fortunately, compared to the conventional approach, by utilizing the additive homomorphism property, the Σ-protocols in our SCV technique could be modified simply and directly to obtain a very efficient designated-verifier zk-SNARK in the BPK model.

8 Implementation

Here, we compare the efficiency of Groth's zk-SNARK with conventional CRS verification [25] (which is batched and denoted by Groth-CV), our zk-SNARK construction (which is batched and denoted by Groth-SCV), and our zk-SNARK with the CWH technique (which is batched and denoted by Groth-CV-CWH) during implementation. In particular, because the main efficiency difference among these schemes comes from the different CRS verification algorithms, we also compare the three CRS verification algorithms from a theoretical point of view.

Similar to the pre-existing implementation of Groth's zk-SNARK in the `bellman` [36] library, we implemented our zk-SNARKs in the `Rust` library using low-level subroutines of `bellman`. 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.

Table 1 compares the schemes Groth-CV, Groth-SCV, and Groth-SCV-CWH. For a circuit with a large proportion of the multiplication gate and several choices of n and l, we report several measures including the running time of CG, CV, P, and V. Additionally, all times are expressed in seconds and all three protocols are batched. We emphasize that the differences between Groth-CV, Groth-SCV, and Groth-SCV-CWH only exist in the CRS generation and CRS verification phases. Thus, the three schemes have the same proof and verification times. From **Table 1**, it can be concluded that the difference in the CRS verification times of Groth-CV and Groth-SCV is extremely small, but Groth-SCV-CWH does have shorter CRS verification time than Groth-CV.

Table 2 compares the main computation of conventional CRS verification algorithm, our Sigma-CRS verification algorithm, and our SCV algorithm with the CWH technique in group $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, and pairing from the asymptotic point of view. We denote the number of multiplication gates by n and the number of wires by m. From **Table 2**, it can be theoretically analyzed that the efficiencies of the CV and SCV are almost the same. Compared to SCV-CWH, the CV and SCV should compute $O(2n - 10\log n)$ more exponential computations in the group $\mathbb{G}_1$. Because m is related to n, we assume that $m = cn$ for convenience, where c is a constant. If we ignore the relatively small number $10\log n$, the PCT can be presented as $PCT = \frac{2n}{(4+3c)n-3l} = \frac{2}{(4+3c)-3\frac{l}{n}}$. Thus our technique is more advantageous when applied to the circuit with more multiplication gates

(note that c decreases as proportion of multiplication gates in all circuit gates increases). It is also evident that with the same c, the PCT decreases as n increases with the same l and increases as l becomes larger with the same n. However, the n is often much larger than l, thus the main factor that determine the value of PCT is the proportion of multiplication gates.

Table 1: Performance of the implementations of the Groth-CV, Groth-SCV, and Groth-SCV-CWH for different values of n and l.

			Groth-CV		Groth-SCV		Groth-SCV-CWH		
Protocol	P(s)	V(s)	CG(s)	CV(s)	CG(s)	CV(s)	CG(s)	CV(s)	PCT
$n = 2^{13}$,$l = 2^{6}$	0.151	0.008	0.898	0.387	2.065	0.350	2.076	0.298	22.9%
$n = 2^{14}$,$l = 2^{6}$	0.285	0.008	1.626	0.707	3.932	0.666	3.943	0.581	17.8%
$n = 2^{15}$,$l = 2^{6}$	0.529	0.008	3.090	1.209	7.650	1.192	7.662	1.017	15.9%
$n = 2^{15}$,$l = 2^{10}$	0.522	0.096	3.197	1.138	7.418	1.114	7.430	0.945	17.0%
$n = 2^{16}$,$l = 2^{10}$	0.977	0.097	6.224	2.212	15.007	2.326	15.020	1.938	12.4%
$n = 2^{17}$,$l = 2^{10}$	1.906	0.096	12.059	4.243	29.510	4.378	29.524	3.844	9.4%
$n = 2^{18}$,$l = 2^{10}$	3.637	0.094	23.241	7.671	58.480	7.555	58.493	6.558	14.5%
$n = 2^{19}$,$l = 2^{10}$	7.525	0.098	48.127	14.682	120.399	15.100	120.414	13.330	9.2%
$n = 2^{20}$,$l = 2^{10}$	14.290	0.096	94.246	27.874	235.155	27.585	235.171	24.246	13.0%

This table show us the specific performance comparison. "CG" in this table implies the time of the CRS generation process that could be completed offline, while "V" represents the time of the online verification algorithm. "CV" and "P" denote the time of the CRS verification process and the time of the proof process, respectively. "PCT" represents the efficiency improvement rate of the CRS verification in Groth-SCV-CWH over the CRS verification in Groth-CV. By n and l, we denote the number of multiplication gates and the length of public input, respectively.

Table 2: CRS verification comparison

	$\mathbb{G}_1$ Exp	$\mathbb{G}_2$ Exp	$\mathbb{G}_T$ Exp	P
[1]CV	$O(4n+3m-3l)$	$O(n+m-l)$	\	$O(1)$
[2]SCV	$O(4n+3m-3l)$	$O(n+m-l)$	$O(1)$	\
[3]SCV-CWH	$O(2n+10\log n+3m-3l)$	$O(n+m-l)$	$O(1)$	\

This table show us the prover's computational complexity asymptotically in different protocol. "$\mathbb{G}_1$ Exp", "$\mathbb{G}_2$ Exp", and "$\mathbb{G}_T$ Exp" represent the exponential computation in group $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, respectively. "P" means the pairing operation.

[1] "CV" means the traditional CRS verification algorithm.

[2] "SCV" means the Sigma-CRS verification algorithm presented in this work.

[3] "SCV-CWH" means our Sigma-CRS verification with the CWH technique in addition.

References

1. Abdolmaleki, B., Baghery, K., Lipmaa, H., Zając, M.: A subversion-resistant snark. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 3–33. Springer International Publishing, Cham (2017)
2. Abdolmaleki, B., Glaeser, N., Ramacher, S., Slamanig, D.: Circuit-succinct universally-composable nizks with updatable crs. Cryptology ePrint Archive, Paper 2023/097 (2023), https://eprint.iacr.org/2023/097, https://eprint.iacr.org/2023/097
3. Abdolmaleki, B., Lipmaa, H., Siim, J., Zając, M.: On qa-nizk in the bpk model. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) Public-Key Cryptography – PKC 2020. pp. 590–620. Springer International Publishing, Cham (2020)
4. Abdolmaleki, B., Lipmaa, H., Siim, J., Zając, M.: On subversion-resistant snarks. Journal of Cryptology **34**(3), 17 (Apr 2021). https://doi.org/10.1007/s00145-021-09379-y, https://doi.org/10.1007/s00145-021-09379-y
5. Abdolmaleki, B., Ramacher, S., Slamanig, D.: Lift-and-shift: Obtaining simulation extractable subversion and updatable snarks generically. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020. pp. 1987–2005. ACM (2020). https://doi.org/10.1145/3372297.3417228, https://doi.org/10.1145/3372297.3417228
6. Amine, O., Baghery, K., Pindado, Z., Ràfols, C.: Simulation extractable versions of groth's zk-snark revisited. Cryptology ePrint Archive, Paper 2020/1306 (2020), https://eprint.iacr.org/2020/1306, https://eprint.iacr.org/2020/1306
7. Baghery, K.: Subversion-resistant simulation (knowledge) sound nizks. In: Albrecht, M. (ed.) Cryptography and Coding. pp. 42–63. Springer International Publishing, Cham (2019)
8. Baghery, K., Pindado, Z., Ràfols, C.: Simulation extractable versions of groth's zk-snark revisited. In: Cryptology and Network Security: 19th International Conference, CANS 2020, Vienna, Austria, December 14–16, 2020, Proceedings. p. 453–461. Springer-Verlag, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-65411-5_22, https://doi.org/10.1007/978-3-030-65411-5_22

9. Bellare, M., Fuchsbauer, G., Scafuro, A.: Nizks with an untrusted crs: Security in the face of parameter subversion. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016. pp. 777–804. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
10. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. pp. 701–732. Springer International Publishing, Cham (2019)
11. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for r1cs. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019. pp. 103–128. Springer International Publishing, Cham (2019)
12. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct Non-Interactive zero knowledge for a von neumann architecture. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 781–796. USENIX Association, San Diego, CA (Aug 2014), https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson
13. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7785, pp. 315–333. Springer (2013). https://doi.org/10.1007/978-3-642-36594-2_18, https://doi.org/10.1007/978-3-642-36594-2_18
14. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA. pp. 103–112. ACM, New York (1988)
15. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 327–357. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
16. Bowe, S., Gabizon, A.: Making groth's zk-snark simulation extractable in the random oracle model. Cryptology ePrint Archive, Paper 2018/187 (2018), https://eprint.iacr.org/2018/187
17. Bünz, B., Fisch, B., Szepieniec, A.: Transparent snarks from dark compilers. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020. pp. 677–706. Springer International Publishing, Cham (2020)
18. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 315–334 (2018). https://doi.org/10.1109/SP.2018.00020
19. Campanelli, M., Faonio, A., Fiore, D., Querol, A., Rodríguez, H.: Lunar: A toolbox for more efficient universal and updatable zksnarks and commit-and-prove extensions. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021. pp. 3–33. Springer International Publishing, Cham (2021)
20. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., Ward, N.: Marlin: Preprocessing zksnarks with universal and updatable srs. Cryptology ePrint Archive, Paper 2019/1047 (2019), https://eprint.iacr.org/2019/1047
21. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020. pp. 769–793. Springer International Publishing, Cham (2020)

22. Danezis, G., Fournet, C., Groth, J., Kohlweiss, M.: Square span programs with applications to succinct nizk arguments. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014. pp. 532–550. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
23. Fauzi, P., Lipmaa, H., Siim, J., Zając, M., Ødegaard, A.T.: Verifiably-extractable owfs and their applications to subversion zero-knowledge. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021. pp. 618–649. Springer International Publishing, Cham (2021)
24. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology — CRYPTO' 86. pp. 186–194. Springer Berlin Heidelberg, Berlin, Heidelberg (1987)
25. Fuchsbauer, G.: Subversion-zero-knowledge snarks. In: Abdalla, M., Dahab, R. (eds.) Public-Key Cryptography – PKC 2018. pp. 315–347. Springer International Publishing, Cham (2018)
26. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953 (2019), https://eprint.iacr.org/2019/953
27. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. pp. 626–645. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
28. Goldreich, O., Oren, Y.: Definitions and properties of zero-knowledge proof systems. Journal of Cryptology **7**(1), 1–32 (Dec 1994). https://doi.org/10.1007/BF00195207, https://doi.org/10.1007/BF00195207
29. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1), 186–208 (1989). https://doi.org/10.1137/0218012
30. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) Advances in Cryptology - ASIACRYPT 2010. pp. 321–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
31. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 305–326. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
32. Groth, J., Ishai, Y.: Sub-linear zero-knowledge argument for correctness of a shuffle. In: Smart, N. (ed.) Advances in Cryptology – EUROCRYPT 2008. pp. 379–396. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
33. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-snarks. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018. pp. 698–728. Springer International Publishing, Cham (2018)
34. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 581–612. Springer (2017). https://doi.org/10.1007/978-3-319-63715-0_20, https://doi.org/10.1007/978-3-319-63715-0_20
35. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Kosaraju, S.R., Fellows, M., Wigderson, A., Ellis, J.A. (eds.) Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada. pp. 723–732. ACM, New York (1992)

36. Labs, M.: Bellman : Bellman zksnark library for community with ethereum's bn256 support, github https://github.com/matter-labs/bellman
37. Lindell: Parallel coin-tossing and constant-round secure two-party computation. Journal of Cryptology **16**(3), 143–184 (Jun 2003). https://doi.org/10.1007/s00145-002-0143-7, https://doi.org/10.1007/s00145-002-0143-7
38. Lipmaa, H.: Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In: Cramer, R. (ed.) Theory of Cryptography. pp. 169–189. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
39. Lipmaa, H.: A unified framework for non-universal snarks. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13177, pp. 553–583. Springer (2022). https://doi.org/10.1007/978-3-030-97121-2_20, https://doi.org/10.1007/978-3-030-97121-2_20
40. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge snarks from linear-size universal and updateable structured reference strings. Cryptology ePrint Archive, Paper 2019/099 (2019), https://eprint.iacr.org/2019/099
41. Micali, S.: Computationally sound proofs. SIAM Journal on Computing **30**(4), 1253–1298 (2000). https://doi.org/10.1137/S0097539795284959
42. Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U. (ed.) Advances in Cryptology — EUROCRYPT '96. pp. 387–398. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
43. Setty, S.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. pp. 704–737. Springer International Publishing, Cham (2020)
44. Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zk-snarks without trusted setup. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 926–943 (2018). https://doi.org/10.1109/SP.2018.00060

Acknowledgements We would like to thank Xinxuan Zhang, Shunli Ma and anonymous reviewers for their theoretical guidance and valuable suggestions.

A Sub-protocols Summary

Here, we summarize the sub-protocols witch are introduced in **Section 4** and required in our zk-SNARK protocol. Recall that we now have many protocols to prove different consistencies:

1. $\Pi_0^1 : P = \left[\sum_{j=0}^{n-1} t_j x^j\right]_T$, $Q = \left[\sum_{j=0}^{n-1} t_j x^j\right]_2$
2. $\Pi_0^2 : P = \left[t_n x^{n-1}\right]_T$, $Q = \left[t_n x^{n-1}\right]_2$
3. $\Pi_0^3 : P = [\alpha]_T$, $Q = [\alpha]_1$
4. $bat\Pi_2 : P_i = \left[x^i\right]_1$, $Q_i = \left[x^{i-1}\right]_1$, $T = [x]_1$ for $i \in [1, n-1]$
5. $bat\Pi_3 : P_i = \left[x^i\right]_1$, $Q_i = \left[x^i\right]_2$ for $i \in [1, n-1]$
6. $\Pi_4 : P = [\delta]_1$, $Q = [\delta]_2$, $T = [\delta]_T$ and $P = [\beta]_1$, $Q = [\beta]_2$, $T = [\beta]_T$
7. $bat\Pi_5^1 : P = \left[\sum_{i=0}^{n-2} x^i t(x) c^i\right]_T$, $Q_i = \left[(x^i t(x))/\delta\right]_1$, $T = [\delta]_T$ for $i \in [0, n-2]$
8. $bat\Pi_5^2 : P = \left[\sum_{i=0}^{n-2} x^i \sum_{j=0}^{n-1} t_j x^j c^i\right]_T$, $Q_i = \left[x^i\right]_1$, $T = \left[\sum_{j=0}^{n-1} t_j x^j\right]_T$ for $i \in [0, n-2]$
9. $bat\Pi_5^3 : P = \left[\sum_{i=0}^{n-2} x^{i+1} t_n x^{n-1} c^i\right]_T$, $Q_i = \left[x^{i+1}\right]_1$, $T = \left[t_n x^{n-1}\right]_T$ for $i \in [0, n-2]$
10. $bat\Pi_6^1 : P = \left[\sum_{i=l+1}^{m} (\beta u_i(x) + \alpha v_i(x) + w_i(x)) c^{i-l-1}\right]_T$, $Q_i = [(\beta u_i(x) + \alpha v_i(x) + w_i(x))/\delta]_1$, $T = [\delta]_T$ for $i \in [l+1, m]$
11. $bat\Pi_6^2 : P = \left[\sum_{i=l+1}^{m} \beta \sum_{j=0}^{n-1} u_{i,j} x^j c^{i-l-1}\right]_T$, $Q_i = \left[\sum_{j=0}^{n-1} u_{i,j} x^j\right]_1$, $T = [\beta]_T$ for $i \in [l+1, m]$
12. $bat\Pi_6^3 : P = \left[\sum_{i=l+1}^{m} \alpha \sum_{j=0}^{n-1} v_{i,j} x^j c^{i-l-1}\right]_T$, $Q_i = \left[\sum_{j=0}^{n-1} v_{i,j} x^j\right]_2$, $T = [\alpha]_T$ for $i \in [l+1, m]$
13. $bat\Pi_0^4 : \left[\sum_{i=l+1}^{m} \sum_{j=0}^{n-1} w_{i,j} x^j c^{i-l-1}\right]_T$, $Q_i = \left[\sum_{j=0}^{n-1} w_{i,j} x^j\right]_1$ for $i \in [l+1, m]$

Notably, there is a trick that after the computation of $\left[\sum_{i=1}^{n-1} x^{i-1} c_3^{i-1}\right]_1$ in $bat\Pi_5^2$, the verifier could also help the prover to compute the $\left[\sum_{i=1}^{n-1} x^i c_3^{i-1}\right]_1$ in $bat\Pi_5^3$ by proving the exponential multiplication relation for $\left(\left[\sum_{i=1}^{n-1} x^i c_3^{i-1}\right]_1, \left[\sum_{i=1}^{n-1} x^{i-1} c_3^{i-1}\right]_1, [x]_1\right)$. However, this trick does not generate any advantage in term of prover efficiency. This is because with the knowledge of c_3, it is easy to compute $\left[\sum_{i=1}^{n-1} x^{i-1} c_3^{i-1}\right]_1$ and $\left[\sum_{i=1}^{n-1} x^i c_3^{i-1}\right]_1$ together through $O(n)$ exponential multiplication operations.

B Definitions

B.1 Witness Extended Emulation

In this appendix subsection, we will introduce the definition of *witness extended emulation*. It is another definition of knowledge soundness which has

been used for example in [15, 18] and defined in [37, 32]. Informally, A protocol is said to have this property if for any prover $\overline{P}$ there exists an efficient algorithm, with rewindable oracle access to $\overline{P}$, that outputs a transcript and, if this transcript is accepting then it outputs, with overwhelming probability, a witness as well. The transcripts generated by this algorithm are required to be indistinguishable from conversations between $\overline{P}$ and an honest verifier.

Definition 6. *(Statistical witness extended emulation [15]).* (P, V) *has statistical witness extended emulation if for all deterministic polynomial time* $\overline{P}$ *there exists an expected polynomial time emulator* $\mathcal{E}$ *such that for all interactive adversaries* $\mathcal{A}$

$$\begin{aligned} &\Pr\left[(u,s) \leftarrow \mathcal{A}\left(1^{\lambda}\right); tr \leftarrow \left\langle \overline{P}(u,s), V(u)\right\rangle : \mathcal{A}(tr) = 1\right] \\ \overset{s}{\approx} &\Pr\left[\begin{array}{l}(u,s) \leftarrow \mathcal{A}\left(1^{\lambda}\right); (tr,w) \leftarrow \mathcal{E}^{\left\langle \overline{P}(u,s), V(u)\right\rangle}(u) : \\ \mathcal{A}(tr) = 1 \text{ and if } tr \text{ is accepting then } (u,w) \in R\end{array}\right] \end{aligned}$$

where the oracle called by $\mathcal{E}^{\left\langle \overline{P}(u,s), V(u)\right\rangle}$ *permits rewinding to a specific point and resuming with fresh randomness for the verifier from this point onwards.*

In the definition, s can be interpreted as the state of $\overline{P}$, including the randomness. So, whenever $\overline{P}$ is able to make a convincing argument when in state s, $\mathcal{E}$ can extract a witness. This is why we call it an argument of knowledge.

B.2 Generalized Forking Lemma

In this appendix subsection, we will introduce the generalized forking lemma which has been formally discussed in [15].

Suppose that we have a $(2\mu+1)$-move public-coin argument with μ challenges, $x_1, \ldots, x_\mu$ in sequence. Let $k_i \geq 1$ for $1 \leq i \leq \mu$. Consider $\prod_{i=1}^{\mu} k_i$ accepting transcripts with challenges in the following tree format. The tree has depth μ and $\prod_{i=1}^{\mu} k_i$ leaves. The root of the tree is labelled with the statement. Each node of depth $i < \mu$ has exactly k_i children, each labelled with a distinct value for the i th challenge x_i.

This can be referred to as an $(k_1, \ldots, k_\mu)$-tree of accepting transcripts. All of our arguments allow a witness to be extracted efficiently from an appropriate tree of accepting transcripts. This is a natural generalisation of special-soundness for Σ-protocols, where $\mu = 1$ and $k = 2$. For simplicity in the following lemma, we assume that the challenges are chosen uniformly from $\mathbb{Z}_p$ where $|p| = \lambda$, but any sufficiently large challenge space would suffice. A public-coin protocol is said to be (unconditionally) $(k_1, \ldots, k_\mu)$ special sound if there exists a polynomial time algorithm that on input a statement x and a $(k_1, k_2, \ldots, k_\mu)$ tree of accepting transcripts, outputs a witness w for x.

Lemma 1 (Generalized Forking Lemma [15]). *Let* (P, V) *be a* $(2\mu + 1)$*-move, public-coin interactive protocol. Let* χ *be a witness extraction algorithm*

that always succeeds in extracting a witness from an $(k_1, \ldots, k_\mu)$*-tree of accepting transcripts in probabilistic polynomial time. Assume that* $\prod_{i=1}^{\mu} k_i$ *is bounded above by a polynomial in the security parameter* λ*. Then* (P, V) *has witness-extended emulation.*

This lemma show us that the $(k_1, \ldots, k_\mu)$-special soundness property implies the witness extended emulation property.

C Proofs

An appendix contains supplementary information that is not an essential part of the text itself but which may be helpful in providing a more comprehensive understanding of the research problem or it is information that is too cumbersome to be included in the body of the paper.

C.1 proof of Theorem 1

Proof. **Completeness**: If the prover run this protocol honestly, then we can do a simple verification:

$$P_i^e \cdot t_1 = \left(g^{x^i}\right)^e \cdot g^r = g^{ex^i+r} = g^z$$
$$Q_i^e \cdot t_2 = \left(h^{x^i}\right)^e \cdot h^r = h^{ex^i+r} = h^z$$

So, the completeness property is satisfied.

Special Soundness: We show that there exists an efficient extractor that on input two accepting transcripts computes a witness for relation R_3.

Let (t_1, t_2, e, z) and (t_1, t_2, e', z') be two accepting transcripts for distinct challenges $e, e' \in \mathbb{Z}_q$. The fact that they all pass the verification implies the following equation:

$$g^z = P_i^e \cdot t_1, h^z = Q_i^e \cdot t_2$$
$$g^{z'} = P_i^{e'} \cdot t_1, h^{z'} = Q_i^{e'} \cdot t_2$$

Then we have

$$g^{z-z'} = P_i^{e-e'} = g^{x^i(e-e')}$$
$$h^{z-z'} = Q_i^{e-e'} = h^{x^i(e-e')}$$

Thus we get $log_g P_i = log_h Q_i = \frac{z-z'}{e-e'} = x^i$, the soundness of the protocol follows.

Honest-Verifier Zero-Knowledge:To show that the protocol achieves the honest-verifier zero-knowledge property, we construct a simulator S to simulate the view of the verifier V. The simulator S first picks the challenges $e \leftarrow \mathbb{Z}_p$. Then S generate $z \leftarrow \mathbb{Z}_p$ and computes

$$t_1 = \frac{g^z}{P_i^e}, t_2 = \frac{h^z}{Q_i^e}$$

We note that the distribution of (t_1, t_2, e, z) in this simulation is perfectly indistinguishable from that of a real execution. This is due to the fact that given random $e \in \mathbb{Z}_p$, the z is uniformly random both in a real execution and in simulation. As for t_1, t_2, they are uniquely determined via verification equations.

C.2 Proof of Theorem 2

Proof. **Completeness**: If the prover run this protocol honestly, then we can do a simple verification:

$$P_i^e \cdot t = \left(T^{x^{i-1}}\right)^e \cdot T^r = T^{ex^{i-1}+r} = T^z$$
$$Q_i^e \cdot t_1 = \left(g^{x^{i-1}}\right)^e \cdot g^r = g^{ex^{i-1}+r} = g^z$$

So, the completeness property is satisfied.

Special Soundness: We show that there exists an efficient extractor that on input two accepting transcripts computes a witness for relation R_2.

Let (t, t_1, e, z) and (t, t_1, e', z') be two accepting transcripts for distinct challenges $e, e' \in \mathbb{Z}_q$. The fact that they all pass the verification implies the following equation:

$$T^z = P_i^e \cdot t, g^z = Q_i^e \cdot t_1$$
$$T^{z'} = P_i^{e'} \cdot t, g^{z'} = Q_i^{e'} \cdot t_1$$

Then we have

$$T^{z-z'} = P_i^{e-e'}$$
$$g^{z-z'} = Q_i^{e-e'}$$

We get $log_T P_i = \frac{z-z'}{e-e'}, log_g Q_i = \frac{z-z'}{e-e'}$, thus the witness $x^{i-1} = \frac{z-z'}{e-e'}$, the soundness of the protocol follows.

Honest-Verifier Zero-Knowledge:To show that the protocol achieves the honest-verifier zero-knowledge property, we construct a simulator S to simulate the view of the verifier V. The simulator S first picks the challenges $e \leftarrow \mathbb{Z}_p$. Then S generate $z \leftarrow \mathbb{Z}_p$ and computes

$$t = \frac{T^z}{P_i^e}, t_1 = \frac{g^z}{Q_i^e}$$

We note that the distribution of (t, t_1, e, z) in this simulation is perfectly indistinguishable from that of a real execution. This is due to the fact that given random $e \in \mathbb{Z}_p$, the z is uniformly random both in a real execution and in simulation. As for t, t_1, they are uniquely determined via verification equations.

C.3 Proof of Theorem 4

Proof. **Completeness**: If the prover run this protocol honestly, then we can do a simple verification:

$$\left(\prod_{i=1}^{n-1} P_i^{c^{i-1}}\right)^e \cdot t_1 = g^{e(\sum_{i=1}^{n-1} x^i c^{i-1})+r} = g^z$$
$$\left(\prod_{i=1}^{n-1} Q_i^{c^{i-1}}\right)^e \cdot t_2 = h^{e(\sum_{i=1}^{n-1} x^i c^{i-1})+r} = h^z$$

So, the completeness property is satisfied.

Special Soundness: We show that there exists an efficient extractor that on input two accepting transcripts computes a witness.

For one fixed $c_j, j \in [1, n-1]$ we let $(c_j, t_{1j}, t_{2j}, e_j, z_j)$ and $(c_j, t_{1j}, t_{2j}, e_j^{'}, z_j^{'})$ be two accepting transcripts for distinct challenges $e, e' \in \mathbb{Z}_q$. The fact that they all pass the verification implies the following equation:

$$g^z = \left(\prod_{i=1}^{n-1} P_i^{c_j^{i-1}}\right)^{e_j} \cdot t_1, h^z = \left(\prod_{i=1}^{n-1} Q_i^{c_j^{i-1}}\right)^{e_j} \cdot t_2$$
$$g^{z'} = \left(\prod_{i=1}^{n-1} P_i^{c_j^{i-1}}\right)^{e_j^{'}} \cdot t_1, h^{z'} = \left(\prod_{i=1}^{n-1} Q_i^{c_j^{i-1}}\right)^{e_j^{'}} \cdot t_2$$

Then we have

$$g^{z-z'} = P^{e_j - e_j^{'}} = g^{\sum_{i=1}^{n-1} x^i c_j^{i-1}(e_j - e_j^{'})}$$
$$h^{z-z'} = Q^{e_j - e_j^{'}} = h^{\sum_{i=1}^{n-1} x^i c_j^{i-1}(e_j - e_j^{'})}$$

Thus we get $\sum_{i=1}^{n-1} x^i c_j^{i-1} = \frac{z-z'}{e_j - e_j^{'}}$.

Then, for every $c_j, j \in [1, n-1]$ we get

$$\begin{aligned} \textstyle\sum_{i=1}^{n-1} x^i c_1^{i-1} &= \tfrac{z-z'}{e_1 - e_1^{'}} \\ \textstyle\sum_{i=1}^{n-1} x^i c_2^{i-1} &= \tfrac{z-z'}{e_2 - e_2^{'}} \\ &\cdots \\ \textstyle\sum_{i=1}^{n-1} x^i c_{n-1}^{i-1} &= \tfrac{z-z'}{e_{n-1} - e_{n-1}^{'}} \end{aligned}$$

By the properties of vandermond matrix, we can find the unique solution for every x^i, the soundness of the protocol follows.

Honest-Verifier Zero-Knowledge:To show that the protocol achieves the honest-verifier zero-knowledge property, we construct a simulator S to simulate the view of the verifier V. The simulator S first picks the challenges $e, c \leftarrow \mathbb{Z}_p$. Then S generate $z \leftarrow \mathbb{Z}_p$ and computes

$$t_1 = \frac{g^z}{\left(\prod_{i=1}^{n-1} P_i^{c^{i-1}}\right)^e}, t_2 = \frac{h^z}{\left(\prod_{i=1}^{n-1} Q_i^{c^{i-1}}\right)^e}$$

We note that the distribution of (c, t_1, t_2, e, z) in this simulation is perfectly indistinguishable from that of a real execution. This is due to the fact that given random $e, c \in \mathbb{Z}_p$, the z is uniformly random both in a real execution and in simulation. As for t_1, t_2, they are uniquely determined via verification equations.

C.4 Proof of Theorem 6

Proof. **Completeness**: If the prover run this protocol honestly, then we can do a simple verification:

$$\left(g_1^{q_1(x)}\right)^e \cdot t_1 = g_1^{eq_1(x)+r} = g_1^z$$
$$T^e \cdot t_2 = g_\tau^{(eq_1(x)+r)h(x)} = \left(g_\tau^{h(x)}\right)^z$$

So, the completeness property is satisfied.

Special Soundness: We show that there exists an efficient extractor that on input two accepting transcripts computes the witness $q_1(x)$.

Let (T, t_1, t_2, e, z) and (T, t_1, t_2, e', z') be two accepting transcripts for distinct challenges $e, e' \in \mathbb{Z}_q$. The fact that they all pass the verification implies the following equation:

$$g_1^z = \left(g_1^{q_1(x)}\right)^e \cdot t_1,\ \left(g_\tau^{h(x)}\right)^z = T^e \cdot t_2$$
$$g_1^{z'} = \left(g_1^{q_1(x)}\right)^{e'} \cdot t_1,\ \left(g_\tau^{h(x)}\right)^{z'} = T^{e'} \cdot t_2$$

Then we have

$$g_1^{z-z'} = \left(g_1^{q_1(x)}\right)^{e-e'} = g_1^{q_1(x)(e-e')}$$
$$\left(g_\tau^{h(x)}\right)^{z-z'} = T^{e-e'} = g_\tau^{q_1(x)h(x)(e-e')}$$

Thus we get $log_{g_\tau^{h(x)}} T = \frac{z-z'}{e-e'} = q_1(x)$, the special soundness of the protocol follows.

Honest-Verifier Zero-Knowledge:To show that the protocol achieves the honest-verifier zero-knowledge property, we construct a simulator S to simulate the view of the verifier V. The simulator S first picks the challenges $e \leftarrow \mathbb{Z}_p$, Then S generate $z \leftarrow \mathbb{Z}_p$. It is obvious that the S could compute the $g_1^{q_1(x)}$, $g_\tau^{h(x)}$ and $T = g_\tau^{q_1(x)h(x)}$ by itself with the public input

$$\left(\left\{g_\tau^{x^i}\right\}_{i\in[0,d],\tau\in\{1,2\}}, h(X), q_1(X) \in \mathbb{F}_{<d}[X]\right)$$

Finally, the S compute

$$t_1 = \frac{g_1^z}{\left(g_1^{q_1(x)}\right)^e}, t_2 = \frac{\left(g_\tau^{h(x)}\right)^z}{T^e}$$

We note that the distribution of (T, t_1, t_2, e, z) in this simulation is perfectly indistinguishable from that of a real execution. This is due to the fact that given random $e \in \mathbb{Z}_p$, the z is uniformly random both in a real execution and in simulation. As for t_1, t_2 (T is trivial), they are uniquely determined via verification equations.

D Some Σ-protocol schemes

Protocol 3 Σ-protocol Π_5^1 for relation R_5^1
Σ-protocol to prove the exponential multiplication

PUBLIC PARAMETERS:pp
FOR EVERY $i \in [0, n-2]$
INPUT:$(pp, P_i = \left[x^i t(x)\right]_T, Q_i = \left[\frac{x^i t(x)}{\delta}\right]_1, T = [\delta]_T ; \frac{x^i t(x)}{\delta} \in \mathbb{Z}_p)$
$P_i = T^{\frac{x^i t(x)}{\delta}} \wedge Q_i = g^{\frac{x^i t(x)}{\delta}}$

Prover		Verifier
Pick $r \in \mathbb{Z}_p$ Compute $t = T^r$ $t_1 = g^r$		
	$\xrightarrow{t,t_1}$	
		Pick $e \in \mathbb{Z}_p$
	$\xleftarrow{e}$	
Compute $z = e\frac{x^i t(x)}{\delta} + r$		
	$\xrightarrow{z}$	
		Check $T^z \stackrel{?}{=} P_i^e \cdot t$ $g^z \stackrel{?}{=} Q_i^e \cdot t_1$

Fig. 5: The Σ-protocol for Relation R_5^1

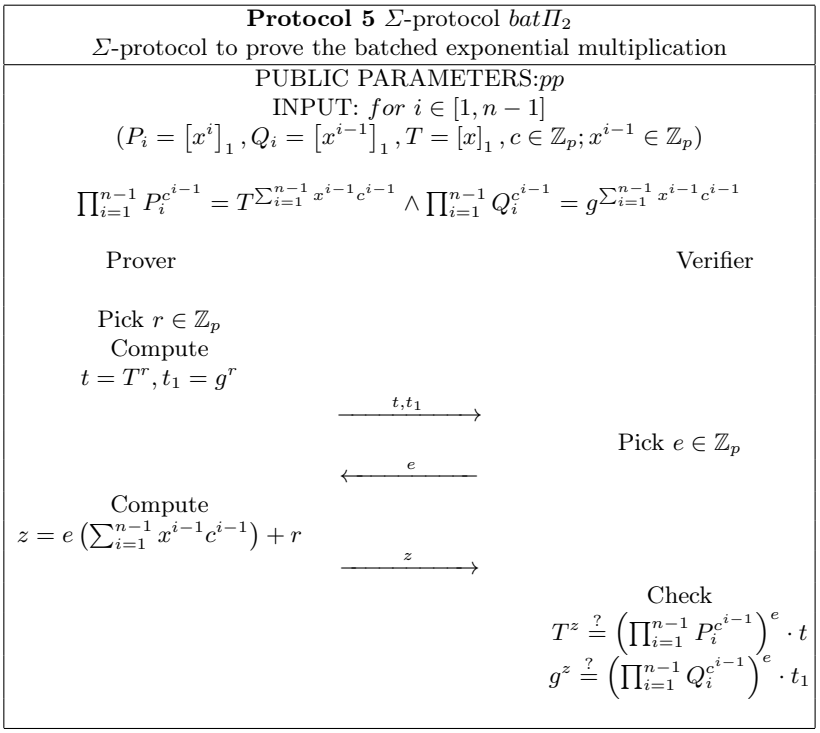

Fig. 6: The Batched Σ-protocol for Relation R_2

Protocol 6 Σ-protocol $batII_5^1$
The batched exponential multiplication proof with extra elements

PUBLIC PARAMETERS:pp

INPUT: $for\ i \in [0, n-2]$

$\left(P = \left[\sum_{i=0}^{n-2} x^i t(x) c^i\right]_T, Q_i = \left[\frac{x^i t(x)}{\delta}\right]_1, T = [\delta]_T, c \in \mathbb{Z}_p; \frac{x^i t(x)}{\delta} \in \mathbb{Z}_p\right)$

$$P = T^{\sum_{i=0}^{n-2} \frac{x^i t(x)}{\delta} c^i} \wedge \prod_{i=0}^{n-2} Q_i^{c^i} = g^{\sum_{i=0}^{n-2} \frac{x^i t(x)}{\delta} c^i}$$

Prover		Verifier
Pick $r \in \mathbb{Z}_p$ Compute $t = T^r, t_1 = g^r$	$\xrightarrow{t, t_1}$	
		Pick $e \in \mathbb{Z}_p$
	$\xleftarrow{e}$	
Compute $z = e\left(\sum_{i=0}^{n-2} \frac{x^i t(x)}{\delta} c^i\right) + r$	$\xrightarrow{z}$	
		Check $T^z \stackrel{?}{=} P^e \cdot t$ $g^z \stackrel{?}{=} \left(\prod_{i=0}^{n-2} Q_i^{c^i}\right)^e \cdot t_1$

Fig. 7: The Batched Version of Σ-protocol II_5^1

Protocol 7 Σ-protocol $bat\Pi_0^4$
The batched exponential multiplication proof with extra elements

PUBLIC PARAMETERS:pp

INPUT: $for\ i \in [l+1, m]$

$\left(P_i = \left[\sum_{j=0}^{n-1} w_{i,j} x^j\right]_1, Q = \left[\sum_{i=l+1}^{m} c^i \sum_{j=0}^{n-1} w_{i,j} x^j\right]_T, c \in \mathbb{Z}_p; \sum_{j=0}^{n-1} w_{i,j} x^j \in \mathbb{Z}_p\right)$

$\prod_{i=l+1}^{m} (P_i)^{c^i} = g^{\sum_{i=l+1}^{m} c^i \sum_{j=0}^{n-1} w_{i,j} x^j} \wedge Q = e(g,h)^{\sum_{i=l+1}^{m} c^i \sum_{j=0}^{n-1} w_{i,j} x^j}$

Prover		Verifier
Pick $r \in \mathbb{Z}_p$ Compute $t_1 = g^r, t_2 = e(g,h)^r$		
	$\xrightarrow{t_1, t_2}$	
		Pick $e \in \mathbb{Z}_p$
	$\xleftarrow{e}$	
Compute $z = e\left(\sum_{i=l+1}^{m} c^i \sum_{j=0}^{n-1} w_{i,j} x^j\right) + r$		
	$\xrightarrow{z}$	
		Check $g^z \stackrel{?}{=} \left(\prod_{i=l+1}^{m} (P_i)^{c^i}\right)^e \cdot t_1$ $e(g,h)^z \stackrel{?}{=} Q^e \cdot t_2$

Fig. 8: The Batched Version of Σ-protocol Π_0^4