VECTIS: Efficient Batching Framework for Group-based CP-SNARKs

Byeongjun Jang
Kookmin University
Seoul, Republic of Korea
sunjbs@kookmin.ac.kr

Gweonho Jeong
Hanyang University
Seoul, Republic of Korea
kwonhojeong@hanyang.ac.kr

Hyuktae Kwon
Hanyang University
Seoul, Republic of Korea
hyuktaekwon00@gmail.com

Jihye Kim
Kookmin University
Zkrypto Inc
Seoul, Republic of Korea
jihyek@kookmin.ac.kr

Hyunok Oh
Hanyang University
Zkrypto Inc
Seoul, Republic of Korea
hoh@hanyang.ac.kr

ABSTRACT

Blockchain applications in finance and identity management increasingly require scalable and privacy-preserving solutions. Cryptographic commitments secure sensitive data on-chain, but verifying properties of these commitments efficiently remains challenging, particularly in large-scale scenarios. For multiple commitments, CP-SNARKs, a family of zk-SNARKs, enhance prover efficiency by shifting large-cost operations outside the circuit and verifying linkages between commitments, but incur verifier-side overhead due to linkage checks. Verification costs grow with the number of commitments, leading to inefficiencies in key size, proof size, and verification time.

We propose **VECTIS**, an efficient batching framework for proving multiple commitments. Our approach aggregates multiple commitments into a single batched commitment, enabling the linking proof system to operate on the aggregated commitment instead of individual commitments, thereby significantly reducing the overall verification cost.

Experimental results show meaningful efficiency gains. For 2^{16} commitments, **VECTIS** reduces the verification time to 0.064s, achieving over 30× improvement compared to LegoSNARK's 1.972s. These results show **VECTIS**'s potential for enabling scalable and efficient privacy-preserving solutions in blockchain applications.

1 INTRODUCTION

With the rise of blockchain-based applications across various domains—ranging from finance (e.g., decentralized exchanges, lending platforms, and asset tokenization) to identity (e.g., self-sovereign identity solutions, credential management, and cross-border verification)—privacy concerns have emerged as a pressing issue that must be addressed. One common strategy to preserve privacy is to publish sensitive data on the ledger in the form of commitments, cryptographic constructs that conceal the underlying information while retaining its authenticity. However, commitments alone are insufficient; an appropriate proof mechanism is required to verify properties of the hidden data without revealing it. To address this, zk-SNARKs have emerged as a powerful cryptographic tool, enabling efficient verification of statements—such as whether a user's assets are fully accounted for within an exchange's total liabilities or whether they meet multi-attribute eligibility requirements (e.g., a series of lending or insurance criteria)—without exposing the underlying confidential details [12, 14, 17, 21, 22].

Despite the advantages of zk-SNARKs, significant challenges arise when applying them to large-scale scenarios involving numerous commitments. Generating proofs with zk-SNARKs typically requires provers to perform computations that are orders of magnitude more expensive than executing the original code. While this overhead is manageable for small-scale commitments, generating proofs for a large number of commitments becomes computationally impractical.

To address prover-side inefficiencies, Commit-and-Prove SNARKs (CP-SNARKs) [2, 6], a class of zk-SNARKs, have been introduced. They significantly reduce the complexity of generating proofs by eliminating the need to compute complex commitment operations, such as group exponentiations, within the proof circuit. In traditional zk-SNARKs, commitments are often handled as part of the arithmetic circuit, increasing circuit size and computation costs for the prover. CP-SNARKs bypass this issue by performing heavy computations outside the circuit and treating commitments as external inputs to the proof. This approach minimizes the circuit size, reduces proving time and memory usage, and improves efficiency for large-scale commitments.

While CP-SNARKs optimize the proving process, this approach requires an additional proof system, known as the linking proof system. This system ensures the linkage between pre-committed values and proof-dependent commitment, guaranteeing the consistency of multiple commitments within a single proof. However, verification cost in such systems is proportional to the number of commitments, posing significant scalability challenges for the verifier, particularly in blockchain environments. For example, when QA-NIZK [16] is employed for the linking proof, verification involves $O(l)$ pairing operations with $O(l)$-sized verification key, where l is the number of commitments, although the proof size remains constant. Pairing operations, which require computationally intensive elliptic curve computations, are especially costly in blockchain systems. As a concrete example, verifying 2^{10} commitments can cost approximately 135 USD, assuming an ETH price of 3,000 USD, a gas price of 1 Gwei, and an average pairing operation cost of 0.135 USD[1].

[1]On Ethereum, a single pairing operation consumes 45,000 gas, storing data in the storage consumes 22,100 gas, and loading data from the storage consumes 2,100 gas.

Compressed Σ-protocols [3, 4] aim to mitigate the verifier cost by replacing pairing operations with lighter group exponentiations. This approach reduces computational overhead for individual verifications but retains linear scaling in verifier complexity because each commitment must still be processed separately. Furthermore, the logarithmic growth of proof size and the linear growth of verification key size in these systems not only increases on-chain storage costs but also affects transaction throughput and verification latency, creating additional barriers for real-world deployments.

In this paper, we propose **VECTIS**, a novel batching framework for CP-SNARKs that utilizes the homomorphic properties of group-based commitments, such as Pedersen commitments, to significantly reduce verifier overhead. Our approach aggregates multiple commitments into a single aggregated commitment using a random linear combination, ensuring the independence and separability of each committed value. By enabling the linking proof system to handle the aggregated commitment rather than individual commitments, the number of verification operations is reduced from $O(l)$ to a constant. Consequently, the proof size and verification key size are also reduced to constants, regardless of the batch size.

This aggregation eliminates the need for equality checks on individual commitments within the linking proof system, significantly reducing computational overhead. To construct the aggregated commitment, a random linear combination of pre-committed values is applied, preserving the independence of each witness. The same randomness is then used to compute a proof-dependent commitment for the aggregated witness, ensuring consistency between the pre-committed and aggregated values. Instead of verifying $O(l)$ individual witnesses, the linking proof system verifies only the equality of two aggregated commitments, resulting in constant verification time and key size.

The prover ensures the correctness of this aggregation process by generating a proof within the SNARK circuit, which validates that all individual commitments are correctly aggregated. This additional proof relies on lightweight field operations within the circuit, making the process efficient and minimizing computational overhead for the prover. Furthermore, this design ensures that the aggregated commitment maintains the integrity and independence of individual messages, preventing fabrication during aggregation.

By replacing the verifier overhead of the linking proof system with lightweight aggregation, our framework achieves significant performance improvements. While aggregation introduces additional group computations for the verifier, these computations remain far lighter or fewer than those required in traditional linking systems. When integrated with a QA-NIZK-based linking system [16], pairing operations required for verification are reduced from $O(l)$ to a constant. Similarly, applying our aggregated commitment to a compressed-Σ protocol [3, 4] reduces both the proof size and the verification key size to constants, while also decreasing the number of group operations performed by the verifier by approximately threefold.

Table 1 presents a comparison of the linking costs, including aggregation computations, incurred when constructing CP-SNARKs, referred to as **Lego**. It highlights the significant efficiency improvements achieved by our proposed scheme **VECTIS** over existing linking proof systems. First, compared to the **Lego** scheme with QA-NIZK [16], our approach eliminates $l - 1$ pairing operations, replacing them with l+1 group exponentiations, which are computationally less expensive. Second, when compared to the compressed Σ-protocol, the proposed scheme reduces approximately $2l + 4 \log l$ group exponentiations. These reductions significantly lower the computational overhead for the verifier. Additionally, the proposed scheme achieves a constant proof size, independent of l, whereas the compressed Σ-protocol requires a proof size of $O(\log l)$. Similarly, the verification key size is reduced from $O(l)$ to a constant. These improvements make the scheme more efficient and scalable for large-scale applications, as reflected in Table 1.

1.1 Our Contributions

- **A new batching framework for cc-SNARK**. We propose a new bathcing framework, called **VECTIS**, to commit-carrying SNARK that efficiently proves multiple Pedersen commitments with a single proof, significantly reducing the computational overhead compared to the traditional approach (i.e. in-the-circuit). Additionally, we enhance the efficiency of commit-and-prove SNARKs by leveraging our batching technique, which aggregates multiple commitments into a single commitment. This approach significantly reduces the verification cost while maintaining the integrity of the proof, providing a practical advantage in scenarios requiring efficient verification of commitments.
- **Implementation and Evaluations**. We have implemented and empirically tested our scheme, demonstrating its practical efficiency and scalability in handling large batches of commitments. For 2^{10} commitments, our approach can generate proofs in just 59 ms, whereas the in-the-circuit method takes approximately 57.203s (970x faster). In the comparison experiments, LegoSNARK, at 2^{16} with an empty relation, exhibits a prover time of approximately 0.177s, while our system shows 1.413s, making our system about 8 times slower. However, for the verifier time, our system takes 0.064s compared to 1.972s for LegoSNARK. From an application perspective, performance metrics on the blockchain demonstrate that our system can verify 2^{10} commitments at about 5.2 transactions per second (TPS), offering more practical utility compared to LegoSNARK's 0.5 TPS. Furthermore, we have conducted an experiment under an age checking relation. Proving 2^{13} commitments takes 14.85s in our system, while LegoSNARK takes 16.11s. This highlights our scheme's efficiency when handling complex relations such as non-arithmetic operations.

1.2 Applications

Commitment can be employed to validate confidential data while preserving privacy. However, if numerous commitments are involved, verifying each one can be inefficient. Motivated by this, we propose a batching framework that enables simultaneous verification of multiple commitments. To emphasize the importance of our framework, we present high-level use cases of batching functionality in the following applications.

1.2.1 Proof of solvency. It is a fundamental concept in financial applications, providing a formal mechanism for institutions to demonstrate their ability to meet all outstanding liabilities. It involves

Table 1: Comparison of additional costs required for constructing CP-SNARKs [2, 6] under Pedersen engine using different linking proof systems. We denote l as the number of commitments, E as group exponentiations, and P as pairings. NIZK **and** Comp-Σ **correspond to QA-NIZK [16] and the compressed Σ-protocol [3, 4], respectively. $\mathcal{P}$ represents the prover computational cost, $\mathcal{V}$ represents the verifier computational cost, $|\pi|$ denotes the proof size, and $|\mathsf{vk}|$ denotes the verification key size. For the prover, we analyze the computational cost assuming the Groth16 [11] as the underlying SNARK system.**

	$\mathcal{P}$	$\mathcal{V}$	$\|\pi\|$	$\|\mathsf{vk}\|$
Lego$_{\mathsf{NIZK}}$	$O(l)\ E_1$	$(l+2)\ P$	$1\ \mathbb{G}_1$	$(l+2)\ G_2$
VECTIS$_{\mathsf{NIZK}}$	$O(l)\ E_1,\ O(l)\ E_2$	$(l+1)\ E_1, 3P$	$2\ \mathbb{G}_1$	$4\ G_2$
Lego$_{\text{Comp-}\Sigma}$	$O(l)\ E_1$	$(3l+4\log l)\ E_1$	$(4\log l+2)\ \mathbb{G}_1, 4\ \mathbb{F}$	$(2l+2)\ \mathbb{G}_1$
VECTIS$_{\text{Comp-}\Sigma}$	$O(l)\ E_1,\ O(l)\ E_2$	$(l+7)\ E_1$	$3\ \mathbb{G}_1, 2\ \mathbb{F}$	$4\ \mathbb{G}_1$

verifying that an institution's total assets exceed its liabilities without disclosing sensitive financial information. In such scenarios, each customer's balance must remain confidential, and the individual can check their balance against the institution's reported totals. To protect individual privacy, account-related commitments are published on the blockchain. The batching technique enables proving that each commitments has been properly formed using a single proof. This functionality is essential for institutions that manage numerous individual customer balances, providing a robust and privacy-preserving solution in the financial sector.

1.2.2 Digital credentials. In scenarios where individuals or members of organizations need to maintain anonymity while proving ownership of credentials—such as digital certificates issued by authorities—each user's credentials are committed and stored by a third-party service, enhancing privacy and reducing data storage burdens on the individual. For instance, in settings where a service provider frequently validates their users' credentials against public commitments, the batching technique simplifies the process. Instead of generating a separate proof for each user, the service provider can accumulate multiple requests and generate a single proof that collectively validates all of them. This batching approach not only ensures privacy but also significantly reduces the computational and time costs associated with proof generation.

1.3 Related works

The approach on integrating different proof systems has progressed with the goal of getting computational efficiency, as evidenced by several studies [1, 2, 5–7, 9, 18, 20]. One important work of these studies, Chase et al. [7], provides a method that combines algebraic-based proofs, such as Σ-protocols, with garbled circuit proofs. This technique efficiently computes algebraic operations through algebraic-based proofs and handles non-algebraic operations using garbled circuit proofs. However, since this approach leverages a private garbling scheme from JKO13 [13], the necessity for private garbled circuits imposes limitations on the applicability to proof systems that do not employ such circuits. LegoSNARK [6] introduces a generic framework for constructing composite systems from different proof systems by linking different systems using a generic compiler to build the generic integration of proof systems. In the paper, it shows a high-performance commit-and-prove proof system for proving Pedersen commitment, instantiated in a modular manner. This approach is more efficient than traditional methods (i.e. the commitment is encoded in the circuit). Eclipse [2] has crafted a compiler that transforms a proof system based on algebraic holographic proof into CP-SNARK. Using compressed Σ-protocols, the proof systems such as Plonk [10], Sonic [19], and Marlin [8] can be instantiated into CP-SNARK with logarithmic proof size. In the recent study detailed in [20], the authors provide techniques for offloading non-native arithmetic operations from zero-knowledge circuits. By employing Σ-protocols for proving algebraic operations and SNARK for non-algebraic parts, the paper reduces the computational burden typically associated with embedding complex arithmetic in zero-knowledge circuits.

1.4 Organizations

Section 2 outlines the notations and informal definitions of the building blocks used in this paper. Section 3 introduces our batching framework, followed by the security proofs presented in Section 4. Section 5 of our scheme, and Section 6 evaluates its performance. Finally, Section 7 concludes the paper.

2 PRELIMINARIES

2.1 Notations

We use $\boldsymbol{a}$ or $\{a_i\}$ for the list of elements, which is equivalent to a vector. We denote by λ a security parameter and by $\epsilon(\cdot)$ as a negligible function. Let $\mathbb{F}$ denote a finite field and $\mathbb{G}$ denote a group. A bilinear group generator $\mathcal{BG}$ takes a security parameter as input in unary and returns a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ consisting of cyclic groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of prime order p and a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Given a security parameter 1^λ, a relation generator $\mathcal{RG}$ returns a polynomial time decidable relation $\mathcal{R} \leftarrow \mathcal{RG}(1^\lambda)$. For $(\boldsymbol{x}, \boldsymbol{w}) \in \mathcal{R}$ we say $\boldsymbol{w}$ is a witness to the instance $\boldsymbol{x}$ being in the relation. We use the bracket for any bilinear group such as $[a]_s \equiv a \cdot g_s \in \mathbb{G}_s$.

2.2 Pedersen vector commitments

Pedersen vector commitment for vector $\boldsymbol{w}$ of size n can be expressed succinctly with the following algorithms:

- Ped.Setup(1^λ): chooses $g \xleftarrow{\$} \mathbb{G}$, $\boldsymbol{h} \xleftarrow{\$} \mathbb{G}^n$ from a domain $\mathcal{D}$. It outputs a commit key $\mathsf{ck} := (g, \boldsymbol{h})$.

- $\mathsf{Ped.Commit}(\mathsf{ck}, \boldsymbol{m}; o)$: returns $\mathsf{cm} := (o, \boldsymbol{m})^\top \cdot \mathsf{ck}$.
- $\mathsf{Ped.VerCom}(\mathsf{ck}, \mathsf{cm}, \boldsymbol{m}, o)$: returns true if $\mathsf{cm} = (o, \boldsymbol{m})^\top \cdot \mathsf{ck}$. Otherwise, false.

Lemma 2.1. *The Pedersen vector commitment is perfectly hiding and computationally binding if the discrete logarithm assumption holds.*

2.3 Succinct Non-interactive arguments of knowledge

Definition 2.2. A succinct non-interactive arguments of knowledge (SNARK) for $\mathcal{R}$ is a tuple of algorithms $\mathbf{\Pi}$ = (Setup, Prove, Verify) working as follows:

- $\mathsf{crs} := (\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{Setup}(\mathcal{R})$: takes a relation $\mathcal{R} \leftarrow \mathcal{RG}(1^\lambda)$ as input and returns a common reference string crs consisting of an evaluation key ek and a verification key vk.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{ek}, \boldsymbol{x}; \boldsymbol{w})$: takes an evaluation key ek, a statement $\boldsymbol{x}$, and a witness $\boldsymbol{w}$ as inputs, and returns a proof π.
- $\mathsf{true/false} \leftarrow \mathsf{Verify}(\mathsf{vk}, \boldsymbol{x}, \pi)$: takes a verification key vk, a statement $\boldsymbol{x}$, and a proof π as inputs and returns false ($reject$) or true ($accept$).

It satisfies the completeness, knowledge soundness, and succinctness:

Completeness. Given a true statement $\boldsymbol{x}$, for all relation $\mathcal{R}$ and for all $(\boldsymbol{x}; \boldsymbol{w}) \in \mathcal{R}$,

$$\Pr\left[\begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(\mathcal{R}), \\ \pi \leftarrow \mathsf{Prove}(\mathsf{ek}, \boldsymbol{x}, \boldsymbol{w}) \end{array} : \mathsf{Verify}(\mathsf{crs}, \boldsymbol{x}, \pi) = 1 \right] = 1$$

Knowledge Soundness. Knowledge soundness states that a prover must know a witness and such knowledge can be efficiently extracted from π by a knowledge extractor $\mathcal{E}$. Formally, the following is *negligible* for any PPT adversary $\mathcal{A}$.

$$\Pr\left[\begin{array}{lcl} \mathsf{crs} \leftarrow \mathsf{Setup}(\mathcal{R}), & & \mathsf{Verify}(\mathsf{crs}, \boldsymbol{x}^*, \pi^*) = \mathsf{true} \\ (\boldsymbol{x}^*, \pi^*) \leftarrow \mathcal{A}(\mathsf{crs}), & : & \wedge \\ \boldsymbol{w} \leftarrow \mathcal{E}_{\mathcal{A}}(trans_{\mathcal{A}}), & & (\boldsymbol{x}^*; \boldsymbol{w}) \notin \mathcal{R} \end{array}\right]$$

Succinctness. Succinctness states that the argument generates the proof of polynomial size in the security parameter, and the verifier's computation time is polynomial in the security parameter and in statement size.

Remark. A SNARK may also satisfy *zero-knowledge*. It states that the system does not leak any information besides the truth of the statement. This is modeled by a simulator that does not know the witness (but has some trapdoor information that enables it to simulate proofs). We refer to it as a zk-SNARK in this scenario.

2.3.1 Commit-carrying SNARK. There exists a variant of commit-and-prove SNARK ($\mathsf{SNARK_{cp}}$), referred to as a commit-carrying SNARK($\mathsf{SNARK_{cc}}$), which is a SNARK whose proof includes a commitment to the portion of witnesses. A $\mathsf{SNARK_{cc}}$ consists of a tuple of algorithms as follows:

- $\mathsf{crs} := (\mathsf{ck}, \mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{Setup}(\mathcal{R})$: takes a relation $\mathcal{R}$ as input, and outputs a common reference string which includes a commitment key ck, an evaluation key ek, and a verification key vk.
- $(\widetilde{\mathsf{cm}}, \pi; \widetilde{o}) \leftarrow \mathsf{Prove}(\mathsf{ek}, \boldsymbol{x}; \boldsymbol{w})$: takes an evaluation key ek, a statement $\boldsymbol{x}$ and a witness $\boldsymbol{w} := (\boldsymbol{u}, \boldsymbol{\omega})$ such that the relation $\mathcal{R}$ holds as inputs, and outputs a proof π, a proof-dependent commitment $\widetilde{\mathsf{cm}}$ and an opening $\widetilde{o}$ such that $\mathsf{VerCom}(\mathsf{ck}, \widetilde{\mathsf{cm}}, \boldsymbol{u}, \widetilde{o}) =$ true.
- $\mathsf{true/false} \leftarrow \mathsf{Verify}(\mathsf{vk}, \boldsymbol{x}, \widetilde{\mathsf{cm}}, \pi)$: takes a verification key vk, a statement $\boldsymbol{x}$, a proof-dependent commitment $\widetilde{\mathsf{cm}}$, a proof π as inputs, and outputs true if $(\boldsymbol{x}, \widetilde{\mathsf{cm}}, \pi) \in \mathcal{R}$, or false otherwise.

The $\mathsf{SNARK_{cc}}$ satisfies the properties of completeness, succinctness, knowledge soundness, zero-knowledge, and binding.

2.4 Σ-protocols

With an arbitrary relation $\mathcal{R}(\boldsymbol{x}, \boldsymbol{w})$, we briefly recapitulate Σ-protocols. A Σ-protocol for the relation $\mathcal{R}$ is a three-round interactive proof system between a prover (with $\boldsymbol{x}$ and $\boldsymbol{w}$) and a verifier (with $\boldsymbol{x}$). $\mathbf{\Pi}_\Sigma$ consists of a tuple of efficient algorithms (Com, Chl, Res) run as follows:

- $\mathcal{P}$ runs $\mathsf{Com}(\boldsymbol{x}, \boldsymbol{w}) \rightarrow a$: sends a commitment a
- $\mathcal{V}$ runs $\mathsf{Chl}(\cdot) \rightarrow c$: chooses a challenge c is distributed uniformly at random and sends c to $\mathcal{P}$.
- $\mathcal{P}$ runs $\mathsf{Res}(\boldsymbol{x}, \boldsymbol{w}, c) \rightarrow z$: returns some response value z.
- $\mathcal{V}$ runs $\mathsf{Verify}(\boldsymbol{x}, (a, c, z))$ returns a bit $b \in \{0, 1\}$. If $b = 1$, the verifier accepts the proof, otherwise rejects.

where (a, c, z) is called *transcript.* A Σ-protocol satisfies *completeness, special soundness, (honest verifier) zero-knowledge.*

Completeness. δ-completeness is satisfied if honestly-generated transcripts always verify, unless the prover aborts, which occurs with a probability of δ. Formally, $(\boldsymbol{x}, \boldsymbol{w}) \in \mathcal{R}$ we have that, for all honestly generated transcripts (a, c, z)

$$\Pr[\ \mathsf{Verify}(\boldsymbol{x}, a, c, z) = 1 \mid z \neq \perp] = 1, \text{and } \Pr[z = \perp] = \delta$$

Special soundness. Special soundness is satisfied if there exists an efficient extractor $\mathcal{E}$ that, for any PPT adversary $\mathcal{A}$, returns a statement $\boldsymbol{x}$ and two distinct accepting transcripts $(a, c_0, z_0), (a, c_1, z_1)$ where $c_0 \neq c_1$ such that $\mathcal{E}(\boldsymbol{x}, (a, c_0, z_0), (a, c_1, z_1))$ extracts a valid witness $\boldsymbol{w}$ with an exception probability ϵ, known as the *knowledge soundness error.*

Honest verifier zero-knowledge. Honest verifier zero-knowledge is satisfied if there exists a simulator $\mathcal{S}$ such that for all $(\boldsymbol{x}, \boldsymbol{w}) \in \mathcal{R}$ the following distributions are *indistinguishable.*

$$\{(a, z) | c \leftarrow \mathsf{Chl}(); a, z \leftarrow \mathcal{S}(\boldsymbol{x}, c)\}$$

$$\{(a, z) | c \leftarrow \mathsf{Chl}(); a \leftarrow \mathsf{Com}(\boldsymbol{x}, \boldsymbol{w}); z \leftarrow \mathsf{Res}(\boldsymbol{x}, \boldsymbol{w}, c)\}$$

3 A NEW BATCHING FRAMEWORK

3.1 Bifurcate commitment key

In the setup phase of cc-SNARK, a common reference string $\mathsf{crs} := (\mathsf{ck}, \mathsf{ek}, \mathsf{vk})$ is generated. Then we split the commitment key ck into two parts, denoted as

$$\mathsf{ck} := (\mathsf{ck}_1, \mathsf{ck}_2)$$

Specifically, ck_1 serves as the commitment key enabling the prover to generate multiple commitments that the prover aims to prove, while ck_2 acts as a bridge by encapsulating the knowledge of these commitments (e.g., m_i, o_i) as committed witnesses within the cc-SNARK. Each commitment cm_i is constructed as:

$$(\mathsf{cm}_i, o_i) = \mathsf{Com}(\mathsf{ck}_1, m_i)$$

We define $\mathsf{List}_{\mathsf{cm}}$ as a commitment list where each cm_i and o_i is generated by commitment scheme Com with commitment key ck_1 and values of m_i, for $i \in [l]$. At this point, the committed witness $\boldsymbol{u}$ consists of pairs (m_i, o_i) indexed by $i \in [l]$, which is used to compute a proof-dependent commitment $\widetilde{\mathsf{cm}}$ under a commitment key ck_2:

$$\boldsymbol{u} = \{m_i, o_i\}_{i \in [l]} \qquad \widetilde{\mathsf{cm}} = \mathsf{Com}(\mathsf{ck}_2, \boldsymbol{u}; \widetilde{o})$$

The prover's claim is that each commitment cm_i is committed to m_i, and the proof-dependent commitment $\widetilde{\mathsf{cm}}$ is committed to the committed witness $\boldsymbol{u}$. It can be expressed as:

$$\{\mathsf{VerCom}(\mathsf{ck}_1, \mathsf{cm}_i, m_i, o_i)\}_{i \in [l]} \wedge \mathsf{VerCom}(\mathsf{ck}_2, \widetilde{\mathsf{cm}}, \boldsymbol{u}, \widetilde{o})$$

3.2 Batched commitment with Σ-protocol

Recall that in the context of cc-SNARKs, each pair (m_i, o_i) of $\widetilde{\mathsf{cm}}$ can be viewed as a committed witness $\boldsymbol{u}$. However, $\widetilde{\mathsf{cm}}$ cannot be considered as the proof-dependent commitment for the multiple commitment relation $\mathcal{R}$, since we must prove the knowledge of each of commitment based on the *identical* commitment key ck_1. In other words, the linking proof system must be required since the existing cc-SNARK approach cannot prove the multiple commitments as a whole. To improve this limitation and facilitate the aggregation of multiple commitments into a single commitment, we use a randomness τ to apply unique encoding to each message and opening such as

$$\mathsf{agg}_m = \sum_{i=1}^{l} \tau^i \cdot m_i \qquad \mathsf{agg}_o = \sum_{i=1}^{l} \tau^i \cdot o_i.$$

By attaching a unique identifier through the linear combination with the randomness, each element is independently encoded. Then we add aggregated values $(\mathsf{agg}_m, \mathsf{agg}_o)$ into committed witness $\boldsymbol{u}$. We can prove that accumulated values $(\mathsf{agg}_m, \mathsf{agg}_o)$ are correctly derived from the pairs (m_i, o_i) within the circuit, which requires the prover to engage in only $O(l)$ field operations to evaluate agg_m and agg_o. The witness $\boldsymbol{w}$ can be expressed as follows.

$$\boldsymbol{w} = (\boldsymbol{u}, \boldsymbol{\omega}) = \left(\left\{\mathsf{agg}_m, \mathsf{agg}_o, \{m_i, o_i\}_{i \in [l]}\right\}, \boldsymbol{\omega}\right)$$

The prover sends a proof-dependent commitment $\widetilde{\mathsf{cm}}$ along with the proof π to the verifier, who then checks the validity of π using the commitments cm_i. However, there remains an issue to consider in our protocol: while it is possible to combine the knowledge of each commitment into a single value using randomness to ensure knowledge integrity, the prover can compute a simulated proof-dependent commitment $\widetilde{\mathsf{cm}}$. Our verifier knows the commitments cm_i but does not know the underlying knowledge for each commitment. This means that if the prover does not fix the proof-dependent commitment, it would be impossible to extract the knowledge of each commitment. To prevent this, we employ a Σ-protocol. Rather than simultaneously transmitting the proof-dependent commitment $\widetilde{\mathsf{cm}}$ and the proof, we first bind the knowledge within $\widetilde{\mathsf{cm}}$ and send it prior to the proof. Subsequently, the verifier sends a challenge τ to the prover in the Chl phase. This procedure ensures that the prover cannot generate the knowledge of $\mathsf{cm}_{\mathsf{agg}}$ before receiving the challenge. Upon receiving τ, the prover constructs $\mathsf{cm}_{\mathsf{agg}}$, and then sends the proof π for the $\mathsf{SNARK}_{\mathsf{cc}}$, excluding the proof-dependent commitment in the Res phase. The verifier can verify the proof with $\mathsf{List}_{\mathsf{cm}}$.

3.3 Putting Together

Our protocol is also a three-move protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ with a triple of algorithms: $\mathsf{Com}, \mathsf{Chl}, \mathsf{Res}$. We present the interactive form of the protocol, as it can straightforwardly be converted to a non-interactive version by applying the Fiat-Shamir transform, ensuring security in the random oracle model (ROM). Additionally, our protocol leverages commit-carrying SNARK ($\mathsf{SNARK}_{\mathsf{cc}}$) under Pedersen engines, which means that proof is constructed as group linear encoding.

Protocol. By $\boldsymbol{\Pi}_{\mathsf{cc}}.\mathsf{Setup}$ algorithm, the prover and verifier have a common reference string $\mathsf{crs} := (\mathsf{ck}, \mathsf{ek}, \mathsf{vk})$ for the following relation $\mathcal{R}_{\mathsf{Batch}}$.

$$\mathcal{R}_{\mathsf{Batch}}(\boldsymbol{x}; \boldsymbol{u}) = \left\{ \begin{array}{c} \tau; \\ (\mathsf{agg}_m, \mathsf{agg}_o), \\ (\{m_i\}_{i \in [l]}, \{o_i\}_{i \in [l]}) \end{array} : \begin{array}{c} \mathsf{agg}_m = \sum_{i=1}^{l} \tau^i \cdot m_i \\ \mathsf{agg}_o = \sum_{i=1}^{l} \tau^i \cdot o_i \end{array} \right\}$$

In our protocol, the prover's inputs are $\{m_i, o_i\}_{\in [l]}$ and a commitment list $\mathsf{List}_{\mathsf{cm}} := \{\mathsf{cm}_i\}_{i \in [l]}$ committing to values m_i with the opening o_i under the partial commitment key ck_1. The verifier's input to the protocol is a commitment list $\mathsf{List}_{\mathsf{cm}}$. In the committing phase, the prover $\mathcal{P}$ computes a proof-dependent commitment $\widetilde{\mathsf{cm}}$ as $\mathsf{Ped.Commit}(\mathsf{ck}_2, \{m_i, o_i\}_{i \in [l]}; \widetilde{o})$ where $\widetilde{o}$ is randomly chosen. Then $\mathcal{P}$ sends a message $\widetilde{\mathsf{cm}}$ to the verifier $\mathcal{V}$. Given the proof-dependent commitment $\widetilde{\mathsf{cm}}$, the verifier chooses a challenge $\tau \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{P}$. The prover runs $\boldsymbol{\Pi}_{\mathsf{cc}}.\mathsf{Prove}(\mathsf{ek}, \boldsymbol{x}, \boldsymbol{w})$ to generate a proof π_{cc}. The prover returns the proof π_{cc} as a response to the verifier $\mathcal{V}$. The verifier, given the commitment list $\mathsf{List}_{\mathsf{cm}}$ and the challenge τ, verifies the proof. We formally describe the protocol as an interactive Σ-protocol in Figure 1.

3.4 Extending to CP-SNARKs

Our protocol can be extended to a $\mathsf{SNARK}_{\mathsf{cp}}$ for multiple Pedersen commitments. We assume the existence of an external commitment scheme, denoted as $\mathsf{Com}_{\mathsf{Ext}}$, which satisfies the additively-homomorphic property. The goal is to prove multiple pre-computed l-commitments, $\mathsf{List}_{\hat{\mathsf{cm}}}$, through a commit-and-prove framework. Typically, a conventional commit-and-prove approach from $\mathsf{SNARK}_{\mathsf{cc}}$ requires that all l-commitments be included in the relation. However, since we can prove multiple commitments within a single proof by applying our protocol, we leverage this advantage in a commit-and-prove manner. More specifically, if our protocol verifies correctly, the verifier becomes aware of the validity of $\mathsf{cm}_{\mathsf{agg}}$. Assuming that the committed messages $\hat{m}_i$ of the l-commitments

Initialization : A trusted party $\mathcal{T}$, given a relation $\mathcal{R}_{\mathsf{Batch}}$ and a security parameter 1^{λ}, generates $(\mathsf{ck}, \mathsf{ek}, \mathsf{vk})$ through $\mathbf{\Pi}_{\mathsf{cc}}.\mathsf{Setup}$. The commmitment key ck can be split into $(\mathsf{ck}_1, \mathsf{ck}_2)$. We denote l-Pedersen commitments by $\mathsf{List}_{\mathsf{cm}} := \{\mathsf{cm}_i\}_{i\in[l]}$, each based on the batching key ck_1.

Prover and Verifier : A prover $\mathcal{P}$ knows the message m_i and the opening o_i corresponding to each commitment cm_i, $[\mathsf{cm}_i]_1$, as well as an auxiliary instance $\boldsymbol{x}$ and a witness $\boldsymbol{w}$. $\mathcal{P}$ then follows the subsequent procedure to generate a zk-SNARK proof π. The prover and verifier run the following Σ-protocol.

(1) $\mathcal{P}$ chooses a random $\widetilde{o} \overset{\$}{\leftarrow} \mathbb{F}$. Then $\mathcal{P}$ computes a proof-dependent commitment $\widetilde{\mathsf{cm}}$ as follows, where m_i and o_i refer to the message and opening of each commitment:
$$\widetilde{\mathsf{cm}} := \mathsf{Ped.Commit}(\mathsf{ck}_2, \{m_i, o_i\}_{i\in[l]}\,; \widetilde{o})$$
(2) $\mathcal{P}$ sends the proof-dependent commitment $\widetilde{\mathsf{cm}}$ to $\mathcal{V}$.
(3) $\mathcal{V}$ chooses a random $\tau \overset{\$}{\leftarrow} \mathbb{F}$, and returns it to $\mathcal{P}$.
(4) $\mathcal{P}$ computes the aggregated message agg_m and opening agg_o through the powers of τ as follows:
$$\mathsf{agg}_m := \sum_{i=1}^{l} \tau^i \cdot m_i, \quad \mathsf{agg}_o := \sum_{i=1}^{l} \tau^i \cdot o_i$$
(5) $\mathcal{P}$ runs $\mathbf{\Pi}_{\mathsf{cc}}.\mathsf{Prove}(\mathsf{ek}, \boldsymbol{x}, \boldsymbol{w})$, and generates a proof π_{cc} where $\boldsymbol{w} := (\boldsymbol{u}, \boldsymbol{\omega})$ and $\boldsymbol{u} := \{\mathsf{agg}_m, \mathsf{agg}_o, \{m_i, o_i\}_{i\in[l]}\}$.
(6) Finally, $\mathcal{P}$ returns π_{cc} to $\mathcal{V}$.
(7) $\mathcal{V}$ runs the following algorithm to verify the proof π_{cc}:
   (a) $\mathcal{V}$ computes $\mathsf{cm}_{\mathsf{agg}}$ using the powers of τ.
   (b) Lastly $\mathcal{V}$ runs
$$\mathbf{\Pi}_{\mathsf{cc}}.\mathsf{Verify}(\mathsf{vk}, \boldsymbol{x}, \mathsf{cm}_{\mathsf{agg}} \cdot \widetilde{\mathsf{cm}}, \pi_{\mathsf{cc}})$$
   where $\boldsymbol{x} := \tau$.

Figure 1: Our protocol for proving multiple Pedersen commitments

$\mathsf{List}_{\hat{\mathsf{cm}}}$, computed by $\mathsf{Com}_{\mathsf{Ext}}$, are identical to the messages m_i committed in our protocol, we prove their coherence not by individually proving each commitment corresponds to the same message, but by employing the challenge used in our protocol. By leveraging CP-Link (denoted in LegoSNARK [6]), which proves that two distinct Pedersen-like commitments under different keys open to the same vector, we can efficiently design a CP-SNARK system using a condensed form, represented as $\hat{\mathsf{cm}}_{\mathsf{agg}}$ and $\mathsf{cm}_{\mathsf{agg}}$. The relation $\mathcal{R}^{\mathsf{Batch}}_{\mathsf{Eq}}$ can be expressed as follows:

$$\mathcal{R}^{\mathsf{Batch}}_{\mathsf{Eq}}(\boldsymbol{x};\boldsymbol{w}) = \left\{ \begin{array}{c} (\hat{\mathsf{ck}}, \mathsf{ck}_1);\\ (\hat{\mathsf{cm}}_{\mathsf{agg}}, \mathsf{cm}_{\mathsf{agg}})\\ (\mathsf{agg}_m, \mathsf{agg}_o) \end{array} : \begin{array}{c} \hat{\mathsf{cm}}_{\mathsf{agg}} = \mathsf{Com}_{\mathsf{Ext}}(\hat{\mathsf{ck}}, \mathsf{agg}_m, \mathsf{agg}_o)\\ \mathsf{cm}_{\mathsf{agg}} = \mathsf{Com}(\mathsf{ck}_1, \mathsf{agg}_m, \mathsf{agg}_o) \end{array} \right\}$$

We can prove the above relation $\mathcal{R}^{\mathsf{Batch}}_{\mathsf{Eq}}$ using Σ-protocol, but similar to LegoSNARK [6], we can also use several schemes such as QA-NIZK [16] or compressed-Σ protocol [3]. The concrete construction is described in Figure 2.

We assume that there exists an external commitment scheme, denoted by $\mathsf{Com}_{\mathsf{Ext}}$, which satisfies the additively-homomorphic property. The algorithms of $\mathsf{Com}_{\mathsf{Ext}}$ consists of a tuple of algorithms (Setup, Com, VerCom). Setup outputs a commitment key $\widehat{\mathsf{ck}}$ and the committing algorithm Com computes a commitment $\widehat{\mathsf{cm}}$ upon receiving an input message m and a randomly chosen opening o. Note that we denote the CP-Link protocol as $\mathbf{\Pi}_{\mathsf{Link}}$, where $\mathsf{ek} = (\mathsf{ek}_{\mathsf{cc}}, \mathsf{ek}_{\mathsf{Link}})$, $\mathsf{vk} = (\mathsf{vk}_{\mathsf{cc}}, \mathsf{vk}_{\mathsf{Link}})$, and $\mathsf{ck} = (\widehat{\mathsf{ck}}, \mathsf{ck}_1, \mathsf{ck}_2)$.

In our commit-and-prove SNARK, we define the relation for multiple Pedensen commitments as follows:

$$\mathcal{R}^{\mathsf{Batch}}_{\mathsf{cp}}(\boldsymbol{x};\boldsymbol{w}) = \left\{ \begin{array}{c} (\hat{\mathsf{ck}}), (\{\hat{\mathsf{cm}}_i\}_{i\in[l]});\\ (\{m_i\}_{i\in[l]}, \{o_i\}_{i\in[l]}) \end{array} : \hat{\mathsf{cm}}_i = \mathsf{Com}_{\mathsf{Ext}}(\hat{\mathsf{ck}}, m_i, o_i) \right\}$$

We note that $\mathcal{R}^{\mathsf{Batch}}_{\mathsf{cp}}$ can be decomposed into two sub-relations, $\mathcal{R}_{\mathsf{Batch}}$ and $\mathcal{R}^{\mathsf{Batch}}_{\mathsf{Eq}}$, such that

$$\mathcal{R}^{\mathsf{Batch}}_{\mathsf{cp}}(\boldsymbol{x};\boldsymbol{w}) \iff (\mathcal{R}_{\mathsf{Batch}}(\boldsymbol{x};\boldsymbol{w}) \wedge \mathcal{R}^{\mathsf{Batch}}_{\mathsf{Eq}}(\boldsymbol{x};\boldsymbol{w})).$$

i.e., $\mathcal{R}^{\mathsf{Batch}}_{\mathsf{cp}}$ holds if and only if both $\mathcal{R}_{\mathsf{Batch}}$ and $\mathcal{R}^{\mathsf{Batch}}_{\mathsf{Eq}}$ are satisfied.

4 SECURITY PROOFS

Theorem 4.1. *Our protocol for the relation $\mathcal{R}_{\mathsf{Batch}}$ satisfies completeness, computational special soundness, and honest verifier zero-knowledge.*

Proof. We focus on special soundness and zero-knowledge in priority since completeness is relatively straightforward.

Completeness. It reduces to the completeness of commit-carrying SNARK. Therefore, the verification equation is always satisfied.

Special soundness. It reduces to the knowledge soundness of commit-carrying SNARK and the binding property of Pedersen commitments.

We define a knowledge extractor $\mathcal{E}$ that on input $\mathsf{List}_{\mathsf{cm}} \in \mathbb{G}_1^l$, and two accepting transcripts $\mathsf{Tr}_0 := (\widetilde{\mathsf{cm}}, \tau^*, \pi^*_{\mathsf{cc}})$ and $\mathsf{Tr}_1 := (\widetilde{\mathsf{cm}}, \tau', \pi'_{\mathsf{cc}})$ we must recover $\{m_i, o_i\}_{i\in[l]}$ such that

$$\{\mathsf{cm}_i = \mathsf{Ped.Commit}(\mathsf{ck}_1, m_i; o_i) \mid i \in [l]\}$$

If given the proofs are valid, by leveraging the knowledge extractor for the commit-carrying SNARK proofs we can extract the following with all but ϵ_{cc} which is the extractor failed error

$$\left\{\mathsf{agg}^*_m, \mathsf{agg}^*_o, \{m^*_i, o^*_i\}_{i\in[l]}, \widetilde{o}^*\right\}, \ \left\{\mathsf{agg}'_m, \mathsf{agg}'_o, \{m'_i, o'_i\}_{i\in[l]}, \widetilde{o}'\right\}$$

such that

$$\begin{aligned} \widetilde{\mathsf{cm}} &= \mathsf{Ped.Commit}(\mathsf{ck}_2, \{m^*_i, o^*_i\}_{i\in[l]}\,; \widetilde{o}^*) \\ &= \mathsf{Ped.Commit}(\mathsf{ck}_2, \{m'_i, o'_i\}_{i\in[l]}\,; \widetilde{o}') \end{aligned}$$

However, by the binding of **Lemma** 2.1, the probability of having two pairs of a Pedersen commitment is *negligible.* This means that $\{m^*_i, o^*_i\} = \{m'_i, o'_i\}$ with all but negligible probability $\widetilde{\epsilon}_{\mathsf{binding}}$. Therefore we have that

$$\mathsf{agg}^*_m = \sum_{i\in[l]} {\tau^*}^i \cdot m^*_i, \qquad \mathsf{agg}^*_o = \sum_{i\in[l]} {\tau^*}^i \cdot o^*_i, \qquad (1)$$

$$\mathsf{agg}'_m = \sum_{i\in[l]} {\tau'}^i \cdot m^*_i, \qquad \mathsf{agg}'_o = \sum_{i\in[l]} {\tau'}^i \cdot o^*_i. \qquad (2)$$

Protocol

$\mathcal{P}(\mathsf{ek}, \mathsf{ck}, \boldsymbol{x}, \boldsymbol{\omega}, \{m_i, o_i\}_{i\in[l]}, \mathsf{List}_{\widehat{\mathsf{cm}}})$		$\mathcal{V}(\mathsf{vk}, \mathsf{ck}, \boldsymbol{x}, \mathsf{List}_{\widehat{\mathsf{cm}}})$
$(\widetilde{\mathsf{cm}}, o) \leftarrow \mathsf{Com}(\mathsf{ck}_2, \{m_i, o_i\}_{i\in[n]}; \widetilde{o})$	$\xrightarrow{\widetilde{\mathsf{cm}}}$	
		$\tau \xleftarrow{\$} \mathbb{Z}_p^*$
	$\xleftarrow{\tau}$	
$\mathsf{agg}_m = \sum_{i=1}^{n} \tau^i \cdot m_i, \mathsf{agg}_o = \sum_{i=1}^{n} \tau^i \cdot o_i$		
$\mathsf{cm}_{\mathsf{agg}} := \mathsf{Com}(\mathsf{ck}, \mathsf{agg}_m; \mathsf{agg}_o)$		
$\widehat{\mathsf{cm}}_{\mathsf{agg}} := \mathsf{Com}(\widehat{\mathsf{ck}}, \mathsf{agg}_m; \mathsf{agg}_o)$		
$u := \left\{\mathsf{agg}_m, \mathsf{agg}_o, \{m_i, o_i\}_{i\in[n]}\right\}$		
$\boldsymbol{x}_{\mathsf{cc}} := \{\tau, \boldsymbol{x}\}$		
$\boldsymbol{x}_{\mathsf{Link}} := \{\mathsf{cm}_{\mathsf{agg}}, \widehat{\mathsf{cm}}_{\mathsf{agg}}\}$		
$\boldsymbol{\omega}_{\mathsf{Link}} := \{\mathsf{agg}_m, \mathsf{agg}_o\}$		
$\pi_{\mathsf{Link}} \leftarrow \Pi_{\mathsf{Link}}.\mathsf{Prove}(\mathsf{ek}_{\mathsf{Link}}, \boldsymbol{x}_{\mathsf{Link}}; \boldsymbol{\omega}_{\mathsf{Link}})$		
$\pi_{\mathsf{cc}} \leftarrow \Pi_{\mathsf{cc}}.\mathsf{Prove}(\mathsf{ek}_{\mathsf{cc}}, \boldsymbol{x}_{\mathsf{cc}}; (u, \boldsymbol{\omega}))$	$\xrightarrow{\pi_{\mathsf{cc}}, \pi_{\mathsf{Link}}, \mathsf{cm}_{\mathsf{agg}}}$	
		$\widehat{\mathsf{cm}}_{\mathsf{agg}} \leftarrow \prod_{i=1}^{l} \widehat{\mathsf{cm}}_i^{\tau^i}$
		$\boldsymbol{x}_{\mathsf{Link}} := \{\mathsf{cm}_{\mathsf{agg}}, \widehat{\mathsf{cm}}_{\mathsf{agg}}\}$
		$\Pi_{\mathsf{cc}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{cc}}, \boldsymbol{x}_{\mathsf{cc}}, \mathsf{cm}_{\mathsf{agg}} \cdot \widetilde{\mathsf{cm}}, \pi_{\mathsf{cc}})$
		$\Pi_{\mathsf{Link}}.\mathsf{Verify}(\mathsf{vk}_{\mathsf{Link}}, \boldsymbol{x}_{\mathsf{Link}}, \pi_{\mathsf{Link}})$

Figure 2: A CP-SNARK applying our framework for multiple Pedersen commitments where $\mathsf{List}_{\widehat{\mathsf{cm}}} := \{\widehat{\mathsf{cm}}_i\}_{i\in[l]}$

From the verification equation, we can compute a batched commitment $\mathsf{cm}_{\mathsf{agg}}$ using the randomized aggregation for $\mathsf{List}_{\mathsf{cm}}$ and the two challenges τ^*, τ' under the commitment key ck_1. For the legibility we denote these elements by $\overline{\mathsf{cm}}^*_{\mathsf{agg}}$ and $\overline{\mathsf{cm}}'_{\mathsf{agg}}$ respectively. We can express these batched commitments as

$$\overline{\mathsf{cm}}^*_{\mathsf{agg}} = \mathsf{Ped.Commit}(\mathsf{ck}_1, \overline{\mathsf{agg}}^*_m, \overline{\mathsf{agg}}^*_o)$$
$$\overline{\mathsf{cm}}'_{\mathsf{agg}} = \mathsf{Ped.Commit}(\mathsf{ck}_1, \overline{\mathsf{agg}}'_m, \overline{\mathsf{agg}}'_o).$$

where

$$\overline{\mathsf{agg}}^*_m = \sum_{i\in[l]} {\tau^*}^i \cdot m_i, \qquad \overline{\mathsf{agg}}^*_o = \sum_{i\in[l]} {\tau^*}^i \cdot o_i, \tag{3}$$
$$\overline{\mathsf{agg}}'_m = \sum_{i\in[l]} {\tau'}^i \cdot m_i, \qquad \overline{\mathsf{agg}}'_o = \sum_{i\in[l]} {\tau'}^i \cdot o_i. \tag{4}$$

Since the proofs are verified, we can say that with all but $\overline{\epsilon}_{\mathsf{binding}}$,

$$\mathsf{agg}^*_m = \overline{\mathsf{agg}}^*_m, \quad \mathsf{agg}^*_o = \overline{\mathsf{agg}}^*_o, \quad \mathsf{agg}'_m = \overline{\mathsf{agg}}'_m, \quad \mathsf{agg}'_o = \overline{\mathsf{agg}}'_o$$

Combining equations (1) to (4), we obtain the following result:

$$\sum_{i\in[l]} \left({\tau^*}^i - {\tau'}^i\right)\left(m^*_i - m_i\right) = 0,$$
$$\sum_{i\in[l]} \left({\tau^*}^i - {\tau'}^i\right)\left(o^*_i - o_i\right) = 0$$

Since the challenges (τ^*, τ') are distinct, $\left({\tau^*}^i - {\tau'}^i\right)$ terms cannot be 0, and for all $i \in [l]$

$$m^*_i = m'_i = m_i, \qquad o^*_i = o'_i = o_i.$$

Therefore $\mathcal{E}$ extracts a valid witness for the commitment list $(\mathsf{List}_{\mathsf{cm}})$ with error $\epsilon = \epsilon_{\mathsf{cc}} + \widetilde{\epsilon}_{\mathsf{binding}} + \overline{\epsilon}_{\mathsf{binding}}$. □

Honest Verifier Zero-knowledge. We informally show that it is hard for an adversary to distinguish simulated transcripts from real transcripts generated by an honest prover via a hybrid argument on the distribution of prover transcripts. Note that we denote a simulator as $\mathcal{S}$, which can choose $\widetilde{\mathsf{cm}}, \tau$ and simulate π_{cc} for the statement $\mathsf{List}_{\mathsf{cm}}$.

- Game_0: An honestly-generated prover transcript is the following tuple $(\widetilde{\mathsf{cm}}, \tau, \pi_{\mathsf{cc}})$.
- Game_1: π^*_{cc} is computed using $\mathcal{S}$ for $(\mathsf{List}_{\mathsf{cm}}, \widetilde{\mathsf{cm}}, \tau)$. The two distributions are indistinguishable by the zero-knowledge property of π_{cc}.

Since the simulated transcript is indistinguishable from the real transcript via hybrid argument, our protocol satisfies honest verifier zero-knowledge.

Theorem 4.2. *Our protocol for the relation $\mathcal{R}_{\mathsf{cp}}^{\mathsf{Batch}}$ satisfies completeness, computational special soundness, and honest verifier zero-knowledge.*

Proof. The proof of completeness, special soundness, and honestly verifier zero-knowledge for $\mathcal{R}_{\mathsf{cp}}^{\mathsf{Batch}}$ follows a similar structure to the proof of Theorem 4.1.

Completeness. It reduces to the completeness of linking proof system and commit-carrying SNARK. Therefore, the verification equation is always satisfied.

Special soundness. It reduces to the knowledge soundness of linking proof system, commit-carrying SNARK and the binding property of Pedersen commitments.

Honest Verifier Zero-knowledge. Our protocol achieves honest verifier zero-knowledge (HVZK) since both the underlying linking proof system and the commit-carrying SNARK guarantee HVZK. This ensures that a simulated transcript is indistinguishable from a real transcript generated by an honest prover. □

5 CONCRETE INSTANTIATIONS

This section provides a description of specific instantiations built upon two zk-SNARK schemes, Groth16 [11] and Plonk [10].

5.1 Instantiation based on Groth16 [11]

LegoSNARK [6] introduces the commit-carrying SNARK version (ccGro16) based on the SNARK of Gro16. The scheme is constructed from the Non-Interactive Linear Proof (NILP), a cryptograhpic primitive of Gro16.

Non-Interactive Linear Proof (NILP). A NILP comprises a tuple of algorithms (Setup, PrfMtx, Test) operating in the following manner:

- Setup: takes a relation $\mathcal{R}$ (e.g., QAP) as input, and returns vectors $\boldsymbol{\sigma} := (\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2) \in \mathbb{F}^{\kappa_1} \times \mathbb{F}^{\kappa_2}$.
- PrfMtx: given a relation $\mathcal{R}$ and a pair $(\boldsymbol{x}, \boldsymbol{w})$, outputs two matrices $(\boldsymbol{\Pi}_1, \boldsymbol{\Pi}_2) \in \mathbb{F}^{m_1 \times \kappa_1} \times \mathbb{F}^{m_2 \times \kappa_2}$. This facilitates the computation of a proof $(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2)$ as $(\boldsymbol{\Pi}_1 \cdot \boldsymbol{\sigma}_1, \boldsymbol{\Pi}_2 \cdot \boldsymbol{\sigma}_2)$.
- Test: upon receiving a relation $\mathcal{R}$ and a statement $\boldsymbol{x}$, yields a set of matrices $T_1, \dots, T_\mu \in F^{(m_1+\kappa_1)\times(m_2+\kappa_2)}$, with the acceptance condition for a proof $(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2)$ being $(\boldsymbol{\sigma}_1^\top, \boldsymbol{\pi}_1^\top) \cdot T_i \cdot (\boldsymbol{\sigma}_2^\top, \boldsymbol{\pi}_2^\top) = 0$ for all $i = \{1, \dots, \mu\}$.

Also, NILP satisfies completeness, statistical knowledge soundness, and zero-knowledge properties.

Here is a brief overview of how commit-carrying SNARK is derived from Groth16, as described in Figure 3. This construction aims to design a commit-carrying SNARK that provides double binding when proving the satisfiability of QAP relations s.t. $\mathcal{R}(\boldsymbol{x}, (\boldsymbol{u}, \boldsymbol{w}))$. This scheme includes a binding commitment to a portion $\boldsymbol{u}$ of the witness, with the public input being void (i.e. $\boldsymbol{x} = \perp$). LegoSNARK leverages the fact that witness-encoded polynomials are linearly independent, and its structure can be seen as linear group encoding (e.g., Pedersen commitment). Therefore, LegoSNARK adds a blinding factor, reconstructs the common reference string crs, and generates a new term $[D]_1$, which is a proof-dependent commitment. The term $[D]_1$ is structurally similar to a Pedersen commitment

```
NILP.Setup($\mathcal{R}$) $\to \boldsymbol{\sigma}$
$\alpha, \beta, \gamma, \delta, \boxed{\eta}, x \overset{\$}{\leftarrow} \mathbb{F}$, and define $y_i(x) := \beta a_i(x) + \alpha b_i(x) + c_i(x)$
$\boldsymbol{\sigma}_1 \leftarrow \left( \begin{matrix} 1, \alpha, \beta, \delta, \{x^i\}_{i=1}^{d-1}, \\ \left\{\frac{y_i(x)}{\gamma}\right\}_{i=1}^{l}, \left\{\frac{y_i(x)}{\delta}\right\}_{i=l+1}^{n}, \left\{\frac{x^i t(x)}{\delta}\right\}_{i=0}^{d-2}, \boxed{\frac{\eta}{\gamma}, \frac{\eta}{\delta}} \end{matrix} \right)$
$\boldsymbol{\sigma}_2 \leftarrow \left(1, \beta, \gamma, \delta, \{x^i\}_{i=1}^{d-1}\right)$
return $\boldsymbol{\sigma} := (\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2) \in \mathbb{F}^{(m+2d+6)} \times \mathbb{F}^{d+4}$

NILP.PrfMtx($\mathcal{R}, \boldsymbol{\sigma}, \boldsymbol{w}$) $\to (\boldsymbol{\Pi}_1, \boldsymbol{\Pi}_2)$
parse $\boldsymbol{w}$ as $(\boldsymbol{u}, \boldsymbol{\omega})$, $\boldsymbol{\sigma}$ as $(\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2)$ , and $r, s, \boxed{v} \overset{\$}{\leftarrow} \mathbb{F}$
$\boldsymbol{\Pi}_1 \in \mathbb{F}^{3\times(m+2d+6)}, \boldsymbol{\Pi}_2 \in \mathbb{F}^{1\times(d+4)}$
 s.t. $(A, C, D)^\top = \boldsymbol{\Pi}_1 \cdot \boldsymbol{\sigma}_1, B = \boldsymbol{\Pi}_2 \cdot \boldsymbol{\sigma}_2$
$A \leftarrow \alpha + \sum_{i=0}^{n} w_i a_i + r\delta \,;\, B \leftarrow \beta + \sum_{i=0}^{n} w_i b_i(x) + s\delta$
$C \leftarrow \sum_{i=l+1}^{n} w_i \frac{y_i(x)}{\delta} + \sum_{i=0}^{d-2} \frac{h_i x^i t(x)}{\delta} + As + Br - rs\delta - \boxed{\frac{v\eta}{\delta}}$
$\boxed{D \leftarrow \sum_{i=0}^{l} \frac{w_i y_i(x)}{\gamma} + \frac{v\eta}{\gamma}}$

NILP.Test($\mathcal{R}$) $\to$ true/false
check $A \cdot B = \alpha \cdot \beta + C \cdot \delta + D \cdot \gamma$
```

Figure 3: A NILP tailored for an augmented QAP relation, underpinning the ccGro16 [6]. The boxed elements indicate terms introduced in the modification from Gro16 [11] to the construction of ccGro16.

and is verified through the following verification equation as

$$\mathsf{Ped.VerCom}(\mathsf{ck}, [D]_1, \boldsymbol{u}, v) \overset{?}{=} \mathsf{true/false}$$

5.1.1 Our instantiation. We introduce $\mathbf{VECTIS}_{\mathsf{Gro16}}$ described in Figure 4, which is designed from ccGro16 by applying our Σ-protocol. The commitment key ck generated during Setup can be viewed as $\left\{\frac{y_i(x)}{\gamma}, \frac{\eta}{\gamma}\right\}$. Without loss of generality, assume the public input consists solely of τ and the non-committed witness is empty. We denote the starting indices for ck_1 and ck_2 as pfx_1 and pfx_2, respectively. Thus ck_1 and ck_2 are represented as follows:

$$\mathsf{ck}_1 := \left\{\frac{y_{\mathsf{pfx}_1+i}(x)}{\gamma}\right\}, \mathsf{ck}_2 := \frac{\eta}{\gamma}, \left\{\frac{y_{\mathsf{pfx}_2+i}(x)}{\gamma}\right\}$$

Each commitment cm_i in $\mathsf{List}_{\mathsf{cm}}$ is computed under ck_1. The proof-dependent commitment $\widetilde{\mathsf{cm}}$ is computed as $\widetilde{\mathsf{cm}} = \mathsf{Ped.Commit}(\mathsf{ck}_2, \{m_i, o_i\}\,; v)$ where v is an opening chosen by ccGro16. In the verification, the final proof-dependent commitment $[D]_1$ can be computed by computing $\mathsf{cm}_{\mathsf{agg}}$ and performing a group addition of three group elements: $[\mathsf{cm}_{\mathsf{agg}}]_1$, $[\widetilde{\mathsf{cm}}]_1$, and $[PI]_1$, where $[PI]_1$ represents the result of the linear encoding of the public input.

Initialization A trusted party $\mathcal{T}$, given an arbitrary relation $\mathcal{R}_{\text{Batch}}$ and a security parameter 1^λ, generates $(\mathsf{ck}, \mathsf{ek}, \mathsf{vk})$ through $\mathbf{\Pi}_{\text{ccGro16}}.\mathsf{Setup}$ where ck consists of $(\mathsf{ck}_1, \mathsf{ck}_2)$.

Prover and Verifier The prover and verifier execute the Σ-protocol described in Figure 1, applying the specific procedures outlined below, with all other remaining processes the same.

(1) In committing phase, $\mathcal{P}$ computes a proof-dependent commitment $\widetilde{\mathsf{cm}}$ by mapping each commitment's message $\boldsymbol{u}$ and opening o as follows:

$$\widetilde{\mathsf{cm}} := v \cdot \frac{\eta}{\gamma} + \sum_{i=0}^{l-1} \left(m_i \cdot \frac{y_{\text{pfx}+2i}(x)}{\gamma} + o_i \cdot \frac{y_{\text{pfx}+2i+1}(x)}{\gamma} \right)$$

where pfx refers to the prefix index related to the committed witness for each of cm.

(2) In proving phase, $\mathcal{P}$ runs $\mathbf{\Pi}_{\text{ccGro16}}.\mathsf{Prove}(\mathsf{ek}, \boldsymbol{x}; \boldsymbol{w})$, and generates a proof $\pi_{\text{cc}} := ([A]_1, [B]_2, [C]_1)$ in which $\mathcal{P}$ uses the chosen opening v in committing phase.

(3) In verification phase, the verifier $\mathcal{V}$ computes cm_{agg}, and then $\mathcal{V}$ generates a $[D]_1$ such that

$$[D]_1 := [\mathsf{cm}_{\text{agg}}]_1 \cdot [\widetilde{\mathsf{cm}}]_1 \cdot [PI]_1$$

Lastly, $\mathcal{V}$ runs the following verification as

$$e([A]_1, [B]_2) \stackrel{?}{=} e([a]_1, [b]_2) \cdot ([D]_1, [d]_2) \cdot ([C]_1, [c]_2)$$

where $[a]_1, [b]_2, [c]_1, [d]_1$ denote the group elements within the verification key vk of Gro16.

Figure 4: Our construction based on ccGro16

5.2 Instantiation based on Plonk [10]

Plonk [10] is a universal SNARK, which uses a polynomial commitment scheme to prove knowledge of any arbitrary relation $\mathcal{R}$. A polynomial commitment scheme (PCS) enables a prover to generate a commitment for a polynomial, valid at any given point of evaluation. Subsequently, the prover sends an opening proof for the verifier's evaluation. If the used PC is under group linear encoding for the public parameters such as KZG commitment [15], our batch commit-carrying SNARK scheme can also be applied in Plonk.

In the Plonk protocol, the prover proves knowledge of fan-in 2 and fan-out 1 gate values for each of the N gates. The constraint verification within Plonk is divided into *gate constraints* and *copy constraints*. PC is used to prove the validity of two constraints. We briefly describe Plonk [10] protocol.

Constraint system. Plonk designs its constraint system that requires satisfying the following equation through the use of *selector vectors* $(\boldsymbol{q_l}, \boldsymbol{q_r}, \boldsymbol{q_o}, \boldsymbol{q_m}, \boldsymbol{q_c})$ and *wire vectors* $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$

$$q_{l,i} \cdot a_i + q_{r,i} \cdot b_i + q_{o,i} \cdot c_i + q_{m,i} \cdot (a_i b_i) + q_{c,i} = 0$$

where we denote the left, right, and output wire as $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$.

Lagrange basis. Given a characteristic q of $\mathbb{F}$ and n satisfying $q \equiv 1 \mod n$, the multiplicative group of $\mathbb{F}^*$ reduces a subgroup $\mathbb{H} = \{\zeta, \zeta^2, \ldots, \zeta^n\}$ generated by an n-th primitive root of unity $\zeta \in \mathbb{F}^*$. By $\mathbb{H}$, we can construct a zero-polynomial $z_{\mathbb{H}}(X) = X^n - 1$, which can be expressed as $X^n - 1 = \prod_{i=1}^{n}(X - \zeta^i)$. There exists a Lagrange basis $Ł_i(X)$ for each $i \in [n]$ such that:

$$L_i(X) = \frac{\zeta^i (X^n - 1)}{n(X - \zeta^i)}$$

where

$$L_i(\zeta^i) = 1 \land L_i(\zeta^j) = 0 \ (i \neq j)$$

Copy constraint. Let multiple polynomials be $\boldsymbol{f} = (f_1, f_2, \ldots, f_\ell) \in \mathbb{F}[X]_\ell$ and $\sigma : \ell n \to \ell n$ be a permutation. For $\boldsymbol{g} = (g_1, g_2, \ldots, g_\ell) \in \mathbb{F}[X]_\ell$, we say that $\boldsymbol{f} = \sigma(\boldsymbol{g})$ if for each $i \in [n], j \in [\ell]$ the following holds for all $l \in [\ell n]$,

$$f_{((j-1)\cdot n+i)} := f(\zeta^i)_j \quad g_{((j-1)\cdot n+i)} := g_j(\zeta^i) \quad : g_l = f_{\sigma(l)}$$

Initialization A trusted party $\mathcal{T}$, given a security parameter 1^λ, generates a commitment key ck through $\mathbf{\Pi}_{PC}.\mathsf{Setup}$ where ck consists of $(\mathsf{ck}_1, \mathsf{ck}_2)$.

Prover and Verifier The prover and verifier execute the Σ-protocol described in Figure 1, applying the specific procedures outlined below, with all other remaining processes the same.

- In committing phase, $\mathcal{P}$ computes a proof-dependent commitment $\widetilde{\mathsf{cm}}$ by mapping each commitment's message $\boldsymbol{u}$ and opening v_1, v_2 as follows:

$$\widetilde{\mathsf{cm}} := (v_1 X + v_2) z_{\mathbb{H}}(X) + \sum_{i=0}^{l-1} \left(m_i \cdot L_{\text{pfx}_{2,2i}}(X) + o_i \cdot L_{\text{pfx}_{2,2i+1}}(X) \right)$$

  where pfx refers to the position related to the committed witness for each of cm.
- During the proving phase, the prover $\mathcal{P}$ runs $\mathbf{\Pi}_{\text{Plonk}}.\mathsf{Prove}(\mathsf{ek}, \boldsymbol{x}, \boldsymbol{w})$ to generate a Plonk proof π using a commitment and an evaluation to the polynomial defined as

$$f_u(X) := (v_1 X + v_2) z_{\mathbb{H}}(X) + \sum_{i \in [l]} -u_i L_i(X)$$

  Here, $\mathcal{P}$ utilizes the chosen opening v during the committing phase.
- In verification phase, the verifier $\mathcal{V}$ computes a batched commitment $[u_{\text{agg}}]_1$, and then $\mathcal{V}$ generates a $[u]_1$ such that

$$[u]_1 := [u_{\text{agg}}]_1 \cdot [\widetilde{\mathsf{cm}}]_1$$

  Lastly, $\mathcal{V}$ verifies the proof with $[u]_1$.

Figure 5: Our construction based on ccPlonk

5.2.1 *Our instantiation.* Plonk leverages a polynomial commitment scheme (PCS). If the PCS employs the [15] scheme, which uses a discrete log-based group encoding of polynomial, a polynomial $p(X)$ can be represented as follows, with a commitment key

$\mathsf{ck} := \left\{ g^{x^i} \right\}_{i \in [n]}$:

$$\mathsf{PCS.Com}(\mathsf{ck}, p(X)) := \prod_{i=0}^{n-1} g^{p_i \cdot x^i}$$

In the Plonk protocol, there exist the wire polynomials, which can be shortly expressed as follows, with random blinding scalars $(v_1, v_2, \dots, v_6) \in \mathbb{F}$

$$f_L(X) = (v_1 X + v_2) z_{\mathbb{H}}(X) + \sum_{i \in [n]} w_i L_i(X),$$
$$f_R(X) = (v_3 X + v_4) z_{\mathbb{H}}(X) + \sum_{i \in [n]} w_{n+i} L_i(X),$$
$$f_O(X) = (v_5 X + v_6) z_{\mathbb{H}}(X) + \sum_{i \in [n]} w_{2i+i} L_i(X)$$

For the public input polynomial $f_{pi}(X)$, we can separate the public input $\boldsymbol{pi} \in \mathbb{F}^l$ into two parts $(\boldsymbol{x}, \boldsymbol{u})$, where we regard $\boldsymbol{u}$ as a committed witness. As in previous descriptions, assume that the public input consists solely of τ, and the non-committed witness is empty. We denote the starting indices for ck_1 and ck_2 as pfx_1 and pfx_2, respectively. By integrating blinding factors (v_1, v_2) into the committed witness encoded polynomial $f_u(X)$ to ensure zero-knowledge, its form aligns with that of wire polynomials such as

$$f_u(X) = (v_1 X + v_2) z_{\mathbb{H}}(X) + \sum_{i \in [l]} -u_i L_i(X)$$

If we employ the KZG10 scheme, the polynomial commitment of $f_u(X)$ can be recast as a Pedersen vector commitment using a specific commitment key ck as

$$\left\{ [z_{\mathbb{H}}(X)], [z_{\mathbb{H}}(X)X], [(L_i(X))_{i \in [n]}] \right\}$$

We can split the commitment key into $(\mathsf{ck}_1, \mathsf{ck}_2)$ as

$$\mathsf{ck}_1 := [L_{\mathsf{pfx}_{1,i}}(X)], \mathsf{ck}_2 := [z_{\mathbb{H}}(X)], [z_{\mathbb{H}}(X)X], \left\{ [L_{\mathsf{pfx}_{2,i}}(X)] \right\}$$

Hence we can construct a proof-dependent commitment for the committed witness u, as $\widetilde{\mathsf{cm}} := \mathsf{Ped.Commit}(\mathsf{ck}_2, \{m_i, o_i\}; \widetilde{o})$ where the opening $\widetilde{o}$ consists of (s_1, s_2).

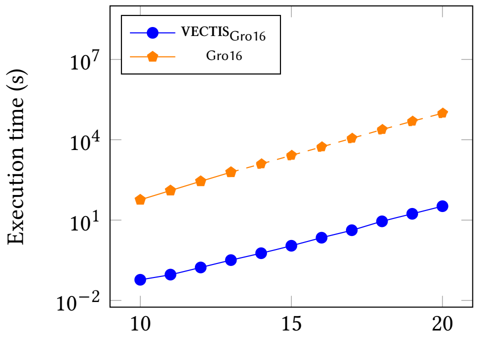
6 EXPERIMENTS

We implement our system on top of the Rust `Arkworks` library[2], which provides useful cryptographic primitives such as finite fields and elliptic curves. We adopted BN254 and BLS12-381, which are pairing-friendly elliptic curves offering 128 bits of security. BN254 is used to compare linking proof systems and to demonstrate practicality on a smart contract by comparing TPS and gas costs. Meanwhile, BLS12-381 is utilized to compare the proving time with the naive approach. We implement and evaluate our scheme based on Groth16 [11] , denoted as $\textbf{VECTIS}_{\mathsf{Gro16}}$. We evaluate the performance on an Apple M1 Pro with 32GB of RAM.

[2]https://github.com/arkworks-rs

6.1 Microbenchmark

Execution time. Figure 6 illustrates the performance for varying batch sizes, showing an increase in execution time as the batch size grows. To more explicitly demonstrate our performance, we also add results measured using a naive approach (i.e., in-the-circuit) within Groth16 [11]. In the scheme, although the number of constraints grows linearly with each exponential increment in batch size, the operations (e.g., group scalar exponentiation) in the naive approach are more expensive than the constraints required in our system. Specifically, for Groth16 [11], we can prove commitments for 2^{20} batches in approximately 33.024s, whereas the estimated time for the naive approach, which could not be measured in our device, would be around 98,000s showing a high-performance improvement. The dashed line in the graph represents estimated values.



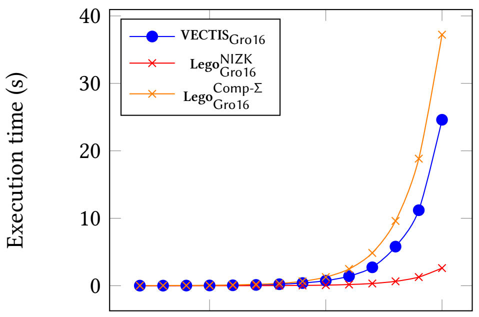
(a) Prover time for Groth16 [11] and VECTIS$_{\mathsf{Gro16}}$

Figure 6: Prover time for varying batch sizes, where the x-axis represents the batch size in log **and the dashed line represents the estimated value.**

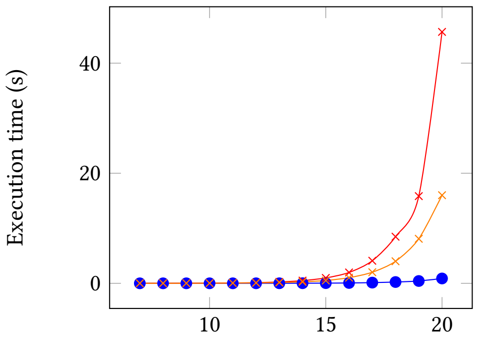
6.2 Comparison

Our framework efficiently supports the construction and batch verification of multiple commitments by commit-carrying SNARKs. For a more concrete analysis, we also provide a detailed examination of the commit-and-prove SNARK. To better demonstrate the practicality of our system, we have conducted a comparative analysis with the widely recognized LegoSNARK in Figure 7, Table 2, and Table 3. We have combined commit-carrying SNARKs based on Groth16 [11] with linking proof systems—the QA-NIZK [16] and the compressed-Σ-protocol—resulting in $\textbf{VECTIS}^{\mathsf{NIZK}}_{\mathsf{Gro16}}$, $\textbf{VECTIS}^{\mathsf{Comp}\text{-}\Sigma}_{\mathsf{Gro16}}$, $\textbf{Lego}^{\mathsf{NIZK}}_{\mathsf{Gro16}}$, and $\textbf{Lego}^{\mathsf{Comp}\text{-}\Sigma}_{\mathsf{Gro16}}$. We do not present the performance of each linking proof system unless there is significant performance difference using our scheme.

Prover and Verifier time. When comparing prover times in the Figure 9a, $\textbf{Lego}^{\mathsf{NIZK}}_{\mathsf{Gro16}}$ outperforms our scheme due to the absence of additional operations required to generate aggregated elements for a batched commitment in our approach. Interestingly, in $\mathcal{P}$'s time, $\textbf{Lego}^{\mathsf{Comp}\text{-}\Sigma}_{\mathsf{Gro16}}$ has asymptotically fewer computations than our scheme, but it is slower. This is because the aggregation of commitments can be computed more quickly using multi-scalar exponentiation. Specifically, at 2^{16}, our prover time is 1.413s, but $\textbf{Lego}^{\text{NIZK}}_{\text{Gro16}}$ achieves a performance of 0.177s while $\textbf{Lego}^{\text{Comp-}\Sigma}_{\text{Gro16}}$ requires 2.476s. Since the linking system is independent of the batch size, our scheme exhibits superior performance in verifier time as shown in Figure 7b.



(a) Prover time



(b) Verifier time

Figure 7: Comparison of prover time and verifier time between Lego$_{\text{Gro16}}$ and VECTIS$_{\text{Gro16}}$ for varying batch sizes in log scale

Key size. Examining the key sizes from Table 2 and 3, the evaluation key sizes generated by Gro16 are 4× to 6× bigger in our scheme, which necessitates additional keys due to the nature of aggregating commitments in the circuit. However, evaluation and verification key for linkable proof system in our scheme increased less than 1 **KB**. The verification key size in our scheme could be reduced to a constant if there is no need to maintain the committing key. In contrast, the sizes of both $\textbf{Lego}^{\text{NIZK}}_{\text{Gro16}}$ and $\textbf{Lego}^{\text{Comp-}\Sigma}_{\text{Gro16}}$ scale linearly with the batch size in the linking proof system.

6.3 Application

We provide detailed performance metrics in applications such as verifying proofs on blockchain platforms. We consider a scenario within smart contracts on the blockchain where users' commitments are stored, and a prover (e.g., bank, authority, etc.) must demonstrate the validity of these commitments by including proofs in transactions. This scenario aligns with simplified versions of applications such as proof of solvency or digital credentials. To compare the performance, we have measured the transactions per second (TPS) and gas costs for each system. Specifically, we generate 1,000 transactions and measure the time taken for these transactions to be confirmed on the network. Additionally, in this experiment, we utilize the BN254 curve, which is particularly advantageous as it is supported by precompiled functions in smart contract. TPS is computed by dividing the total number of transactions by the total time.

Table 2: Comparison of evaluation key sizes for varying batch sizes

Batch Size (log)	**VECTIS$_{\text{Gro16}}$ ek (KB)**	**Lego$^{\text{NIZK}}_{\text{Gro16}}$ ek (KB)**	**Lego$^{\text{Comp-}\Sigma}_{\text{Gro16}}$ ek (KB)**
7	130	34	21
8	259	66	42
9	517	132	83
10	1,033	264	164
11	2,065	526	328
12	4,129	1,050	656
13	8,258	2,099	1,311
14	16,516	4,196	2,622
15	33,031	8,390	5,244
16	66,061	16,779	10,486

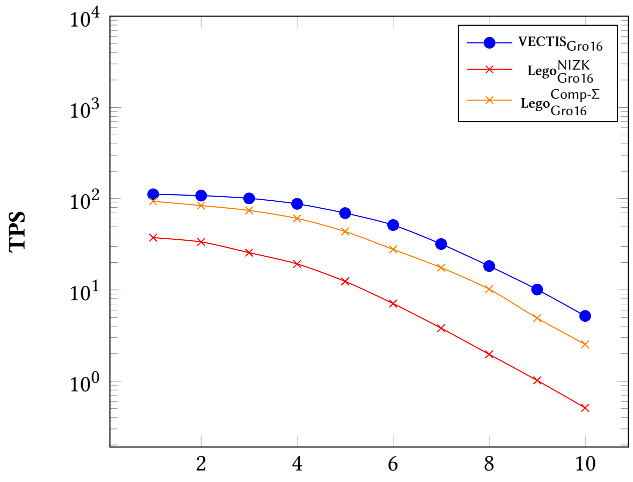
Figure 8 shows the performance comparison between $\textbf{Lego}_{\text{Gro16}}$ and our scheme ($\textbf{VECTIS}_{\text{Gro16}}$). As batch sizes increase, the difference in TPS becomes more pronounced. For instance, at a batch size of 2^{10}, $\textbf{Lego}^{\text{NIZK}}_{\text{Gro16}}$ and $\textbf{Lego}^{\text{Comp-}\Sigma}_{\text{Gro16}}$ can handle approximately 0.51 and 2.533 transactions per second respectively, whereas our scheme can process about 5.19 transactions per second, which indicates that we can verify about 5,300 commitments per second. This notable performance discrepancy is due to the computational overhead. $\textbf{Lego}^{\text{NIZK}}_{\text{Gro16}}$ needs $O(l)$ pairings and $\textbf{Lego}^{\text{Comp-}\Sigma}_{\text{Gro16}}$ needs $O(l)$ E_1 to rescale commitment keys, whereas in our scheme, $O(l)$ E_1 with a smaller constant factor is needed to aggregate commitments. Additionally, concerning verification key (vk), $\textbf{Lego}^{\text{NIZK}}_{\text{Gro16}}$ and $\textbf{Lego}^{\text{Comp-}\Sigma}_{\text{Gro16}}$ require the number of $\mathbb{G}_1$ and $\mathbb{G}_2$ elements to scale linearly with l.

Relation with age checking. To evaluate the practicality of our scheme, we assume simple application with age-checking relation as an example. Figure 9 presents the measured prover time compared to the baseline CP-SNARK approach. While our prover time appeared significantly slower when measured with an empty relation in Figure 7, the inclusion of actual relations shows that our scheme achieves comparable prover time performance.

In our scheme, the circuit involves a simple random linear combination, resulting in a complexity of $O(l)$, where l is the number of witnesses. This introduces only $O(l)$ additional field operations. For instance, in a 254-bit field, this overhead accounts for less than 1% of the total computational cost for the age-checking relation. Specifically, for proving 2^{13} commitments, our scheme requires only

Table 3: Comparison of verification key sizes for varying batch sizes

Batch Size	$\text{VECTIS}_{\text{Gro16}}$				$\text{Lego}^{\text{NIZK}}_{\text{Gro16}}$		$\text{Lego}^{\text{Comp-}\Sigma}_{\text{Gro16}}$
	cc	NIZK	Comp-Σ				
(log)	vk (B)	vk (B)	vk (B)	ck (KB)	vk (KB)	ck (KB)	vk (KB)
7				8	9	4	5
8				17	17	8	9
9				33	33	16	17
10				66	66	33	33
11				131	131	66	66
12	296	560	360	262	263	131	131
13				524	525	262	263
14				1,049	1,049	524	525
15				2,097	2,098	1,049	1,049
16				4,194	4,195	2,097	2,098



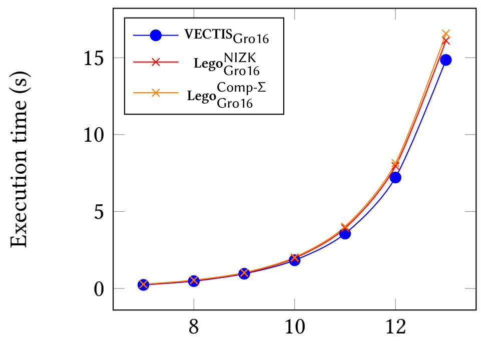
(a) TPS for $\text{Lego}_{\text{Gro16}}$ and $\text{VECTIS}_{\text{Gro16}}$

Batch size (log)	$\text{VECTIS}_{\text{Gro16}}$	$\text{Lego}^{\text{NIZK}}_{\text{Gro16}}$	$\text{Lego}^{\text{Comp-}\Sigma}_{\text{Gro16}}$
1	294K	488K	429K
2	309K	583K	527K
3	340K	773K	678K
4	403K	1,161K	932K
5	527K	1,918K	1,397K
6	777K	3,440K	2,277K
7	1,283K	6,485K	3,997K
8	2,282K	12,579K	7,389K
9	4,308K	24,779K	14,145K
10	8,323K	49,214K	27,648K

(b) Gas costs for $\text{Lego}_{\text{Gro16}}$ and $\text{VECTIS}_{\text{Gro16}}$

Figure 8: Performance for varying batch sizes.

14.85s, outperforming $\text{Lego}^{\text{NIZK}}_{\text{Gro16}}$ by 6.64% (16.11s) and $\text{Lego}^{\text{Comp-}\Sigma}_{\text{Gro16}}$ by 10.4% (16.57 s) Moreover, the performance gap widens as the number of commitments increases. These results highlight that our scheme efficiently handles practical applications with minimal overhead.



(a) Prover time

Figure 9: Comparison of prover time between $\text{Lego}_{\text{Gro16}}$ and $\text{VECTIS}_{\text{Gro16}}$ for varying batch sized in log scale, with age check constraints

7 CONCLUSION

Our paper proposes a batching framework **VECTIS**, which can efficiently prove and verify multiple commitments. As batch sizes increase, our performance surpasses that of other works in terms of verifier efficiency (i.e., time, key size) and proof size, although the proving time is slightly longer than in other works. Our work has significant potential for applications that demand efficient proving and verification, particularly when dealing with numerous commitments. It offers a far more efficient approach compared to the traditional method of verifying each commitment individually. Consequently, our scheme proves to be highly effective in applications that heavily rely on the use of commitments.

REFERENCES

[1] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. 2018. Non-Interactive Zero-Knowledge Proofs for Composite Statements. In *Advances in Cryptology – CRYPTO 2018*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer International Publishing, Cham, 643–673.

[2] Diego F. Aranha, Emil Madsen Bennedsen, Matteo Campanelli, Chaya Ganesh, Claudio Orlandi, and Akira Takahashi. 2022. ECLIPSE: Enhanced Compiling method for Pedersen-committed zkSNARK Engines. Springer-Verlag.

[3] Thomas Attema and Ronald Cramer. 2020. Compressed Σ-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics. In *Advances in Cryptology – CRYPTO 2020*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer International Publishing, Cham, 513–543.

[4] Thomas Attema, Ronald Cramer, and Serge Fehr. 2021. Compressing Proofs of k-Out-Of-n Partial Knowledge. In *Advances in Cryptology – CRYPTO 2021*, Tal Malkin and Chris Peikert (Eds.). Springer International Publishing, Cham, 65–91.

[5] Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. 2021. Lunar: a toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27*. Springer, 3–33.

[6] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2075–2092.

[7] Melissa Chase, Chaya Ganesh, and Payman Mohassel. 2016. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In *Advances in Cryptology–CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III 36*. Springer, 499–530.

[8] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. 2020. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In *Advances in Cryptology – EUROCRYPT 2020*, Anne Canteaut and Yuval Ishai (Eds.). Springer International Publishing, Cham, 738–768.

[9] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. 2015. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 253–270.

[10] Ariel Gabizon, Zachary J. Williamson, and Oana-Madalina Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. *IACR Cryptol. ePrint Arch.* 2019 (2019), 953.

[11] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*. 305–326. https://doi.org/10.1007/978-3-662-49896-5_11

[12] Zhangshuang Guan, Zhiguo Wan, Yang Yang, Yan Zhou, and Butian Huang. 2022. BlockMaze: An Efficient Privacy-Preserving Account-Model Blockchain Based on zk-SNARKs . *IEEE Transactions on Dependable and Secure Computing* 19, 03 (May 2022), 1446–1463. https://doi.org/10.1109/TDSC.2020.3025129

[13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. 2013. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 955–966. https://doi.org/10.1145/2508859.2516662

[14] Gweonho Jeong, Nuri Lee, Jihye Kim, and Hyunok Oh. 2023. Azeroth: Auditable Zero-Knowledge Transactions in Smart Contracts. *IEEE Access* 11 (2023), 56463–56480. https://doi.org/10.1109/ACCESS.2023.3279408

[15] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *Advances in Cryptology - ASIACRYPT 2010*, Masayuki Abe (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 177–194.

[16] Eike Kiltz and Hoeteck Wee. 2015. Quasi-Adaptive NIZK for Linear Subspaces Revisited. In *Advances in Cryptology - EUROCRYPT 2015*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 101–128.

[17] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 839–858.

[18] Helger Lipmaa. 2016. Prover-efficient commit-and-prove zero-knowledge SNARKs. In *Progress in Cryptology–AFRICACRYPT 2016: 8th International Conference on Cryptology in Africa, Fes, Morocco, April 13-15, 2016, Proceedings 8*. Springer, 185–206.

[19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. 2019. Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 2111–2128. https://doi.org/10.1145/3319535.3339817

[20] Michele Orrù, George Kadianakis, Mary Maller, and Greg Zaverucha. 2024. Beyond the circuit: How to Minimize Foreign Arithmetic in ZKP Circuits. Cryptology ePrint Archive, Paper 2024/265. https://eprint.iacr.org/2024/265 https://eprint.iacr.org/2024/265.

[21] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE symposium on security and privacy*. IEEE, 459–474.

[22] Karl Wüst, Kari Kostiainen, Noah Delius, and Srdjan Capkun. 2022. Platypus: A Central Bank Digital Currency with Unlinkable Transactions and Privacy-Preserving Regulation. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 2947–2960. https://doi.org/10.1145/3548606.3560617

A QUASI-ADAPTIVE NIZK ARGUMENTS FOR LINEAR SPACES

Intuitively, a QA-NIZK argument, as defined by Jutla and Roy, allows proving the membership of an instance $\boldsymbol{x}$ with a witness $\boldsymbol{w}$ in a language $\mathcal{L}$, defined by a relation $\mathcal{R}(\boldsymbol{x}, \boldsymbol{w})$. The QA-NIZK arguments consist of a set of PPT algorithms $\boldsymbol{\Pi}_{\text{.QA-NIZK}}$ = (KeyGen, Prove, Verify, Sim).

Kiltz and Wee [16] introduce constructions for QA-NIZK arguments for linear spaces. The linear space language $\mathcal{L}_{LS}$ can be represented as,

$$\mathcal{L}_{LS} = \left\{ [\boldsymbol{x}]_1 \in \mathbb{G}_1^n : \exists \boldsymbol{w} \in \mathbb{Z}_p \text{ s.t. } \boldsymbol{x} = M \cdot \boldsymbol{w} \right\}$$

, where the relation $\mathcal{R}$ is defined as

$$\mathcal{R}_M(\boldsymbol{x}; \boldsymbol{w}) = \left\{ (\boldsymbol{x}; \boldsymbol{w}) \in \mathbb{G}_1^n \times \mathbb{Z}_p^m : \boldsymbol{x} = M \cdot \boldsymbol{w} \right\}$$

We provide the construction of the Kiltz-Wee's QA-NIZK arguments for linear subspaces in the CRS model, described in Figure 10.

```
KeyGen($[M]_1 \in \mathbb{G}_1^{n \times m}$) → crs, td
    $\boldsymbol{K} \xleftarrow{\$} \mathbb{Z}_p^{n \times \hat{k}}, a \xleftarrow{\$} \mathbb{Z}_p, \boldsymbol{C} \leftarrow \boldsymbol{K} \cdot a$
    $\boldsymbol{P} \leftarrow [M]_1^\top \cdot \boldsymbol{K} \in \mathbb{G}_1^{n \times \hat{k}}$
    ek := $\boldsymbol{P}$, vk := $([C]_2, [a]_2)$, td := $\boldsymbol{K}$
    return crs := (ek, vk), td

Prove(ek, $\boldsymbol{x}$, $\boldsymbol{w}$) → $[\pi]_1$
    $\pi_1 \leftarrow \boldsymbol{w}^\top \cdot P \in \mathbb{G}_1$
    return $[\pi]_1 \in \mathbb{G}_1^{\hat{k}}$

Verify(vk, $\boldsymbol{x}$, $[\pi]_1$) → (true/false)
    Check that $[\boldsymbol{x}]_1^\top \odot [C]_2 \stackrel{?}{=} [\pi]_1 \odot [a]_2$

Sim($[M]_1$, td, $\boldsymbol{x}$) → $[\pi]_1$
    $\pi_1 \leftarrow \boldsymbol{K}^\top \cdot [\boldsymbol{x}]_1 \in \mathbb{G}_1^{\hat{k}}$
```

Figure 10: KW15 [16] QA-NIZK $\boldsymbol{\Pi}_{\text{QA-NIZK}}$

Similar to the approach in LegoSNARK, we set $\hat{k} = 1$. In LegoSNARK, it is demonstrated that when $\hat{k} = 1$, knowledge soundness is achieved under the discrete logarithm assumption within the algebraic group model (AGM). A comparable proof for the application of this scheme in a non-falsifiable setting is also provided in KW15 [16]. We recall the proof from LegoSNARK and describe it simply as follows:

THEOREM A.1. *Assuming that $\mathcal{D}$ is a witness-sampleable matrix distribution, under the discrete logarithm assumption in AGM, the QA-NIZK $\boldsymbol{\Pi}_{\text{QA-NIZK}}$ from KW15 [16] (with $\hat{k} = 1$) is a knowledge-sound SNARK for the relation $\mathcal{R}_{LS}$ with matrices from $\mathcal{D}$.*

PROOF. Let $\mathcal{A}$ be an algebraic adversary against the knowledge soundness of $\boldsymbol{\Pi}_{\text{QA-NIZK}}$. The adversary takes the matrix M, the CRS (i.e., $\boldsymbol{P}$, C), and the auxiliary input (aux) as inputs (i.e. $\mathcal{A}([M, \boldsymbol{P}]_1, [a, C])$. Consider $[\boldsymbol{z}]_1$, a vector comprising M, elements from *aux* in the group $\mathbb{G}_1$, and the generator of $\mathbb{G}_1$. Then the adversary $\mathcal{A}$ outputs a pair $([x]_1, [\pi]_1)$ and coefficients $\boldsymbol{w}$ that express these elements as linear combinations of its input in $\mathbb{G}_1$. We denote the coefficients for $\boldsymbol{x}$ and π by $(\boldsymbol{X}_0, \boldsymbol{X}_1)$ and $(\boldsymbol{\pi}_0, \boldsymbol{\pi}_1)$ respectively,

$$[\boldsymbol{x}]_1 = \boldsymbol{X}_0[M^\top \boldsymbol{K}]_1 + \boldsymbol{X}_1[\boldsymbol{z}]_1$$
$$[\pi]_1 = \boldsymbol{\pi}_0^\top[M^\top \boldsymbol{K}]_1 + \boldsymbol{\pi}_1^\top[\boldsymbol{z}]_1$$

Now, we define the extractor $\mathcal{E}$ that extracts the witness $\boldsymbol{\pi}_0$. Then we prove that the following probability is negligible:

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, [\boldsymbol{x}]_1, [\pi]_1) = \mathsf{true} \wedge [\boldsymbol{x}]_1 \neq [M]\cdot \boldsymbol{w}\right]$$

If $\mathcal{A}$ returns such a tuple with non-negligible probability, we construct an algorithm $\mathcal{B}$ that, on input $([\boldsymbol{K}]_1, [\boldsymbol{K}]_2)$, outputs the elements $(\boldsymbol{a}, \boldsymbol{b}, c)$ such that:

$$\boldsymbol{K}^\top \cdot \boldsymbol{a} \cdot \boldsymbol{K} + \boldsymbol{K}^\top \cdot \boldsymbol{b} + c = 0$$

The algorithm $\mathcal{B}$ proceeds as follows,

(1) it uses $\mathcal{D}$ to sample $([M]_1, aux)$ along with its witness over $\mathbb{G}_1$, which is a vector $\boldsymbol{z}$ where each element of $\boldsymbol{z}$ is an entry from $\mathbb{Z}_p$.
(2) it samples $a \xleftarrow{\$} \mathbb{Z}_p$ and runs $\mathcal{A}([\boldsymbol{z}, \boldsymbol{P}]_1, [a, a\cdot \boldsymbol{K}]_2)$.
(3) Upon receiving the output from $\mathcal{A}$, $\mathcal{B}$ sets:

$$\boldsymbol{a} := \boldsymbol{X}_0 \cdot M^\top,\ \boldsymbol{b} = \boldsymbol{X}_1\boldsymbol{z} - M\cdot \boldsymbol{\pi}_0,\ \boldsymbol{c} = -\boldsymbol{\pi}_1^\top \cdot \boldsymbol{z}$$

At least one of $\boldsymbol{a}$, $\boldsymbol{b}$, or $\boldsymbol{c}$ must be nonzero. If all are zero, then $\boldsymbol{X}_1\boldsymbol{z} - M\boldsymbol{\pi}_0 = 0$, which implies $\boldsymbol{x} = M\cdot\boldsymbol{\pi}_0$ since $\boldsymbol{X}_0 \cdot M^\top = 0$, contradicting our assumption about $\mathcal{A}$'s output.

Using algorithm $\mathcal{B}$, we construct an algorithm $\mathcal{B}'$ that deals with the discrete logarithm problem. On input $([y]_1, [y]_2)$, the algorithm $\mathcal{B}'$ chooses $\boldsymbol{r}, \boldsymbol{s} \in \mathbb{Z}_p^n$ and sets $\boldsymbol{K} := y\cdot \boldsymbol{r} + \boldsymbol{s}$. It can be shown that $([\boldsymbol{K}]_1, [\boldsymbol{K}]_2)$ can be simulated with a distribution identical to the one expected by $\mathcal{B}$. Given a solution $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$, one can find (a_0, b_0, c_0) such that:

$$\begin{aligned} 0 &= (y\boldsymbol{r}+\boldsymbol{s})^\top \cdot \boldsymbol{a}\cdot(y\cdot\boldsymbol{r}+\boldsymbol{s}) + (y\cdot\boldsymbol{r}+\boldsymbol{s})^\top\cdot\boldsymbol{b}+\boldsymbol{c} \\ &= a_0\cdot y^2 + b_0\cdot y + c_0 \end{aligned}$$

With high probability, $c_0 \neq 0$. From this $\mathcal{B}'$ can extract y.

B COMPRESSED-Σ PROTOCOL

Compressed-Σ protocols are interactive protocols that maintain the same functionality and remain honest-verifier zero-knowledge proofs of knowledge for a given relation $\mathcal{R}$. These protocols achieve succinct communication complexity, reducing from linear to logarithmic size.

In this section, we introduce a protocol for proving the equality of committed vectors. By the protocols proposed in [3] and [4], the proposed protocol can serve the same role as CP_{link}, providing a proof for N Pedersen commitments with a size of $O(\log N)$. Referencing the relation described in Eclipse [2], the relation $\mathcal{R}_{\mathsf{Eq}}^{\mathsf{Batch}}$ that we aim to prove can be described as follows:

$$\mathcal{R}_{\mathsf{Eq}}^{\mathsf{Batch}}(\boldsymbol{x};\boldsymbol{w}) = \left\{ \begin{array}{c} (\boldsymbol{g}, \boldsymbol{h}, \tilde{\boldsymbol{g}}, \tilde{\boldsymbol{h}}, n, d, d_1, d_2), \\ (C, \{D_i\}_{i\in[n]}); \\ (\boldsymbol{m}, \boldsymbol{o}, \{\boldsymbol{o}_i\}_{i\in n}) \end{array} : \begin{array}{c} C = \boldsymbol{g}^{\boldsymbol{m}}\cdot \boldsymbol{h}^{\boldsymbol{o}}, D_i = \tilde{\boldsymbol{g}}^{\boldsymbol{m}_i}\cdot \tilde{\boldsymbol{h}}^{\boldsymbol{o}_i}, \\ \boldsymbol{g} \in \mathbb{Z}_q^{nd}, \tilde{\boldsymbol{g}} \in \mathbb{Z}_q^{d}, \\ \boldsymbol{h} \in \mathbb{Z}_q^{d_1}, \tilde{\boldsymbol{h}} \in \mathbb{Z}_q^{d_2}, \\ \boldsymbol{m} = \{\boldsymbol{m}_i\}_{i\in[n]}, \\ \boldsymbol{o} \in \mathbb{Z}_q^{d_1}, \boldsymbol{o}_i \in \mathbb{Z}_q^{d_2} \end{array} \right\}$$

The compressed version of Σ-protocol for the above relation $\mathcal{R}_{\mathsf{Eq}}^{\mathsf{Batch}}$ is described as follows.

(1) The verifier $\mathcal{V}$ samples a random challenge $\delta \in \mathbb{Z}_q$, and sends it to the prover $\mathcal{P}$. Then both parties scale out $\tilde{\boldsymbol{g}}$ as follows:

$$\tilde{\boldsymbol{g}} := \left\{\tilde{\boldsymbol{g}}^{\delta^i}\right\}_{i=0}^{n-1} \in \mathbb{G}^{nd}$$

(2) The prover $\mathcal{P}$ chooses random $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma} \in \mathbb{Z}_q^{nd\times d_1\times d_2}$, and sends the following elements to the verifier $\mathcal{V}$

$$X = \boldsymbol{g}^{\boldsymbol{\alpha}}\cdot \boldsymbol{h}^{\boldsymbol{\beta}}, \qquad \tilde{X} = \tilde{\boldsymbol{g}}^{\boldsymbol{\alpha}}\cdot \tilde{\boldsymbol{h}}^{\boldsymbol{\gamma}}$$

(3) The verifier samples a challenge $e \in \mathbb{Z}_q$ and sends it to the prover $\mathcal{P}$.
(4) The prover $\mathcal{P}$ computes

$$\boldsymbol{z} = \boldsymbol{\alpha} + e\cdot \boldsymbol{m}, \qquad \boldsymbol{k} = \boldsymbol{\beta} + e\cdot \boldsymbol{o}, \qquad \boldsymbol{\omega} = \boldsymbol{\gamma} + e\cdot \sum_{i=1}^{n} \boldsymbol{o}_i \cdot \delta^{i-1}$$

(5) Let

$$\boldsymbol{g} = \boldsymbol{g}_L \| \boldsymbol{g}_R, \qquad \tilde{\boldsymbol{g}} = \tilde{\boldsymbol{g}}_L\|\tilde{\boldsymbol{g}}_R, \qquad \boldsymbol{z} = \boldsymbol{z}_L\|\boldsymbol{z}_R$$

and

$$Y = X\cdot C^e\cdot \boldsymbol{h}^{-\boldsymbol{k}}, \qquad \tilde{Y} = \tilde{X}\cdot(\prod_{i=1}^{n} D_i^{\delta^{i-1}})^e\cdot \tilde{\boldsymbol{h}}^{-\boldsymbol{\omega}}$$

(6) The prover $\mathcal{P}$ sends

$$L = \boldsymbol{g}_R^{\boldsymbol{z}_L}, \qquad R = \boldsymbol{g}_L^{\boldsymbol{z}_R}$$
$$\tilde{L} = \tilde{\boldsymbol{g}}_R^{\boldsymbol{z}_L}, \qquad \tilde{R} = \tilde{\boldsymbol{g}}_L^{\boldsymbol{z}_R}$$

(7) The verifier $\mathcal{V}$ sends a challenge $c \in \mathbb{Z}_q$
(8) The prover $\mathcal{P}$ computes

$$\boldsymbol{z}' = \boldsymbol{z}_L + c\cdot \boldsymbol{z}_R$$

and both parties compute

$$Y' = L\cdot Y^c\cdot R^{c^2}, \qquad \tilde{Y}' = \tilde{L}\cdot\tilde{Y}^c\cdot\tilde{R}^{c^2}$$
$$\boldsymbol{g}' = \boldsymbol{g}_L^c \odot \boldsymbol{g}_R, \qquad \tilde{\boldsymbol{g}}' = \tilde{\boldsymbol{g}}_L^c\odot\tilde{\boldsymbol{g}}_R$$

where $\odot$ is an element-wise product.
(9) If $n > 2$, then both parties execute the above step (5)-(8) with

$$((\boldsymbol{g}, \tilde{\boldsymbol{g}}', n/2), (Y', \tilde{Y}'), \boldsymbol{z}')$$

Otherwise, the verifier $\mathcal{V}$ checks

$$\boldsymbol{g}'^{\boldsymbol{z}'} \stackrel{?}{=} Y', \qquad \tilde{\boldsymbol{g}}^{\boldsymbol{z}'} \stackrel{?}{=} \tilde{Y}'$$

Figure 11: Compressed Σ version for the relation $\mathcal{R}_{\mathsf{Eq}}^{\mathsf{Batch}}$

THEOREM B.1. *The protocol described in Fig.11 is a $(2\kappa + 4)$ protocol for the relation $\mathcal{R}_{\mathsf{Eq}}^{\mathsf{Batch}}$ where $\kappa = \lceil \log nd \rceil - 1$. It satisfies completeness, computationally $(n, 2, \{t_i\}_{i\in[\kappa]})$-special sound if finding discrete-logarithm, and special honest verifier zero-knowledge where $t_i = 3$ for all $i \in [\kappa]$.*

PROOF. Since completeness is straightforward, we omit the description.

$(n, 2, \{t_i\}_{i\in[\kappa]})$**-special soundness**. To simplify, we assume a single recursive step execution. Specifically, we analyze the 4-move protocol, where the prover sends the response $\boldsymbol{z}'$ irrespective of it dimension, and proves that this protocol is 4-special sound. Then t_i-special soundness can then be derived through an inductive argument, the details of which are omitted here (i.e. omit j).

First of all, we denote the transcript as Tr, which consists of $(L, R, \tilde{L}, \tilde{R}, Y, \tilde{Y}', c_i, \boldsymbol{z}_i)$. Given three accepting transcripts $(\mathsf{Tr}_0, \mathsf{Tr}_1, \mathsf{Tr}_2)$ for the same challenge δ but the distinct challenge $c_i \in \{0, 1, 2\}$, we can show that there exists an efficient algorithm χ that outputs a valid witness. Given these transcripts, Since $\prod_{0\le i<k\le 2}(c_k - c_i) \neq 0$, we define (v_0, v_1, v_2) such that

$$\sum_{i=0}^{2} v_i = 0, \quad \sum_{i=0}^{2} v_i \cdot c_i = 1, \quad \sum_{i=0}^{2} v_i \cdot c_i^2 = 0$$

Define $\bar{z}_i = (v_i c_i \boldsymbol{z}_i \| v_i \boldsymbol{z}_i)$. Then let $\boldsymbol{w} = \sum_{i=0}^{2} \bar{z}_i$ be the extracted value. We show the correctness of the extracted value as follows:

$$\begin{aligned}
\boldsymbol{g}^{\boldsymbol{w}} &= \boldsymbol{g}^{(\sum_{i=0}^{2} v_i c_i \boldsymbol{z}_i) \| (\sum_{i=0}^{2} v_i \boldsymbol{z}_i)} \\
&= \boldsymbol{g}_L^{v_0 c_0 \boldsymbol{z}_0} \cdot \boldsymbol{g}_L^{v_1 c_1 \boldsymbol{z}_1} \cdot \boldsymbol{g}_L^{v_2 c_2 \boldsymbol{z}_2} \cdot \boldsymbol{g}_R^{v_0 \boldsymbol{z}_0} \cdot \boldsymbol{g}_R^{v_1 \boldsymbol{z}_1} \cdot \boldsymbol{g}_R^{v_2 \boldsymbol{z}_2} \\
&= \prod_{i=0}^{2} \left((\boldsymbol{g}_L^{c_i} \odot \boldsymbol{g}_R)^{\bar{\boldsymbol{z}}_{i,L} + c_i \bar{\boldsymbol{z}}_{i,R}} \right)^{v_i} \\
&= \prod_{i=0}^{2} (\boldsymbol{g}_L^{c_i \bar{\boldsymbol{z}}_{i,L}} \cdot \boldsymbol{g}_L^{c_i^2 \bar{\boldsymbol{z}}_{i,R}} \cdot \boldsymbol{g}_R^{\bar{\boldsymbol{z}}_{i,L}} \cdot \boldsymbol{g}_R^{c_i \bar{\boldsymbol{z}}_{i,R}})^{v_i} \\
&= \prod_{i=0}^{2} \left((\boldsymbol{g}^{\bar{\boldsymbol{z}}_i})^{c_i} \cdot \boldsymbol{g}_R^{\bar{\boldsymbol{z}}_{i,L}} \cdot (\boldsymbol{g}_L^{\bar{\boldsymbol{z}}_{i,R}})^{c_i^2} \right)^{v_i} \\
&= \prod_{i=0}^{2} (Y^{c_i} \cdot L \cdot R^{c_i^2})^{v_i} \\
&= Y
\end{aligned}$$

where $\odot$ denotes the element-wise product. In a similar vein, extraction can also be performed for $\tilde{\boldsymbol{g}}$.

Special honest verifier zero-knowledge. With the challenge x and e provided, the simulator randomly samples $\boldsymbol{z}$, $\boldsymbol{k}$, and $\boldsymbol{\omega}$, subsequently using these to perfectly simulate the remaining messages as follows:

$$X := \boldsymbol{g}^{\boldsymbol{z}} \cdot \boldsymbol{h}^{\boldsymbol{k}} \cdot C^{-e}, \quad \tilde{X} := \tilde{\boldsymbol{g}}^{\boldsymbol{z}} \cdot \tilde{\boldsymbol{h}}^{\boldsymbol{k}} \cdot (\prod_{i=1}^{l} D_i^{x^{i-1}})^{-e}$$