zkMatrix: Batched Short Proof for Committed Matrix Multiplication

Mingshu Cong[1][0000−0001−5858−3411], Tsz Hon Yuen *[2][0000−0002−0629−6792], and Siu Ming Yiu **[1][0000−0002−3975−8500]

[1] Department of Computer Science, The University of Hong Kong, Pokfulam Road,
Hong Kong SAR, China
{mscong,smyiu}@cs.hku.hk

[2] Faculty of Information Technology, Monash University, Australia
John.tszhonyuen@monash.edu

Abstract. Matrix multiplication is a common operation in applications like machine learning and data analytics. To demonstrate the correctness of such an operation in a privacy-preserving manner, we propose *zkMatrix*, a zero-knowledge proof for the multiplication of committed matrices. Among the succinct non-interactive zero-knowledge protocols that have an $O(\log n)$ transcript size and $O(\log n)$ verifier time, zkMatrix stands out as the *first* to achieve $O(n^2)$ prover time and $O(n^2)$ RAM usage for multiplying two $n \times n$ matrices. Significantly, zkMatrix distinguishes itself as the *first* zk-SNARK protocol specifically designed for matrix multiplication. By batching multiple proofs together, each additional matrix multiplication only necessitates $O(n)$ group operations in prover time.

Keywords: zero-knowledge proof, matrix multiplication, zk-SNARK

1 Introduction

Matrix multiplication is a fundamental operation with wide applications in applied mathematics, statistics, finance, engineering, and more. It plays an essential role in handling high-dimensional datasets in data analytics and machine learning. In these applications, sometimes we need to protect the privacy of sensitive data. While zero-knowledge proofs offer a solution, providing an efficient zero-knowledge proof for large-scale matrix multiplications, which may contain thousands of rows, remains challenging.

Current protocols often struggle to simultaneously meet the following three key requirements:

- *Succinctness*: When dealing with big datasets, it is a standard requirement for a polylogarithmic transcript size and polylogarithmic verifier time.

* Both authors contributed equally to this research.

** Siu Ming Yiu is the corresponding author.

- *Linear prover time with respect to the witness size*: Many zero-knowledge proofs based on arithmetic circuits require prover time linear to the number of multiplication gates, which is roughly proportional to the 1.5-power of the number of elements in the matrices. [3]
- *Consistency with commitments*: In many applications, matrix multiplications are performed sequentially (e.g., neural networks in Subsection 1.4). Given the non-uniqueness of matrix multiplication outcomes (i.e., for any given matrix $\boldsymbol{c}$, there are infinite distinct matrices $\boldsymbol{a}$ and $\boldsymbol{b}$ such that $\boldsymbol{c} = \boldsymbol{ab}$), it is vital to verify that the private inputs are the same across multiple matrix multiplications. Therefore, the zero-knowledge proof should guarantee that the private matrices are consistent with the same commitments. Ideally, the commitment scheme should be independent of specific matrix operations.

Providing proof for the multiplication of two $n \times n$ matrices necessitates roughly n^3 multiplication gates. Direct application of zero-knowledge Succinct Non-interactive ARgument of Knowledge (zk-SNARK) protocols, like those in [Groth(2016), Parno et al.(2016), Bowe et al.(2019), Chiesa et al.(2020b), Maller et al.(2019), Bünz et al.(2020), Gabizon et al.(2019), Chen et al.(2023), Chiesa et al.(2020a)], results in $O(n^3)$ prover time and potential memory overflow due to the $O(n^3)$ circuit size. Thaler's specialized protocol [Thaler(2013)] transforms matrix multiplication into a sumcheck relation, resulting in greater efficiency in prover time. However, it is not a zk-SNARK protocol and necessitates linear time for verification. Despite subsequent improvements by LegoSNARK [Campanelli et al.(2019)] and Libra [Xie et al.(2019)], achieving $O(n^2)$ prover time with polylogarithmic verifier time remains unaccomplished. QuickSilver uses an interactive protocol and secure multiparty computation, trimming down the prover time to $O(n^2)$ but introducing a higher communication overhead of $O(n^2)$. A detailed comparison is provided in Table 1. Among protocols with logarithmic verifier time, such as Pinocchio [Parno et al.(2016)] and Libra [Xie et al.(2019)], our zkMatrix protocol is the first to achieve $O(n^2)$ prover time.

1.1 Main Result

This paper seeks to construct a zero-knowledge proof for witness matrices $\boldsymbol{a}, \boldsymbol{b}$ and $\boldsymbol{c}$ such that $\boldsymbol{c} = \boldsymbol{ab}$. The matrices $\boldsymbol{a}, \boldsymbol{b}$, and $\boldsymbol{c}$ are committed using Pedersen vector commitments C_a, C_b, and C_c respectively. Throughout the paper, we will use bold letters for matrices (e.g., $\boldsymbol{a}$) and smaller letters with index i, j for elements inside the matrices (e.g., a_{ij}). We use the symbol $\oplus$ for point addition in the elliptic curve group $\mathbb{G}$ of prime order p.

[3] For $n \times n$ square matrices, the number of multiplication gates is n^3 using the schoolbook algorithm. The best algorithm in the literature is $> n^{2.37}$ multiplications.

Table 1: Comparison of the Zero-Knowledge Proofs for Matrix Multiplication of $n \times n$ Matrices

Protocol	Communication	RAM Usage		Timing		Consistency
		Prover	Verifier	Prover	Verifier	
Pinocchio [Parno et al.(2016)]	$O(1)$	$O(n^3)$	$O(1)$	$O(n^3)$	$O(1)$	No
Thaler [Thaler(2013)]	$O(\log n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	No
LegoSNARK [Campanelli et al.(2019)]	$O(\log n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	Yes
Libra [Xie et al.(2019)]	$O(\log^2 n)$	$O(n^3)$	$O(\log n)$	$O(n^3)$	$O(\log^2 n)$	No
QuickSilver [Yang et al.(2021)]	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	Yes
zkMatrix (single proof)	$O(\log n)$	$O(n^2)$	$O(\log n)$	$O(n^2)$	$O(\log n)$	Yes
zkMatrix (batched proof for t matrix multiplications)	$O(t \log n)$	$O(n^2 + tn)$	$O(t \log n)$	$O(n^2 + tn)\mathbb{E}$ $+O(tn^2)\mathbb{M}$	$O(t \log n)$	Yes

Note: The timing columns only show the dominating factor(s). For most schemes, it is the number of exponentiations $\mathbb{E}$ in ECC or pairing groups. $\mathbb{M}$ represents the number of multiplications in $\mathbb{Z}_p$, and one exponentiation roughly takes $\log_2 p$ multiplications. All group and modular additions, hashing operations, and non-dominating pairing operations are omitted in the table for simplicity.

We consider the following relation for public group matrices $\boldsymbol{U}, \boldsymbol{G}, \boldsymbol{H}$:

$$\begin{aligned}&\mathcal{R}_{\mathsf{comMatMul}}\\ &= \left\{ \begin{pmatrix} C_c, C_a, C_b \in \mathbb{G}; \\ \boldsymbol{U} \in \mathbb{G}^{m \times n}, \\ \boldsymbol{G} \in \mathbb{G}^{m \times l}, \\ \boldsymbol{H} \in \mathbb{G}^{l \times n} \end{pmatrix} : \begin{pmatrix} \boldsymbol{c} \in \mathbb{Z}_p^{m \times n}, \\ \boldsymbol{a} \in \mathbb{Z}_p^{m \times l}, \\ \boldsymbol{b} \in \mathbb{Z}_p^{l \times n} \end{pmatrix} \middle| \begin{pmatrix} \boldsymbol{c} = \boldsymbol{a}\boldsymbol{b}, \\ \wedge\, C_c = \langle \boldsymbol{c}, \boldsymbol{U} \rangle \\ \wedge\, C_a = \langle \boldsymbol{a}, \boldsymbol{G} \rangle \\ \wedge\, C_b = \langle \boldsymbol{b}, \boldsymbol{H} \rangle \end{pmatrix} \right\},\end{aligned} \tag{1}$$

where $\langle \boldsymbol{a}, \boldsymbol{G} \rangle$ is the Pedersen vector commitment of the matrix $\boldsymbol{a}$:

$$\begin{array}{rlllllll} \langle \boldsymbol{a}, \boldsymbol{G} \rangle := & a_{11}G_{11} & \oplus & a_{12}G_{12} & \oplus \cdots \oplus & a_{1l}G_{1l} \\ \oplus & a_{21}G_{21} & \oplus & a_{22}G_{22} & \oplus \cdots \oplus & a_{2l}G_{2l} \\ \oplus & \vdots & & \vdots & \quad \vdots & \vdots \\ \oplus & a_{m1}G_{m2} & \oplus & a_{m2}G_{m2} & \oplus \cdots \oplus & a_{ml}G_{ml}. \end{array} \tag{2}$$

Analogous definitions apply to $\langle \boldsymbol{b}, \boldsymbol{H} \rangle$ and $\langle \boldsymbol{c}, \boldsymbol{U} \rangle$.[4]

[4] Mathematically, $\langle \boldsymbol{a}, \boldsymbol{G} \rangle$ is defined as $\vec{\boldsymbol{1}}^\top (\boldsymbol{a} \circ \boldsymbol{G}) \vec{\boldsymbol{1}}$, where $\vec{\boldsymbol{1}}$ is a unit vector with a suitable length, $\top$ stands for transpose and $\circ$ is the Hadamard product.

Our paper introduces the *zkMatrix* protocol for the relation $\mathcal{R}_{\mathsf{comMatMul}}$ characterized by a logarithmic transcript size, $O(n^2)$ prover time, and logarithmic verifier time. Specifically, the prover time is $O(n^2)$ ($\sim 14n^2$ exponentiations), and the verifier runs $O(\log n)$ exponentiations and $O(1)$ pairings. Given that one optimal Ate pairing operation takes around 12 exponentiations and 4 field multiplications, the verifier time is on the order of $O(\log n)$.

We can optimize the prover time though batch processing. When providing proof for t instances of $n \times n$ matrix multiplications (with the witness matrices collectively comprising $3tn^2$ elements), the prover's time complexity is $O(n^2 + nt)$ in terms of group operations , while the verifier's time complexity is $O(t \log n)$. For each additional matrix multiplication, the marginal prover time takes roughly $\sim 8n$ exponentiations and $\sim 5n^2$ field multiplications.

1.2 High-Level Idea

Reducing matrix multiplication to four inner-product relations. The foundational idea is derived from Groth's seminal work on providing sublinear zero-knowledge proofs for linear algebra relations [Groth(2009)]. Linear algebra relations can be reduced to high-dimensional inner-product relations (as seen in Section 1.3 of [Groth(2009)]).

Using the following equivalence, we can prove matrix multiplication by proving four inner-product relations, denoted by the superscripts $\textcircled{1}$, $\textcircled{2}$, $\textcircled{3}$, $\textcircled{4}$ in the equation below:

$$\{\boldsymbol{c} = \boldsymbol{ab}\} \Leftrightarrow \{\forall y \in \mathbb{Z}_p, (\vec{\boldsymbol{y}}_L^\top \boldsymbol{c} \vec{\boldsymbol{y}}_R)^{\textcircled{1}} = ((\vec{\boldsymbol{y}}_L^\top \boldsymbol{a})^{\textcircled{2}} (\boldsymbol{b} \vec{\boldsymbol{y}}_R)^{\textcircled{3}})^{\textcircled{4}}\}. \tag{3}$$

The inner-product relation $\textcircled{1}$ is the *standard polynomial* of the matrix $\boldsymbol{c}$. Specifically, for an $(m \times n)$-matrix $\boldsymbol{c}$, we can multiply it on the left by an m-row-vector $\vec{\boldsymbol{y}}_L^\top = (1, y^n, \cdots, y^{(m-1)n})$ to obtain $\vec{\boldsymbol{y}}_L^\top \boldsymbol{c}$, and then on the right by a n-column-vector $\vec{\boldsymbol{y}}_R = (1, y, \cdots, y^{n-1})^\top$ to obtain $\vec{\boldsymbol{y}}_L^\top \boldsymbol{c} \vec{\boldsymbol{y}}_R$. We can directly compute a degree $(mn-1)$ polynomial of y:

$$\vec{\boldsymbol{y}}_L^\top \boldsymbol{c} \vec{\boldsymbol{y}}_R = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} y^{(i-1)n+(j-1)}.$$

It can be viewed as an inner product of a vector formed by c_{ij} (i.e., a vector of flattened $\boldsymbol{c}$), and a vector $(1, y, y^2, \ldots, y^{mn-1})$. As a result, we can use Bulletproofs [Bünz et al.(2018)] to prove the inner-product relation.

Working with committed vectors. While using Bulletproofs [Bünz et al.(2018)] for proving inner-product relations might seem intuitive, ensuring the soundness of the entire matrix multiplication protocol requires consistent witness vectors across these four inner-product relations. Specifically, the left values of the second and third inner-products, $\vec{\boldsymbol{a}}_y \leftarrow \vec{\boldsymbol{y}}_L^\top \boldsymbol{a}$ and $\vec{\boldsymbol{b}}_y \leftarrow \boldsymbol{b} \vec{\boldsymbol{y}}_R$, must align with the witness of the fourth inner-product relation. One straightforward method to achieve this consistency is by incorporating these intermediate variables into the transcript, but this will result in a large transcript size. We address this by compressing each high-dimensional intermediate variable into a single group element through Pedersen vector commitments.

Bulletproofs cannot ensure that the two witness vectors are consistent with previous commitments. Despite this, we observe that this consistency can be achieved when one vector is constructed from a public-coin challenge. Hence, we present two variants of Bulletproof:

- *Committed Semi-Inner-Product Argument*: It proves the inner-product of a committed vector and a public vector for the inner product ①.
- *Committed High-Dimensional Semi-Inner-Product Argument*: It proves a linear combination of committed vectors for inner products ② and ③.

Combining with the Bulletproof for the inner product ④, we obtain an argument of knowledge for matrix multiplication.

Single zkMatrix: Shifting the verifier's multi-exponentiation computation load to the prover. The primary bottleneck in verification efficiency of Bulletproof is attributed to the elliptic curve multi-exponentiation involved. In short, it involves the verification of a group element V such that:

$$V = [\zeta_1][G_1]\oplus[\zeta_2][G_2]\oplus\cdots\oplus[\zeta_q][G_q],$$

where q is the total size of the prover's secret vectors, $[G_1],\ldots,[G_q]$ are public group elements and $[\zeta_1],\ldots,[\zeta_q] \in \mathbb{Z}_p$ can be computed by the verifier. The verifier time of Bulletproofs is dominated by the multi-exponentiation of size q.

In this paper, we want to transfer the burden of this multi-exponentiation calculation from the verifier to the prover. Bulletproofs require that the public group elements $[G_1],\ldots,[G_q]$ are randomly sampled such that the mutual discrete logarithm between them are unknown. We propose the use of structured bases $[\hat{s}\hat{G}],[\hat{s}^2\hat{G}],\ldots,[\hat{s}^q\hat{G}]$ for some $\hat{s}\in\mathbb{Z}_p$ and $\hat{G}\in\mathbb{G}$. Now consider the multi-exponentiation given by:

$$V = [\zeta_1][\hat{s}\hat{G}]\oplus[\zeta_2][\hat{s}^2\hat{G}]\oplus\cdots\oplus[\zeta_q][\hat{s}^q\hat{G}] := \phi(\hat{s})\hat{G},$$

where $\phi(x) = [\zeta_1]x + \cdots + [\zeta_q]x^q \in \mathbb{Z}_p[X]$ represents a polynomial with known coefficients. Instead of requiring the verifier to compute the multi-exponentiation of size q, the prover can provide a proof that V is valid. In response to a random challenge $[s] \xleftarrow{\$} \mathbb{Z}_p^*$, the prover computes $[W] \leftarrow (\frac{\phi(s)-\phi(\hat{s})}{s-\hat{s}})\hat{G}$, which requires a multi-exponentiation of size q. Subsequently, the verifier evaluates $\phi([s])$ in $\mathbb{Z}_p$ and checks whether $[V]$ aligns with $\phi([s])$ and $[W]$ by using the following pairing equation:

$$e([W],[s][\hat{G}]\ominus[\hat{s}\hat{G}]) = e(\phi([s])[\hat{G}]\ominus V,[\hat{G}]),$$

where $\ominus$ is the inverse of $\oplus$. The pairing equation in our protocol is actually more complicated, and details can be found in Section 3.

Batched zkMatrix: Batch processing before inner-products. For use cases requiring proofs for thousands of matrix multiplications, our proposed solution entails executing t zkMatrix protocols in parallel, utilizing identical public-coin challenges. In essence:

- The prover is restricted to batch computing transcript elements for the inner-product relations ①, ②, and ③.
- The introduction of the batching randomness should come after the prover's commitment to the intermediate variables that serve as public inputs of the sub-protocols. Specifically, this should occur post commitments to $\vec{\boldsymbol{a}}_y$, $\vec{\boldsymbol{b}}_y$, and $\vec{\boldsymbol{y}}_L^\top \boldsymbol{c} \vec{\boldsymbol{y}}_R$.

Zero-knowledge proof via random blinding matrices. To transform an argument of knowledge for the matrix multiplication of $\boldsymbol{c} = \boldsymbol{ab}$ into a zero-knowledge one, we introduce two random blinding matrices $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{Z}_p^{m \times l}, \boldsymbol{\beta} \xleftarrow{\$} \mathbb{Z}_p^{l \times n}$. We then provide a proof for the multiplication of $x(\boldsymbol{\alpha} + x\boldsymbol{a})$ and $x(\boldsymbol{\beta} + x\boldsymbol{b})$, where $x \xleftarrow{\$} \mathbb{Z}_p^*$ is a public-coin challenge. This approach increases the transcript size by a constant factor. In the context of batched proofs, these blinding matrices are introduced within the sub-protocols on batched inputs.

1.3 Our Contribution

Among succinct non-interactive zero-knowledge protocols, characterized by an $O(\log n)$ transcript size and $O(\log n)$ verifier time for the multiplication of two $n \times n$ matrices, our zkMatrix is the *first* to accomplish $O(n^2)$ prover time.

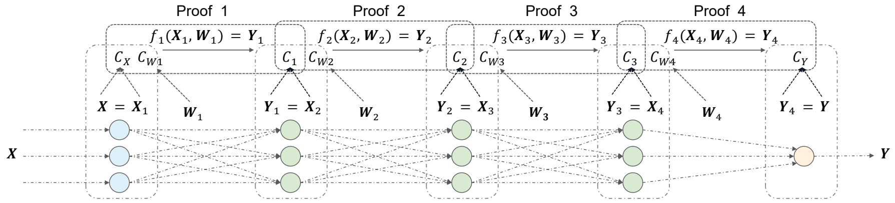

Fig. 1: CNNs with zero-knowledge proofs. X_i and Y_i represent the input and output of the CNN's layer i, with a weight W_i such that $f_i(X_i, W_i) = Y_i$ for some matrix function f_i. C_A stands for the commitment to the secret vector/matrix A, bridging sub-protocols throughout the CNN computation.

First zk-SNARK for matrix multiplication with linear prover time. The zkMatrix protocol achieves a strictly linear prover time while maintaining a succinct transcript. No existing zk-SNARK protocol for matrix multiplication has achieved such a level of asymptotic efficiency.

Bulletproof as zk-SNARK with universal structured reference string. Our methodology reduces the verifier time of Bulletproofs to a logarithmic scale. Bulletproof is not categorized as a zk-SNARK protocol due to its linear verifier time. Our methods can reduce the verifier time of the Bulletproof-version R1CS protocol (see Protocol 3 in [Bünz et al.(2018)]) to logarithmic exponentiations. Though our technique requires a trusted setup phase to generate the structured reference string (SRS), this setup is a one-shot procedure. Consequently, the SRS is universal and upgradable — features that align with contemporary zk-SNARK solutions like Sonic [Maller et al.(2019)], Marlin [Chiesa et al.(2020a)], and PLONK [Gabizon et al.(2019)].

1.4 Potential Applications

Privacy-preserving data mining Privacy-preserving data mining aims to construct data models without direct access to the underlying data records [Agrawal and Srikant(2000)]. A succinct zero-knowledge proof, published alongside the data mining results, can validate the reliability of these results without revealing the underlying data [Duan and Canny([n. d.])]. Given the crucial role of matrix multiplication in data mining techniques such as Linear Regression (LR), Principal Component Analysis (PCA), and graph mining via adjacency matrices, zkMatrix becomes indispensable in enabling privacy-preserving data mining.

Range proof for vectors Bulletproof is a widely-used technique for range proofs. In blockchain applications, confidential transactions that conceal transaction details can be achieved through Pedersen commitments. In this scenario, range proofs are essential to guarantee that the hidden values fall within a specified range. For example, this ensures that the amount sent does not exceed the current balance. Pedersen vector commitments can commit multiple confidential transactions collectively. Just as Bulletproof provides range proofs for single committed values [Bünz et al.(2018)], zkMatrix is capable of providing the necessary range proofs for these committed vectors. Specifically, that all elements of a secret vector $\vec{\boldsymbol{c}} \in \mathbb{Z}_p^m$ fall within the range $(0, 2^n - 1)$ is equivalent to:

$$\exists \boldsymbol{a} \in \mathbb{Z}_p^{m \times n},\ s.t.\ \boldsymbol{a} \circ (\boldsymbol{a} - \mathbf{1}) = \mathbf{0} \wedge \vec{\boldsymbol{c}} = \boldsymbol{a}\vec{\boldsymbol{b}},$$

where the vector $\vec{\boldsymbol{b}} \in \mathbb{Z}_p^n$ is $(1, 2, 2^2, \cdots, 2^{n-1})^T$.

Non-interactive zero-knowledge proofs for machine learning model. Matrix multiplication is also fundamental in machine learning. Zero-knowledge proofs for convolutional neural networks (CNNs) highlight the importance of zk-SNARKs for matrix multiplications [Weng et al.(2021),Liu et al.(2021)]. Figure 1 illustrates a basic CNN architecture, where the convolutional layers are transformed into matrix multiplications [Vasudevan et al.(2017)]. Pedersen vector commitments to the intermediate variables can link the sub-protocols for successive layers. This ensures the output-input consistency across successive layers. Machine learning necessitates floating-point matrix multiplications. The appendix of the online full version of this paper elaborates on the method used to adapt zkMatrix for floating-point matrices.

2 Preliminaries

2.1 Notation

Group and field elements. Let $\mathbb{G}$ represents a cyclic group of prime order p, and let $\mathbb{Z}_p$ denote the ring of integers modulo p. $\mathbb{Z}_p^*$ is defined as $\mathbb{Z}_p^* := \mathbb{Z}_p \backslash \{0\}$. $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}$ denote the sets of natural numbers, integers, and real numbers, respectively. We use the symbol $\overset{\$}{\leftarrow}$ to indicate the uniform sampling of an element from a set. If not otherwise clear from context, group elements are denoted by $G, H, U, V, W, P \in \mathbb{G}$. Field elements are denoted by $a, b, c, d \in \mathbb{Z}_p$, and their Pedersen commitments are denoted by $C_a, C_b, C_c, C_d \in \mathbb{G}$. A random challenge $x \in \mathbb{Z}_p^*$ sent from the verifier to the prover is considered public-coin if it is public information and is selected uniformly at random. We denote this process as $[x] \overset{\$}{\leftarrow} \mathsf{Coin}(\mathbb{Z}_p^*)$, and use x, y, z to denote such public-coin challenges. $\theta, \rho, \vartheta \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ are uniformly distributed randomness for batch processing. We use the superscript $\tilde{\cdot}$ to denote the blinding factors $\tilde{a}, \tilde{b}, \tilde{c} \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ corresponding to a, b, and c, respectively. The base for these blinding factors is denoted by $\tilde{G} \in \mathbb{G}$.

We adopt the additive notation for group operations. We use $\oplus$ to denote group addition, and O is the identity element of $\mathbb{G}$. $\bigoplus_{i=start}^{end}$ represents the summation of multiple group elements indexed by $i \in \mathsf{range}(start, end)$. For comparison, the summation of field elements is denoted by $\Sigma_{i=start}^{end}$. We also employ the term *exponentiation* to indicate the scalar multiplication of group elements.

Pairing A pairing is a bilinear map, denoted by $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$, that maps two cryptographic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ to a third group $\mathbb{G}_T$. This third group $\mathbb{G}_T$ is referred to as the target group. The bilinear property is given by the equation:

$$e(aG, bH) = ab\, e(G, H),\ \forall\, a, b \in \mathbb{Z}_p,\ G \in \mathbb{G}_1, H \in \mathbb{G}_2.$$

When the same group is used for the first two groups (i.e. $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$), the pairing is referred to as symmetric or type-1 pairing. Otherwise, it is asymmetric.

Symmetric pairings are easier to read and analyze, but asymmetric pairings often offer greater computational efficiency. For our purposes, throughout our main text, we will primarily employ symmetric pairings . Our protocols can be readily adapted to use asymmetric pairings.

In the context of asymmetric pairing, the group generators in $\mathbb{G}_1$ and $\mathbb{G}_2$ are denoted by $\hat{G}$ and $\hat{H}$, respectively.

Matrix and matrix operation We employ a slightly unconventional notation for matrices and vectors. Specifically, we use bold letters to represent matrices, $\vec{\cdot}$ to denote column vectors, and $\vec{\cdot}^\top$ to denote row vectors. To distinguish between matrices of field elements (field matrices) and those containing group elements (group matrices), we utilize uppercase and lowercase bold letters. Field matrices are denoted by $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, and group matrices are denoted by $\boldsymbol{G}, \boldsymbol{H}, \boldsymbol{U}$.

Recall that in linear algebra, a matrix can be partitioned into smaller matrices known as blocks. Specifically, we use $\boldsymbol{a} = \{a_{ij}\}$ to represent the mn elements $a_{ij} \in \mathbb{Z}_p$, $i \in \mathsf{range}(1, m)$, $j \in \mathsf{range}(1, n)$ in a matrix. The matrix $\boldsymbol{a}$ can be divided into m row-vectors, denoted by $\vec{\boldsymbol{a}}_{1*}^\top, \cdots, \vec{\boldsymbol{a}}_{m*}^\top$, or n column-vectors, denoted by $\vec{\boldsymbol{a}}_{*1}, \cdots, \vec{\boldsymbol{a}}_{*n}$.

Informally, we use the notation $(\vec{\boldsymbol{a}}_{1*}^\top, \vec{\boldsymbol{a}}_{2*}^\top, \cdots, \vec{\boldsymbol{a}}_{m*}^\top)^\top \xleftarrow{\mathsf{part}} \boldsymbol{a}$ to indicate that we have partitioned $\boldsymbol{a}$ into m row vectors, and
$(\vec{\boldsymbol{a}}_{*1}, \vec{\boldsymbol{a}}_{*2}, \cdots, \vec{\boldsymbol{a}}_{*n}) \xleftarrow{\mathsf{part}} \boldsymbol{a}$ to signify that we have partitioned $\boldsymbol{a}$ into n column vectors. Additionally, we use the expression:

$$\vec{\boldsymbol{a}} := (a_{11}, a_{12}, \cdots, a_{1n}; \cdots; a_{m1}, a_{m2}, \cdots, a_{mn}) \xleftarrow{\mathsf{flat}} \boldsymbol{a},$$

to indicate that we have flattened $\boldsymbol{a}$ into a (mn)-vector. To distinguish between the set of matrices and the set of vectors, we denote the former by $\mathbb{Z}_p^{m\times n}$ and the latter by $\mathbb{Z}_p^{mn}$. Similar notations apply to group matrices and group vectors.

When $\vec{\boldsymbol{a}} \in \mathbb{Z}_p^n$ is a vector, we separate it to the left half $\vec{\boldsymbol{a}}_L \leftarrow (a_1, \cdots, a_{n/2})$ and the right half $\vec{\boldsymbol{a}}_R \leftarrow (a_{n/2+1}, \cdots, a_n)$. We use $(\vec{\boldsymbol{a}}_L || \vec{\boldsymbol{a}}_R) \xleftarrow{\mathsf{half}} \vec{\boldsymbol{a}}$ to denote this procedure. For simplicity, we assume that n is a power of 2. We use the symbol $||$ to denote the concatenation of two vectors into a longer vector.

We can define addition and scalar multiplication on group matrices similarly to those on field matrices. Additionally, we define $\langle \boldsymbol{a}, \boldsymbol{G} \rangle$, the inner product between a field matrix $\boldsymbol{a} \in \mathbb{Z}_p^{m\times n}$ and a group matrix $\boldsymbol{G} \in \mathbb{G}^{m\times n}$, as in Eq. 2. To avoid ambiguity, we use the bullet symbol $\bullet$ instead of $\langle\cdot\rangle$ to denote the inner product operation between field vectors (this happens to be a coincidence with the name Bulletproof).

Bracket representation We introduce the *bracket representation* $[\cdot]$ as a new notation. $[\cdot]$ signifies that the value within the bracket has been known by both the prover and the verifier. To be more specific, we put into brackets the public parameters, the commitments, the public-coin challenges, and the intermediate variables computed from values in brackets. The bracket representation is especially useful from the verifier's viewpoint. As a convention of this paper, we do not use the bracket representation within the prover's algorithm, or in cases where no ambiguity exists.

For instance, in Schnorr's protocol, by employing the bracket representation, we avoid using extra symbols for $[aG]$, $[\alpha G]$, and $[\alpha + xa]$. The verifier checks whether $[\alpha + xa][G] \overset{?}{=} [\alpha G] + [x][aG]$, which holds obviously.

Miscellaneous Throughout this paper, the security parameter is denoted by λ. We use $\mathsf{Setup}(1^\lambda)$ to represent the generation of public parameters, like the order p of $\mathbb{G}$, based on the security parameter. $\mathcal{P}, \mathcal{V}, \mathcal{E}$, $\mathcal{S}$, and $\mathcal{A}$ stand for the prover, the verifier, the emulator, the simulator, and the adversary, respectively. $(\mathcal{P} \rightleftharpoons \mathcal{V})$ represents the interaction between $\mathcal{P}$ and $\mathcal{V}$. We use $\mathcal{R} = \{$ Public Input : Witness | Relation $\}$ to denote a relation with the specified Public Input and Witness. For clarity, We separate the inputs of algorithms with commas ',', semicolons ';', or colons ':', e.g., (Commitments; Bases; Auxiliary bases if applicable: Witnesses). When applicable, we place the parameters corresponding to $\boldsymbol{c}$ before other parameters, e.g., $([C_c], [C_a], [C_b]; [\boldsymbol{U}], [\boldsymbol{G}], [\boldsymbol{H}]; [\boldsymbol{U}'], [\boldsymbol{G}'], [\boldsymbol{H}'] : \boldsymbol{c}, \boldsymbol{a}, \boldsymbol{b})$.

Furthermore, to avoid ambiguity, we use $\leftarrow$ for the assignment operation and $=$ to indicate equivalence. We denote the dimensions of matrices or vectors with n, m, l, the batch size with t, and the indices with i, j, k, τ.

Symbol table We summarize the symbols used in this paper as follows. Symbols not listed below are local and will be defined when they first appear.

- $\boldsymbol{c} \in \mathbb{Z}_p^{m\times n}, \boldsymbol{a} \in \mathbb{Z}_p^{m\times l}, \boldsymbol{b} \in \mathbb{Z}_p^{l\times n}$: Private witness matrices.
- $m, n, l \in \mathbb{N}$: Dimensions of the matrices.
- $i, j, k \in \mathbb{N}$: Indices of the matrix/vector elements.
- $\hat{G} \in \mathbb{G}$ or $\hat{G} \in \mathbb{G}_1, \hat{H} \in \mathbb{G}_2; \hat{s}, \nu \in \mathbb{Z}_p^*$: Group generators and random numbers used to generate the SRS.
- $\boldsymbol{U} \in \mathbb{G}^{m\times n}, \boldsymbol{G} \in \mathbb{G}^{m\times l}, \boldsymbol{H} \in \mathbb{G}^{l\times n}$: Public group matrices serving as bases for $\boldsymbol{c}, \boldsymbol{a}$, and $\boldsymbol{b}$.
- $C_c, C_a, C_b \in \mathbb{G}$: Commitments to the matrices.
- $q \in \mathbb{N}$: The length of the SRS.
- $\vec{\boldsymbol{S}}, \vec{\boldsymbol{S}}' \in \mathbb{G}^q$ or $\mathbb{G}_1^q$: The SRS generated from $\hat{G}$, $\hat{H}$, $\hat{s}$, and ν. Their specific form depends on the setup of the protocols.
- $\vec{\boldsymbol{\zeta}} \in \mathbb{Z}_p^q$: The public coefficients on $\vec{\boldsymbol{S}}$ in the verification check. Its specific form varies in different protocols.
- $\phi(\cdot) \in \mathbb{Z}_p[X]$: A polynomial defined by $\vec{\boldsymbol{\zeta}}$ as in Eq. 10.
- $s \in \mathbb{Z}_p$: The random challenge used to accelerate Bulletproofs and other protocols, as in Subsection 3.1.
- $x_{(j)} \in \mathbb{Z}_p^*$: The random challenge used in the Bulletproof protocol during the jth round of iteration.
- $L_{(j)}, R_{(j)} \in \mathbb{G}$: Transcript elements generated by the Bulletproof protocol during the jth round of iteration.
- $\hat{a}, \hat{b} \in \mathbb{Z}_p$: Transcript elements generated at the end of the Bulletproof protocol.
- $\vec{\boldsymbol{\xi}} \in \mathbb{Z}_p^n$: A public vector computed from $x_{(1)}, \cdots, x_{(\log_2 n)}$ according to the rule in Eq. 7.
- $x \in \mathbb{Z}_p^*$: The random challenge used to multiply C_c in Algorithms 2 and 3.
- $\vec{\boldsymbol{y}}_L \in \mathbb{Z}_p^m, \vec{\boldsymbol{y}}_R \in \mathbb{Z}_p^n, \vec{\boldsymbol{y}} \in \mathbb{Z}_p^{mn}$: The random challenge vectors used in the four inner products in Eq. 3.
- $y \in \mathbb{Z}_p^*$: The random challenge used to compute $\vec{\boldsymbol{y}}_L, \vec{\boldsymbol{y}}_R, \vec{\boldsymbol{y}}$.

- $d \in \mathbb{Z}_p, \vec{\boldsymbol{a}}_y \in \mathbb{Z}_p^l, \vec{\boldsymbol{b}}_y \in \mathbb{Z}_p^l$: Intermediate variables such that $\vec{\boldsymbol{a}}_y = \vec{\boldsymbol{y}}_L^\top \boldsymbol{a}$, $\vec{\boldsymbol{b}}_y = \boldsymbol{b}\vec{\boldsymbol{y}}_R$, and $d = \vec{\boldsymbol{y}}_L^\top \boldsymbol{c}\vec{\boldsymbol{y}}_R$.
- $U' \in \mathbb{G}, \vec{\boldsymbol{G}}' \in \mathbb{G}^l, \vec{\boldsymbol{H}}' \in \mathbb{G}^l$: Public bases for $d, \boldsymbol{a}_y, \boldsymbol{b}_y$.
- $\boldsymbol{\gamma} \in \mathbb{Z}_p^{m\times n}, \boldsymbol{\alpha} \in \mathbb{Z}_p^{m\times l}, \boldsymbol{\beta} \in \mathbb{Z}_p^{l\times n}$: Random matrices used by Algorithm 5 for blinding the matrices $\boldsymbol{c}, \boldsymbol{a}$, and $\boldsymbol{b}$.
- $\rho \in \mathbb{Z}_p^*$: The random challenge used for batch processing.
- $\vartheta, \theta \in \mathbb{Z}_p^*$: Random values used to combine multiple verification equations into a single equation, as in Eq. 13 and 21.

2.2 Cryptographic Assumptions and Definitions

Cryptographic assumptions for our work include the discrete logarithm relation assumption [Bünz et al.(2018)], the q-Power Diffie-Hellman (q-PDH) assumption [Groth(2010), Parno et al.(2016)], the q-Power Knowledge of Exponent (q-PKE) assumption [Groth(2010), Parno et al.(2016)], and the q-Bilinear Strong Diffie-Hellman (q-BSDH) assumption [Kate et al.(2010)]. Our definitions for for Pedersen vector commitment, zero-knowledge arguments of knowledge, perfect completeness, computational knowledge soundness, perfect Special Honest-Verifier Zero-Knowledge (SHVZK), public-coin, Fiat-Shamir heuristics, and fully succinct zk-SNARK align with the definitions presented in [Bünz et al.(2018), Attema et al.(2021), Attema et al.(2022), Gabizon et al.(2019)] These assumptions and definitions are also provided in the appendix of the online full version of this paper.

3 Accelerating Bulletproof Verification

Bulletproof [Bünz et al.(2018)] provides an efficient argument of knowledge for $\vec{\boldsymbol{a}}, \vec{\boldsymbol{b}} \in \mathbb{Z}_p^n$ with a communication complexity of $2\log_2 n$ group elements and 2 field elements for the following relation:

$$\mathcal{R}_{\mathsf{bullet}} = \tag{4}$$
$$\left\{ \begin{pmatrix} [C] \in \mathbb{G}; [U] \in \mathbb{G}; \\ [\vec{\boldsymbol{G}}] \in \mathbb{G}^n, [\vec{\boldsymbol{H}}] \in \mathbb{G}^n \end{pmatrix} : \begin{pmatrix} \vec{\boldsymbol{a}} \in \mathbb{Z}_p^n, \\ \vec{\boldsymbol{b}} \in \mathbb{Z}_p^n \end{pmatrix} \middle| \begin{pmatrix} C = (\vec{\boldsymbol{a}} \bullet \vec{\boldsymbol{b}})U \\ \oplus\langle \vec{\boldsymbol{a}}, \vec{\boldsymbol{G}}\rangle \oplus \langle \vec{\boldsymbol{b}}, \vec{\boldsymbol{H}}\rangle \end{pmatrix} \right\}.$$

To understand Bulletproof in Algorithm 1, let us consider a simplified scenario where $\vec{\boldsymbol{b}} = \vec{\boldsymbol{0}}$. Given the public group vector $[\vec{\boldsymbol{G}}] \in \mathbb{G}^n$ and a commitment $[C] \in \mathbb{G}$, the prover aims to prove the knowledge of a secret vector $\vec{\boldsymbol{a}} \in \mathbb{Z}_p^n$ such that $\langle \vec{\boldsymbol{a}}, [\vec{\boldsymbol{G}}]\rangle = [C]$.

Bulletproof divides the vectors $\vec{\boldsymbol{a}}$ and $[\vec{\boldsymbol{G}}]$ into their left and right halves. There are four inner products of the half vectors as follows:

$$\langle \vec{\boldsymbol{a}}_L, \vec{\boldsymbol{G}}_L\rangle, \quad \langle \vec{\boldsymbol{a}}_R, \vec{\boldsymbol{G}}_R\rangle, \quad \langle \vec{\boldsymbol{a}}_L, \vec{\boldsymbol{G}}_R\rangle, \quad \langle \vec{\boldsymbol{a}}_R, \vec{\boldsymbol{G}}_L\rangle. \tag{5}$$

The last two terms, denoted as:

$$[L_{(1)}] = \langle \vec{\boldsymbol{a}}_L, \vec{\boldsymbol{G}}_R\rangle, \quad [R_{(1)}] = \langle \vec{\boldsymbol{a}}_R, \vec{\boldsymbol{G}}_L\rangle,$$

are included into the transcript.

Upon receiving $[L_{(1)}]$ and $[R_{(1)}]$, the verifier sends a random challenge $[x_{(1)}] \xleftarrow{\$} \mathbb{Z}_p^*$. The prover scales the right halves of $\vec{\boldsymbol{a}}$ and $\vec{\boldsymbol{G}}$ with factors $x_{(1)}^{-1} \in \mathbb{Z}_p^*$ and $x_{(1)} \in \mathbb{Z}_p^*$, respectively, and then folds the right halves with the left halves to derive $\vec{\boldsymbol{a}}' = \vec{\boldsymbol{a}}_L + x_{(1)}^{-1}\vec{\boldsymbol{a}}_R$ and $[\vec{\boldsymbol{G}}'] = [\vec{\boldsymbol{G}}_L] \oplus [x_{(1)}][\vec{\boldsymbol{G}}_R]$. Their inner product becomes:

$$\langle \vec{\boldsymbol{a}}', [\vec{\boldsymbol{G}}'] \rangle = \langle \vec{\boldsymbol{a}}, [\vec{\boldsymbol{G}}] \rangle \oplus ([x_{(1)}][L_{(1)}]) \oplus ([x_{(1)}]^{-1}[R_{(1)}]).$$

This simplifies the original problem to one of half the length. This folding procedure can be recursively applied to reduce the original inner product into a Pedersen commitment to a scalar.

To accelerate Bulletproof, we employ a modified version of the updating rule as described in [Gentry et al.(2022)], which results in an approximate 1/4 reduction in prover time. The corresponding Bulletproof protocol can be found in Algorithm 1.

Algorithm 1: Bulletproof in [Gentry et al.(2022)]

```
Input: Public input: $[C] \in \mathbb{G}$; $[U] \in \mathbb{G}, [\vec{\boldsymbol{G}}], [\vec{\boldsymbol{H}}] \in \mathbb{G}^n$;
       $\mathcal{P}$'s private input: $\vec{\boldsymbol{a}}, \vec{\boldsymbol{b}} \in \mathbb{Z}_p^n$
Output: TRUE ($\mathcal{V}$ accepts) or FALSE ($\mathcal{V}$ rejects)
if $n = 1$ then
    $\mathcal{P} \to \mathcal{V}$: $[a], [b] \in \mathbb{Z}_p$;
    $\mathcal{V}$ checks if $[C] \overset{?}{=} [a][b][U] \oplus [a][G] \oplus [b][H]$:
        if yes, return TRUE; otherwise, return FALSE
else
    // $\mathcal{P}$ and $\mathcal{V}$ compute:
    $n' \leftarrow n/2$;
    $([\vec{\boldsymbol{G}}_L] \,||\, [\vec{\boldsymbol{G}}_R]) \xleftarrow{\mathsf{half}} [\vec{\boldsymbol{G}}]$, $([\vec{\boldsymbol{H}}_L] \,||\, [\vec{\boldsymbol{H}}_R]) \xleftarrow{\mathsf{half}} [\vec{\boldsymbol{H}}]$;
    // $\mathcal{P}$ computes:
    $(\vec{\boldsymbol{a}}_L || \vec{\boldsymbol{a}}_R) \xleftarrow{\mathsf{half}} \vec{\boldsymbol{a}}$, $(\vec{\boldsymbol{b}}_L || \vec{\boldsymbol{b}}_R) \xleftarrow{\mathsf{half}} \vec{\boldsymbol{b}}$;
    $L \leftarrow \langle \vec{\boldsymbol{a}}_L, \vec{\boldsymbol{G}}_R \rangle \oplus \langle \vec{\boldsymbol{b}}_R, \vec{\boldsymbol{H}}_L \rangle \oplus (\vec{\boldsymbol{a}}_L \bullet \vec{\boldsymbol{b}}_R)U \in \mathbb{G}$ ;
    $R \leftarrow \langle \vec{\boldsymbol{a}}_R, \vec{\boldsymbol{G}}_L \rangle \oplus \langle \vec{\boldsymbol{b}}_L, \vec{\boldsymbol{H}}_R \rangle \oplus (\vec{\boldsymbol{a}}_R \bullet \vec{\boldsymbol{b}}_L)U \in \mathbb{G}$ ;
    $\mathcal{P} \to \mathcal{V}$: $[L], [R] \in \mathbb{G}$;
    $\mathcal{V} \to \mathcal{P}$: $[x] \xleftarrow{\$} \mathsf{Coin}(\mathbb{Z}_p^*)$ ;
    // $\mathcal{P}$ computes:
    $\vec{\boldsymbol{a}}' \leftarrow \vec{\boldsymbol{a}}_L + [x]^{-1}\vec{\boldsymbol{a}}_R \in \mathbb{Z}_p^{n'}$, $\vec{\boldsymbol{b}}' \leftarrow \vec{\boldsymbol{b}}_L + [x]\vec{\boldsymbol{b}}_R \in \mathbb{Z}_p^{n'}$;
    // $\mathcal{P}$ and $\mathcal{V}$ compute:
    $[C'] \leftarrow ([x][L]) \oplus [C] \oplus ([x]^{-1}[R]) \in \mathbb{G}$ ;
    $[\vec{\boldsymbol{G}}'] \leftarrow [\vec{\boldsymbol{G}}_L] \oplus [x][\vec{\boldsymbol{G}}_R] \in \mathbb{G}^{n'}$;
    $[\vec{\boldsymbol{H}}'] \leftarrow [\vec{\boldsymbol{H}}_L] \oplus [x]^{-1}[\vec{\boldsymbol{H}}_R] \in \mathbb{G}^{n'}$;
    $\mathcal{P}$ and $\mathcal{V}$ recursively run Bulletproof on input: $([C']; [U], [\vec{\boldsymbol{G}}'], [\vec{\boldsymbol{H}}'] : \vec{\boldsymbol{a}}', \vec{\boldsymbol{b}}')$;
```

3.1 Verification through Multi-Exponentiation

The entire verification process in the Bulletproof protocol can be condensed into a single multi-exponentiation with the following form (refer to Section 3.1 of [Bünz et al.(2018)]):

$$\begin{aligned}[C]\oplus\bigoplus_{j=1}^{\log_2 n}([x_{(j)}][L_{(j)}]\oplus[x_{(j)}]^{-1}[R_{(j)}])\\ =([\hat{a}][\hat{b}])[U]\oplus[\hat{a}]\langle[\vec{\boldsymbol{\xi}}],[\vec{\boldsymbol{G}}]\rangle\oplus[\hat{b}]\langle[\vec{\boldsymbol{\xi}}^{-1}],[\vec{\boldsymbol{H}}]\rangle.\end{aligned} \tag{6}$$

Here, $[x_{(j)}]$ represents the challenge in the jth round, while $[\hat{a}]$ and $[\hat{b}]$ denote field elements in the transcript generated in the final round. $[\vec{\boldsymbol{\xi}}]=([\xi_1],\cdots,[\xi_n])$ is a vector computed from the public-coin challenges as follows:

$$\xi_i=\Pi_{j=1}^{\log_2 n}x_{(j)}^{\iota(i,j)},$$

where $\iota(i,j)$ is the jth bit of $(i-1)$ in big-endian format or, equivalently, the $(\log_2 n-j+1)$th bit in little-endian format. Alternatively, the vector $\vec{\boldsymbol{\xi}}$ can be computed recursively as follows:

$$\vec{\boldsymbol{\xi}}_{(0)}\leftarrow(1);\vec{\boldsymbol{\xi}}_{(j+1)}\leftarrow(\vec{\boldsymbol{\xi}}_{(j)}\,||\,x_{(\log_2 n-j)}\vec{\boldsymbol{\xi}}_{(j)});\vec{\boldsymbol{\xi}}\leftarrow\vec{\boldsymbol{\xi}}_{(\log_2 n)}. \tag{7}$$

In subsequent discussion, where the specific coefficients are not relevant, we define the symbols $\langle[\vec{\boldsymbol{\zeta}}],[\vec{\boldsymbol{S}}]\rangle$ to represent the right hand side of the multi-exponentiation Eq. 6, by setting:

$$\vec{\boldsymbol{\zeta}}=(\hat{a}\hat{b},\hat{a}\xi_1,\hat{b}\xi_1^{-1},\cdots,\hat{a}\xi_n,\hat{b}\xi_n^{-1}),\ \vec{\boldsymbol{S}}=(U,G_1,H_1,\cdots,G_n,H_n). \tag{8}$$

We can see that the verification of Eq.6 requires $2n+2\log_2 n+1$ exponentiations.

Our verification acceleration via prover. In Eq.6, the primary computational burden arises from the evaluation of the multi-exponentiation on the right-hand side. Halo attempts to address this issue by introducing a third party "helper" to collaboratively compute this value [Bowe et al.(2019)]. Our proposition is to let the prover act as the "helper" if they can provide additional proof of the correctness of this computed value.

Our acceleration requires a structured reference string (SRS), and employs the Bulletproof setup described by computing:

$$[\hat{s}^{\omega_i}\hat{G}],\quad \text{where } \omega_i=\omega_0+\epsilon\sum_{j=1}^{\log_2 i}\iota(i,j)2^{\epsilon_{(j)}}. \tag{9}$$

Here, $\iota(i,j)$ represents the jth bit of $i-1$ in little-endian format. The parameters ω_0,ϵ, and $\epsilon_{(j)}$ are chosen such that the sets $U,\boldsymbol{G},\boldsymbol{H}$ contain distinct group elements. For example, by setting $\omega_0=1,2,3$ for $U,\boldsymbol{G},\boldsymbol{H}$, $\epsilon=2$ and $\epsilon_{(j)}=j$, we derive $U=\hat{s}\hat{G}$, $G_i=\hat{s}^{2i}\hat{G}$, and $H_i=\hat{s}^{1+2i}\hat{G}$.

During the setup phase, the setup algorithm selects $\hat{G}$ and randomly chooses $\hat{s}, \nu \in \mathbb{Z}_p^*$. It computes the SRS as:

$$[\vec{\boldsymbol{S}}], [\vec{\boldsymbol{S}}'] := ([\hat{s}\hat{G}], \cdots, [\hat{s}^q\hat{G}]), ([\nu\hat{G}], [\nu\hat{s}\hat{G}], [\nu\hat{s}^2\hat{G}], \cdots, [\nu\hat{s}^q\hat{G}]),$$

where $q > 2n+1$. It subsequently securely discards ν and $\hat{s}$.

By employing a change of notation to $\vec{\boldsymbol{\zeta}} = (\zeta_1, \cdots, \zeta_q)$, where $[\zeta_i] = 0$ for $i > 2n+1$, the right-hand side of Eq. 6 can be represented as:

$$\langle[\vec{\boldsymbol{\zeta}}], [\vec{\boldsymbol{S}}]\rangle = [\zeta_1][\hat{s}\hat{G}] \oplus [\zeta_2][\hat{s}^2\hat{G}] \oplus \cdots \oplus [\zeta_q][\hat{s}^q\hat{G}].$$

This expression can be viewed as a KZG commitment [Kate et al.(2010)] to the following polynomial $\phi(x) \in \mathbb{Z}_p[X]$:

$$\phi(x) = [\zeta_1]x + \cdots + [\zeta_q]x^q. \tag{10}$$

Observe that the polynomial $\phi(x)$ is public knowledge, as it is uniquely determined by the public parameters $[\zeta_1], \ldots, [\zeta_q]$.

During the proving phase, the prover and the verifier runs the original Bulletproof first. Afterward, the verifier sends an additional public-coin challenge $[s] \xleftarrow{\$} \mathsf{Coin}(\mathbb{Z}_p^*)$. In response, the prover computes:

$$[V'] \leftarrow [\zeta_1][\nu\hat{s}\hat{G}] \oplus [\zeta_2][\nu\hat{s}^2\hat{G}] \oplus \cdots \oplus [\zeta_q][\nu\hat{s}^q\hat{G}],$$
$$[W] \leftarrow \bigoplus_{i=1}^{q} \zeta_i\left(\frac{s^i - \hat{s}^i}{s - \hat{s}}\right)\hat{G} = \bigoplus_{i=1}^{q} \psi_i[\hat{s}^{i-1}\hat{G}],$$

where the coefficients $\psi_i, i \in \mathsf{range}(1, q)$ can be computed recursively using the following equations:

$$\psi_q \leftarrow \zeta_q, \quad \psi_i \leftarrow \psi_{i+1}s + \zeta_i.$$

The prover then sends V' and W to the verifier.

The verifier computes:

$$[V] \leftarrow [C] \oplus \bigoplus_{j=1}^{\log_2 n} ([x_{(j)}][L_{(j)}] \oplus [x_{(j)}]^{-1}[R_{(j)}]). \tag{11}$$

The verifier checks the consistency of $[V]$, $[V']$, $[W]$, and the public polynomial $\phi(x)$ by verifying the two pairing equations:

$$\begin{aligned} &e([V], [\nu\hat{G}]) \stackrel{?}{=} e([V'], [\hat{G}]), \\ &e(\phi([s])[\hat{G}] \ominus [V], [\hat{G}]) \stackrel{?}{=} e([s][\hat{G}] \ominus [\hat{s}\hat{G}], [W]). \end{aligned} \tag{12}$$

These two pairing equations can be consolidated into a single pairing equation by using an additional randomness $\vartheta \xleftarrow{\$} \mathbb{Z}_p^*$:

$$\begin{aligned} &e(\phi([s])[\hat{G}] \ominus [V] \oplus [\vartheta][V'], [\hat{G}]) \\ &\qquad \stackrel{?}{=} e([s][\hat{G}] \ominus [\hat{s}\hat{G}], [W]) \oplus e([\vartheta][V], [\nu\hat{G}]). \end{aligned} \tag{13}$$

It is also possible to employ asymmetric pairing to perform our acceleration. Let $\hat{H} \in \mathbb{G}_2$ be the generator of $\mathbb{G}_2$ and include $[\hat{H}], [\nu\hat{H}] \in \mathbb{G}_2$ into the SRS, Eq. 12 becomes:

$$\begin{aligned} & e([V], [\nu\hat{H}]) \overset{?}{=} e([V'], [\hat{H}]), \\ & e(\phi([s])[\hat{G}] \ominus [V], [\hat{H}]) \overset{?}{=} e([W], [s][\hat{H}] \ominus [\hat{s}\hat{H}]). \end{aligned} \tag{14}$$

The cryptographic assumptions for asymmetric pairing should be altered slightly.

Running time evaluation. As shown above, the running time for the verifier includes $2\log_2 n$ exponentiations and $2\log_2 n$ additions in $\mathbb{G}$ (for computing V), 3 pairing computations, 4 exponentiations and 3 additions in $\mathbb{G}$ (for Eq. 13), and the computation of the public polynomial $\phi(s)$ with the form:

$$\begin{aligned} \phi(s) &= \hat{a}\hat{b}s + \hat{a}\sum_{i=1}^{n} \xi_i s^{2i} + \hat{b}\sum_{i=1}^{n} \xi_i^{-1} s^{1+2i} \\ &= \hat{a}\hat{b}s + \hat{a}\vec{\boldsymbol{\xi}} \bullet \vec{\boldsymbol{s}}_G + \hat{b}\vec{\boldsymbol{\xi}}^{-1} \bullet \vec{\boldsymbol{s}}_H, \end{aligned}$$

where $\vec{\boldsymbol{s}}_G = (s^2, \cdots, s^{2n})$ and $\vec{\boldsymbol{s}}_H = (s^3, \cdots, s^{2n+1})$. Remember that $\boldsymbol{\xi}$ and $\boldsymbol{\xi}^{-1}$ can be recursively calculated using Eq. 7. When a vector $\vec{\boldsymbol{s}}$ can be computed through a similar updating rule:

$$\vec{\boldsymbol{s}}_{(0)} \leftarrow (s_1); \quad \vec{\boldsymbol{s}}_{(j+1)} \leftarrow (\vec{\boldsymbol{s}}_{(j)} \,||\, \upsilon_{(j+1)}\vec{\boldsymbol{s}}_{(j)}); \quad \vec{\boldsymbol{s}} \leftarrow \vec{\boldsymbol{s}}_{(\log_2 n)},$$

where $s_1 = s^{\omega_0}$ is the first element of $\vec{\boldsymbol{s}}$, and $\upsilon_{(j)} = \hat{s}^{\epsilon \cdot 2^{\epsilon(j)}}$ by Eq. 9. Then, by utilizing the following formula, the verifier can compute $\vec{\boldsymbol{\xi}} \bullet \vec{\boldsymbol{s}}$ without explicitly calculating $\vec{\boldsymbol{\xi}}$:

$$\vec{\boldsymbol{\xi}} \bullet \vec{\boldsymbol{s}} = s_1 \prod_{j=1}^{\log_2 n} (1 + x_{(\log_2 n - j + 1)} \upsilon_{(j)}). \tag{15}$$

This method demands only $5\log_2 n + 3$ multiplications in $\mathbb{Z}_p$ to compute $\vec{\boldsymbol{\xi}} \bullet \vec{\boldsymbol{s}}$ and $\vec{\boldsymbol{\xi}}^{-1} \bullet \vec{\boldsymbol{s}}$. To compute $\phi(s)$, there are 4 additional multiplications involving $\hat{a}$ and $\hat{b}$. As a result, the time complexity for verifying Eq. 13 is logarithmic.

Theorem 1 (Accelerated Bulletproof). *Under the discrete logarithm relation assumption, the q-PDH assumption, the q-PKE assumption, and the q-BSDH assumption, our accelerated Bulletproof is an argument of knowledge if the original Bulletproof is an argument of knowledge.*

The proof for Theorem 1 is presented in Appendix A.1.

In this section alone, we will be relying on the q-PDH, q-PKE and q-BSDH assumptions. Importantly, it should be noted that this additional proof serves as an accelerating alternative. The verifier retains the option to verify the multi-exponentiation directly by computing it themselves, without being reliant on the accuracy of the additional proof (V', W) provided by the prover, or the q-PDH, q-PKE and q-BSDH assumptions.

Batch processing. An additional useful point is that the prover can provide a batched proof for t multi-exponentiations under the same base by utilizing an extra batching randomness $[\rho] \xleftarrow{\$} \mathbb{Z}_p^*$. Let the multi-exponentiations be expressed as:

$$V_{(1)} = \langle \vec{\boldsymbol{\zeta}}_{(1)}, \vec{\boldsymbol{S}} \rangle, V_{(2)} = \langle \vec{\boldsymbol{\zeta}}_{(2)}, \vec{\boldsymbol{S}} \rangle, \cdots, V_{(t)} = \langle \vec{\boldsymbol{\zeta}}_{(t)}, \vec{\boldsymbol{S}} \rangle.$$

Instead of computing t sets of $V'_{(i)}$ and $W_{(i)}, i \in \mathsf{range}(1,t)$, the prover only needs to compute $(\bar{V}', \bar{W})$ to prove:

$$\bar{V} = \langle \bar{\vec{\boldsymbol{\zeta}}}, \vec{\boldsymbol{S}} \rangle,$$

where $\bar{V} = \bigoplus_{i=1}^{t} \rho^{i-1} V_{(i)}$ and $\bar{\vec{\boldsymbol{\zeta}}} = \sum_{i=1}^{t} \rho^{i-1} \vec{\boldsymbol{\zeta}}_{(i)}$.

4 Building Blocks

4.1 Semi-Inner-Product Argument

Bulletproof is an argument of knowledge that the prover knows the openings of two Pedersen vector commitments $\vec{\boldsymbol{a}}, \vec{\boldsymbol{b}}$ that satisfy the inner product relation $c = (\vec{\boldsymbol{a}} \bullet \vec{\boldsymbol{b}})$, where c is public information. When $\vec{\boldsymbol{b}}$ is a public vector, we can set the bases of $\vec{\boldsymbol{b}}$ to be $\vec{\boldsymbol{O}}$. In this case, $\langle \vec{\boldsymbol{b}}, \vec{\boldsymbol{H}} \rangle$ in $\mathcal{R}_{\mathsf{bullet}}$ is identical to O. We refer to the corresponding protocol as semi-inner-product argument. Consider the extreme case of $n = 1$, we have $c = ab$, where b is public. We have to hide both c and a in commitments C_c and C_a respectively. Hence, we define the relation of semi-inner-product $\mathcal{R}_{\mathsf{sip}}$ as:

$$\mathcal{R}_{\mathsf{sip}} = \tag{16}$$
$$\left\{ \begin{pmatrix} [C_c] \in \mathbb{Z}_p, [C_a] \in \mathbb{G}, \\ [\vec{\boldsymbol{b}}] \in \mathbb{Z}_p^n; \\ [U] \in \mathbb{G}, [\vec{\boldsymbol{G}}] \in \mathbb{G}^n \end{pmatrix} : \left(\vec{\boldsymbol{a}} \in \mathbb{Z}_p^n \right) \middle| \begin{pmatrix} C_c = (\vec{\boldsymbol{a}} \bullet \vec{\boldsymbol{b}}) U \\ \wedge\, C_a = \langle \vec{\boldsymbol{a}}, \vec{\boldsymbol{G}} \rangle \end{pmatrix} \right\}.$$

In the semi-inner-product argument protocol, we do not compute the exponentiations of O. Consequently, the prover's algorithm involves $3n + 4\log_2(n) - 1$ exponentiations. This is similar to the Lightweight Bulletproof proposed by [Gentry et al.(2022)], although the details differ. The verifier checks a single multi-exponentiation with the following form:

$$\begin{aligned} &[x][C_c] \oplus [C_a] \oplus \bigoplus_{j=1}^{\log_2 n} ([x_{(j)}][L_{(j)}] \oplus [x_{(j)}]^{-1}[R_{(j)}]) \\ &\qquad = [\hat{a}][x]([\vec{\boldsymbol{\xi}}] \bullet [\vec{\boldsymbol{b}}])[U] \oplus [\hat{a}] \langle [\vec{\boldsymbol{\xi}}], [\vec{\boldsymbol{G}}] \rangle \end{aligned} \tag{17}$$

Theorem 2 (Semi-Inner-Product). *The semi-inner-product argument protocol is an argument of knowledge if the discrete logarithm relation assumption holds and the underlying Bulletproof is an argument of knowledge.*

The proof for Theorem 2 is given in Appendix A.2.

Algorithm 2: Protocol of Semi-Inner-Product Argument

```
Input: Public input: $[C_c], [C_a] \in \mathbb{G}, [\vec{\boldsymbol{b}}] \in \mathbb{Z}_p^n$;
                     $[U] \in \mathbb{G}, [\vec{\boldsymbol{G}}] \in \mathbb{G}^n$;
       $\mathcal{P}$'s private input: $\vec{\boldsymbol{a}} \in \mathbb{Z}_p^n$
Output: TRUE ($\mathcal{V}$ accepts) or FALSE ($\mathcal{V}$ rejects)
1 $\mathcal{P} \to \mathcal{V}$: $[R] \leftarrow rU \in \mathbb{G}$, where $r \xleftarrow{\$} \mathbb{Z}_p^*$;
2 $\mathcal{V} \to \mathcal{P}$: $[h] \xleftarrow{\$} \mathsf{Coin}(\mathbb{Z}_p^*)$;
3 $\mathcal{P} \to \mathcal{V}$ : $[z] \leftarrow r + h(\vec{\boldsymbol{a}} \bullet \vec{\boldsymbol{b}}) \in \mathbb{Z}_p$ and $[x] \xleftarrow{\$} \mathsf{Coin}(\mathbb{Z}_p^*)$;
4 $\mathcal{V}$ returns FALSE if $[z][U] \neq [R] \oplus [h][C_c]$ ;
5 $\mathcal{P}$ and $\mathcal{V}$ compute $[C] \leftarrow [x][C_c] \oplus [C_a]$ and run Bulletproof on input:
    $([C]; [x][U], [\vec{\boldsymbol{G}}], \vec{\boldsymbol{O}} : \vec{\boldsymbol{a}}, [\vec{\boldsymbol{b}}])$ ;
```

Variations. Our semi-inner-product argument is similar to the sum argument in [Yuen et al.(2021)] with two main differences: (1) the sum argument requires that $\vec{\boldsymbol{b}} = \vec{\boldsymbol{1}}$, (2) the value c is not committed.

It is also worth noting that when $C_c = O$ and $\vec{\boldsymbol{b}} = \vec{\boldsymbol{0}}$, our protocol serves as a proof for the following relation:

$$\mathcal{R}_{\mathsf{com}} = \left\{ [C_a] \in \mathbb{G}, [\vec{\boldsymbol{G}}] \in \mathbb{G}^n : \vec{\boldsymbol{a}} \in \mathbb{Z}_p^n \,\middle|\, C_a = \langle \vec{\boldsymbol{a}}, \vec{\boldsymbol{G}} \rangle \right\},$$

that is, we know a witness $\vec{\boldsymbol{a}}$ for the commitment C_a using the base $\vec{\boldsymbol{G}}$. The protocol for $\mathcal{R}_{\mathsf{com}}$ enables us to develop a protocol for a linear combination of high-dimensional vectors in latter sections.

Optimizations. In Algorithm 2, we use the standard proof of knowledge of discrete logarithm in line 1-4 to ensure that $C_c = cU$ for some $c \in \mathbb{Z}_p$. This increases the transcript size by 1 group element R and 2 field elements (h, z).

To further improve the performance, we can drop the part related to (R, h, z). In that case, we are only able to prove a relaxed relation of $\mathcal{R}_{\mathsf{sip}}$, i.e., $C_c = (\vec{\boldsymbol{a}} \bullet \vec{\boldsymbol{b}})U \oplus \langle \vec{\boldsymbol{c'}}, \vec{\boldsymbol{G}} \rangle$ for some $\vec{\boldsymbol{c'}} \bullet \vec{\boldsymbol{b}} = 0$. Instead of using a proof of knowledge of discrete logarithm, an alternative solution involves demonstrating, through a separate protocol, that $C_c = (\vec{\boldsymbol{a}} \bullet \vec{\boldsymbol{b}})U \oplus \langle \vec{\boldsymbol{c''}}, \vec{\boldsymbol{G}}' \rangle$. Given that $\vec{\boldsymbol{G}}$ and $\vec{\boldsymbol{G}}'$ share no common member, we can confidently deduce that $\vec{\boldsymbol{c'}} = \vec{\boldsymbol{0}}$ and $\vec{\boldsymbol{c''}} = \vec{\boldsymbol{0}}$.

For the relation $\mathcal{R}_{\mathsf{com}}$, the verifier can directly observe that $C_c = 0$. Hence, the lines related to (R, h, z) can also be dropped.

4.2 High-Dimensional Semi-Inner-Product Argument

We can generalize the semi-inner-product argument to the high-dimensional scenario where $\vec{\boldsymbol{a}}_i, i \in \mathsf{range}(1, n)$ are m-dimensional vectors in $\mathbb{Z}_p^m$. We provide a proof for the following relation:

$$\mathcal{R}_{\mathsf{hd-sip}} = \tag{18}$$
$$\left\{ \begin{pmatrix} [C_c], [C_a] \in \mathbb{G}, \\ [b_1], \cdots, [b_n] \in \mathbb{Z}_p; \\ [\vec{\boldsymbol{U}}] \in \mathbb{G}^m, \\ [\vec{\boldsymbol{G}}_1], \cdots, [\vec{\boldsymbol{G}}_n] \in \mathbb{G}^m \end{pmatrix} : \begin{pmatrix} \vec{\boldsymbol{a}}_1, \cdots, \vec{\boldsymbol{a}}_n \\ \in \mathbb{Z}_p^m \end{pmatrix} \middle| \begin{pmatrix} C_c = \langle \Sigma_{i=1}^n \vec{\boldsymbol{a}}_i b_i, \vec{\boldsymbol{U}} \rangle \\ \wedge \\ C_a = \bigoplus_{i=1}^n \langle \vec{\boldsymbol{a}}_i, \vec{\boldsymbol{G}}_i \rangle \end{pmatrix} \right\}.$$

The protocol is presented in Algorithm 3.

Algorithm 3: High-Dimensional Semi-Inner-Product Argument

```
Input: Public input: $[C_c], [C_a] \in \mathbb{G}$, $[\vec{\boldsymbol{b}}] \in \mathbb{Z}_p^n$;
                     $[\vec{\boldsymbol{U}}] \in \mathbb{G}^m$, $[\vec{\boldsymbol{G}}_1], \cdots, [\vec{\boldsymbol{G}}_n] \in \mathbb{G}^m$;
       $\mathcal{P}$'s private input: $\vec{\boldsymbol{a}}_1, \cdots, \vec{\boldsymbol{a}}_n \in \mathbb{Z}_p^m$
Output: TRUE ($\mathcal{V}$ accepts) or FALSE ($\mathcal{V}$ rejects)
1 $\mathcal{V} \rightarrow \mathcal{P}$: $[x] \xleftarrow{\$} \mathsf{Coin}(\mathbb{Z}_p^*)$ ;
  // $\mathcal{P}$ and $\mathcal{V}$ compute:
2 foreach $i \in \mathsf{range}(1, n)$ and $j \in \mathsf{range}(1, m)$ do
3     $[W_{ij}] \leftarrow [G_{ij}] \oplus [x][b_i][U_j] \in \mathbb{G}$ ;
4 $[\vec{\boldsymbol{W}}] \leftarrow ([W_{11}], \cdots, [W_{1m}]; \cdots; [W_{n1}], \cdots, [W_{nm}]) \in \mathbb{G}^{mn}$ ;
5 $\mathcal{P}$ and $\mathcal{V}$ run the Semi-Inner-Product Argument (Algorithm 2) on input:
  $(O, [C_c], \vec{\mathbf{0}}; [G_{11}], [\vec{\boldsymbol{U}}] : \sum_{i=1}^n \vec{\boldsymbol{a}}_i b_i)$;
6 $\mathcal{P}$ and $\mathcal{V}$ run the Semi-Inner-Product Argument (Algorithm 2) on input:
  $(O, [x][C_c] \oplus [C_a], \vec{\mathbf{0}}; [U_1], [\vec{\boldsymbol{W}}] : (\vec{\boldsymbol{a}}_1 || \vec{\boldsymbol{a}}_2 || \cdots || \vec{\boldsymbol{a}}_n))$;
```

Theorem 3. *The high-dimensional semi-inner-product argument is an argument of knowledge if the discrete logarithm relation assumption holds and the underlying semi-inner-product argument is an argument of knowledge.*

The proof for Theorem 3 is given in Appendix A.3.

Optimizations. In Algorithm 3, we incorporate an additional semi-inner-product argument in line 5 to ensure that $C_c = \langle \vec{\boldsymbol{c}}, \vec{\boldsymbol{U}} \rangle$ for some $\vec{\boldsymbol{c}} \in \mathbb{Z}_p^m$. This increases the transcript size by $2 \log_2 m$ group elements and 2 field elements. It is possible to eliminate this additional semi-inner-product argument and verify a relaxed version of this relation $\mathcal{R}_{\mathsf{hd-sip}}$, i.e., $C_c = \langle \sum_{i=1}^n \vec{\boldsymbol{a}}_i b_i, \vec{\boldsymbol{U}} \rangle \oplus \bigoplus_{i=1}^n \langle \vec{\boldsymbol{c}}'_i, \vec{\boldsymbol{G}}_i \rangle$ for some $\sum_{i=1}^n \vec{\boldsymbol{c}}'_i b_i = 0$. We will show in the subsequent section that, in the context of the zkMatrix protocol, where Algorithm 3 is used as a sub-protocol, if another sub-protocol can ensure that $C_c = \langle \vec{\boldsymbol{c}}, \vec{\boldsymbol{U}} \rangle \oplus \langle \vec{\boldsymbol{c}}'', \vec{\boldsymbol{G}}' \rangle$ for some $\vec{\boldsymbol{c}}$ and $\vec{\boldsymbol{c}}''$, we can infer, given that $\cup_{i=1}^n \vec{\boldsymbol{G}}_i$ and $\vec{\boldsymbol{G}}'$ share no common member, that $\vec{\boldsymbol{c}}'_i = \vec{\boldsymbol{c}}'' = \vec{\mathbf{0}}, \forall i$.

Algorithm 4: Protocol MatMul

```
Input: Public input: $[C_c], [C_a], [C_b] \in \mathbb{G}$;
                     $[U'] \in \mathbb{G}, [\vec{\boldsymbol{G}}'], [\vec{\boldsymbol{H}}'] \in \mathbb{G}^l$;
                     $[\boldsymbol{U}] \in \mathbb{G}^{m \times n}, [\boldsymbol{G}] \in \mathbb{G}^{m \times l}, [\boldsymbol{H}] \in \mathbb{G}^{l \times n}$;
       $\mathcal{P}$'s private input: $\boldsymbol{c} \in \mathbb{Z}_p^{m \times n}, \boldsymbol{a} \in \mathbb{Z}_p^{m \times l}, \boldsymbol{b} \in \mathbb{Z}_p^{l \times n}$
Output: TRUE ($\mathcal{V}$ accepts) or FALSE ($\mathcal{V}$ rejects)
$\mathcal{V} \to \mathcal{P}$: $[y] \xleftarrow{\$} \mathsf{Coin}(\mathbb{Z}_p^*)$ ;
// $\mathcal{P}$ and $\mathcal{V}$ compute:
$[\vec{\boldsymbol{y}}] \leftarrow (1, [y], [y]^2, \cdots, [y]^{mn-1}) \in \mathbb{Z}_p^{mn}$ ;
$[\vec{\boldsymbol{y}}_L] \leftarrow (1, [y]^n, [y]^{2n}, \cdots, [y]^{(m-1)n}) \in \mathbb{Z}_p^{m}$ ;
$[\vec{\boldsymbol{y}}_R] \leftarrow (1, [y], [y]^2, \cdots, [y]^{n-1}) \in \mathbb{Z}_p^{n}$ ;
// Partition $\boldsymbol{G}$ into $m$ row vectors
$(\vec{\boldsymbol{G}}_{1*}^\top \in \mathbb{G}^l, \cdots, \vec{\boldsymbol{G}}_{m*}^\top \in \mathbb{G}^l)^\top \xleftarrow{\mathsf{part}} \boldsymbol{G}$ ;
// Partition $\boldsymbol{H}$ into $n$ column vectors
$(\vec{\boldsymbol{H}}_{*1} \in \mathbb{G}^l, \cdots, \vec{\boldsymbol{H}}_{*n} \in \mathbb{G}^l) \xleftarrow{\mathsf{part}} \boldsymbol{H}$ ;
// Flatten $\boldsymbol{U}$ into $(mn)-$dimensional vectors
$\vec{\boldsymbol{U}} = (U_{11}, \cdots, U_{1n}; \cdots; U_{m1}, \cdots, U_{mn}) \xleftarrow{\mathsf{flat}} \boldsymbol{U}$ ;
// $\mathcal{P}$ computes:
$\vec{\boldsymbol{a}}_y \leftarrow \sum_{i=1}^m [y]^{(i-1)n} \vec{\boldsymbol{a}}_{i*} \in \mathbb{Z}_p^l$, where $(\vec{\boldsymbol{a}}_{1*}^\top, \cdots, \vec{\boldsymbol{a}}_{m*}^\top)^\top \xleftarrow{\mathsf{part}} \boldsymbol{a}$ ;
$\vec{\boldsymbol{b}}_y \leftarrow \sum_{j=1}^n [y]^{j-1} \vec{\boldsymbol{b}}_{*j} \in \mathbb{Z}_p^l$, where $(\vec{\boldsymbol{b}}_{*1}, \cdots, \vec{\boldsymbol{b}}_{*n}) \xleftarrow{\mathsf{part}} \boldsymbol{b}$ ;
$d \leftarrow \vec{\boldsymbol{c}} \bullet \vec{\boldsymbol{y}} \in \mathbb{Z}_p$, where $\vec{\boldsymbol{c}} = (c_{11}, \cdots, c_{1n}; \cdots; c_{m1}, \cdots, c_{mn}) \xleftarrow{\mathsf{flat}} \boldsymbol{c}$;
$[C_{ay}] \leftarrow \langle \vec{\boldsymbol{a}}_y, \vec{\boldsymbol{G}}' \rangle \in \mathbb{G}$, $[C_{by}] \leftarrow \langle \vec{\boldsymbol{b}}_y, \vec{\boldsymbol{H}}' \rangle \in \mathbb{G}$, $[C_d] \leftarrow d[U'] \in \mathbb{G}$ ;
$\mathcal{P} \to \mathcal{V}$: $[C_d], [C_{ay}], [C_{by}]$ ;
$\mathcal{P}$ and $\mathcal{V}$ run Semi-Inner-Product Argument on input:
    $([C_d], [C_c], [\vec{\boldsymbol{y}}]; [U'], [\vec{\boldsymbol{U}}] : \vec{\boldsymbol{c}})$ ;
$\mathcal{P}$ and $\mathcal{V}$ run High-Dimensional Semi-Inner-Product Argument on input:
    $([C_{ay}], [C_a], [\vec{\boldsymbol{y}}_L]; [\vec{\boldsymbol{G}}'], ([\vec{\boldsymbol{G}}_{1*}^\top], \cdots, [\vec{\boldsymbol{G}}_{m*}^\top]) : (\vec{\boldsymbol{a}}_{1*}, \cdots, \vec{\boldsymbol{a}}_{m*}))$ ;
$\mathcal{P}$ and $\mathcal{V}$ run High-Dimensional Semi-Inner-Product Argument on input:
    $([C_{by}], [C_b], [\vec{\boldsymbol{y}}_R]; [\vec{\boldsymbol{H}}'], ([\vec{\boldsymbol{H}}_{*1}], \cdots, [\vec{\boldsymbol{H}}_{*n}]) : (\vec{\boldsymbol{b}}_{*1}, \vec{\boldsymbol{b}}_{*2}, \cdots, \vec{\boldsymbol{b}}_{*n}))$ ;
$\mathcal{P}$ and $\mathcal{V}$ run Bulletproof on input: $([C_d] \oplus [C_{ay}] \oplus [C_{by}]; [U'], [\vec{\boldsymbol{G}}'], [\vec{\boldsymbol{H}}'] : \vec{\boldsymbol{a}}_y, \vec{\boldsymbol{b}}_y)$ ;
If all return TRUE, then return TRUE; otherwise return FALSE
```

Accelerating Verification. In this protocol, the verifier checks a single multi-exponentiation as follows:

$$[x][C_c] \oplus [C_a] \oplus \bigoplus_{j=1}^{\log_2(mn)} ([x_{(j)}][L_{(j)}] \oplus [x_{(j)}]^{-1}[R_{(j)}]) \\ = [\hat{a}][x] \langle [\boldsymbol{\xi}][\vec{\boldsymbol{b}}], [\vec{\boldsymbol{U}}] \rangle \oplus [\hat{a}] \langle [\boldsymbol{\xi}], [\boldsymbol{G}] \rangle, \tag{19}$$

where $[\vec{\boldsymbol{G}}] \in \mathbb{G}^{m \times n} = ([\vec{\boldsymbol{G}}_1], \cdots, [\vec{\boldsymbol{G}}_n])$. $[\boldsymbol{\xi}] \in \mathbb{Z}_p^{m \times n} \xleftarrow{\mathsf{fold}} [\vec{\boldsymbol{\xi}}] \in \mathbb{Z}_p^{mn}$ is a matrix determined by the public-coin challenges. Direct mathematical derivation shows that $[\boldsymbol{\xi}] = \vec{\boldsymbol{\xi}}_m \vec{\boldsymbol{\xi}}_n^\top$, where $\vec{\boldsymbol{\xi}}_m$ is computed from the last $\log_2 m$ public-coin challenges via Eq. 7, and $\vec{\boldsymbol{\xi}}_n$ is computed from the first $\log_2 n$ public-coin challenges via Eq. 7.

Here, we adopt the following setup:

$$\begin{aligned}
\vec{\boldsymbol{U}} &= (\hat{s}^{nm+1}\hat{G}, \hat{s}^{nm+2}\hat{G}, \cdots, \hat{s}^{nm+m}\hat{G}) := \vec{\hat{\boldsymbol{s}}}_U \hat{G}, \\
\boldsymbol{G} &= \begin{pmatrix} \hat{s}^1\hat{G} & \hat{s}^{m+1}\hat{G} & \cdots & \hat{s}^{(n-1)m+1}\hat{G} \\ \hat{s}^2\hat{G} & \hat{s}^{m+2}\hat{G} & \cdots & \hat{s}^{(n-1)m+2}\hat{G} \\ \vdots & \vdots & \vdots & \vdots \\ \hat{s}^m\hat{G} & \hat{s}^{2m}\hat{G} & \cdots & \hat{s}^{nm}\hat{G} \end{pmatrix} \\
&= (\hat{s}, \hat{s}^2, \cdots, \hat{s}^m)^T (1, \hat{s}^m, \cdots, \hat{s}^{(n-1)m})\hat{G} \\
&:= \vec{\hat{\boldsymbol{s}}}_{GL} \vec{\hat{\boldsymbol{s}}}_{GR}^\top \hat{G}.
\end{aligned}$$

With the above setup, we can derive the following:

$$\begin{aligned}
\langle \boldsymbol{\xi}\vec{\boldsymbol{b}}, \vec{\boldsymbol{U}} \rangle &= ((\boldsymbol{\xi}\vec{\boldsymbol{b}}) \bullet \vec{\hat{\boldsymbol{s}}}_U)\hat{G} = (\vec{\hat{\boldsymbol{s}}}_U^\top \vec{\boldsymbol{\xi}}_m \vec{\boldsymbol{\xi}}_n^\top \vec{\boldsymbol{b}})\hat{G} \\
&= (\vec{\boldsymbol{\xi}}_m \bullet \vec{\hat{\boldsymbol{s}}}_U)(\vec{\boldsymbol{\xi}}_n \bullet \vec{\boldsymbol{b}})\hat{G}. \\
\langle \boldsymbol{\xi}, \boldsymbol{G} \rangle &= \langle \boldsymbol{\xi}, \vec{\hat{\boldsymbol{s}}}_{GL} \vec{\hat{\boldsymbol{s}}}_{GR}^\top \hat{G} \rangle = \langle \vec{\boldsymbol{\xi}}_m \vec{\boldsymbol{\xi}}_n^\top, \vec{\hat{\boldsymbol{s}}}_{GL} \vec{\hat{\boldsymbol{s}}}_{GR}^\top \hat{G} \rangle \\
&= (\vec{\boldsymbol{\xi}}_m \bullet \vec{\hat{\boldsymbol{s}}}_{GL})(\vec{\boldsymbol{\xi}}_n \bullet \vec{\hat{\boldsymbol{s}}}_{GR})\hat{G}.
\end{aligned}$$

The inner products on the right-hand sides of these equations can be calculated using the formula in Eq. 15. Through these equations, we can apply a procedure similar to the one described in Section 3 to reduce the verifier time to a logarithmic scale.

5 zkMatrix

In this section, we will give a zero-knowledge proof that a matrix $\boldsymbol{c} \in \mathbb{Z}_p^{m \times n}$ is the product of $\boldsymbol{a} \in \mathbb{Z}_p^{m \times l}$ and $\boldsymbol{b} \in \mathbb{Z}_p^{l \times n}$.

5.1 Committed Matrix Multiplication

For any $y \in \mathbb{Z}_p$, we define vectors $\vec{\boldsymbol{y}}_L^\top := (1, y^n, \cdots, y^{(m-1)n})$, $\vec{\boldsymbol{y}}_R := (1, y, \cdots, y^{n-1})^\top$ and a matrix $\boldsymbol{y} := \vec{\boldsymbol{y}}_L \vec{\boldsymbol{y}}_R^T \in \mathbb{Z}_p^{m \times n}$. Recall that $\vec{\boldsymbol{c}} \xleftarrow{\mathsf{flat}} \boldsymbol{c}$ and $\vec{\boldsymbol{y}} \xleftarrow{\mathsf{flat}} \boldsymbol{y}$.

Matrix multiplication can be performed by combining four sub-protocols, based on the following equivalence:

$$\begin{aligned}
\{\boldsymbol{c} = \boldsymbol{a}\boldsymbol{b}\} &\iff \{\forall y, \, \vec{\boldsymbol{y}}_L^\top \boldsymbol{c} \vec{\boldsymbol{y}}_R = (\vec{\boldsymbol{y}}_L^\top \boldsymbol{a})(\boldsymbol{b}\vec{\boldsymbol{y}}_R)\} \\
\iff \{\forall y, \, \exists d, \vec{\boldsymbol{a}}_y, \vec{\boldsymbol{b}}_y \; s.t. \, &(d = \vec{\boldsymbol{c}} \bullet \vec{\boldsymbol{y}}) \wedge (\vec{\boldsymbol{a}}_y = \sum_{i=1}^{m} \vec{\boldsymbol{a}}_{i*} y^{(i-1)n}) \\
&\wedge (\vec{\boldsymbol{b}}_y = \sum_{j=1}^{n} \vec{\boldsymbol{b}}_{*j} y^{j-1}) \wedge (d = \vec{\boldsymbol{a}}_y \bullet \vec{\boldsymbol{b}}_y)\}.
\end{aligned} \tag{20}$$

We first show the equivalence of Eq. 20 above:

1. For the relation $\vec{\boldsymbol{a}}_y = \vec{\boldsymbol{y}}_L^\top \boldsymbol{a}$, recall that $\vec{\boldsymbol{a}}_{i*}$ is the i-th row-vector of $\boldsymbol{a}$. Hence, we have $\vec{\boldsymbol{a}}_y = \sum_{i=1}^m \vec{\boldsymbol{a}}_{i*} y^{(i-1)n}$.
2. For the relation $\vec{\boldsymbol{b}}_y = \boldsymbol{b}\vec{\boldsymbol{y}}_R$, recall that $\vec{\boldsymbol{b}}_{*j}$ is the j-th column-vector of $\boldsymbol{b}$. Hence, we have $\vec{\boldsymbol{b}}_y = \sum_{j=1}^n \vec{\boldsymbol{b}}_{*j} y^{j-1}$.
3. We define $d := \vec{\boldsymbol{a}}_y \bullet \vec{\boldsymbol{b}}_y = (\vec{\boldsymbol{y}}_L^\top \boldsymbol{a})(\boldsymbol{b}\vec{\boldsymbol{y}}_R)$. Hence we have:

$$\begin{aligned} d = \vec{\boldsymbol{y}}_L^\top \boldsymbol{c} \vec{\boldsymbol{y}}_R = (\sum_{i=1}^m \vec{\boldsymbol{c}}_{i*} y^{(i-1)n}) \bullet \vec{\boldsymbol{y}}_R = \sum_{i=1}^m \sum_{j=1}^n c_{ij} \cdot y^{(i-1)n+j-1} \\ = \vec{\boldsymbol{c}} \bullet \vec{\boldsymbol{y}}. \end{aligned}$$

The matrix multiplication argument MatMul is provided in Algorithm 4. We use the relaxed version of semi-inner-product argument for $d = \vec{\boldsymbol{c}} \bullet \vec{\boldsymbol{y}}$, and the relaxed version of high-dimensional semi-inner-product argument for $\vec{\boldsymbol{a}}_y = \sum_{i=1}^m \vec{\boldsymbol{a}}_{i*} y^{(i-1)n}$ and $\vec{\boldsymbol{b}}_y = \sum_{j=1}^n \vec{\boldsymbol{b}}_{*j} y^{j-1}$. We use the (accelerated) Bulletproof for $d := \vec{\boldsymbol{a}}_y \bullet \vec{\boldsymbol{b}}_y$. We employ the commitment to d and the Pedersen vector commitments to $\vec{\boldsymbol{a}}_y$ and $\vec{\boldsymbol{b}}_y$ as the bridges between these four sub-protocols.

Theorem 4 (MatMul Protocol). *The protocol MatMul exhibits perfect completeness and computational knowledge soundness.*

The proof for Theorem 4 is presented in Appendix A.4.

Accelerating Verification. By employing an additional public-coin batching randomness $[\theta] \in \mathbb{Z}_p^*$, we can merge the first four sub-protocols of the MatMul protocol into a single multi-exponentiation check as follows:

$$\begin{aligned} &\theta \cdot [\text{(LHS - RHS) of Eq. 17 for the inner product } \textcircled{1}] \\ &+ \theta^2 \cdot [\text{(LHS - RHS) of Eq. 19 for the inner product } \textcircled{2}] \\ &+ \theta^3 \cdot [\text{(LHS - RHS) of Eq. 19 for the inner product } \textcircled{3}] \\ &+ [\text{(LHS - RHS) of Eq. 6 for the inner product } \textcircled{4}] = 0. \end{aligned} \tag{21}$$

By adopting the following setup and applying the techniques outlined in Section 3, the verifier can efficiently validate this multi-exponentiation equation through the calculation of a known polynomial and three pairings, achieving logarithmic time complexity.

$$\boldsymbol{U} = \begin{pmatrix} \hat{s}\hat{G} & \hat{s}^2\hat{G} & \cdots & \hat{s}^n\hat{G} \\ \hat{s}^{n+1}\hat{G} & \hat{s}^{n+2}\hat{G} & \cdots & \hat{s}^{2n}\hat{G} \\ \vdots & \vdots & \vdots & \vdots \\ \hat{s}^{(m-1)n+1}\hat{G} & \hat{s}^{(m-1)n+2}\hat{G} & \cdots & \hat{s}^{mn}\hat{G} \end{pmatrix},$$

$$\boldsymbol{G} = \begin{pmatrix} \hat{s}\hat{G} & \hat{s}^2\hat{G} & \cdots & \hat{s}^l\hat{G} \\ \hat{s}^{l+1}\hat{G} & \hat{s}^{l+2}\hat{G} & \cdots & \hat{s}^{2l}\hat{G} \\ \vdots & \vdots & \vdots & \vdots \\ \hat{s}^{(m-1)l+1}\hat{G} & \hat{s}^{(m-1)l+2}\hat{G} & \cdots & \hat{s}^{ml}\hat{G} \end{pmatrix},$$

$$
\begin{aligned}
\boldsymbol{H} &= \begin{pmatrix} \hat{s}\hat{G} & \hat{s}^2\hat{G} & \cdots & \hat{s}^n\hat{G} \\ \hat{s}^{n+1}\hat{G} & \hat{s}^{n+2}\hat{G} & \cdots & \hat{s}^{2n}\hat{G} \\ \vdots & \vdots & \vdots & \vdots \\ \hat{s}^{(l-1)n+1}\hat{G} & \hat{s}^{(l-1)n+2}\hat{G} \cdots & & \hat{s}^{ln}\hat{G} \end{pmatrix}, \\
U' &= \hat{s}^{\max\{mn,ml,ln\}+1}\hat{G}, \\
\vec{\boldsymbol{G}}' &= \hat{s}^{\max\{mn,ml,ln\}}(\hat{s}^2\hat{G}, \hat{s}^4\hat{G}, \cdots, \hat{s}^{2l}\hat{G}), \\
\vec{\boldsymbol{H}}' &= \hat{s}^{\max\{mn,ml,ln\}}(\hat{s}^3\hat{G}, \hat{s}^5\hat{G}, \cdots, \hat{s}^{2l+1}\hat{G}). \qquad (22)
\end{aligned}
$$

In this setting, $\boldsymbol{U}$, $\boldsymbol{G}$, and $\boldsymbol{H}$ share elements in common. Despite this, the computational knowledge soundness of the MatMul protocol, as outlined in the proof of Theorem 4, remains unaffected.

Algorithm 5: zkMatrix

```
Input: Public input: $[C_c], [C_a], [C_b] \in \mathbb{G}; \tilde{G} \in \mathbb{G}$;
                     $[U'] \in \mathbb{G}, [\vec{\boldsymbol{G}}'], [\vec{\boldsymbol{H}}'] \in \mathbb{G}^l$;
                     $[\boldsymbol{U}] \in \mathbb{G}^{m\times n}, [\boldsymbol{G}] \in \mathbb{G}^{m\times l}, [\boldsymbol{H}] \in \mathbb{G}^{l\times n}$;
       $\mathcal{P}$'s private input: $\boldsymbol{c} \in \mathbb{Z}_p^{m\times n}, \boldsymbol{a} \in \mathbb{Z}_p^{m\times l}, \boldsymbol{b} \in \mathbb{Z}_p^{l\times n}$,
                     $\tilde{c}, \tilde{a}, \tilde{b} \in \mathbb{Z}_p^*$
Output: TRUE ($\mathcal{V}$ accepts) or FALSE ($\mathcal{V}$ rejects)
// $\mathcal{P}$ chooses the blinding factors at random:
$\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{Z}_p^{m\times l}$, $\boldsymbol{\beta} \xleftarrow{\$} \mathbb{Z}_p^{l\times n}$, $\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\delta} \xleftarrow{\$} \mathbb{Z}_p$;
// $\mathcal{P}$ computes:
$C_\alpha \leftarrow \langle \boldsymbol{\alpha}, \boldsymbol{G} \rangle \oplus \tilde{\alpha}\tilde{G} \in \mathbb{G}$,      $C_\beta \leftarrow \langle \boldsymbol{\beta}, \boldsymbol{H} \rangle \oplus \tilde{\beta}\tilde{G} \in \mathbb{G}$;
$C_\gamma \leftarrow \langle \boldsymbol{\alpha\beta}, \boldsymbol{U} \rangle \oplus \tilde{\gamma}\tilde{G} \in \mathbb{G}$,      $C_\delta \leftarrow \langle \boldsymbol{\alpha b} + \boldsymbol{a\beta}, \boldsymbol{U} \rangle \oplus \tilde{\delta}\tilde{G} \in \mathbb{G}$;
$\mathcal{P} \to \mathcal{V}$: $[C_\alpha], [C_\beta], [C_\gamma], [C_\delta]$ ;
$\mathcal{V} \to \mathcal{P}$ : $[x] \xleftarrow{\$} \mathsf{Coin}(\mathbb{Z}_p^*)$ ;
// $\mathcal{P}$ computes:
$z_a \leftarrow [-\tilde{\alpha} - x\tilde{a}] \in \mathbb{Z}_p$,      $z_b \leftarrow [-\tilde{\beta} - x\tilde{b}] \in \mathbb{Z}_p$ ;
$z_c \leftarrow [-\tilde{\gamma} - x\tilde{\delta} - x^2\tilde{c}] \in \mathbb{Z}_p$ ;
$\mathcal{P} \to \mathcal{V}$: $[z_a], [z_b], [z_c]$ ;
// $\mathcal{P}$ and $\mathcal{V}$ compute:
$[P_a] \leftarrow [x]([C_\alpha] + [x][C_a] + [z_a][\tilde{G}]) \in \mathbb{G}$;
$[P_b] \leftarrow [x]([C_\beta] + [x][C_b] + [z_b][\tilde{G}]) \in \mathbb{G}$ ;
$[P_c] \leftarrow [x]^2([C_\gamma] + [x][C_\delta] + [x]^2[C_c] + [z_c][\tilde{G}]) \in \mathbb{G}$ ;
$\mathcal{P}$ and $\mathcal{V}$ run Protocol MatMul on input:
  $([P_c], [P_a], [P_b]; [U'], [\vec{\boldsymbol{G}}'], [\vec{\boldsymbol{H}}']; [\boldsymbol{U}], [\boldsymbol{G}], [\boldsymbol{H}] : x^2(\boldsymbol{\alpha} + x\boldsymbol{a})(\boldsymbol{\beta} + x\boldsymbol{b})$,
  $x(\boldsymbol{\alpha} + x\boldsymbol{a}), x(\boldsymbol{\beta} + x\boldsymbol{b}))$;
```

5.2 Adding Zero-Knowledge

The zero-knowledge version of committed matrix multiplication corresponds to the following relation:

$$\mathcal{R}_{\mathsf{zkMatMul}} = \tag{23}$$
$$\left\{ \begin{pmatrix} [C_c], [C_a], [C_b] \in \mathbb{G} \\ [\boldsymbol{U}] \in \mathbb{G}^{m\times n}, \\ [\boldsymbol{G}] \in \mathbb{G}^{m\times l}, \\ [\boldsymbol{H}] \in \mathbb{G}^{l\times n}; \\ [\tilde{G}] \in \mathbb{G} \end{pmatrix} : \begin{pmatrix} \boldsymbol{c} \in \mathbb{Z}_p^{m\times n} \\ \boldsymbol{a} \in \mathbb{Z}_p^{m\times l}, \\ \boldsymbol{b} \in \mathbb{Z}_p^{l\times n}, \\ \tilde{c}, \tilde{a}, \tilde{b} \in \mathbb{Z}_p \end{pmatrix} \middle| \begin{pmatrix} \boldsymbol{c} = \boldsymbol{a}\boldsymbol{b} \\ \wedge\, C_c = \langle \boldsymbol{c}, \boldsymbol{U} \rangle \oplus \tilde{c}\tilde{G} \\ \wedge\, C_a = \langle \boldsymbol{a}, \boldsymbol{G} \rangle \oplus \tilde{a}\tilde{G} \\ \wedge\, C_b = \langle \boldsymbol{b}, \boldsymbol{H} \rangle \oplus \tilde{b}\tilde{G} \end{pmatrix} \right\}.$$

Here, blinding factors are used for the commitments to matrices.

To transform a single MatMul protocol into a zero-knowledge protocol, we can introduce two blinding matrices $\boldsymbol{\alpha} \in \mathbb{Z}_p^{m\times l}$ and $\boldsymbol{\beta} \in \mathbb{Z}_p^{l\times n}$, along with additional random factors $\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\delta} \in \mathbb{Z}_p$.

In the first step of zkMatrix, the prover sends a series of commitments: $[C_\alpha], [C_\beta], [C_\gamma], [C_\delta] \in \mathbb{G}$. Subsequently, in response to a public-coin challenge $[x] \xleftarrow{\$} \mathbb{Z}_p^*$, the prover provides an argument of knowledge for a relation between $x(\boldsymbol{\alpha} + x\boldsymbol{a}), x(\boldsymbol{\beta} + x\boldsymbol{b})$, and $x^2(\boldsymbol{\alpha} + x\boldsymbol{a})(\boldsymbol{\beta} + x\boldsymbol{b})$. The transcript size is increased by a constant factor. The zkMatrix protocol is detailed in Algorithm 5.

The perfect completeness and computational knowledge soundness of the zkMatrix protocol is based on the following equivalence:

$$\{\boldsymbol{c} = \boldsymbol{a}\boldsymbol{b}\} \Leftrightarrow \left\{ \begin{aligned} \forall [x],\ [x]^2(\boldsymbol{\alpha}\boldsymbol{\beta} + [x](\boldsymbol{a}\boldsymbol{\beta} + \boldsymbol{\alpha}\boldsymbol{b}) + [x]^2\boldsymbol{c}) \\ = ([x](\boldsymbol{\alpha} + [x]\boldsymbol{a}))([x](\boldsymbol{\beta} + [x]\boldsymbol{b})) \end{aligned} \right\}, \tag{24}$$

and:

$$\begin{aligned} &\langle \boldsymbol{\alpha} + [x]\boldsymbol{a}, \boldsymbol{G} \rangle = [\langle \boldsymbol{\alpha}, \boldsymbol{G} \rangle \oplus \tilde{\alpha}\tilde{G}] \oplus [x][\langle \boldsymbol{a}, \boldsymbol{G} \rangle \oplus \tilde{a}\tilde{G}] \oplus [-\tilde{\alpha} - x\tilde{a}][\tilde{G}], \\ &\langle \boldsymbol{\beta} + [x]\boldsymbol{b}, \boldsymbol{H} \rangle = [\langle \boldsymbol{\beta}, \boldsymbol{H} \rangle \oplus \tilde{\beta}\tilde{G}] \oplus [x][\langle \boldsymbol{b}, \boldsymbol{H} \rangle \oplus \tilde{b}\tilde{G}] \oplus [-\tilde{\beta} - x\tilde{b}][\tilde{G}], \\ &\langle \boldsymbol{\alpha}\boldsymbol{\beta} + [x](\boldsymbol{a}\boldsymbol{\beta} + \boldsymbol{\alpha}\boldsymbol{b}) + [x]^2\boldsymbol{c}, \boldsymbol{U} \rangle \oplus [\tilde{\gamma} + x\tilde{\delta} + x^2\tilde{c}][\tilde{G}] \\ &\quad = [\langle \boldsymbol{\alpha}\boldsymbol{\beta}, \boldsymbol{U} \rangle \oplus \tilde{\gamma}\tilde{G}] \oplus [x][\langle \boldsymbol{a} {*} \boldsymbol{\beta} + \boldsymbol{\alpha}\boldsymbol{b}, \boldsymbol{U} \rangle \oplus \tilde{\delta}\tilde{G}] \oplus [x]^2[\langle \boldsymbol{c}, \boldsymbol{U} \rangle \oplus \tilde{c}\tilde{G}]. \end{aligned}$$

The zero-knowledge property is based on the observation that $x(\boldsymbol{\alpha} + x\boldsymbol{a})$ and $x(\boldsymbol{\beta} + x\boldsymbol{b})$ exhibit uniform distributions, regardless of the distributions of $\boldsymbol{a}$ and $\boldsymbol{b}$. Consequently, the simulator of the proof is able to generate a valid transcript using two random matrices, which is indistinguishable from a genuine one.

Theorem 5. *The zkMatrix protocol exhibits perfect completeness, perfect special honest verifier zero-knowledge, and computational knowledge soundness.*

The proof for Theorem 5 is provided in Appendix A.5.

Line 3 of Algorithm 5 requires $O(n^3)$ filed multiplications for $n \times n$ matrix multiplications. This conflicts with our goal of achieving $O(n^2)$ prover time. Despite this, we note that: (1) Algorithm 5 can be applied to any bilinear matrix or vector operations; (2) The additional prover time reduces to $O(n^2)$ when either $\boldsymbol{a}$ or $\boldsymbol{b}$ is a n-vector, which applies to the inner products ② and ③; (3) Computing the inner products of two n^2-vectors takes $O(n^2)$ field multiplications, which applies to the inner product ①; (4) Blinding matrices are unnecessary for public matrices or vectors, which applies to the inner products ①, ②, and ③. Therefore, in the final zkMatrix protocol, we introduce the blinding matrices within the sub-protocols of lines 13-16 in Algorithm 4. Specifically, we use blinding matrices $\boldsymbol{\gamma} \in \mathbb{Z}_p^{m\times n}$ for the inner product ①, $\boldsymbol{\alpha} \in \mathbb{Z}_p^{m\times l}$ for the inner product ②, $\boldsymbol{\beta} \in \mathbb{Z}_p^{l\times n}$ for the inner product ③, and two blinding vectors $\vec{\boldsymbol{\alpha}}_y, \vec{\boldsymbol{\beta}}_y \in \mathbb{Z}_p^l$ for the inner product ④. As a result, the additional prover time to add zero-knowledge is $O(n^2)$ exponentiations and $O(n^2)$ field multiplications.

6 Optimizations

6.1 Batched zkMatrix

It is feasible to batch process t matrix multiplication relations. This insight stems from the observation that when a matrix $[\boldsymbol{b}]$ is publicly known and shared across t relations, the prover can prove:

$$\begin{aligned} &\left\{\boldsymbol{a}_{(i)}[\boldsymbol{b}] = \boldsymbol{c}_{(i)},\, \forall i \in (1,t)\right\} \\ \iff &\left\{\forall \rho \in \mathbb{Z}_p^*, \quad (\sum_{i=1}^{t} \rho^{i-1}\boldsymbol{a}_{(i)})[\boldsymbol{b}] = \sum_{i=1}^{t} \rho^{i-1}\boldsymbol{c}_{(i)}\right\}, \end{aligned} \tag{25}$$

where $[\rho] \xleftarrow{\$} \mathsf{Coin}(\mathbb{Z}_p^*)$ is an additional public-coin randomness for batch processing. Consequently, the prover is able to batch processing the inner-product relations ①, ②, and ③. In the batched zkMatrix, the marginal prover time for each additional matrix multiplication is on the order of $O(n)$ ($\sim 10n$ exponentiations) for $n \times n$ matrix multiplications.

Theorem 6. *The batched zkMatrix protocol exhibits perfect completeness, perfect special honest verifier zero-knowledge, and computational knowledge soundness.*

The proof for Theorem 6 is provided in Appendix A.6.

6.2 Fixed-Base Optimization and Parallelization

It is standard practice to accelerate multi-exponentiations by using fixed-base optimization and parallelization. Each additional fixed-base multi-exponentiation of length n requires on average $n(\log_2 p)/2$ group additions. Almost all operations in zkMatrix are parallelizable. This means that the performance of zkMatrix can scale with the number of cores available on the machine. It can also leverage GPUs in some applications.

Algorithm 6: zkMatrix (Batched Proof)

```
Input: Public input: $[C_{c(i)}], [C_{a(i)}], [C_{b(i)}] \in \mathbb{G}$,
                                        $i \in \mathsf{range}(1, t)$;
            $[\tilde{G}] \in \mathbb{G}; [U'] \in \mathbb{G}, [\vec{\boldsymbol{G}}'], [\vec{\boldsymbol{H}}'] \in \mathbb{G}^l$;
            $[\boldsymbol{U}] \in \mathbb{G}^{m\times n}, [\boldsymbol{G}] \in \mathbb{G}^{m\times l}, [\boldsymbol{H}] \in \mathbb{G}^{l\times n}$;
       $\mathcal{P}$'s private input: $\boldsymbol{c}_{(i)} \in \mathbb{Z}_p^{m\times n}, \boldsymbol{a}_{(i)} \in \mathbb{Z}_p^{m\times l}$,
                      $\boldsymbol{b}_{(i)} \in \mathbb{Z}_p^{l\times n}$, $\tilde{c}_{(i)}, \tilde{a}_{(i)}, \tilde{b}_{(i)} \in \mathbb{Z}_p$,
                                        $i \in \mathsf{range}(1, t)$
Output: TRUE ($\mathcal{V}$ accepts) or FALSE ($\mathcal{V}$ rejects)
// $\mathcal{V}$ sends the challenge for reducing matrix multiplication:
$\mathcal{V} \to \mathcal{P}$: $[y] \xleftarrow{\$} \mathsf{Coin}(\mathbb{Z}_p^*)$;
// $\mathcal{P}$ makes hiding commitments to $t$ sets of intermediate variables:
foreach $i \in \mathsf{range}(1, t)$ do
    $\mathcal{P}$ runs line 2-13 of MatMul to compute $\vec{\boldsymbol{a}}_{y(i)} \in \mathbb{Z}_p^l, \vec{\boldsymbol{b}}_{y(i)} \in \mathbb{Z}_p^l, d_{(i)} \in \mathbb{Z}_p$;
    $\tilde{d}_{(i)}, \tilde{a}_{y(i)}, \tilde{b}_{y(i)} \xleftarrow{\$} \mathbb{Z}_p$ ;
    $C_{ay(i)} \leftarrow \langle \vec{\boldsymbol{a}}_{y(i)}, \vec{G}' \rangle \oplus \tilde{a}_{y(i)} \tilde{G} \in \mathbb{G}$;
    $C_{by(i)} \leftarrow \langle \vec{\boldsymbol{b}}_{y(i)}, \vec{H}' \rangle \oplus \tilde{b}_{y(i)} \tilde{G} \in \mathbb{G}$;
    $C_{d(i)} \leftarrow d_{y(i)} U' \oplus \tilde{d}_{(i)} \tilde{G} \in \mathbb{G}$;
    $\mathcal{P} \to \mathcal{V}$: $[C_{d(i)}], [C_{ay(i)}], [C_{by(i)}] \in \mathbb{G}$;
// $\mathcal{V}$ sends the batching randomness
$\mathcal{V} \to \mathcal{P}$: $[\rho] \xleftarrow{\$} \mathsf{Coin}(\mathbb{Z}_p^*)$ ;
// $\mathcal{P}$ and $\mathcal{V}$ compute:
$[\bar{C}_c] \leftarrow \sum_{i=1}^t [\rho]^{i-1} C_{c(i)}$, $[\bar{C}_a] \leftarrow \sum_{i=1}^t [\rho]^{i-1} C_{a(i)}$, $[\bar{C}_b] \leftarrow \sum_{i=1}^t [\rho]^{i-1} C_{b(i)}$,
  $[\bar{C}_d] \leftarrow \sum_{i=1}^t [\rho]^{i-1} C_{d(i)}$, $[\bar{C}_{ay}] \leftarrow \sum_{i=1}^t [\rho]^{i-1} C_{ay(i)}$,
  $[\bar{C}_{by}] \leftarrow \sum_{i=1}^t [\rho]^{i-1} C_{by(i)}$;
$\mathcal{P}$ computes $\bar{\boldsymbol{c}}, \bar{\tilde{d}}, \bar{\tilde{c}}, \bar{\boldsymbol{a}}, \bar{\tilde{a}}_y, \bar{\tilde{a}}, \bar{\boldsymbol{b}}; \bar{\tilde{b}}_y, \bar{\tilde{b}}$;
// $\mathcal{P}$ and $\mathcal{V}$ run homomorphic sub-protocols on batched inputs
$\mathcal{P}$ and $\mathcal{V}$ run zk semi-inner-product argument on batched inputs:
  $([\bar{C}_d], [\bar{C}_c], [\vec{\boldsymbol{y}}]; [U'], [\vec{\boldsymbol{U}}], [G_{11}] : \bar{\boldsymbol{c}}; \bar{\tilde{d}}, \bar{\tilde{c}})$ ;
$\mathcal{P}$ and $\mathcal{V}$ run zk high-dimensional semi-inner-product argument on batched
  inputs: $([\bar{C}_{ay}], [\bar{C}_a], [\vec{\boldsymbol{y}}_L]; [\vec{\boldsymbol{G}}'] \; [\bar{\boldsymbol{G}}] : \bar{\boldsymbol{a}}; \bar{\tilde{a}}_y, \bar{\tilde{a}})$ ;
$\mathcal{P}$ and $\mathcal{V}$ run zk high-dimensional semi-inner-product argument on batched
  inputs: $([\bar{C}_{by}], [\bar{C}_b], [\vec{\boldsymbol{y}}_R]; [\vec{\boldsymbol{H}}'], [\boldsymbol{H}] : \bar{\boldsymbol{b}}; \bar{\tilde{b}}_y, \bar{\tilde{b}})$ ;
// $\mathcal{P}$ and $\mathcal{V}$ run $t$ Bulletproofs in parallel
foreach $i \in \mathsf{range}(1, t)$ do
    $\mathcal{P}$ and $\mathcal{V}$ run zk Bulletproof on input: $([C_{d(i)}] \oplus [C_{ay(i)}] \oplus [C_{by(i)}]; [U'],$
      $[\vec{\boldsymbol{G}}'], [\vec{\boldsymbol{H}}'] : \vec{\boldsymbol{a}}_{y(i)}, \vec{\boldsymbol{b}}_{y(i)}; \bar{\tilde{a}}_{y(i)}, \bar{\tilde{b}}_{y(i)})$ ;
If all return TRUE, then return  TRUE; otherwise return FALSE
```

7 Theoretical Performance

The theoretical performance of the protocols discussed in this paper is detailed in Table 2. When estimating this table, we use the same random challenges for concurrent Bulletproof protocols. An inverse operation in $\mathbb{Z}_p$ takes $\log_2 p$ field multiplications. When using the BLS12-381 curve, the size of elements in $\mathbb{Z}_p$ and $\mathbb{G}_1$ are 256bits and 382 bits, respectively. We tested the performance of the Rust BLS12_381 library[5] on a machine equipped with Intel Core i7 and 64G RAM. The execution times for a single exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$ were 0.355ms and 1.123ms, respectively. A doubling operation and a group addition in $\mathbb{G}_1$ took 0.520μs and 0.829μs, respectively. A pairing computation took 1.400ms. A field multiplication took 0.031μs. Based on these metrics and assuming parallelization across 16 threads, we estimated the performance of zkMatrix in Table 3.

Without using pairings for faster verification, we may also implement zkMatrix using curve25519[6]. A single exponentiation on curve25519 took 24.664μs. A single group addition took 0.179μs. Pre-computation for a fixed-base multi-exponentiation of length $1,024$ took 6.527ms; each subsequent fixed-base multi-exponentiation took 1.614ms. A field multiplication took 0.082μs. Given this, the estimated runtimes for the prover and the verifier for a single multiplication of $1,024 \times 1,024$ matrices are 11.37s and 1.64s, respectively. For each additional matrix multiplication in batched zkMatrix, the incremental runtime is 0.44s for the prover and 0.82ms for the verifier.

[5] `https://github.com/zkcrypto/bls12_381`https://github.com/zkcrypto/bls12_381

[6] `https://github.com/dalek-cryptography/curve25519-dalek`https://github.com/dalek-cryptography/curve25519-dalek

Table 2: Theoretical performance for $n \times n$ matrix multiplication

	Transcript Size	RAM Usage		Timing	
Protocol		Prover	Verifier	Prover	Verifier
Single zkMatrix	$(14\log_2 n + 15)\mathbb{G}_1$ $+14\mathbb{Z}_p$	$n^2\mathbb{G}_1$ $+4n^2\mathbb{Z}_p$ $+O(n)$MISC	$O(\log n)$MISC	7 F-ME$_1(n^2)$ $+(n+5)$ F-ME$_1(2n)$ $+(5n^2+6n$ $+2\log_2 n + 38)\mathbb{E}_1$ $+(17n^2+O(n))\mathbb{M}$ $+(3\log_2 n + 9)\mathbb{H}$	$(14\log_2 n + 38)\mathbb{E}_1$ $+(3\log_2 n \cdot \log_2 p$ $+10\log_2 n$ $+O(1))\mathbb{M}$ $+1\mathbb{E}_2 + 4\mathbb{P}$ $+(3\log_2 n + 9)\mathbb{H}$
Batched zkMatrix for t matrix multiplications	$(2t\log_2 n$ $+12\log_2 n$ $+7t+8)\mathbb{G}_1$ $+(5t+9)\mathbb{Z}_p$	$n^2\mathbb{G}_1$ $+4n^2\mathbb{Z}_p$ $+4nt\mathbb{Z}_p$ $+O(n)$MISC $+O(t\log n)$MISC	$O(t\log n)$MISC	7 F-ME$_1(n^2)$ $+(n+3t+2)$ F-ME$_1(2n)$ $+(5n^2+2tn+2t\log_2 n$ $+4n+16t+O(1))\mathbb{E}_1$ $+(5tn^2+12n^2+14tn$ $+O(t)+O(n))\mathbb{M}$ $+(3\log_2 n + 10)\mathbb{H}$	$((2t+12)\log_2 n$ $+13t+25)\mathbb{E}_1$ $+((2t+8)\log_2 n$ $+3\log_2 n \cdot \log_2 p$ $+5t+O(1))\mathbb{M}$ $+1\mathbb{E}_2 + 4\mathbb{P}$ $+(3\log_2 n + 10)\mathbb{H}$

Note: $\mathbb{G}_1$ and $\mathbb{Z}_p$ denote the numbers of elements in $\mathbb{G}_1$ and $\mathbb{Z}_p$, respectively. $\mathbb{M}$, $\mathbb{E}_1$, and $\mathbb{E}_2$ represent the numbers of field multiplications and exponentiations in each respective group. $\mathbb{P}$ and $\mathbb{H}$ represent the numbers of pairings and hashing operations. MISC can generally represent any fixed-size type. F-ME$_1(n^2)$ and F-ME$_1(2n)$ denote the Fixed-base Multi-Exponentiations (F-ME) in $\mathbb{G}_1$ of length n^2 and $2n$, respectively. Each group or field multiplication is accompanied by either one or no group or field addition, which has been omitted for simplicity.

Table 3: Estimated performance using BLS12-381 for multiplication of $1,024 \times 1,024$ matrices

Protocol	Transcript Size	RAM Usage	Prover Time	Verifier Time
Single zkMatrix	7.67KB	$176MB$	197.04s	70.15ms
Batched zkMatrix				
(total)	1.42MB	$304MB$	512.48s	12.06s
(per proof)	1.42KB	$304KB$	0.50s	11.77ms

Note: The batched proof handles $1,024$ instances of multiplications of $1,024 \times 1,024$ matrices. We assume parallelization of the multi-exponentiations on an 8-core machine with 16 threads.

References

[Agrawal and Srikant(2000)] Rakesh Agrawal and Ramakrishnan Srikant. 2000. Privacy-Preserving Data Mining. In *SIGMOD 2000* (Dallas, Texas, USA) *(Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data)*. ACM, 439–450.

[Attema et al.(2021)] Thomas Attema, Ronald Cramer, and Lisa Kohl. 2021. A compressed Σ-protocol theory for lattices. In *CRYPTO 2021 (Lecture Notes in Computer Science, Vol. 12826)*, Tal Malkin and Chris Peikert (Eds.). Springer, 549–579.

[Attema et al.(2022)] Thomas Attema, Serge Fehr, and Michael Klooß. 2022. Fiat-shamir transformation of multi-round interactive proofs. In *TCC 2022 (Lecture Notes in Computer Science, Vol. 13747)*, Eike Kiltz and Vinod Vaikuntanathan (Eds.). Springer, 113–142.

[Bowe et al.(2019)] Sean Bowe, Jack Grigg, and Daira Hopwood. 2019. Halo: Recursive proof composition without a trusted setup. *Cryptology ePrint Archive* (2019), 1021. https://eprint.iacr.org/2019/1021

[Bünz et al.(2018)] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *S&P 2018 (Proceedings of 2018 IEEE symposium on security and privacy)*. IEEE, 315–334.

[Bünz et al.(2020)] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. 2020. Transparent SNARKs from DARK compilers. In *EUROCRYPT 2020 (Lecture Notes in Computer Science, Vol. 12105)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, 677–706.

[Campanelli et al.(2019)] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *CCS 2019 (Proceedings of the ACM SIGSAC Conference on Computer and Communications Security)*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 2075–2092.

[Chen et al.(2023)] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. 2023. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *EUROCRYPT 2023 (Lecture Notes in Computer Science, Vol. 14005)*, Carmit Hazay and Martijn Stam (Eds.). Springer, 499–530.

[Chiesa et al.(2020a)] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. 2020a. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *EUROCRYPT 2020 (Lecture Notes in Computer Science, Vol. 12105)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, 738–768.

[Chiesa et al.(2020b)] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. 2020b. Fractal: Post-quantum and transparent recursive proofs from holography. In *EUROCRYPT 2020 (Lecture Notes in Computer Science, Vol. 12105)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, 769–793.

[Duan and Canny([n. d.])] Yitao Duan and John Canny. [n. d.]. Practical Private Computation and Zero-Knowledge Tools for Privacy-Preserving Distributed Data Mining. In *SDM 2008 (Proceedings of the SIAM International Conference on Data Mining)*. SIAM, 265–276.

[Gabizon et al.(2019)] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. 2019. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive* (2019), 953. https://eprint.iacr.org/2019/953

[Gentry et al.(2022)] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. 2022. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In *EUROCRYPT 2022 (Lecture Notes in Computer Science, Vol. 13275)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, 458–487.

[Groth(2004)] Jens Groth. 2004. *Honest verifier zero-knowledge arguments applied.* Ph. D. Dissertation. BRICS.

[Groth(2009)] Jens Groth. 2009. Linear algebra with sub-linear zero-knowledge arguments. In *CRYPTO 2009 (Lecture Notes in Computer Science, Vol. 5677)*, Shai Halevi (Ed.). Springer, 192–208.

[Groth(2010)] Jens Groth. 2010. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT 2010 (Lecture Notes in Computer Science, Vol. 6477)*, Masayuki Abe (Ed.). Springer, 321–340.

[Groth(2016)] Jens Groth. 2016. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016 (Lecture Notes in Computer Science, Vol. 9666)*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer, 305–326.

[Kate et al.(2010)] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *ASIACRYPT 2010 (Lecture Notes in Computer Science, Vol. 6477)*, Masayuki Abe (Ed.). Springer, 177–194.

[Liu et al.(2021)] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *CCS 2021 (Proceedings of the ACM SIGSAC Conference on Computer and Communications Security)*, Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi (Eds.). ACM, 2968–2985.

[Maller et al.(2019)] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. 2019. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In *CCS 2019 (Proceedings of the ACM SIGSAC Conference on Computer and Communications Security)*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 2111–2128.

[Parno et al.(2016)] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2016. Pinocchio: Nearly practical verifiable computation. *Commun. ACM* 59, 2 (2016), 103–112.

[Thaler(2013)] Justin Thaler. 2013. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO 2013 (Lecture Notes in Computer Science, Vol. 8043)*, Ran Canetti and Juan A. Garay (Eds.). Springer, 71–89.

[Vasudevan et al.(2017)] Aravind Vasudevan, Andrew Anderson, and David Gregg. 2017. Parallel Multi Channel convolution using General Matrix Multiplication. In *ASAP 2017 (Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors)*. IEEE, 19–24. `https://doi.org/10.1109/ASAP.2017.7995254`

[Weng et al.(2021)] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. 2021. Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. In *USENIX 2021 (Proceedings of the 28th USENIX Conference on Security Symposium)*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 501–518.

[Xie et al.(2019)] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. 2019. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *CRYPTO 2019 (Lecture Notes in Computer Science, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, 733–764.

[Yang et al.(2021)] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. 2021. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *CCS 2021 (Proceedings of the ACM SIGSAC Conference on Computer and Communications Security)*, Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi (Eds.). ACM, 2986–3001.

[Yuen et al.(2021)] Tsz Hon Yuen, Muhammed F. Esgin, Joseph K. Liu, Man Ho Au, and Zhimin Ding. 2021. DualRing: Generic Construction of Ring Signatures with Efficient Instantiations. In *CRYPTO 2021 (Lecture Notes in Computer Science, Vol. 12825)*, Tal Malkin and Chris Peikert (Eds.). Springer, 251–281.

A Proofs of Theorems

A.1 Proof of Theorem 3.1

Proof (Accelerated Bulletproof). Perfect completeness. To establish perfect completeness, we demonstrate that if $\mathcal{P}$ possesses a valid witness satisfying the relation $\mathcal{R}_{\mathsf{bullet}}$ as outlined in Eq. 4, then $\mathcal{P}$ can generate a valid transcript:

$$[L_{(1)}], [R_{(1)}], \cdots, [L_{(\log_2 n)}], [R_{(\log_2 n)}], [\hat{a}], [\hat{b}], [V'], [W'],$$

such that Eq. 12 is satisfied.

Utilizing the perfect completeness of the original Bulletproof, we assert that if $\mathcal{P}$ holds a valid witness for $\mathcal{R}_{\mathsf{bullet}}$, then the $[V]$ computed by the verifier as per Eq. 11 must satisfy:

$$\begin{aligned}[V] &= [C] \oplus \bigoplus_{j=1}^{\log_2 n} ([x_{(j)}][L_{(j)}] \oplus [x_{(j)}]^{-1}[R_{(j)}]) \\ &= ([\hat{a}][\hat{b}])[U] \oplus [\hat{a}]\langle[\vec{\boldsymbol{\xi}}], [\vec{\boldsymbol{G}}]\rangle \oplus [\hat{a}]\langle[\vec{\boldsymbol{\xi}}^{-1}], [\vec{\boldsymbol{H}}]\rangle.\end{aligned}$$

Here, the coefficients on the RHS can be computed by both parties. These coefficients are collectively denoted by $[\vec{\boldsymbol{\zeta}}] \in \mathbb{Z}_p^{2n+1}$, in accordance with the notation change in Eq. 8. We assert:

$$[V] = [\zeta_1][\hat{s}\hat{G}] + [\zeta_2][\hat{s}^2\hat{G}] + \cdots + [\zeta_{2n+1}][\hat{s}^{2n+1}\hat{G}] + .. + [\zeta_q][\hat{s}^q\hat{G}],$$

where $[\zeta]^{2n+2} = [\zeta]^{2n+3} \cdots = [\zeta]^q = 0$.

Given a random challenge $[s] \in \mathbb{Z}_p^*$, the verifier can compute:

$$\phi([s]) = [\zeta_1][s] + [\zeta_2][s]^2 + \cdots + [\zeta_{2n+1}][s]^{2n+1}.$$

From the computation formula of $[V']$ and $[W']$, we deduce:

$$\begin{aligned}&e([V'], [\hat{G}]) = e([\zeta_1][\nu\hat{s}\hat{G}] + [\zeta_2][\nu\hat{s}^2\hat{G}] + \cdots + [\zeta_q][\nu\hat{s}^q\hat{G}], [\hat{G}]) \\ &= e([\zeta_1][\hat{s}\hat{G}] + [\zeta_2][\hat{s}^2\hat{G}] + \cdots + [\zeta_q][\hat{s}^q\hat{G}], [\nu\hat{G}]) = e([V], [\nu\hat{G}]), \\ &e([\hat{s}][\hat{G}] \ominus [\hat{s}\hat{G}], [W]) = e\left((\hat{s} - s)\hat{G}, \bigoplus_{i=1}^{q} \zeta_i(\frac{s^i - \hat{s}^i}{s - \hat{s}})\hat{G}\right)\end{aligned}$$

$$
\begin{aligned}
&= e\left(\bigoplus\nolimits_{i=1}^{q} \zeta_i(s^i - \hat{s}^i)\hat{G}, \hat{G}\right) = e\left((\sum_{i=1}^{q} \zeta_i s^i)\hat{G} - (\bigoplus\nolimits_{i=1}^{q} \zeta_i \hat{s}^i \hat{G}), \hat{G}\right) \\
&= e(\phi([s])[\hat{G}] \ominus [V], [\hat{G}]).
\end{aligned}
$$

As a consequence, the perfect completeness property of the accelerated Bulletproof protocol is established.

Computational knowledge soundness. To establish computational knowledge soundness (implied by witness-extended emulation [Groth(2004)] or special soundness [Attema et al.(2021)]), we aim to demonstrate that if an adversarial prover $\mathcal{A}$ can generate transcript elements $[V], [V'], [W]$ that satisfy Eq.12, then we show how to construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ and a q-PKE assumption knowledge extractor to break either the q-PDH assumption, the q-BSDH assumption or the witness-extended emulation of the (original) Bulletproof with high probability.

The Setup algorithm picks a random $\nu \in \mathbb{Z}_p$ and uses the q-PDH problem instance $\hat{G}, \hat{s}\hat{G}, \ldots, \hat{s}^q\hat{G}$, to generate the SRS for the original Bulletproof. $\mathcal{B}$ receives the Bulletproof challenges from the challenger of the original Bulletproof.

Suppose that $\mathcal{A}$ wants to prove a relation with size $2n + 1 \leq q$. In the accelerated Bulletproof, $\mathcal{B}$ picks a random challenge $[s] \overset{\$}{\leftarrow} \mathbb{Z}_p^*$. $\mathcal{A}$ generates valid transcript elements $[V], [V'], [W]$. From the first pairing equation of Eq.12, we have $V' = \nu V$. By using the q-PKE assumption knowledge extractor $\mathcal{K}_{\mathcal{A}}$, with probability no less than $1 - \mathsf{negl}_{\mathsf{PKE}}$, $\mathcal{K}_{\mathcal{A}}$ can extract $\zeta'_0, \cdots, \zeta'_q$ such that:

$$V = (\zeta'_0 + \zeta'_1\hat{s} + \zeta'_2\hat{s}^2 + \cdots + \zeta'_q\hat{s}^q)\hat{G}.$$

If:

$$(\zeta'_0, \zeta'_1, \zeta'_2, \cdots, \zeta'_q) = (0, \zeta_1, \zeta_2, \cdots, \zeta_{2n+1}, 0, \ldots, 0), \tag{26}$$

then V is equal to the right-hand side of Eq.6 of the original Bulletproof. Hence, $\mathcal{B}$ can use them to break the witness-extended emulation of the original Bulletproof.

If Eq.26 does not hold, we define a bivariate polynomial $g : \mathbb{Z}_p \times \mathbb{Z}_p \mapsto \mathbb{Z}_p$ as:

$$
\begin{aligned}
g(s, \hat{s}) :=&(\zeta_1 s + \zeta_2 s^2 + \cdots + \zeta_{2n+1} s^{2n+1}) \\
&- (\zeta'_0 + \zeta'_1\hat{s} + \zeta'_2\hat{s}^2 + \cdots + \zeta'_q\hat{s}^q) \neq 0.
\end{aligned}
\tag{27}
$$

First, if $(s - \hat{s})$ (considered as a bivariate function) divides $g(s, \hat{s})$, then, it also divides:

$$
\begin{aligned}
g'(s, \hat{s}) :=& -\zeta'_0 + (\zeta_1 - \zeta'_1)\hat{s} + \cdots + (\zeta_{2n+1} - \zeta'_{2n+1})\hat{s}^{2n+1} \\
&- \zeta'_{2n+2}\hat{s}^{2n+2} - \ldots - \zeta'_q\hat{s}^q.
\end{aligned}
$$

In this case, when $\hat{s}$ is fixed, and both $(s - \hat{s})$ and $g'(s, \hat{s})$ are treated as univariate functions, $(s - \hat{s})$ must divide $g'(s, \hat{s})$. Considering that the degree of s in $g'(s, \hat{s})$

is 0, this division only occurs under the conditions when:

$$\begin{aligned}-\zeta'_0+(\zeta_1-\zeta'_1)\hat{s}+\cdots+(\zeta_{2n+1}-\zeta'_{2n+1})\hat{s}^{2n+1}\\-\zeta'_{2n+2}\hat{s}^{2n+2}-\ldots-\zeta'_q\hat{s}^q=0.\end{aligned}$$

We define $\zeta_0=\zeta_{2n+2}=\cdots=\zeta_q:=0$, and let $q^*\in[1,q]$ represents the biggest index for which $(\zeta_{q^*}-\zeta'_{q^*})\neq 0$, we obtain:

$$(\zeta_{q^*}-\zeta'_{q^*})\hat{s}^{q^*}=-\sum_{i=0}^{q^*-1}(\zeta_i-\zeta'_i)\hat{s}^i.$$

Consequently, by multiplying $(\zeta_{q^*}-\zeta'_{q^*})^{-1}\hat{s}^{q+1-q^*}\hat{G}$ to both sides of the equation, we have:

$$\hat{s}^{q+1}\hat{G}=\ominus(\zeta_{q^*}-\zeta'_{q^*})^{-1}\bigoplus\nolimits_{i=1}^{q^*-1}(\zeta_i-\zeta'_i)[\hat{s}^{i+q+1-q^*}\hat{G}].$$

Here, $\mathcal{B}$ solves the q-PDH problem. By the q-PDH assumption, the probability of this occurrence is no more than $\mathsf{negl}_{\mathsf{PDH}}$.

Otherwise, if $g(s,\hat{s})$ is coprime with $s-\hat{s}$, the prover can find two bivariate functions, $\alpha(s,\hat{s})$ and $\beta(s,\hat{s})$, both with degrees less than q, such that:

$$\alpha(s,\hat{s})g(s,\hat{s})+\beta(s,\hat{s})(s-\hat{s})=1.$$

Then $\mathcal{B}$ can compute $E\leftarrow\alpha(s,\hat{s})e(\hat{G},W)\oplus\beta(s,\hat{s})e(\hat{G},\hat{G})$ by using s, the coefficients of $\alpha(s,\hat{s})$, $\beta(s,\hat{s})$, and $[\hat{G}],[\hat{s}\hat{G}],\ldots,[\hat{s}^q\hat{G}]$. According to the second pairing equation in Eq. 12, E satisfies:

$$\begin{aligned}&(s-\hat{s})E\\=&(s-\hat{s})\{\alpha(s,\hat{s})e(\hat{G},W)\oplus\beta(s,\hat{s})e(\hat{G},\hat{G})\}\\=&\{\alpha(s,\hat{s})e((s-\hat{s})\hat{G},W)\}\oplus\{\beta(s,\hat{s})(s-\hat{s})e(\hat{G},\hat{G})\}\\=&\{\alpha(s,\hat{s})e(\phi(s)\hat{G}-V,\hat{G})\}\oplus\{\beta(s,\hat{s})(s-\hat{s})e(\hat{G},\hat{G})\}&&\text{by Eq. 12}\\=&\{\alpha(s,\hat{s})g(s,\hat{s})e(\hat{G},\hat{G})\}\oplus\{\beta(s,\hat{s})(s-\hat{s})e(\hat{G},\hat{G})\}&&\text{by Eq.27}\\=&e(\hat{G},\hat{G}).\end{aligned}$$

As a result, $\mathcal{B}$ obtains a solution $(\ominus E,s)$ to the q-BSDH problem. According to the q-BSDH assumption, the probability for this case is no greater than $\mathsf{negl}_{\mathsf{BSDH}}$.

Consequently, the probability that Eq.6 holds true is no less than $(1-\mathsf{negl}_{\mathsf{PKE}})(1-\mathsf{negl}_{\mathsf{PDH}}-\mathsf{negl}_{\mathsf{BSDH}})$. Suppose $\mathcal{A}$ generates an acceptable transcript with probability $p(\mathcal{A})$ and the knowledge error of the original Bulletproof protocol is κ_{bullet}. Then, $\mathcal{B}$ can output a valid witness with probability:

$$\begin{aligned}&(p(\mathcal{A})(1-\mathsf{negl}_{\mathsf{PKE}})(1-\mathsf{negl}_{\mathsf{PDH}}-\mathsf{negl}_{\mathsf{BSDH}})-\kappa_{\mathsf{bullet}})/\mathsf{poly}(n)\\\geq&(p(\mathcal{A})-\mathsf{negl}_{\mathsf{PKE}}-\mathsf{negl}_{\mathsf{PDH}}-\mathsf{negl}_{\mathsf{BSDH}}-\kappa_{\mathsf{bullet}})/\mathsf{poly}(n)\end{aligned}$$

As a result, we obtain the computational knowledge soundness of the accelerated Bulletproof protocol.

Variation. We can also accelerate Bulletproof using asymmetric pairings. Under the assumptions of the asymmetric versions of q-PKE and q-BSDH, the pairing equations in Eq. 12 evolve to:

$$\begin{aligned}&e([V],[\nu\hat{H}])\overset{?}{=}e([V'],[\hat{H}]),\\&e(\phi([s])[\hat{G}]\ominus[V],[\hat{H}])\overset{?}{=}e([W],[s][\hat{H}]\ominus[\hat{s}\hat{H}]).\end{aligned}$$

Here, $[\hat{H}]\in\mathbb{G}_2$ is a generator of $\mathbb{G}_2$. $[\hat{H}],[\nu\hat{H}]$, and $[\hat{s}\hat{H}]$ are also included into the SRS.

The demonstration of computational knowledge soundness for the accelerated Bulletproof using asymmetric pairings is in essence identical to that using symmetric pairing. Further details are omitted here due to page limitations.

A.2 Proof of Theorem 4.1

Proof (Semi-Inner-Product). Perfect completeness. If $\mathcal{P}$ possesses a valid witness for $\mathcal{R}_{\mathsf{sip}}$, then it also holds a valid witness for $\mathcal{R}_{\mathsf{bullet}}$ under the following notation substitution:

$$\begin{aligned}&U\leftarrow xU,\quad \vec{\boldsymbol{G}}\leftarrow\vec{\boldsymbol{G}},\quad \vec{\boldsymbol{H}}\leftarrow\vec{\boldsymbol{O}},\\&C=(\vec{\boldsymbol{a}}\bullet\vec{\boldsymbol{b}})U\oplus\langle\vec{\boldsymbol{a}},\vec{\boldsymbol{G}}\rangle\oplus\langle\vec{\boldsymbol{b}},\vec{\boldsymbol{H}}\rangle\leftarrow(\vec{\boldsymbol{a}}\bullet\vec{\boldsymbol{b}})xU\oplus\langle\vec{\boldsymbol{a}},\vec{\boldsymbol{G}}\rangle=xC_c\oplus C_a.\end{aligned}$$

Given the perfect completeness of Bulletproof, $\mathcal{P}$ can generate a valid transcript acceptable by the verifier of Bulletproof. Consequently, the verification check in line 5 of Algorithm 2 returns true. Additionally, since:

$$[z][U]=[r+h(\vec{\boldsymbol{a}}\bullet\vec{\boldsymbol{b}})][U]=[rU]\oplus[h][(\vec{\boldsymbol{a}}\bullet\vec{\boldsymbol{b}})U]=[R]\oplus[h][C_c],$$

we know that the verification check in line 4 of Algorithm 2 also returns true. Consequently, $\mathcal{P}$ generates a valid transcript that is accepted by the verifier of the semi-inner-product argument.

Computational knowledge soundness. To prove the witness-extended emulation (this implies computational knowledge soundness as per [Groth(2004)]), we first notice that lines 1 to 4 are the argument of knowledge of the discrete logarithm of C_c with respect to the base U. We can construct an extractor $\mathcal{K}_{\mathsf{DL}}$ to output c such that $C_c=cU$ if the discrete logarithm assumption holds.

Next, we notice that in line 5 we run Bulletproof with $\vec{\boldsymbol{H}}=\vec{\boldsymbol{O}}$. This special case of Bulletproof is actually the same as the algorithm 10 of the full version of [Yuen et al.(2021)]. Yuen et al. showed that there exists an efficient extractor $\mathcal{K}_{\mathsf{semi}}$ that can output either a discrete logarithm relation among U, $\vec{\boldsymbol{G}}$, or a witness $\vec{\boldsymbol{a}}$ such that:

$$C=\langle\vec{\boldsymbol{a}},\vec{\boldsymbol{G}}\rangle\oplus(\vec{\boldsymbol{a}}\bullet\vec{\boldsymbol{b}})xU.$$

(It is relation 6 of the full version of [Yuen et al.(2021)]). Since $C=xC_c\oplus C_a=xcU\oplus C_a$, we have:

$$xcU\oplus C_a=\langle\vec{\boldsymbol{a}},\vec{\boldsymbol{G}}\rangle\oplus(\vec{\boldsymbol{a}}\bullet\vec{\boldsymbol{b}})xU. \tag{28}$$

For our protocol, we construct an extractor $\mathcal{K}$ which receives a witness c using $\mathcal{K}_{\mathsf{DL}}$, and receives witnesses $\vec{\boldsymbol{a}}^{(1)}$, $\vec{\boldsymbol{a}}^{(2)}$ using $\mathcal{K}_{\mathsf{semi}}$ twice with different challenges $x^{(1)}, x^{(2)}$. By Eq. 28, we have:

$$(x^{(1)} - x^{(2)})cU = \langle \vec{\boldsymbol{a}}^{(1)} - \vec{\boldsymbol{a}}^{(2)}, \vec{\boldsymbol{G}} \rangle \oplus ((x^{(1)}\vec{\boldsymbol{a}}^{(1)} - x^{(2)}\vec{\boldsymbol{a}}^{(2)}) \bullet \vec{\boldsymbol{b}})U.$$

Assuming that we cannot find a non-trivial discrete logarithm relation among U, $\vec{\boldsymbol{G}}$, we have $\vec{\boldsymbol{a}}^{(1)} = \vec{\boldsymbol{a}}^{(2)}$, and hence $c = (\vec{\boldsymbol{a}}^{(1)} \bullet \vec{\boldsymbol{b}})$.

By Eq. 28, we also have:

$$\begin{aligned}(x^{(1)} - x^{(2)})C_a =& \langle x^{(1)}\vec{\boldsymbol{a}}^{(2)} - x^{(2)}\vec{\boldsymbol{a}}^{(1)}, \vec{\boldsymbol{G}} \rangle \\ & \oplus ((\vec{\boldsymbol{a}}^{(1)} - \vec{\boldsymbol{a}}^{(2)}) \bullet \vec{\boldsymbol{b}}) x^{(1)} x^{(2)} U \\ =& \langle x^{(1)}\vec{\boldsymbol{a}}^{(1)} - x^{(2)}\vec{\boldsymbol{a}}^{(2)}, \vec{\boldsymbol{G}} \rangle.\end{aligned}$$

Hence we have $C_a = \langle \vec{\boldsymbol{a}}^{(1)}, \vec{\boldsymbol{G}} \rangle$ and $C_c = cU = (\vec{\boldsymbol{a}}^{(1)} \bullet \vec{\boldsymbol{b}})U$. As a result, we conclude that our protocol has witness-extended emulation.

For the relaxed version of the semi-inner-product argument, the subsequent relation remains valid for each $x^{(\tau)}$ with $\tau = 1, 2$:

$$x^{(\tau)} C_c \oplus C_a = \langle \vec{\boldsymbol{a}}^{(\tau)}, \vec{\boldsymbol{G}} \rangle \oplus (\vec{\boldsymbol{a}}^{(\tau)} \bullet \vec{\boldsymbol{b}}) x^{(\tau)} U, \quad \tau = 1, 2. \tag{29}$$

We can compute:

$$\begin{aligned} a_U &:= \frac{x^{(1)}x^{(2)}(\vec{\boldsymbol{a}}^{(2)} - \vec{\boldsymbol{a}}^{(1)}) \bullet \vec{\boldsymbol{b}}}{x^{(1)} - x^{(2)}}, \quad & \vec{\boldsymbol{a}}_G &:= \frac{x^{(1)}\vec{\boldsymbol{a}}^{(2)} - x^{(2)}\vec{\boldsymbol{a}}^{(1)}}{x^{(1)} - x^{(2)}}, \\ c_U &:= \frac{(x^{(1)}\vec{\boldsymbol{a}}^{(1)} - x^{(2)}\vec{\boldsymbol{a}}^{(2)}) \bullet \vec{\boldsymbol{b}}}{x^{(1)} - x^{(2)}}, \quad & \vec{\boldsymbol{c}}_G &:= \frac{\vec{\boldsymbol{a}}^{(1)} - \vec{\boldsymbol{a}}^{(2)}}{x^{(1)} - x^{(2)}}. \end{aligned}$$

Then, we have:

$$C_a = a_U U \oplus \langle \vec{\boldsymbol{a}}_G, \vec{\boldsymbol{G}} \rangle, \quad C_c = c_U U \oplus \langle \vec{\boldsymbol{c}}_G, \vec{\boldsymbol{G}} \rangle. \tag{30}$$

By comparing Eq.29 and Eq.30, if the pairwise discrete logarithm between U and elements in $\vec{\boldsymbol{G}}$ is difficult, we obtain:

$$(x^{(\tau)})(\vec{\boldsymbol{a}}^{(\tau)} \bullet \vec{\boldsymbol{b}}) = (x^{(\tau)})c_U + a_U, \qquad \vec{\boldsymbol{a}}^{(\tau)} = (x^{(\tau)})\vec{\boldsymbol{c}}_G + \vec{\boldsymbol{a}}_G.$$

Putting them together, we have:

$$(x^{(\tau)})^2 \vec{\boldsymbol{c}}_G \bullet \vec{\boldsymbol{b}} + (x^{(\tau)}) \vec{\boldsymbol{a}}_G \bullet \vec{\boldsymbol{b}} = (x^{(\tau)}) c_U + a_U.$$

By comparing the coefficients corresponding to different degrees of $x^{(\tau)}$, we deduce:

$$a_U = 0, \qquad \vec{\boldsymbol{c}}_G \bullet \vec{\boldsymbol{b}} = 0, \qquad \vec{\boldsymbol{a}}_G \bullet \vec{\boldsymbol{b}} = c_U.$$

By the definition of a_U, and $x^{(1)} \neq 0, x^{(2)} \neq 0$, we have:

$$\frac{x^{(1)}x^{(2)}((\vec{\boldsymbol{a}}^{(2)} - \vec{\boldsymbol{a}}^{(1)}) \bullet \vec{\boldsymbol{b}})}{x^{(1)} - x^{(2)}} = 0.$$

Hence, we have $\vec{\boldsymbol{a}}^{(1)} \bullet \vec{\boldsymbol{b}} = \vec{\boldsymbol{a}}^{(2)} \bullet \vec{\boldsymbol{b}}$. Putting it back to the definition of c_U, we have:

$$c_U = \vec{\boldsymbol{a}}^{(1)} \bullet \vec{\boldsymbol{b}}.$$

Let $\vec{\boldsymbol{a}} = \vec{\boldsymbol{a}}_G$ and $\vec{\boldsymbol{c'}} = \vec{\boldsymbol{c}}_G$. Then, we obtain a valid witness for the relaxed version of $\mathcal{R}_{\mathsf{sip}}$.

A.3 Proof of Theorem 4.2

Proof (High-dimensional semi-inner-product).

Perfect completeness. If $\mathcal{P}$ possesses a valid witness $\vec{\boldsymbol{a}}_1, \cdots, \vec{\boldsymbol{a}}_n \in \mathbb{Z}_p^m$ for $\mathcal{R}_{\mathsf{hd-sip}}$, then it also holds a valid witness $(\vec{\boldsymbol{a}}_1 || \cdots || \vec{\boldsymbol{a}}_n)$ for $\mathcal{R}_{\mathsf{sip}}$ under the following notation substitution:

$$\begin{aligned}
&U \leftarrow U_1, \quad \vec{\boldsymbol{G}} \leftarrow (\vec{\boldsymbol{G}}_1 \oplus x b_1 \vec{\boldsymbol{U}} || \cdots || \vec{\boldsymbol{G}}_n \oplus x b_n \vec{\boldsymbol{U}}), \quad \vec{\boldsymbol{b}} \leftarrow \vec{\boldsymbol{0}},\\
&C = (\vec{\boldsymbol{a}} \bullet \vec{\boldsymbol{b}}) U \oplus \langle \vec{\boldsymbol{a}}, \vec{\boldsymbol{G}} \rangle\\
&\quad \leftarrow (\vec{\boldsymbol{a}} \bullet \vec{\boldsymbol{0}}) U_1 \oplus \langle (\vec{\boldsymbol{a}}_1 || \cdots || \vec{\boldsymbol{a}}_n), (\vec{\boldsymbol{G}}_1 \oplus x b_1 \vec{\boldsymbol{U}} || \cdots || \vec{\boldsymbol{G}}_n \oplus x b_n \vec{\boldsymbol{U}}) \rangle\\
&\quad = O \oplus \{ \bigoplus\nolimits_{i=1}^{n} \langle \vec{\boldsymbol{a}}_i, (\vec{\boldsymbol{G}}_i \oplus x b_i \vec{\boldsymbol{U}} \rangle \}\\
&\quad = \{ \bigoplus\nolimits_{i=1}^{n} \langle \vec{\boldsymbol{a}}_i, \vec{\boldsymbol{G}}_i \rangle \} \oplus x \langle \Sigma_{i=1}^{n} \vec{\boldsymbol{a}}_i b_i, \vec{\boldsymbol{U}} \rangle)\} = x C_c \oplus C_a.
\end{aligned}$$

Given the perfect completeness of Bulletproof, $\mathcal{P}$ can generate a valid transcript acceptable by the verifier. Consequently, the verification check in line 6 of Algorithm 2 returns true.

Similarly, $\mathcal{P}$ can also compute a valid witness $\sum_{i=1}^{n} \vec{\boldsymbol{a}}_i b_i \in \mathbb{Z}_p^m$ for $\mathcal{R}_{\mathsf{sip}}$ under the following notation substitution:

$$\begin{aligned}
&U \leftarrow G_{11}, \quad \vec{\boldsymbol{G}} \leftarrow \vec{\boldsymbol{U}}, \quad \vec{\boldsymbol{b}} \leftarrow \vec{\boldsymbol{0}},\\
&C = (\vec{\boldsymbol{a}} \bullet \vec{\boldsymbol{b}}) U \oplus \langle \vec{\boldsymbol{a}}, \vec{\boldsymbol{G}} \rangle \leftarrow (\vec{\boldsymbol{a}} \bullet \vec{\boldsymbol{0}}) G_{11} \oplus \langle \Sigma_{i=1}^{n} \vec{\boldsymbol{a}}_i b_i, \vec{\boldsymbol{U}} \rangle = O \oplus C_c = C_c.
\end{aligned}$$

Hence, the verification check in line 5 of Algorithm 2 also returns true. As a result, $\mathcal{P}$ generates a valid transcript acceptable by the verifier of the high-dimensional semi-inner-product argument.

Computational knowledge soundness. To demonstrate computational knowledge soundness, let $\mathcal{K}_{\mathsf{com}}$ and $\mathcal{K}_{\mathsf{sip}}$ be the extractor for the relation $\mathcal{R}_{\mathsf{com}}$ and $\mathcal{R}_{\mathsf{sip}}$ in line 5 and line 6 respectively. By $\mathcal{K}_{\mathsf{com}}$, it outputs $\vec{\boldsymbol{c}}$ such that $C_c = \langle \vec{\boldsymbol{c}}, \vec{\boldsymbol{U}} \rangle$.

Then, for challenges $x^{(\tau)}, \tau = 1, 2$ in the high-dimensional semi-inner-product argument protocol, $\mathcal{K}_{\mathsf{sip}}$ can extract, with very high probability, $a_{ij}^{(\tau)}, i \in \mathsf{range}(1, n), j \in \mathsf{range}(1, m)$ such that:

$$\begin{aligned}
C_a \oplus \{(x^{(\tau)}) C_c\} &= \bigoplus\nolimits_{i=1}^{n} \bigoplus\nolimits_{j=1}^{m} \{ a_{ij}^{(\tau)} (G_{ij} \oplus x^{(\tau)} b_i U_j) \}\\
=&\{ \bigoplus\nolimits_{i=1}^{n} \bigoplus\nolimits_{j=1}^{m} a_{ij}^{(\tau)} G_{ij} \} \oplus \{ x^{(\tau)} \bigoplus\nolimits_{j=1}^{m} (\sum_{i=1}^{n} a_{ij}^{(\tau)} b_i) U_j \}.
\end{aligned} \tag{31}$$

By Eq. 31, we have:

$$(x^{(1)} - x^{(2)})C_c = \{\bigoplus_{i=1}^{n}\bigoplus_{j=1}^{m}(a_{ij}^{(1)} - a_{ij}^{(2)})G_{ij}\} \oplus \{\bigoplus_{j=1}^{m}(\sum_{i=1}^{n}(x^{(1)}a_{ij}^{(1)} - x^{(2)}a_{ij}^{(2)})b_i)U_j\}.$$

Suppose that the pairwise discrete logarithms between elements in $\vec{\boldsymbol{U}}$ and all G_{ij} are difficult. By $\mathcal{K}_{\mathsf{com}}$, we have $a_{ij}^{(1)} = a_{ij}^{(2)}$, $\forall i, j$ and:

$$C_c = \langle \vec{\boldsymbol{c}}, \vec{\boldsymbol{U}} \rangle = \bigoplus_{j=1}^{m}(\sum_{i=1}^{n} a_{ij}^{(1)} b_i)U_j := \langle \sum_{i=1}^{n} \vec{\boldsymbol{a}}_i^{(1)} b_i, \vec{\boldsymbol{U}} \rangle.$$

By Eq. 31 and $a_{ij}^{(1)} = a_{ij}^{(2)}$, we also have:

$$\begin{aligned}(x^{(1)} - x^{(2)})C_a =& \{\bigoplus_{i=1}^{n}\bigoplus_{j=1}^{m}(x^{(1)}a_{ij}^{(2)} - x^{(2)}a_{ij}^{(1)})G_{ij}\} \\ & \oplus \{x^{(1)}x^{(2)}\bigoplus_{j=1}^{m}(\sum_{i=1}^{n}(a_{ij}^{(2)} - a_{ij}^{(1)})b_i)U_j\} \\ =& \bigoplus_{i=1}^{n}\bigoplus_{j=1}^{m}(x^{(1)} - x^{(2)})a_{ij}^{(1)}G_{ij}.\end{aligned}$$

Hence we obtain:

$$C_a = \bigoplus_{i=1}^{n}\bigoplus_{j=1}^{m} a_{ij}^{(1)} G_{ij} = \bigoplus_{i=1}^{n} \langle \vec{\boldsymbol{a}}_i^{(1)}, \vec{\boldsymbol{G}}_i \rangle.$$

Consequently, $\vec{\boldsymbol{a}}_1^{(1)}, \cdots, \vec{\boldsymbol{a}}_n^{(1)}$ form a valid witness for $\mathcal{R}_{\mathsf{hd-sip}}$.

Utilizing reasoning analogous to what was presented in the proof of Theorem 4.1, we can demonstrate that the relaxed version of the high-dimensional semi-inner-product protocol serves as an argument of knowledge for the relaxed version of $\mathcal{R}_{\mathsf{hd-sip}}$.

A.4 Proof of Theorem 5.1

Proof (MatMul Protocol). Perfect completeness. Building on the equivalence established in Eq. 20, if $\mathcal{P}$ possesses a valid witness for $\mathcal{R}_{\mathsf{comMatMul}}$, then for any $y \in \mathbb{Z}_p$, $\mathcal{P}$ is capable of generating witnesses for the four inner products ①, ②, ③, and ④ in Eq. 3. Given the perfect completeness of Bulletproof, the semi-inner-product argument, and the high-dimensional semi-inner-product argument, it follows that $\mathcal{P}$ can produce transcripts that are acceptable to the verifiers of the corresponding sub-protocols.

Computational knowledge soundness. To demonstrate computational knowledge soundness, we rewind the MatMul protocol mn times and provide it with mn pairwise distinct challenges $y^{(\tau)}, \tau \in \mathsf{range}(1, mn)$. Utilizing Theorem 2 and Theorem 3, we employ the extractors $\mathcal{K}_{\mathsf{sip}}$ and $\mathcal{K}_{\mathsf{hd-sip}}$ to obtain $\boldsymbol{a}^{(\tau)}, \boldsymbol{b}^{(\tau)}, \boldsymbol{c}^{(\tau)}$, $d^{(\tau)}$, $C_d^{(\tau)}$, $C_{ay}^{(\tau)}, C_{by}^{(\tau)}$, $\vec{\boldsymbol{a}}_y^{(\tau)}, \vec{\boldsymbol{b}}_y^{(\tau)}$ as follows.

First, we employ the extractor $\mathcal{K}_{\mathsf{sip}}$ to obtain $\vec{\boldsymbol{c}}^{(\tau)}$ and $\vec{\boldsymbol{c}}^{'(\tau)}_{cy}$ such that:

$$C_c = \langle \vec{\boldsymbol{c}}^{(\tau)}, \vec{\boldsymbol{U}} \rangle, \quad C_d^{(\tau)} = (\vec{\boldsymbol{c}}^{(\tau)} \bullet \vec{\boldsymbol{y}}^{(\tau)})U' \oplus \langle \vec{\boldsymbol{c}}^{'(\tau)}_{cy}, \vec{\boldsymbol{U}} \rangle. \tag{32}$$

Since $C_c = \langle \vec{\boldsymbol{c}}^{(1)}, \vec{\boldsymbol{U}} \rangle = \ldots = \langle \vec{\boldsymbol{c}}^{(mn)}, \vec{\boldsymbol{U}} \rangle$ and the pairwise discrete logarithm between elements in $\vec{\boldsymbol{U}}$ is unknown, we have $\vec{\boldsymbol{c}} := \vec{\boldsymbol{c}}^{(1)} = \ldots = \vec{\boldsymbol{c}}^{(mn)}$ such that $C_c = \langle \vec{\boldsymbol{c}}, \vec{\boldsymbol{U}} \rangle$.

We use the extractor $\mathcal{K}_{\mathsf{hd-sip}}$ to obtain $\vec{\boldsymbol{a}}_i^{(\tau)}$ and $\vec{\boldsymbol{c}}^{'(\tau)}_{ay}$ such that:

$$\begin{aligned} C_a &= \bigoplus\nolimits_{i=1}^{m} \langle \vec{\boldsymbol{a}}_i^{(\tau)}, \vec{\boldsymbol{G}}_{i*}^{\top} \rangle, \\ C_{ay} &= \langle \sum_{i=1}^{m} \vec{\boldsymbol{a}}_i^{(\tau)} (y^{(\tau)})^{(i-1)n}, \vec{\boldsymbol{G}}' \rangle \oplus \langle \vec{\boldsymbol{c}}^{'(\tau)}_{ay}, \vec{\boldsymbol{G}} \rangle. \end{aligned} \tag{33}$$

Similarly, we have $\vec{\boldsymbol{a}}_i := \vec{\boldsymbol{a}}_i^{(1)} = \ldots = \vec{\boldsymbol{a}}_i^{(mn)}$ such that $C_a = \bigoplus_{i=1}^{m} \langle \vec{\boldsymbol{a}}_i, \vec{\boldsymbol{G}}_{i*}^{\top} \rangle$, if the pairwise discrete logarithm between elements in $\vec{\boldsymbol{G}}_{i*}^{\top}$ is unknown.

We use the extractor $\mathcal{K}_{\mathsf{hd-sip}}$ to obtain $\vec{\boldsymbol{b}}_j^{(\tau)}$ and $\vec{\boldsymbol{c}}^{'(\tau)}_{by}$ such that:

$$\begin{aligned} C_b &= \bigoplus\nolimits_{i=1}^{n} \langle \vec{\boldsymbol{b}}_j^{(\tau)}, \vec{\boldsymbol{H}}_{*j} \rangle, \\ C_{by} &= \langle \sum_{j=1}^{n} \vec{\boldsymbol{b}}_j^{(\tau)} (y^{(\tau)})^{j-1}, \vec{\boldsymbol{H}}' \rangle \oplus \langle \vec{\boldsymbol{c}}^{'(\tau)}_{by}, \vec{\boldsymbol{H}} \rangle. \end{aligned} \tag{34}$$

Similarly, we have $\vec{\boldsymbol{b}}_j := \vec{\boldsymbol{b}}_j^{(1)} = \ldots = \vec{\boldsymbol{b}}_j^{(mn)}$ such that $C_b = \bigoplus_{i=1}^{n} \langle \vec{\boldsymbol{b}}_j, \vec{\boldsymbol{H}}_{*j} \rangle$, if the pairwise discrete logarithm between elements in $\vec{\boldsymbol{H}}_{*j}$ is unknown.

Next, we apply the Bulletproof extractor to obtain $\vec{\boldsymbol{a}}_y^{'(\tau)}, \vec{\boldsymbol{b}}_y^{'(\tau)}$:

$$C_d^{(\tau)} \oplus C_{ay}^{(\tau)} \oplus C_{by}^{(\tau)} = (\vec{\boldsymbol{a}}_y^{'(\tau)} \bullet \vec{\boldsymbol{b}}_y^{'(\tau)})U' \oplus \langle \vec{\boldsymbol{a}}_y^{'(\tau)}, \vec{\boldsymbol{G}}' \rangle \oplus \langle \vec{\boldsymbol{b}}_y^{'(\tau)}, \vec{\boldsymbol{H}}' \rangle.$$

Assume that the pairwise discrete logarithm between elements in $U', \vec{\boldsymbol{G}}', \vec{\boldsymbol{H}}'$, $\vec{\boldsymbol{U}} \cup \vec{\boldsymbol{G}} \cup \vec{\boldsymbol{H}}$ are difficult[7]. Comparing the above equation with Eq. 32, Eq. 33 and Eq. 34, we deduce:

$$\begin{aligned} &\vec{\boldsymbol{a}}_y^{'(\tau)} = \sum_{i=1}^{m} \vec{\boldsymbol{a}}_i (y^{(\tau)})^{(i-1)n}, \quad \vec{\boldsymbol{b}}_y^{'(\tau)} = \sum_{j=1}^{n} \vec{\boldsymbol{b}}_j (y^{(\tau)})^{j-1}, \\ &\vec{\boldsymbol{a}}_y^{'(\tau)} \bullet \vec{\boldsymbol{b}}_y^{'(\tau)} = \vec{\boldsymbol{c}} \bullet \vec{\boldsymbol{y}}^{(\tau)}. \end{aligned}$$

We define $\boldsymbol{a}$ as a matrix formed by row vectors $\vec{\boldsymbol{a}}_1, \ldots, \vec{\boldsymbol{a}}_m$, $\boldsymbol{b}$ as a matrix formed by column vectors $\vec{\boldsymbol{b}}_1, \ldots, \vec{\boldsymbol{b}}_n$, $\boldsymbol{c}$ is an $m \times n$ matrix formed by the reverse operation of flat over $\vec{\boldsymbol{c}}$. By Eq. 20, the above relation is equivalent to:

[7] Here we require that elements in $U', \vec{\boldsymbol{G}}', \vec{\boldsymbol{H}}'$ has no known discrete logarithm with elements in $\vec{\boldsymbol{U}} \cup \vec{\boldsymbol{G}} \cup \vec{\boldsymbol{H}}$. It is possible to have common elements within $\vec{\boldsymbol{U}}, \vec{\boldsymbol{G}}$ and $\vec{\boldsymbol{H}}$. In fact, we use common elements within these sets for accelerating verification.

$$\vec{\boldsymbol{y}}_L^{(\tau)T}\boldsymbol{c}\vec{\boldsymbol{y}}_R^{(\tau)} = \vec{\boldsymbol{y}}_L^{(\tau)T}\boldsymbol{a}\boldsymbol{b}\vec{\boldsymbol{y}}_R^{(\tau)}\ , \quad \forall\tau\in\mathsf{range}(1,mn).$$

Consider that $\vec{\boldsymbol{y}}_L^T\boldsymbol{c}\vec{\boldsymbol{y}}_R$ is a polynomial of y with degree $mn-1$, and its coefficients are determined by $\boldsymbol{c}$. Similarly, we can define $\vec{\boldsymbol{y}}_L^T\boldsymbol{a}\boldsymbol{b}\vec{\boldsymbol{y}}_R$ as a polynomial of y with degree $mn-1$, and its coefficients are determined by $\boldsymbol{a},\boldsymbol{b}$. Since the equation above holds for mn different values of y, we conclude that $\boldsymbol{c}=\boldsymbol{a}\boldsymbol{b}$. Hence, the extracted witness $\boldsymbol{c},\boldsymbol{a},\boldsymbol{b}$ satisfies $\mathcal{R}_{\mathsf{comMatMul}}$.

A.5 Proof of Theorem 5.2

Proof (zkMatrix (Single Proof)).

Perfect completeness. Given the perfect completeness of the MatMul protocol and the equivalence established in Eq. 24, if $\mathcal{P}$ holds a valid witness $\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}$ such that $\boldsymbol{c}=\boldsymbol{a}\boldsymbol{b}$, then for any $x\in\mathbb{Z}_p$, $\mathcal{P}$ is able to produce the valid witness $x(\boldsymbol{\alpha}+x\boldsymbol{a})$, $x(\boldsymbol{\beta}+x\boldsymbol{b})$, and $(x(\boldsymbol{\alpha}+x\boldsymbol{a}))(x(\boldsymbol{\beta}+x\boldsymbol{b}))$, which will be accepted by the verifier of the MatMul protocol.

Computational knowledge soundness. To demonstrate computational knowledge soundness, we will show that by rewinding the zkMatrix protocol multiple times and applying the extractor of the MatMul protocol, we can extract a valid witness for $\mathcal{R}_{\mathsf{zkMatMul}}$.

In the zkMatrix protocol, by rewinding six times with different values of $x^{(\tau)},\tau=1,\dots,6$ and applying the extractor $\mathcal{K}_{\mathsf{comMatMul}}$, we obtain $\boldsymbol{c}^{'(\tau)},\boldsymbol{a}^{'(\tau)},\boldsymbol{b}^{'(\tau)},P_c^{(\tau)}$, $P_a^{(\tau)},P_b^{(\tau)}$, $z_c^{(\tau)},z_a^{(\tau)},z_b^{(\tau)}$ such that:

$$P_c^{(\tau)}=\langle\boldsymbol{c}^{'(\tau)},\boldsymbol{U}\rangle,\quad P_a^{(\tau)}=\langle\boldsymbol{a}^{'(\tau)},\boldsymbol{G}\rangle,\quad P_b^{(\tau)}=\langle\boldsymbol{b}^{'(\tau)},\boldsymbol{H}\rangle, \tag{35}$$

$$\boldsymbol{c}^{'(\tau)}=\boldsymbol{a}^{'(\tau)}*\boldsymbol{b}^{'(\tau)}, \tag{36}$$

$$P_c^{(\tau)}=(x^{(\tau)})^2(C'_\gamma\oplus x^{(\tau)}C'_\delta\oplus(x^{(\tau)})^2C_c\oplus z_c^{(\tau)}\tilde{G}), \tag{37}$$

$$P_a^{(\tau)}=(x^{(\tau)})(C'_\alpha\oplus x^{(\tau)}C_a\oplus z_a^{(\tau)}\tilde{G}), \tag{38}$$

$$P_b^{(\tau)}=(x^{(\tau)})(C'_\beta\oplus x^{(\tau)}C_b\oplus z_b^{(\tau)}\tilde{G}). \tag{39}$$

Drawing from Eq. 35, 37, 38, 39, we can represent $C'_\gamma,C'_\delta,C'_\alpha,C'_\beta$, C_c,C_a,C_b in the following form:

$$C'_\gamma=\langle\boldsymbol{\gamma}^*,\boldsymbol{U}\rangle\oplus\tilde{\gamma}^*\tilde{G},\ C'_\delta=\langle\boldsymbol{\delta}^*,\boldsymbol{U}\rangle\oplus\tilde{\delta}^*\tilde{G}, \tag{40}$$

$$C'_\alpha=\langle\boldsymbol{\alpha}^*,\boldsymbol{G}\rangle\oplus\tilde{\alpha}^*\tilde{G},\ C'_\beta=\langle\boldsymbol{\beta}^*,\boldsymbol{H}\rangle\oplus\tilde{\beta}^*\tilde{G}, \tag{41}$$

$$C_c=\langle\boldsymbol{c}^*,\boldsymbol{U}\rangle\oplus\tilde{c}^*\tilde{G},\ C_a=\langle\boldsymbol{a}^*,\boldsymbol{G}\rangle\oplus\tilde{a}^*\tilde{G},\ C_b=\langle\boldsymbol{b}^*,\boldsymbol{H}\rangle\oplus\tilde{b}^*\tilde{G}. \tag{42}$$

Upon comparing Eq. 40, 41, 42 with Eq. 35, 37, 38, 39, we derive either a non-trivial logarithm relation or the subsequent equations:

$$\begin{aligned}
\boldsymbol{c}'^{(\tau)} &= (x^{(\tau)})^2(\boldsymbol{\gamma}^* + x^{(\tau)}\boldsymbol{\delta}^* + (x^{(\tau)})^2\boldsymbol{c}^*), && (43)\\
z_c^{(\tau)} &= (x^{(\tau)})^2(\tilde{\gamma}^* + x^{(\tau)}\tilde{\delta}^* + (x^{(\tau)})^2\tilde{c}^*), \\
\boldsymbol{a}'^{(\tau)} &= (x^{(\tau)})(\boldsymbol{\alpha}^* + x^{(\tau)}\boldsymbol{a}^*),\ z_a^{(\tau)} = (x^{(\tau)})(\tilde{\alpha}^* + x^{(\tau)}\tilde{a}^*), && (44)\\
\boldsymbol{b}'^{(\tau)} &= (x^{(\tau)})(\boldsymbol{\beta}^* + x^{(\tau)}\boldsymbol{b}^*),\ z_b^{(\tau)} = (x^{(\tau)})(\tilde{\beta}^* + x^{(\tau)}\tilde{b}^*). && (45)
\end{aligned}$$

We have:

$$\begin{pmatrix}
\boldsymbol{c}'^{(1)} & z_c^{(1)} & \boldsymbol{a}'^{(1)} & z_a^{(1)} & \boldsymbol{b}'^{(1)} & z_b^{(1)} \\
\boldsymbol{c}'^{(2)} & z_c^{(2)} & \boldsymbol{a}'^{(2)} & z_a^{(2)} & \boldsymbol{b}'^{(2)} & z_b^{(2)} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\boldsymbol{c}'^{(6)} & z_c^{(6)} & \boldsymbol{a}'^{(6)} & z_a^{(6)} & \boldsymbol{b}'^{(6)} & z_b^{(6)}
\end{pmatrix}$$

$$= \begin{pmatrix}
1 & x^{(1)} & \dots & (x^{(1)})^5 \\
1 & x^{(2)} & \dots & (x^{(2)})^5 \\
\vdots & \vdots & \vdots & \vdots \\
1 & x^{(6)} & \dots & (x^{(6)})^5
\end{pmatrix}
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \boldsymbol{\alpha}^* & \tilde{\alpha}^* & \boldsymbol{\beta}^* & \tilde{\beta}^* \\
\boldsymbol{\gamma}^* & \tilde{\gamma}^* & \boldsymbol{a}^* & \tilde{a}^* & \boldsymbol{b}^* & \tilde{b}^* \\
\boldsymbol{\delta}^* & \tilde{\delta}^* & 0 & 0 & 0 & 0 \\
\boldsymbol{c}^* & \tilde{c}^* & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}.$$

By utilizing the inverse of the Vandermonde matrix, we can compute $\boldsymbol{\gamma}^*, \boldsymbol{\delta}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*, \tilde{c}^*, \tilde{a}^*, \tilde{b}^*, \gamma^*, \delta^*, \alpha^*, \beta^*, \boldsymbol{c}^*, \boldsymbol{a}^*, \boldsymbol{b}^*$. Substituting Eq. 43, 44, 45 into Eq. 36 and comparing the coefficients of $(x^{(\tau)})^4$, we obtain:

$$\boldsymbol{c}^* = \boldsymbol{a}^*\boldsymbol{b}^*. \tag{46}$$

Eq. 42 and Eq. 46 show that $\boldsymbol{c}^*, \boldsymbol{a}^*, \boldsymbol{b}^*$ and $\tilde{c}^*, \tilde{a}^*, \tilde{b}^*$ form a valid witness for the zkMatrix protocol. As a result, we obtain the extractor $\mathcal{K}_{\mathsf{zkMatrix}}$.

Perfect special honest-verifier zero-knowledge. When $\mathcal{S}$ knows the public-coin challenge $[x]$ in advance, it randomly selects $\boldsymbol{a'} \overset{\$}{\leftarrow} \mathbb{Z}_p^{m\times l}$, $\boldsymbol{b'} \overset{\$}{\leftarrow} \mathbb{Z}_p^{l\times n}$. $\mathcal{S}$ computes:

$$P_c^* = \langle \boldsymbol{a'}\boldsymbol{b'}, \boldsymbol{U}\rangle, \quad P_a^* = \langle \boldsymbol{a'}, \boldsymbol{G}\rangle, \quad P_b^* = \langle \boldsymbol{b'}, \boldsymbol{H}\rangle.$$

Then, $\mathcal{S}$ can generate the transcript elements for the zkMatrix protocol by utilizing the random matrices $\boldsymbol{a'}$ and $\boldsymbol{b'}$.

$\mathcal{S}$ randomly chooses $z_c^*, z_a^*, z_b^*, \tilde{\delta}^* \overset{\$}{\leftarrow} \mathbb{Z}_p$ and calculates C_γ^*, C_α^*, C_β^*, and C_δ^*, using the following method:

$$\begin{aligned}
&C_\delta^* \leftarrow \tilde{\delta}^*\tilde{G}, && C_\gamma^* \leftarrow x^{-2}(P_c^* \ominus x C_\delta^* \ominus x^2 C_c \ominus z_c^*\tilde{G}),\\
&C_\alpha^* \leftarrow x^{-1}(P_a^* \ominus x C_a \ominus z_a^*\tilde{G}), && C_\beta^* \leftarrow x^{-1}(P_b^* \ominus x C_b \ominus z_b^*\tilde{G}).
\end{aligned}$$

Additionally, these transcript elements conform to uniform distributions, identical to the distributions of a genuine transcript. To observe this, we utilize the fact that when $\tilde{\alpha} \in \mathbb{Z}_p$ follows a uniform distribution over $\mathbb{Z}_p$, then for any arbitrary $P \in \mathbb{G}$ that is independent of $\tilde{\alpha}$, $P \oplus \tilde{\alpha}\tilde{G} \in \mathbb{G}$ follows a uniform distribution that is independent of the distribution of P. The input matrices of the zkMatrix protocol in line 12 also follow uniform distributions.

Therefore, no PPT distinguisher $\mathcal{D}$ can differentiate between a simulated transcript and a genuine one. Consequently, the zkMatrix protocol exhibits the SHVZK property.

A.6 Proof of Theorem 6

Proof (zkMatrix (Batched Proof)).

Perfect completeness. The perfect completeness follows directly from the perfect completeness of the protocols in Section 4, the equivalence in Eq. 25, and the equation:

$$\langle \sum_{i=1}^{t} \rho^{i-1} \boldsymbol{a}_{(i)}, \boldsymbol{G} \rangle \oplus (\sum_{i=1}^{t} \rho^{i-1} \tilde{a}_{(i)}) \tilde{G} = \bigoplus_{i=1}^{t} \rho^{i-1} [\langle \boldsymbol{a}_{(i)}, \boldsymbol{G} \rangle \oplus \tilde{a}_{(i)} \tilde{G}],$$

as well as similar equations for $\boldsymbol{b}_{(i)}, \boldsymbol{c}_{(i)}, i \in \mathsf{range}(1, t)$.

Computational knowledge soundness. By rewinding batched zkMatrix $mn \times t$ times, we provide it with mn values of $y^{(\tau)}, \tau \in \mathsf{range}(1, mn)$. For each $y^{(\tau)}$, we provide t instances of $\rho^{(\tau')}, \tau' \in \mathsf{range}(1, t)$.

By using the extractors of the zero-knowledge semi-inner-product argument and the zero-knowledge high-dimensional-inner-product argument, we extract $\bar{\boldsymbol{a}}^{(\tau')}, \bar{\boldsymbol{b}}^{(\tau')}, \bar{\vec{\boldsymbol{c}}}^{(\tau')}, \bar{\tilde{a}}^{(\tau')}, \bar{\tilde{b}}^{(\tau')}, \bar{\tilde{c}}^{(\tau')}$ such that:

$$\begin{aligned} &\bar{C}_a^{(\tau')} = \langle \bar{\boldsymbol{a}}^{(\tau')}, \boldsymbol{G} \rangle \oplus \bar{\tilde{a}}^{(\tau')} \tilde{G}, \quad \bar{C}_b^{(\tau')} = \langle \bar{\boldsymbol{b}}^{(\tau')}, \boldsymbol{H} \rangle \oplus \bar{\tilde{b}}^{(\tau')} \tilde{G}, \\ &\bar{C}_c^{(\tau')} = \langle \bar{\vec{\boldsymbol{c}}}^{(\tau')}, \vec{\boldsymbol{U}} \rangle \oplus \bar{\tilde{d}}^{(\tau')} \tilde{G}. \end{aligned}$$

By applying the inverse of the Vandermonde matrix composed of t values of $\rho^{(\tau')}$, we can extract $\boldsymbol{a}_{(i)}, \boldsymbol{b}_{(i)}, \vec{\boldsymbol{c}}_{(i)}, \tilde{a}_{(i)}, \tilde{b}_{(i)}, \tilde{c}_{(i)}$ for each $i \in \mathsf{range}(1, t)$ such that:

$$\begin{aligned} &C_{a(i)} = \langle \boldsymbol{a}_{(i)}, \boldsymbol{G} \rangle \oplus \tilde{a}_{(i)} \tilde{G}, \quad C_{b(i)} = \langle \boldsymbol{b}_{(i)}, \boldsymbol{H} \rangle \oplus \tilde{b}_{(i)} \tilde{G}, \\ &C_{c(i)} = \langle \vec{\boldsymbol{c}}_{(i)}, \vec{\boldsymbol{U}} \rangle \oplus \tilde{c}_{(i)} \tilde{G}. \end{aligned}$$

Further, applying the extractor of the zero-knowledge Bulletproof and proceeding with a process analogous to the one described in Appendix A.4, we can ensure that:

$$\boldsymbol{c}_{(i)} = \boldsymbol{a}_{(i)} \boldsymbol{b}_{(i)}, \forall i \in \mathsf{range}(1, t). \tag{47}$$

Perfect special honest-verifier zero-knowledge. Every element in the transcript of Algorithm 6 adheres to a uniform distribution.

When the randomness involved in Algorithm 6 is known in advance, $\mathcal{S}$ can randomly select values for $C_{d(i)}, C_{ay(i)}, C_{by(i)} \xleftarrow{\$} \mathbb{G}$ for $i \in \mathsf{range}(1, t)$. Then, $\mathcal{S}$ uses the sub-protocol's simulators to generate a simulated transcript. The simulated transcript cannot be distinguished from a genuine one.

B Cryptographic Preliminaries

B.1 Assumptions

Assumption for knowledge soundness of zkMatrix. The computational knowledge soundness of the fundamental zkMatrix protocol (without acceleration) is dependent solely on the assumption of the following discrete logarithm relation.

Assumption 1 (Discrete Logarithm Relation [Bünz et al.(2018)]) *For all PPT adversaries $\mathcal{A}$ and for all $n \geq 2$ there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that:*

$$\mathsf{Pr}\left(\begin{matrix}\exists a_i \neq 0 \\ \wedge \bigoplus_{i=1}^{n} a_i G_i = O\end{matrix}\left|\begin{matrix}(p, \mathbb{G}) \leftarrow \mathsf{Setup}(1^\lambda), G_1, \cdots, G_n \xleftarrow{\$} \mathbb{G}; \\ a_1, \cdots, a_n \in \mathbb{Z}_p \leftarrow \mathcal{A}(p, \mathbb{G}, G_1, \cdots, G_n)\end{matrix}\right.\right)$$
$$\leq \mathsf{negl}(\lambda).$$

We say $\bigoplus_{i=1}^{n} a_i G_i = O$ is a non-trivial discrete logarithm relation between $G_1, \cdots, G_n$. The discrete logarithm relation assumption states that an adversary cannot find a non-trivial discrete logarithm relation between randomly chosen group elements. This assumption implies the standard discrete logarithm assumption (i.e., given $(p, \mathbb{G}, G_1, G_2)$, output $(1, -x)$ such that $G_1 \oplus (-x) G_2 = O$).

Assumption for accelerating zkMatrix. To accelerate zkMatrix, we will also require assumptions related to a bilinear pairing $e : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}_T$, including q-PDH, q-PKE, and q-BSDH.

Assumption 2 (q-PDH [Groth(2010), Parno et al.(2016)]) *The q-Power Diffie-Hellman (q-PDH) assumption holds for the bilinear group generator* Setup *if for all PPT adversary $\mathcal{A}$ we have:*

$$\mathsf{Pr}\left(C = \hat{s}^{q+1} \hat{G} \left| \begin{matrix} (p, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathsf{Setup}(1^\lambda); \\ \hat{G} \xleftarrow{\$} \mathbb{G}; \hat{s} \xleftarrow{\$} \mathbb{Z}_p^*; \\ \mathsf{srs} \leftarrow (p, \mathbb{G}, \mathbb{G}_T, e; [\hat{G}], [\hat{s}\hat{G}], \cdots, [\hat{s}^q \hat{G}]); \\ C \leftarrow \mathcal{A}(\mathsf{srs}) \end{matrix} \right.\right)$$
$$\leq \mathsf{negl}(\lambda).$$

Assumption 3 (q-PKE [Groth(2010), Parno et al.(2016)]) *The q-Power Knowledge of Exponent (q-PKE) assumption holds for $(\mathbb{G}, \mathbb{G}_T)$ if for all PPT adversary $\mathcal{A}$, there exists a non-uniform PPT extractor $\mathcal{K}_{\mathcal{A}}$ such that:*

$$\mathsf{Pr}\left(\begin{matrix} V' = \nu V \\ \wedge V \neq (\sum_{i=0}^{q} \alpha_i \hat{s}^i) \hat{G} \end{matrix} \left| \begin{matrix} (p, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathsf{Setup}(1^\lambda); \\ \hat{G} \xleftarrow{\$} \mathbb{G}; \nu, \hat{s} \xleftarrow{\$} \mathbb{Z}_p^*; \\ \mathsf{srs} \leftarrow (p, \mathbb{G}, \mathbb{G}_T, e; [\hat{G}], [\hat{s}\hat{G}], \cdots, [\hat{s}^q \hat{G}], \\ \quad [\nu\hat{G}], [\nu\hat{s}\hat{G}], \cdots, [\nu\hat{s}^q \hat{G}]); \\ (V', V; \alpha_0, \cdots, \alpha_q) \leftarrow (\mathcal{A} || \mathcal{K}_{\mathcal{A}})(\mathsf{srs}, \boldsymbol{z}) \end{matrix} \right.\right)$$
$$\leq \mathsf{negl}(\lambda),$$

for any auxiliary information $\boldsymbol{z} \in \{0,1\}^{\mathsf{poly}(\lambda)}$ that is generated independently of ν. Note that $(y_1; y_2) \leftarrow (\mathcal{A}||\mathcal{K}_{\mathcal{A}})(x)$ means that on input x, $\mathcal{A}$ outputs y_1 and that $\mathcal{K}_{\mathcal{A}}$ with the same input x and $\mathcal{A}$'s random tape, produces y_2.

Assumption 4 (q-BSDH [Kate et al.(2010)]) *The q-Bilinear Strong Diffie-Hellman (q-BSDH) assumption holds for $(\mathbb{G}, \mathbb{G}_T)$ if for all PPT adversary $\mathcal{A}$ we have:*

$$\mathsf{Pr}\left(E = (\hat{s} - r)^{-1} e(\hat{G}, \hat{G}) \;\middle|\; \begin{array}{l} (p, \mathbb{G}, \mathbb{G}_T, e) \leftarrow \mathsf{Setup}(1^\lambda); \\ \hat{G} \xleftarrow{\$} \mathbb{G}; \hat{s} \xleftarrow{\$} \mathbb{Z}_p^*; \\ \mathsf{srs} \leftarrow (p, \mathbb{G}, \mathbb{G}_T, e; [\hat{G}], [\hat{s}\hat{G}], \cdots, [\hat{s}^q \hat{G}]) \\ (E \in \mathbb{G}_T, r \in \mathbb{Z}_p^*) \leftarrow \mathcal{A}(\mathsf{srs}) \end{array} \right)$$

$$\leq \mathsf{negl}(\lambda).$$

B.2 Pedersen Vector Commitment

Definition 1 (Commitment). *A non-interactive commitment scheme consists of a pair of probabilistic polynomial time algorithms* $(\mathsf{Setup}, \mathsf{Com})$. *The setup algorithm $pp \leftarrow \mathsf{Setup}(1^\lambda)$ generates public parameters* pp *for the scheme, for security parameter λ. The commitment algorithm* $\mathsf{Com}_{\mathsf{pp}}$ *defines a function $\mathbb{M}_{\mathsf{pp}} \times \mathbb{R}_{\mathsf{pp}} \mapsto \mathbb{C}_{\mathsf{pp}}$ for the message space $\mathbb{M}_{\mathsf{pp}}$, the randomness space $\mathbb{R}_{\mathsf{pp}}$, and the commitment space $\mathbb{C}_{\mathsf{pp}}$ determined by* pp. *for a message $a \in \mathbb{M}_{\mathsf{pp}}$, the algorithm draws $\tilde{a} \xleftarrow{\$} \mathbb{R}_{\mathsf{pp}}$ uniformly at random, and computes the commitment $C_a \leftarrow \mathsf{Com}_{\mathsf{pp}}(a; \tilde{a})$.*

Definition 2 (Pedersen Vector Commitment). *Let the message space $\mathbb{M}_{\mathsf{pp}} = \mathbb{Z}_p^n$, the randomness space $\mathbb{R}_{\mathsf{pp}} = \mathbb{Z}_p$, and the commitment space $\mathbb{C}_{\mathsf{pp}} = \mathbb{G}$ with $\mathbb{G}$ of order p. The Pedersen vector commitment of a vector $\vec{\boldsymbol{a}} \in \mathbb{Z}_p^n$ and a randomness $\tilde{a} \in \mathbb{Z}_p$ is*

$$\mathsf{Setup}: \vec{\boldsymbol{G}} \xleftarrow{\$} \mathbb{G}^n, \tilde{G} \xleftarrow{\$} \mathbb{G}, \quad \mathsf{Com}(\boldsymbol{a}; \tilde{a}): C_a \leftarrow \langle \vec{\boldsymbol{a}}, \vec{\boldsymbol{G}} \rangle \oplus \tilde{a}\tilde{G}.$$

With the bracket representation introduced in Section 2.1, we write the Pedersen vector commitment as $[\langle \vec{\boldsymbol{a}}, \vec{\boldsymbol{G}} \rangle \oplus \tilde{a}\tilde{G}]$. We often set $\tilde{a} = 0$, in which case the commitment is binding but not hiding.

Pedersen vector commitment for a matrix is the Pedersen vector commitment to the flattened vector of the matrix.

B.3 Zero-Knowledge Arguments of Knowledge

An Argument of Knowledge (AoK) is a protocol that allows a computationally bounded prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that he knows a witness w for a certain statement u. Zero-knowledge Arguments of Knowledge (zkAoK) or zero-knowledge proofs further require that the prover can convince the verifier that the statement holds without revealing any information about why it holds.

Let $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$ be a polynomial-time-decidable ternary relation. Given a structured reference string srs, we call w a witness for a statement u if $(\mathsf{srs}, u, w) \in \mathcal{R}$, and define the SRS-dependent language:

$$\mathcal{L}_{\mathsf{srs}} = \{u \mid \exists w \text{ s.t. } (\mathsf{srs}, u, w) \in \mathcal{R}\},$$

as the set of statements u that have a witness w in the relation $\mathcal{R}$.

We consider arguments consisting of three interactive algorithms $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$, all running in probabilistic polynomial time. On input 1^λ, the algorithm Setup produces a structured reference string (SRS) srs. Then, $\mathcal{P}$ and $\mathcal{V}$ interact on input $(\mathsf{srs}, u : w)$, responding to the verifier's randomness rand and producing a transcript tr. This process is denoted by $\mathsf{tr} \leftarrow (\mathcal{P} \rightleftharpoons \mathcal{V})(\mathsf{srs}, u : w)$, where the prover's view is $(\mathsf{srs}, u : w)$ and the verifier's view is $(\mathsf{srs}, u; \mathsf{tr}; \mathsf{rand})$. We mark $\mathsf{flag} = \mathcal{V}(\mathsf{srs}, u; \mathsf{tr}; \mathsf{rand})$ depending on whether the verifier rejects, $\mathsf{flag} = \mathsf{FALSE}$, or accepts, $\mathsf{flag} = \mathsf{TRUE}$.

Definition 3 (Argument of Knowledge). *The triple* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *is called an argument of knowledge for the relation* $\mathcal{R}$ *if it satisfies perfect completeness and knowledge soundness as defined below.*

Definition 4 (Perfect Completeness [Attema et al.(2021)]). $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *has perfect completeness if for all non-uniform polynomial time adversaries* $\mathcal{A}$*:*

$$\mathsf{Pr}\left(\begin{matrix}(\mathsf{srs}, u : w) \notin \mathcal{R} \\ \vee\, \mathcal{V}(\mathsf{srs}, u; \mathsf{tr}; \mathsf{rand}) = \mathsf{TRUE}\end{matrix} \;\middle|\; \begin{matrix}\mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda); \\ (u, w) \leftarrow \mathcal{A}(\mathsf{srs}); \\ \mathsf{tr} \leftarrow (\mathcal{P} \rightleftharpoons \mathcal{V})(\mathsf{srs}, u : w)\end{matrix}\right) = 1.$$

Definition 5 (Computational Knowledge Soundness [Attema et al.(2021)]).

$(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *for the relation* $\mathcal{R}$ *is knowledge sound with error* $\kappa_\lambda(|u|) : \mathbb{N} \mapsto [0,1]$ *if there exists an algorithm* $\mathcal{K}$*, called a knowledge extractor, with the following properties. Given input u and -box oracle access to a (potentially dishonest) prover* $\mathcal{P}^*$*, the extractor* $\mathcal{K}$ *runs in an expected number of steps that is polynomial in* $|u|$ *(counting queries to* $\mathcal{P}$ *as a single step) and outputs a witness* $(\mathsf{srs}, u : w) \in \mathcal{R}$ *with probability:*

$$\Pr((\mathsf{srs}, u : \mathcal{K}(u)) \in \mathcal{R}) \geq \frac{p(\mathcal{P}^*, u) - \kappa_\lambda(|u|)}{\mathsf{poly}(|u|)},$$

where $p(\mathcal{P}^*, u)$ *is the probability that* $\mathcal{P}^*$ *generates an accepted transcript.*

Previous studies have shown that with the utilization of public-coin randomness, as defined in the subsequent subsection, witness-extended emulation is an alternative approach to define knowledge soundness [Groth(2004), Attema et al.(2022), Attema et al.(2021)].

Definition 6 (Witness-Extended Emulation [Bünz et al.(2018)]). $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *has witness-extended emulation if for all deterministic polynomial time* $\mathcal{P}^*$*, and*

PPT adversaries $\mathcal{A}$ and all PPT distinguishers $\mathcal{D}$, there exists a negligible function $\mathsf{negl}_\lambda(|u|)$ *and an expected polynomial time emulator* $\mathcal{E}^{(\mathcal{P}^* \rightleftharpoons \mathcal{V})}$ *such that:*

$$\left| \begin{array}{l} \mathsf{Pr}\left(\begin{array}{r} \mathcal{D}(\mathsf{srs}, u; \mathsf{tr}^*; \mathsf{rand}) \\ = \mathsf{TRUE} \end{array} \middle| \begin{array}{l} \mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda); \\ \mathsf{rand} \xleftarrow{\$} \{0,1\}^*; \\ \mathsf{tr}^* \leftarrow (\mathcal{P}^* \rightleftharpoons \mathcal{V})(\mathsf{srs}, u; \mathsf{rand}) \end{array} \right) \\ -\mathsf{Pr}\left(\begin{array}{r} (\mathcal{D}(\mathsf{srs}, u; \mathsf{tr}^*; \mathsf{rand}) \\ = \mathsf{TRUE}) \quad \wedge \\ (\mathcal{V}(\mathsf{srs}, u; \mathsf{tr}^*; \mathsf{rand}) \\ = \mathsf{TRUE} \quad \Rightarrow \\ (\mathsf{srs}, u : w^*) \in \mathcal{R}) \end{array} \middle| \begin{array}{l} \mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda); \\ \mathsf{rand} \xleftarrow{\$} \{0,1\}^*; \\ \mathsf{tr}^* \leftarrow (\mathcal{P}^* \rightleftharpoons \mathcal{V})(\mathsf{srs}, u; \mathsf{rand}) \\ w^* \leftarrow \mathcal{E}^{(\mathcal{P}^* \rightleftharpoons \mathcal{V})}(\mathsf{srs}, u) \end{array} \right) \end{array} \right|$$
$$\leq \mathsf{negl}_\lambda(|u|),$$

where $\mathcal{E}^{(\mathcal{P}^* \rightleftharpoons \mathcal{V})}$ *is the emulator on the oracle* $(\mathcal{P}^* \rightleftharpoons \mathcal{V})$ *that permits rewinding to a specific point and resuming with fresh randomness* rand *for the verifier from this point onwards.*

Definition 7 (Perfect Zero-Knowledge). *A non-interactive argument of knowledge* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *is perfect zero-knowledge if there exists a PPT simulator* $\mathcal{S}$ *and a trapdoor* trap *such that for all stateful distinguishers* $\mathcal{D}$ *the following two probabilities are equal:*

$$\mathsf{Pr}\left(\mathcal{D}(\mathsf{srs}, u; \mathsf{tr}; \mathsf{trap}) = \mathsf{TRUE} \middle| \begin{array}{l} (\mathsf{srs}, \mathsf{trap}) \leftarrow \mathsf{Setup}(1^\lambda); \\ \mathsf{tr} \leftarrow (\mathcal{P} \rightleftharpoons \mathcal{V})(\mathsf{srs}, u : w) \end{array} \right)$$
$$= \mathsf{Pr}\left(\mathcal{D}(\mathsf{srs}, u; \mathsf{tr}^*; \mathsf{trap}) = \mathsf{TRUE} \middle| \begin{array}{l} (\mathsf{srs}, \mathsf{trap}) \leftarrow \mathsf{Setup}(1^\lambda); \\ \mathsf{tr}^* \leftarrow \mathcal{S}(\mathsf{srs}, u; \mathsf{trap}) \end{array} \right).$$

When using public-coin challenges, perfect Special Honest-Verifier Zero-Knowledge (SHVZK) offers a more convenient means of defining perfect zero-knowledge. It is feasible to transform an SHVZK argument into a perfect zero-knowledge argument, utilizing established techniques [Groth(2004)].

Definition 8 (Perfect Special Honest-Verifier Zero-Knowledge). *An argument of knowledge* $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ *has perfect special honest-verifier zero-knowledge if there exists a PPT simulator* $\mathcal{S}$ *such that for all pairs of interactive adversaries* $\mathcal{A}$ *and* $\mathcal{D}$*, the following two probabilities are equal:*

$$\mathsf{Pr}\left(\begin{array}{l} (\mathsf{srs}, u, w) \in \mathcal{R} \\ \wedge\, \mathcal{D}(\mathsf{srs}, u; \mathsf{tr}; \mathsf{rand}) = \mathsf{TRUE} \end{array} \middle| \begin{array}{l} \mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda); \\ (u, w, \mathsf{rand}) \leftarrow \mathcal{A}(\mathsf{srs}); \\ \mathsf{tr} \leftarrow (\mathcal{P} \rightleftharpoons \mathcal{V})(\mathsf{srs}, u, w) \end{array} \right)$$
$$= \mathsf{Pr}\left(\begin{array}{l} (\mathsf{srs}, u, w) \in \mathcal{R} \\ \wedge\, \mathcal{D}(\mathsf{srs}, u; \mathsf{tr}^*; \mathsf{rand}) = \mathsf{TRUE} \end{array} \middle| \begin{array}{l} \mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda); \\ (u, w, \mathsf{rand}) \leftarrow \mathcal{A}(\mathsf{srs}); \\ \mathsf{tr}^* \leftarrow \mathcal{S}(\mathsf{srs}, u, \mathsf{rand}) \end{array} \right),$$

where rand *is the public-coin randomness used by the verifier.*

Definition 9 (Fully Succinct zk-SNARK [Gabizon et al.(2019)]). *A non-interactive zero-knowledge argument of knowledge is fully succinct if:*

- *The SRS generation time is quasilinear relative to the witness size;*
- *The prover time is quasilinear relative to the witness size;*
- *The transcript size is logarithmic relative to the witness size;*
- *The verifier time is polylogarithmic relative to the witness size.*

B.4 Fiat-Shamir Heuristics

Definition 10 (Public-Coin). *A $(2\mu+1)-$move public-coin interactive protocol is an argument of knowledge $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ where in the $(2j)$th move, $\mathcal{V}$ sends the jth challenge $x_j \overset{\$}{\leftarrow} \mathbb{X}_j$. All challenges $x_1, \cdots, x_\mu$ sent from the verifier to the prover are public information and chosen uniformly at random and independently of the prover's messages.*

The Fiat-Shamir heuristic replaces the public-coin challenge sent by the verifier with a hash function, thereby turning an interactive protocol into a non-interactive one [Attema et al.(2022)].

Definition 11 (Adaptive Fiat-Shamir Transformation). *The adaptive Fiat-Shamir transformation on an interactive argument of knowledge $\mathsf{FS}(\mathsf{Setup}, \mathcal{P}, \mathcal{V}) \mapsto (\mathsf{Setup}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$ is a two-round argument of knowledge where $\mathcal{P}_{\mathsf{FS}}(\mathsf{srs}, u, w)$ runs $\mathcal{P}(\mathsf{srs}, u, w)$ but instead of asking the verifier for the challenge x_i on message a_i, the challenges are computed as:*

$$x_i = \mathsf{Hash}_i(u, a_1, \cdots, a_{i-1}, a_i);$$

the output is then the transcript $\mathsf{tr} = (a_1, \cdots, a_i, a_{i+1})$. On input of a statement u and a proof $\mathsf{tr} = (a_1, \cdots, a_i, a_{i+1})$, $\mathcal{V}_{\mathsf{FS}}(\mathsf{srs}, u, w)$ accepts if, for x_i as above, $\mathcal{V}$ accepts the transcript $\mathsf{tr} = (a_1, \cdots, a_i, a_{i+1})$ on input u.

C Adaptation to Floating-Point Matrix Multiplication

In floating-point arithmetic, the mantissa of the product of two floating-point numbers is obtained by multiplying their mantissas and rounding to the specified precision. According to the IEEE 754 standard, a double-precision floating-point number consists of 64 bits: 1 bit for the sign, 11 bits for the exponent, and 52 bits for the mantissa. In the context of floating-point matrices, the floating-point numbers within a matrix can share a common exponent.

Consider a matrix $\boldsymbol{a} = \{a_{ij} = a_{ij}^{(m)} \cdot 2.0^{e_{ij}}\} \in \mathbb{R}^{m \times n}$ with elements $a_{ij}^{(m)} \cdot 2.0^{e_{ij}} \in \mathbb{R}$, where $a_{ij}^{(m)} \in \mathbb{Z}$ and $e_{ij} \in \mathbb{Z}$ are the signed mantissa and exponent, respectively. Let $\eta_a := e_{\mathsf{min}}$ represent the smallest and e_{max} the largest exponents. Each element can then be expressed as:

$$a_{ij} = a_{ij}^{(m)} \cdot 2.0^{e_{ij}} = (2^{e_{ij}-\eta_a} \cdot a_{ij}^{(m)}) \cdot 2.0^{\eta_a},$$

where powers of 2.0 are computed in the real field and powers of 2 in the integer field. The new mantissa $2^{e_{ij}-\eta_a} \cdot a_{ij}^{(m)}$ is a signed integer with at most $(e_{\mathsf{max}} - e_{\mathsf{min}} + 52)$ bits. If the ratio between the largest and smallest non-zero elements of the matrix does not exceed 2.0^{32}, the new mantissa is at most $32 + 52 = 84$ bits. Therefore, committing to the floating-point matrix $\boldsymbol{a} \in \mathbb{R}^{m \times n}$ is equivalent to committing to $\boldsymbol{a}^{(m)} := \{2^{e_{ij}-\eta_a} \cdot a_{ij}^{(m)}\} \in \mathbb{Z}_p^{m \times n}$ and $\eta_a \in \mathbb{Z}_p$.

Consider three floating-point matrices $\boldsymbol{a}, \boldsymbol{b}$, and $\boldsymbol{c}$ satisfying:

$$\boldsymbol{c} = \boldsymbol{c}^{(m)} \cdot 2.0^{-\eta_c}, \quad \boldsymbol{a} = \boldsymbol{a}^{(m)} \cdot 2.0^{-\eta_a}, \quad \boldsymbol{b} = \boldsymbol{b}^{(m)} \cdot 2.0^{-\eta_b},$$

where $\boldsymbol{c}^{(m)}, \boldsymbol{a}^{(m)}, \boldsymbol{b}^{(m)}$ are the signed mantissa matrices, and η_c, η_a, and η_b are the common exponents. We assert:

$$\begin{aligned} &\boldsymbol{c} = \boldsymbol{a}\boldsymbol{b} \iff \{\exists \boldsymbol{c}^{(m)'} \in \mathbb{Z}_p^{m \times n}, \eta \in \mathbb{Z}_p, s.t.\, \boldsymbol{c}^{(m)'} = \boldsymbol{a}^{(m)}\boldsymbol{b}^{(m)} \wedge \\ &\eta = \eta_a + \eta_b - \eta_c \wedge -2^{84} \cdot \mathbf{1} < \boldsymbol{c}^{(m)'} \cdot 2^{84-\eta} - \boldsymbol{c}^{(m)} \cdot 2^{84} < 2^{84} \cdot \mathbf{1}\}, \end{aligned}$$

where $\mathbf{1} \in \mathbb{Z}_p^{m \times n}$ represents the matrix with all elements being 1. If $\boldsymbol{a}$ has at most 2^{86} columns, then the elements of $\boldsymbol{c}^{(m)'}$ in the equation each have at most $84 + 84 + 86 = 254$bits. Therefore, the matrix multiplication in $\mathbb{Z}_p$ is identical to that in $\mathbb{Z}$. This relation can be verified using the zkMatrix protocol and range proofs for committed matrices, with further details to be discussed in subsequent work.