zkMatrix: Batched Short Proof for Committed Matrix Multiplication

Mingshu Cong, Tsz Hon Yuen, Siu Ming Yiu

The University of Hong Kong

AsiaCCS Presentation 2024.7

Contents

Why We Target Committed Matrix Multiplication

Problem Definition: zkSNARK for Matrix Multiplication

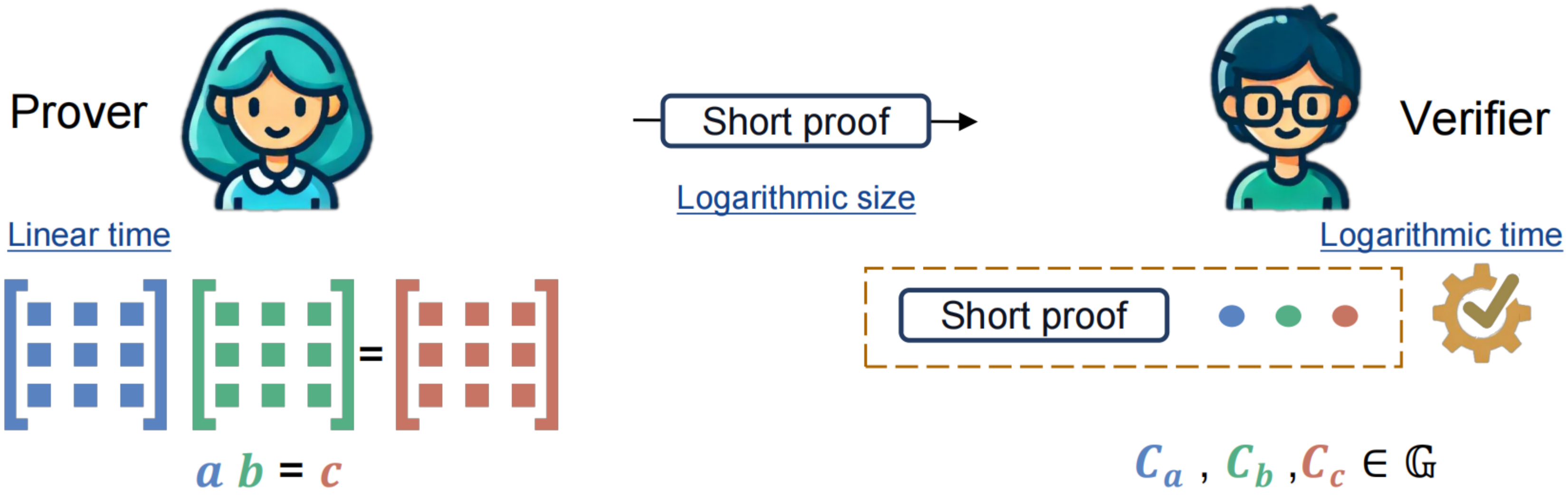

Figure: The prover holds three $n \times n$ secret matrices $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{Z}_p^{n\times n}$. The verifier knows their commitments $C_a, C_b, C_c \in \mathbb{G}$. The prover provides a short proof to let the verifier verify that $\boldsymbol{ab} = \boldsymbol{c}$.

Matrix Commitment

- Commit a matrix $\boldsymbol{a} \in \mathbb{Z}_p^{m\times n}$ to a group element $C_a \in \mathbb{G}$.
- Pedersen vector commitment to $\boldsymbol{a} \in \mathbb{Z}_p^{m\times n}$ using bases $\boldsymbol{G} \in \mathbb{G}^{m\times n}$:

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

- The committed matrix multiplication relation:

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

Application: zkSNARK for Neural Networks

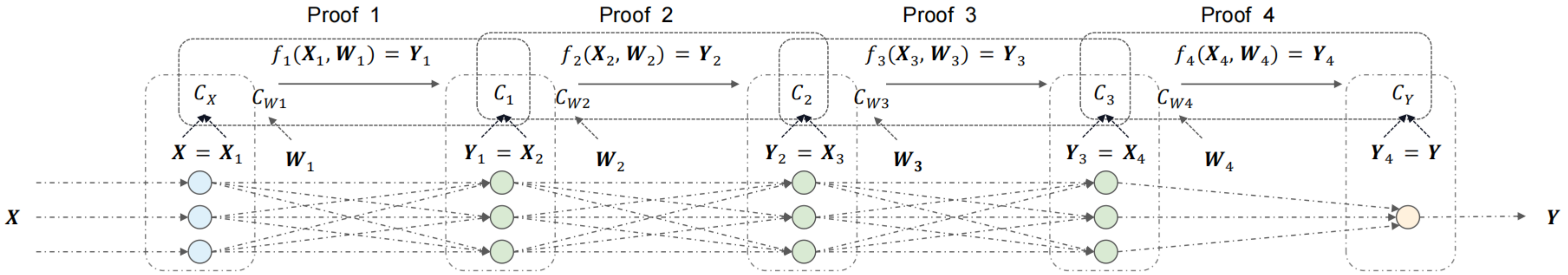

Figure: Neural networks with zero-knowledge proofs. X_i and Y_i represent the input and output of the neural network's layer i, with a weight W_i such that $f_i(X_i, W_i) = Y_i$ for some matrix function f_i. C_A stands for the commitment to the secret vector/matrix A, bridging sub-protocols throughout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

Application: Verifiable Statistics on Private Dataset

- Example: A health institute may publish reliable statistical analysis on the efficacy of a medical treatment without leaking sensitive patient data.
- Estimation of coefficients for the linear regression (LR) model: $y = \boldsymbol{\beta}\boldsymbol{x} + \epsilon$:

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}(\boldsymbol{X}^\top \boldsymbol{Y}).$$

Difficulty for Matrix Multiplication

Inherent superlinearity of matrix multiplication:

Schoolbook algorithm requires n^3 multiplication gates for $n \times n$ matrix multiplication. The best algorithm in the literature needs $> n^{2.37}$ multiplications.

- General-purpose zkSNARKs (e.g. Groth16 [2]) requires RAM usage and prover time proportional to the number of multiplication gates.
- Interactive protocols (e.g. Quicksilver [6] , Thaler's protocol [4]) enable $O(n^2)$ prover time but lack succinct proof size or logarithmic verifier time.

How We Solve This Problem

Freivalds' Algorithm

Freivalds' Algorithm

Freivalds' Algorithm verifies matrix multiplication probabilistically via:

$$\boldsymbol{a}\boldsymbol{b} = \boldsymbol{c} \quad \Longleftrightarrow \quad \boldsymbol{a}\boldsymbol{b}\vec{\boldsymbol{r}} = \boldsymbol{c}\vec{\boldsymbol{r}}, \text{ where } \vec{\boldsymbol{r}} \in \mathbb{Z}_p^{*n} \text{ is a random vector.}$$

- We apply Freivalds' Algorithm twice and verify:

$$\vec{\boldsymbol{l}}^\top \boldsymbol{a}\boldsymbol{b}\vec{\boldsymbol{r}} = \vec{\boldsymbol{l}}^\top \boldsymbol{c}\vec{\boldsymbol{r}}, \text{ where } \vec{\boldsymbol{l}} \in \mathbb{Z}_p^{*m}, \vec{\boldsymbol{r}} \in \mathbb{Z}_p^{*n} \text{ are random vectors.}$$

- We use powers of a random challenge $y \xleftarrow{\$} \mathbb{Z}_p^*$ to generate $\vec{\boldsymbol{l}}, \vec{\boldsymbol{r}}$:

$$\vec{\boldsymbol{l}} \leftarrow \vec{\boldsymbol{y}}_L = (1, y^n, \cdots, y^{(m-1)n}), \vec{\boldsymbol{r}} \leftarrow \vec{\boldsymbol{y}}_R = (1, y, \cdots, y^{n-1}).$$

- We verify:

$$\vec{\boldsymbol{y}}_L^\top (\boldsymbol{a}\boldsymbol{b})\vec{\boldsymbol{y}}_R = \vec{\boldsymbol{y}}_L^\top \boldsymbol{c}\vec{\boldsymbol{y}}_R.$$

Matrix Multiplication Through Four Inner Products

Matrix Multiplication to Four Inner Product Relations

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

- Inner product ①: $d := \vec{\boldsymbol{y}}_L^\top \boldsymbol{c} \vec{\boldsymbol{y}}_R = \sum_{i=1}^m \sum_{j=1}^n c_{ij} y^{(i-1)n+(j-1)}$.
- (High-dimensional) inner product:
  ② $\vec{\boldsymbol{a}}_y := \sum_{i=1}^m \vec{\boldsymbol{a}}_{i*} y^{(i-1)n}$, ③ $\vec{\boldsymbol{b}}_y := \sum_{j=1}^n \vec{\boldsymbol{b}}_{*j} y^{j-1}$.
- Inner product ④: $d = \vec{\boldsymbol{a}}_y \bullet \vec{\boldsymbol{b}}_y$.

Verify Four Inner Products Using Bulletproofs.

- *Committed Inner-Product Argument* for ④.
- *Committed Semi-Inner-Product Argument* for the inner-product between a committed vector and a public vector in ①.
- *Committed High-Dimensional Semi-Inner-Product Argument* for high-dimensional inner products ② and ③.

Improvement on Bulletproofs

Faster Verifier Time Via Trusted Setup

Shift the verifier's multi-exponentiation computation load to the prover.

- Verification of classical Bulletproofs involves computing 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 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.
- 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 $\hat{G} \in \mathbb{G}_1$ in a pairing group.
- Define $\phi(x) = \zeta_1 x + \cdots + \zeta_q x^q \in \mathbb{Z}_p[X]$. By using a random challenge $s \in \mathbb{Z}_p^*$, the prover computes $W \leftarrow (\frac{\phi(s)-\phi(\hat{s})}{s-\hat{s}})\hat{G}$, and the verifier verifies:

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

Verification of High-Dimensional Inner Product

High-Dimensional Inner Product to Normal Inner Product

$$\vec{\boldsymbol{c}} = \boldsymbol{a}\vec{\boldsymbol{b}} \wedge C_a = \langle \boldsymbol{a}, \boldsymbol{G} \rangle \wedge C_c = \langle \vec{\boldsymbol{c}}, \vec{\boldsymbol{U}} \rangle \iff \forall x \in \mathbb{Z}_p^*, C_a \oplus x C_c = \langle \boldsymbol{a}, \{G_{ij} \oplus x U_i b_j\} \rangle.$$

- By using a random challenge $x \in \mathbb{Z}_p^*$, verification of the high-dimensional inner product is transformed into verifying inner product between $\boldsymbol{a} \in \mathbb{Z}_p^{m \times n}$ and the base matrix $\{G_i \oplus x b_j U_i\} \in \mathbb{G}^{m \times n}$.
- This transformed inner product is verified by Bulletproof.

Batch Proving for Multiple Matrix Multiplications

Batched zkMatrix for t Matrix Multiplications

- For a common $\boldsymbol{b}$, batched zkMatrix is achieved using a challenge $\rho \xleftarrow{\$} \mathbb{Z}_p^*$:

$$\boldsymbol{a}_{(i)}\boldsymbol{b} = \boldsymbol{c}_{(i)}, \forall i \in (1, t) \iff \forall \rho \in \mathbb{Z}_p^*, \quad (\sum_{i=1}^{t} \rho^{i-1}\boldsymbol{a}_{(i)})\boldsymbol{b} = \sum_{i=1}^{t} \rho^{i-1}\boldsymbol{c}_{(i)}.$$

- For distinct matrices $\boldsymbol{a}_{(i)}, \boldsymbol{b}_{(i)}, \boldsymbol{c}_{(i)}$ such that $\boldsymbol{a}_{(i)}\boldsymbol{b}_{(i)} = \boldsymbol{c}_{(i)}$, batched zkMatrix is achieved by batch processing inner products ①, ②, and ③.

Batch Proving for Multiple Matrix Multiplications (Continued)

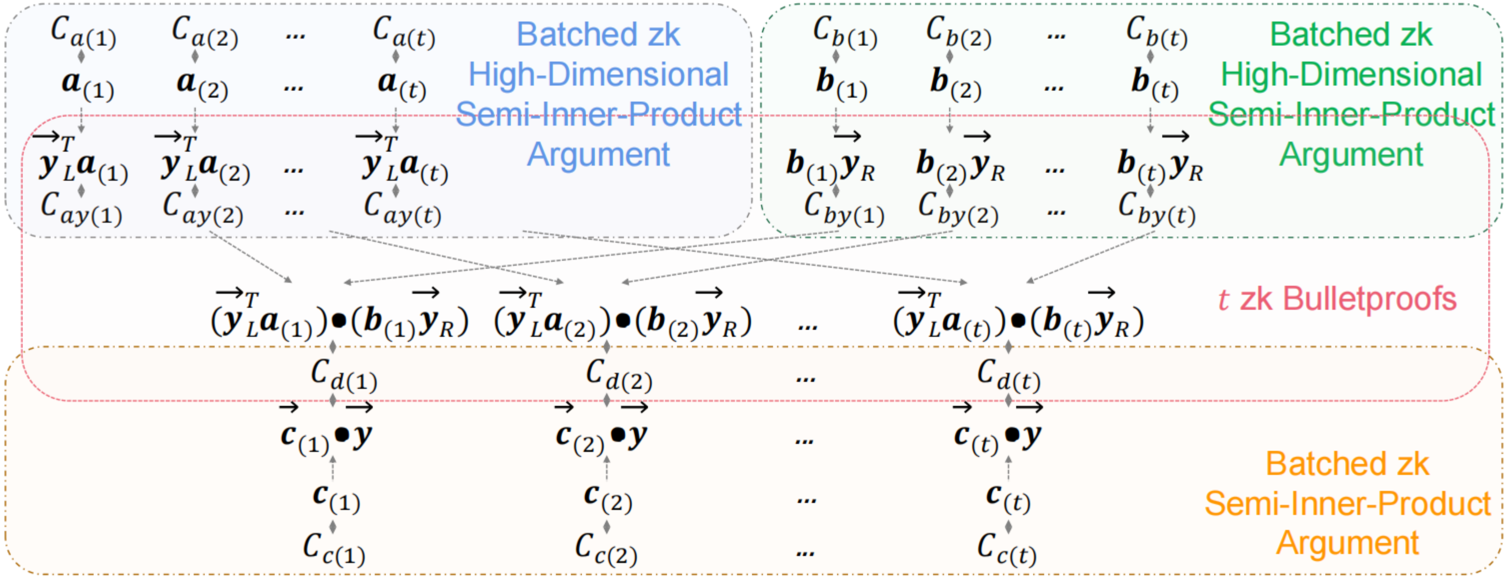

Figure: Batched zkMatrix for t matrix multiplications $\boldsymbol{a}_{(i)}\boldsymbol{b}_{(i)} = \boldsymbol{c}_{(i)}, i = 1, \cdots, t$. Inner products arguments for ①, ②, and ③ are batch processed, while those for ④ are processed individually for t matrix multiplications. The prover time is $O(n^2 + tn)$ group operations.

Adding Zero Knowledge

Zero-knowledge proof via random masking matrices.

We use two random masking matrices $\boldsymbol{\alpha} \in \mathbb{Z}_p^{m\times l}$ and $\boldsymbol{\beta} \in \mathbb{Z}_p^{l\times n}$:

$$\{\boldsymbol{c} = \boldsymbol{a}\boldsymbol{b}\} \Leftrightarrow \left\{ \begin{aligned} &\forall x \in \mathbb{Z}_p^*,\ x^2(\boldsymbol{\alpha}\boldsymbol{\beta} + x(\boldsymbol{a}\boldsymbol{\beta} + \boldsymbol{\alpha}\boldsymbol{b}) + x^2\boldsymbol{c}) \\ &\qquad\qquad = (x(\boldsymbol{\alpha} + x\boldsymbol{a}))(x(\boldsymbol{\beta} + x\boldsymbol{b})) \end{aligned} \right\}.$$

Why zkMatrix is Better

Comparison With Existing Methods

First zkSNARK With Linear Prover Time for Matrix Multiplication

For $n \times n$ matrix multiplication, we achieve $O(n^2)$ prover time, $O(\log n)$ proof size, and $O(\log n)$ verifier time.

Protocol	Communi-cation	RAM Usage Prover	RAM Usage Verifier	Timing Prover	Timing Verifier	Consistency
Pinocchio [3]	$O(1)$	$O(n^3)$	$O(1)$	$O(n^3)$	$O(1)$	No
Thaler [4]	$O(\log n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	No
LegoSNARK [1]	$O(\log n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	Yes
Libra [5]	$O(\log^2 n)$	$O(n^3)$	$O(\log n)$	$O(n^3)$	$O(\log^2 n)$	No
QuickSilver [6]	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	Yes
Single zkMatrix	$O(\log n)$	$O(n^2)$	$O(\log n)$	$O(n^2)$	$O(\log n)$	Yes
Batched zkMatrix	$O(t \log n)$	$O(n^2 + tn)$	$O(t \log n)$	$O(n^2 + tn)\mathbb{E} + O(tn^2)\mathbb{M}$	$O(t \log n)$	Yes

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

Evaluations

Our Codes is Available

We have implemented DualMatrix, an improvement over zkMatrix. You can access the code at: `https://github.com/mirandaprivate/dualmatrix`.

	DualMatrix	Thaler's[a]	zkMatrix[b]	Libra[c]	Spartan[d]	HyperPlonk[d]
Setup time	1.94s	NA	419.54s	26.93s	427.23ms	48.22s
SRS size	361.61KB	NA	801.11MB	2.06GB	1.44MB	3.22GB
Preprocessing	23.38s	NA	17.45s	6.76s	23.47s	87.46s
Prover time	5.63s	77.54ms	33.42s	258.63s	357.49s	149.91s
Verifier time	469.50ms	942.78ms	78.87ms	150.38ms	682.00ms	25.32ms
Transcript size	131.13KB	168B	7.85KB	48.00KB	501.35KB	17.96KB

Table: Benchmarking Performance on Dense Matrices of Size $1,024 \times 1,024$. All experiments were conducted on the same machine, utilizing 64 threads for parallelization. DualMatrix, Spartan, and HyperPlonk are implemented in Rust, whereas Thaler's protocol and Libra are implemented in C++. We exclude the time for computing the underlying matrix multiplication from the prover times.

[a] In Thaler's protocol, the matrices $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ are public. Its performance is not fairly comparable with those of other protocols.
[b] The performance metrics of zkMatrix are estimated from the theoretical performance and execution times of single group and field operations on the same machine.
[c] Libra encountered a memory overflow on our machine when processing $1,024 \times 1,024$ matrices. Consequently, we report its performance on smaller 512×512 matrices.
[d] The performance metrics for Spartan and Hyperplonk are measured for a mock circuit with 2^{24} constraints, under the assumption of the ideal matrix-multiplication-to-circuit conversion. These are lower bounds for real implementations of the matrix multiplication circuit.

References

[1] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSnark: Modular design and composition of succinct zero-knowledge proofs. In *CCS*, pages 2075–2092, 2019.

[2] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326, 2016.

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

[4] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Crypto*, pages 71–89, 2013.

[5] Tiacheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Crypto*, pages 733–764, 2019.

[6] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *CCS*, pages 2986–3001, 2021.

Thank you!
Q & A