Efficient Homomorphic Matrix Computation for Secure Transformer Inference

Miran Kim

Hanyang University

Joint work with Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang

Motivation

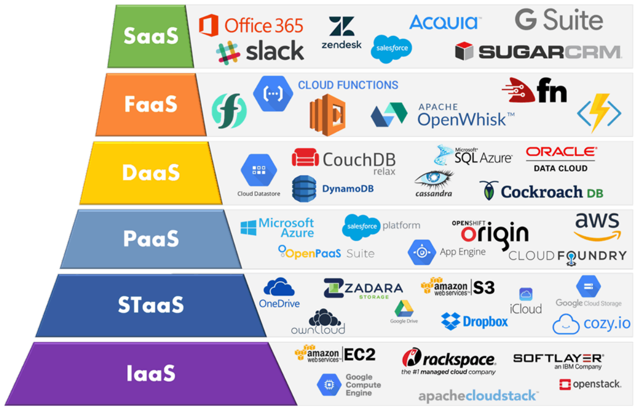
- Privacy issue with AI
  - **Personalized services** require personal information
  - Cloud computing services: Google Gemini, Meta LLaMA, OpenAI ChatGPT
  - **Data abuse**: if data is available, it will be used

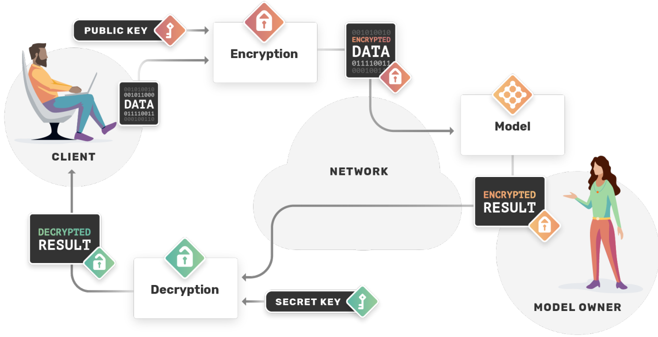
Motivation

Question. Can ***Prediction as a Service*** be made trustworthy and efficient?

(Privacy-preserving Personalized Service)

- Traditional encryption protects only storage and transmission—not the computation.
- HE enables computations to be performed without decrypting the data.






Homomorphic Encryption and LLM : Is ChatGPT end to end encrypted ?

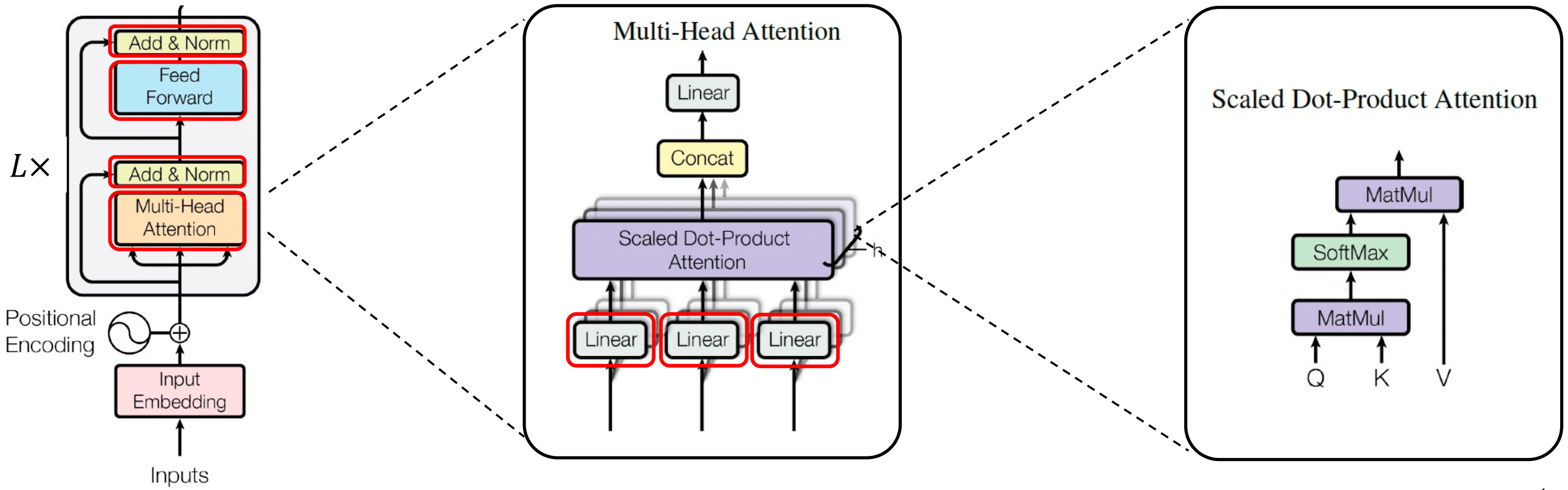
April 19, 2023 — Rand Hindi

This is the first post in a series dedicated to making large language models (LLMs) encrypted end-to-end with homomorphic encryption. We will publish more details on how to achieve it technically as we make progress towards this goal in the coming years.

Since most of the challenges in FHE are already solved (or will be in the near future), **we can confidently expect to have end-to-end encrypted AI within 5 years.** I strongly believe that when this happens, nobody will care about privacy anymore, not because it's unimportant, but because it will be guaranteed by design.

Transformer-based Model

- What is ***Transformer***?
  - **Self-attention** based architecture [V+18] (how strongly each token is related to every other token)
  - Foundation of modern NLP models like BERT, GPT, T5, BART
  - *Parallelizable* and efficient processing of sequences



[V+18] Attention is all you need, NeurIPS 2018

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

Table of Contents

What is Homomorphic Encryption (HE)?

- HE can evaluate arithmetic functions on encrypted data.
  - For $x, y \in \mathcal{M} = \mathbb{Z}_q[X]/(X^N + 1)$,

$$Dec\big(Enc(x) + Enc(y)\big) = Dec\big(Enc(x)\big) + Dec\big(Enc(y)\big) = x + y$$

$$Dec\big(Enc(x) * Enc(y)\big) = Dec\big(Enc(x)\big) * Dec\big(Enc(y)\big) = x * y$$

- HE offers lots of flexibility in parameter selection
  - Plaintext modulus, vector-size, polynomial degree …

- ***Ciphertext packing technique*** to improve performance
  - Each ciphertext encrypts multiple plaintext elements and perform ciphertext computation in a *SIMD* manner.
  - Even with a fixed parameter set, determining how to put each data element into the ciphertexts requires careful design to achieve optimal efficiency.

Ciphertext Packing Technique

- Slot-encoding
  - Use *DFT-like algorithm* to transform a plaintext vector into an element of a cyclotomic ring (BGV,BFV,CKKS)
  - Support *vectorized* operations in a *SIMD* manner
    - Entry-wise addition/multiplication
    - Rotation: moving values between slots
  - Computing the same function on l inputs at the price of one computation (better amortized size and timing)

- Coefficient-encoding
  - Encode plaintext values *directly* as an element of a cyclotomic ring (FHEW, TFHE)

Homomorphic Matrix Computation

Question. How to efficiently perform matrix computation over encrypted data?

Inside FHE

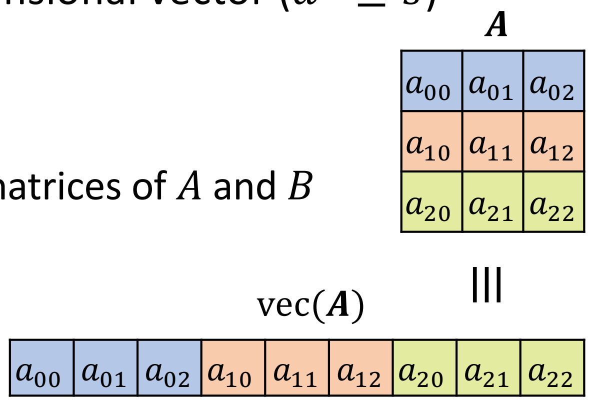
- **Slot**-encoding & HE-friendly expression of MM
  - Matrix ⇒ $1d$-vector
  - Matrix operations ⇒ composite of vectorized operations
- Related Work
  - Row-major: JKLS'18
  - Column-major: BOLT (PC-MM)
  - Diagonal-major: HS'14 (Mat×vec)

Outside FHE

- **Coefficient**-encoding & reduction from encrypted MM to plaintext MMs
  - Multiply matrix directly on the decryption equation
  - Enable regular matrix computation optimizations
  - Related work: [B+24, P25]

Inside FHE

- Row-major matrix encoding: identify $(d{\times}d)$ matrix to d^2-dimensional vector ($d^2 \leq s$)
- **(Vectorized) matrix computation** [JKLS'18]
  - $\mathrm{vec}(A+B) = \mathrm{vec}(A) \oplus \mathrm{vec}(B)$
  - $\mathrm{vec}(AB) = \sum \mathrm{vec}(A_i) \odot \mathrm{vec}(B_i)$ where A_i, B_i are permuted matrices of A and B
    - $\mathrm{vec}(A), \mathrm{vec}(B) \xrightarrow{\text{Linear Transf}} \mathrm{vec}(A_0), \mathrm{vec}(B_0)$
    - A_i = Column-shift(A_0,i); B_i = Row-shift(B_0,i)

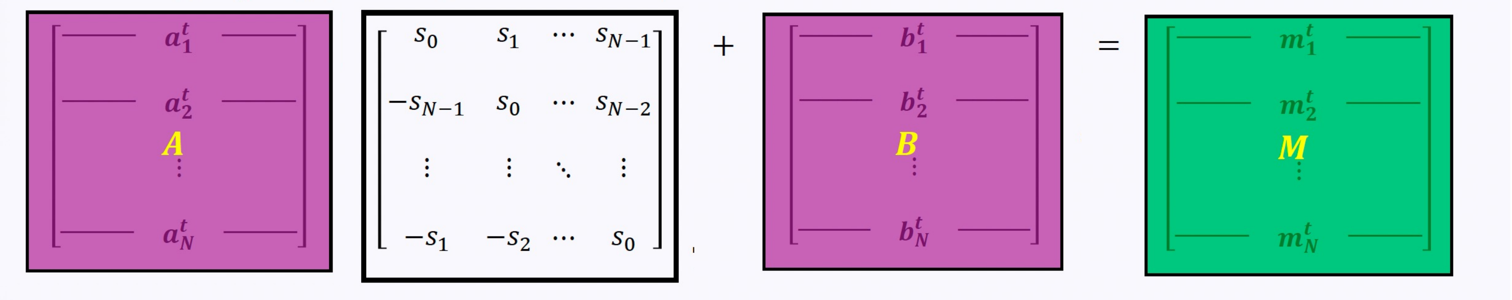


- **Limitation**
  - Small matrix size assumption: $d^2 \leq s$
    - But need large-scale MM (e.g., (768x768)x(768x128))
    - Block-wise computation [C+20]
  - Originally proposed for CC-MM; Not optimal for plaintext-ciphertext multiplication (PC-MM)
    - A_i or B_i can be provided in cleartext (e.g., feed-forward, linear projection)
  - Matrix transposition: $\mathcal{O}(d)$ complexity

[JKLS18] X. Jiang, **M. Kim**, K. Lauter, Y. Song. "Secure outsourced matrix computation and application to neural networks", CCS 2018
[C+20] H. Chen, **M. Kim,** I. Razenshteyn, D. Rotaru, Y. Song, S. Wagh. "Maliciously secure MM with applications to Private DL". Asiacrypt 2020.

Outside FHE

- **Reduction from encrypted MM to plaintext MMs** [B+24, P25]
  - A RLWE ciphertext of $m(X) = \sum m_j X^j \Leftrightarrow$ plaintext vector of (m_j)
  - RLWE ciphertexts of $m_i(X) = \sum m_{ij} X^j \Leftrightarrow$ plaintext matrix of (m_{ij})
  - $AS + B = M \ (mod\ q) \quad \Rightarrow (WA)S + (WB) \approx WM (mod\ q)$
- **Limitation**
  - Optimal when #(matrices) $= N$
  - Need a conversion to slot-encoded ciphertext when performing coefficient-wise operations (e.g., nonlinear function evaluation)

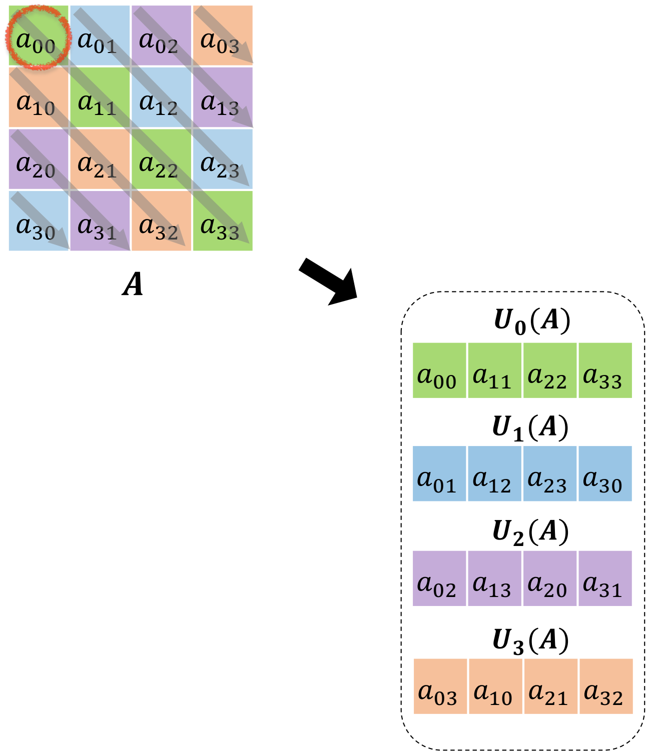


[B+24] Y. Bae, J.H. Cheon, G. Hanrot, J.H. Park, D. Stehle. "Plaintext-ciphertext MM and FHE BTS: fast and fused", CRYPTO 2024
[P25] J.H. Park. "Ciphertext-ciphertext MM: fast for large matrices", Eurocrypt 2025

Slide courtesy to Dr. Park

Table of Contents

Diagonal-major Matrix Encoding

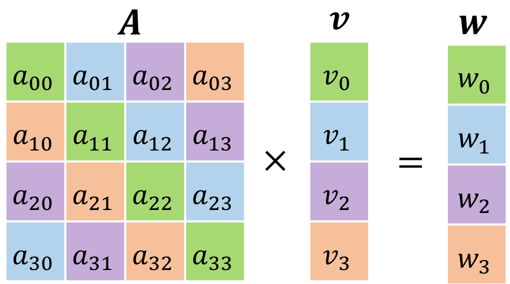


Upper diagonal

[HS14] S. Halevi, V. Shoup. “Algorithms in HElib”, CRYPTO 2014

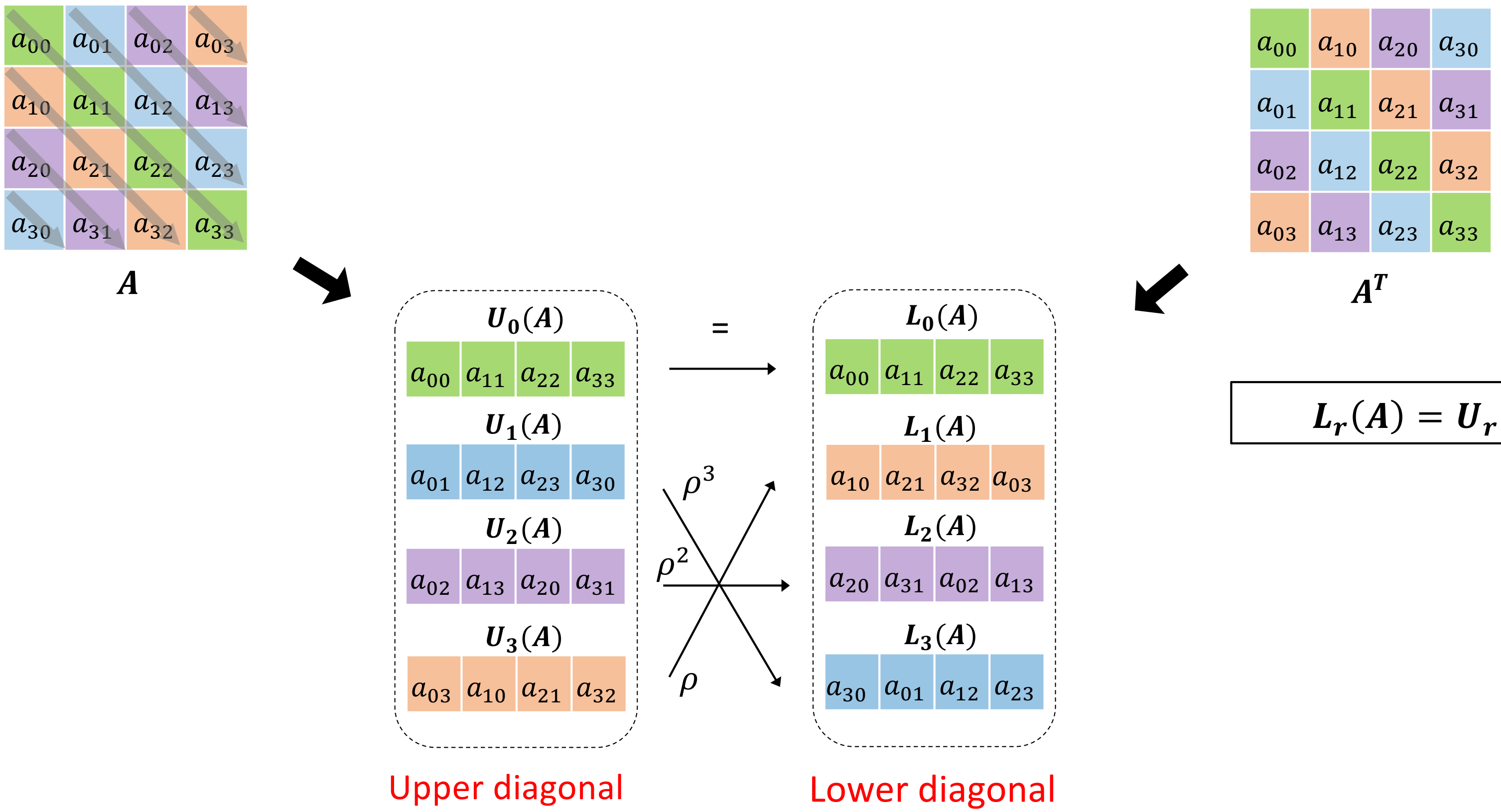
Matrix-Vector Multiplication

- [HS14] "If an input matrix is provided in plaintext or we can process it before the multiplication, the best solution is to put it in diagonal order, which would lets us use the parallel *systolic* multiplication algorithm."
  - $w = U_0(A) \odot v + U_1(A) \odot \rho(v) + \cdots + U_{d-1}(A) \odot \rho^{d-1}(v)$
  - Perform in a systolic array
    - Multiply between diagonal and rotated vector
    - Accumulate
    - Pass data to neighbor



[HS14] S. Halevi, V. Shoup. "Algorithms in HElib", CRYPTO 2014

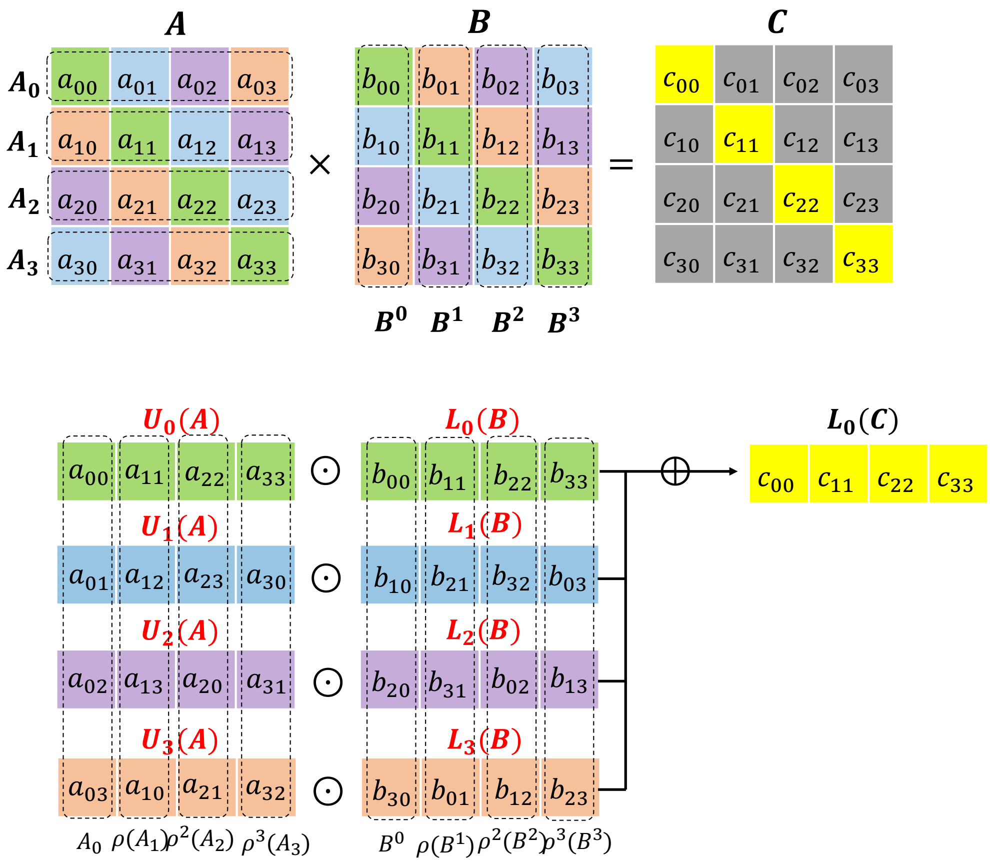
Diagonal-major Matrix Encoding



$$L_r(A) = U_r(A^T)$$

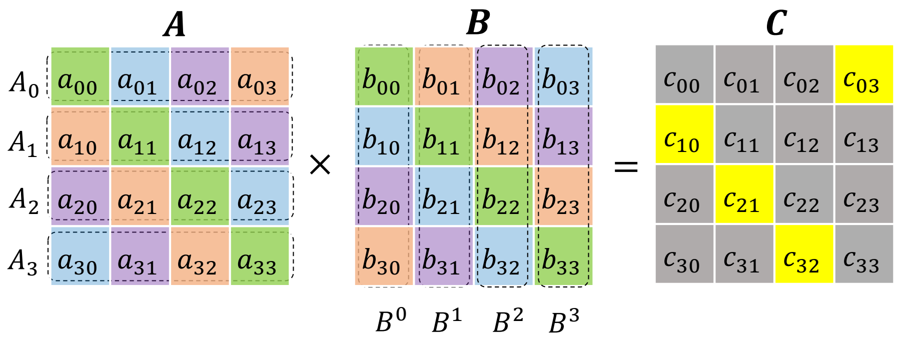
*ρ^k: *left-rotation by k positions*

Main Idea

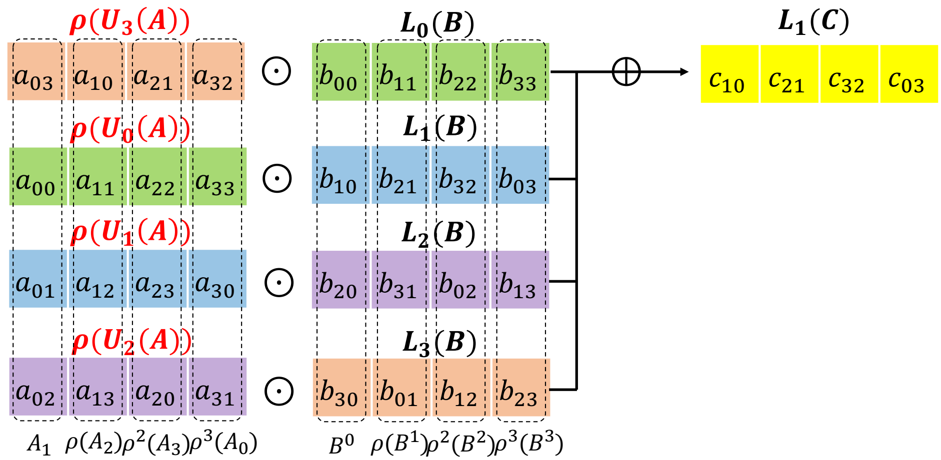


- $\boldsymbol{L_0(C)}$: the 0th lower diagonal
  - $c_{00} = \langle A_0, B^0 \rangle$
  - $c_{11} = \langle A_1, B^1 \rangle = \langle \rho(A_1), \rho(B^1) \rangle$
  - $c_{22} = \langle A_2, B^2 \rangle = \langle \rho^2(A_2), \rho^2(B^2) \rangle$
  - $c_{33} = \langle A_3, B^3 \rangle = \langle \rho^3(A_3), \rho^3(B^3) \rangle$

Main Idea



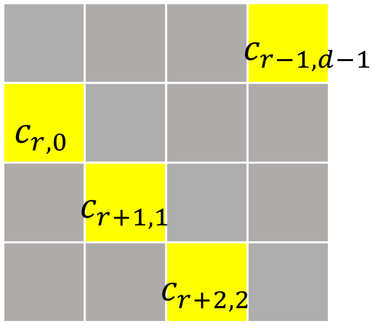
- $\boldsymbol{L_1(C)}$: the 1st lower diagonal
  - $c_{10} = \langle A_1, B^0 \rangle$
  - $c_{21} = \langle A_2, B^1 \rangle = \langle \rho(A_2), \rho(B^1) \rangle$
  - $c_{32} = \langle A_3, B^2 \rangle = \langle \rho^2(A_3), \rho^2(B^2) \rangle$
  - $c_{03} = \langle A_0, B^3 \rangle = \langle \rho^3(A_0), \rho^3(B^3) \rangle$



Basic Square Matrix Multiplication

- Given $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d \times d}$, the r-th lower diagonal $\boldsymbol{L_r}(\boldsymbol{C})$ can be obtained as follows:
  - $c_{r,0} = \langle \boldsymbol{A_r}, \boldsymbol{B^0} \rangle$
  - $c_{r+1,1} = \langle \boldsymbol{A_{r+1}}, \boldsymbol{B^1} \rangle = \langle \rho(\boldsymbol{A_{r+1}}), \rho(\boldsymbol{B^1}) \rangle$

    $\vdots$
  - $c_{r-1,d-1} = \langle \boldsymbol{A_{r-1}}, \boldsymbol{B^{d-1}} \rangle = \langle \rho^{d-1}(\boldsymbol{A_{r-1}}), \rho^{d-1}(\boldsymbol{B^{d-1}}) \rangle$



$$L_r(C) = \left(c_{r,0}, c_{r+1,1}, \dots, c_{r-1,d-1}\right) = \sum_{0 \le l < d} \rho^r\left(U_{l-r}(\boldsymbol{A})\right) \odot L_l(\boldsymbol{B})$$

- In the context of HE computation,
  - PC-MM: When $L_l(\boldsymbol{B})$ is encrypted, $[\![L_r(\boldsymbol{C})]\!] = \sum_{l=0}^{d-1} \mathrm{PMult}(\rho^r\left(U_{l-r}(\boldsymbol{A})\right), [\![L_l(\boldsymbol{B})]\!])$
  - CC-MM: When both are encrypted, $[\![L_r(\boldsymbol{C})]\!] = \sum_{l=0}^{d-1} \mathrm{Mult}(\rho^r([\![U_{l-r}(\boldsymbol{A})]\!]), [\![L_l(\boldsymbol{B})]\!])$

HE-friendly Matrix Multiplication

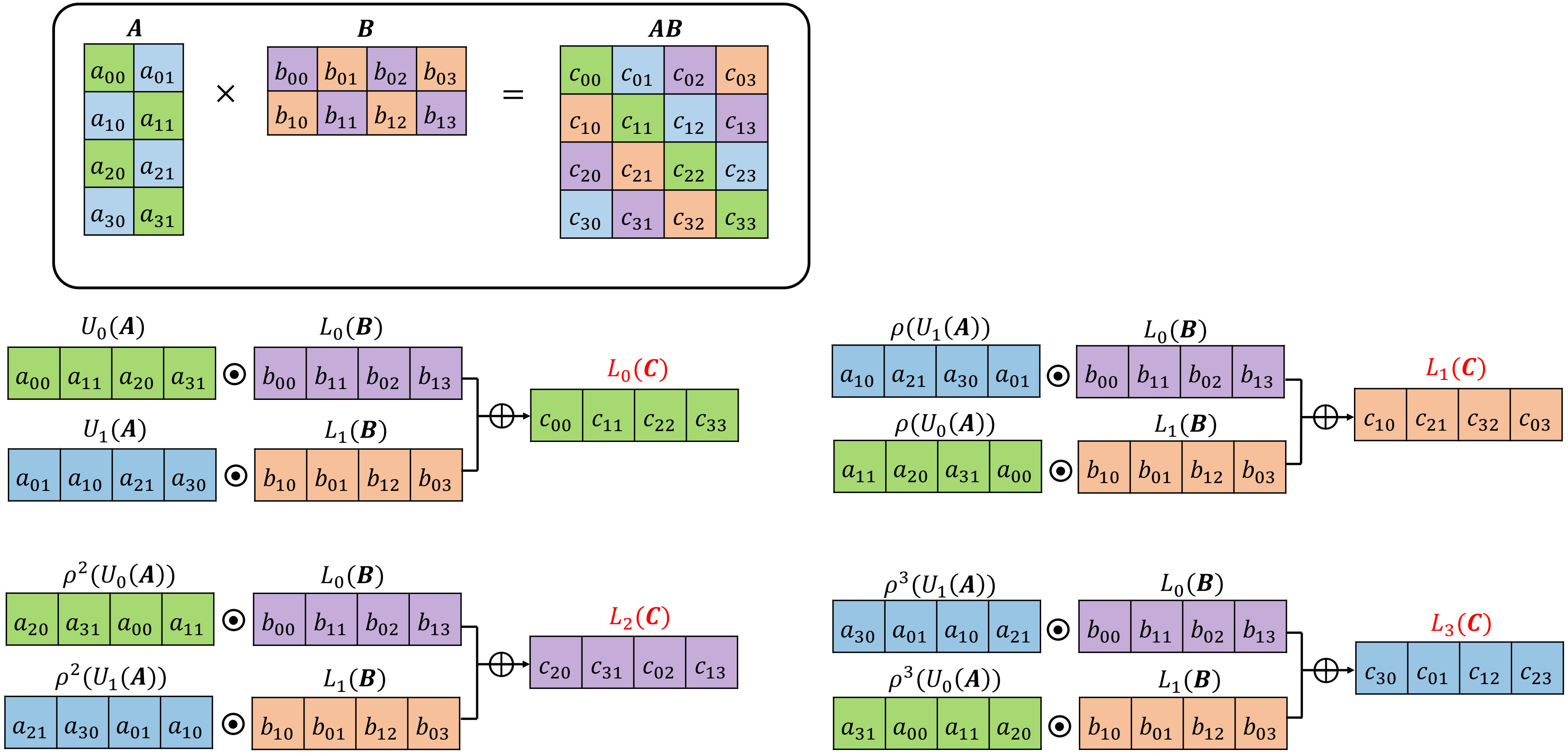
$$\boldsymbol{L_r}(\boldsymbol{C}) = \left(C_{r,0}, C_{r+1,1}, \dots, C_{r-1,d-1}\right)$$
$$= \sum_{l=0}^{d-1} \rho^r\left(U_{l-r}(A)\right) \odot L_l(B)$$

(PC-MM) $[\![L_r(\boldsymbol{C})]\!] = \sum_{l=0}^{d-1} \mathrm{Mult}(\rho^r\left(U_{l-r}(\boldsymbol{A})\right), [\![L_l(\boldsymbol{B})]\!])$

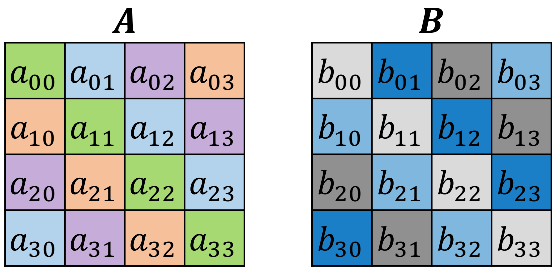
(CC-MM) $[\![L_r(\boldsymbol{C})]\!] = \sum_{l=0}^{d-1} \mathrm{Mult}(\rho^r([\![U_{l-r}(\boldsymbol{A})]\!]), [\![L_l(\boldsymbol{B})]\!])$

- ✓ Easy to implement
- ✓ Optimized for both PC-MM and CC-MM
- ✓ Matrix transposition for free but a different format ($\boldsymbol{L_r}(\boldsymbol{A}) = \boldsymbol{U_r}(\boldsymbol{A}^T)$)
- ✓ *Unified* encrypted matrix representation (reusable for subsequent computations)
- ✓ Easily extended to parallel matrix multiplications
  (interlace multiple matrices & batch the computation)
- ✓ Easily extended to matrix multiplication of various sizes

Non-Square Matrix Multiplication

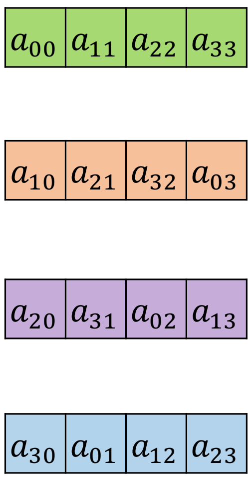


Multi-diagonal Batched Encryption

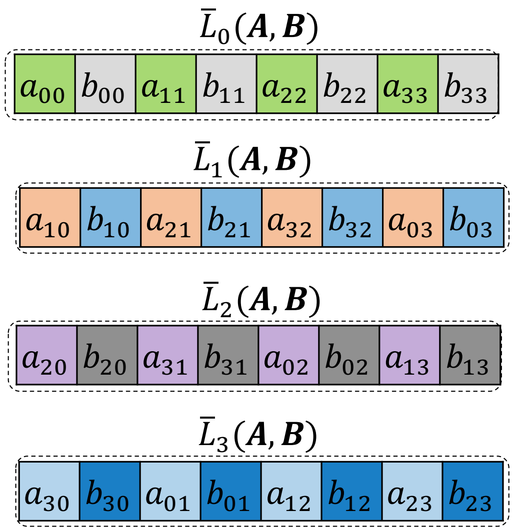


- Diagonal packing capacity $c := s/(nH)$
  (n: dimension, H: number of parallelized matrices)
- ***Batch ciphertext computation***

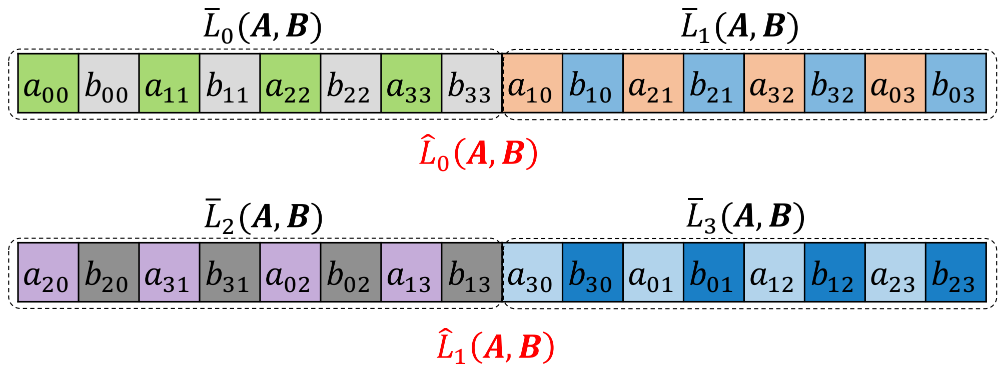
- **Case 1**: $s = 4$



- **Case 2**: $s = 8$



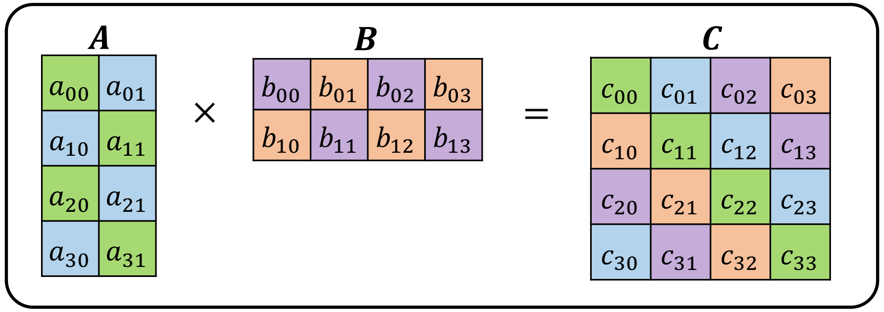
- **Case 3**: $s = 16 \quad \Rightarrow c = 2$



e.g., $s = 2^{15}, H = 16,\ n = 2^7$

$\Rightarrow c = \frac{2^{15}}{2^7 \cdot 16} = 2^4$ different diagonals in one ciphertext

Non-Square Matrix Multiplication (Packed Version)



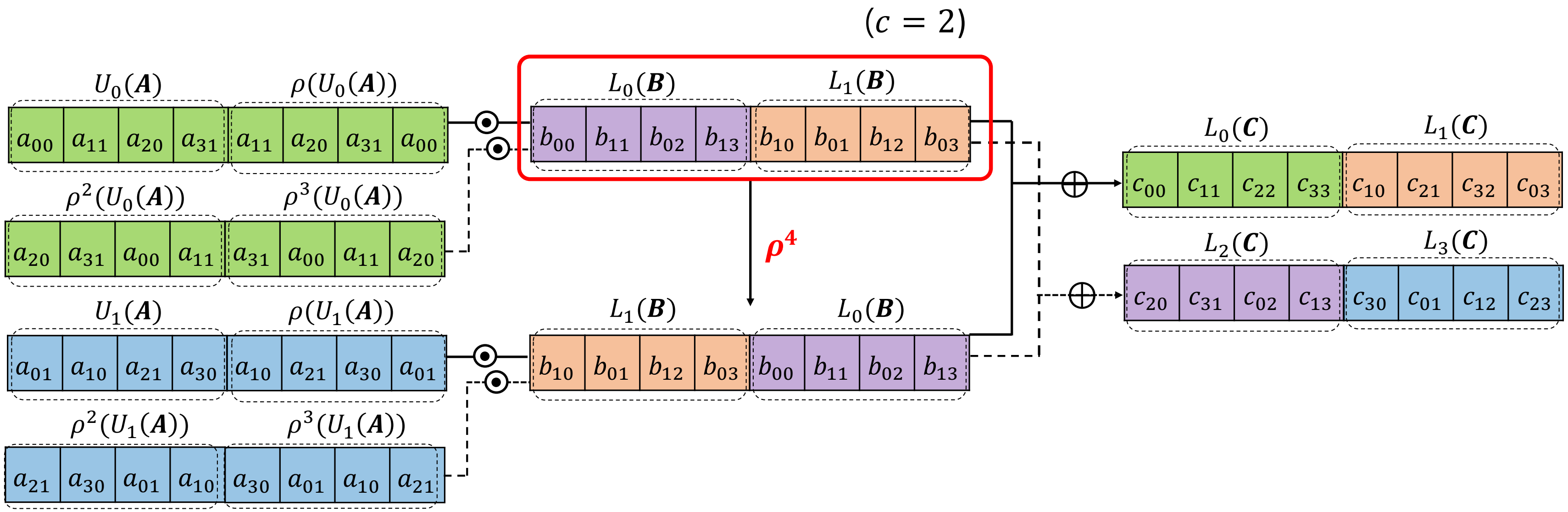
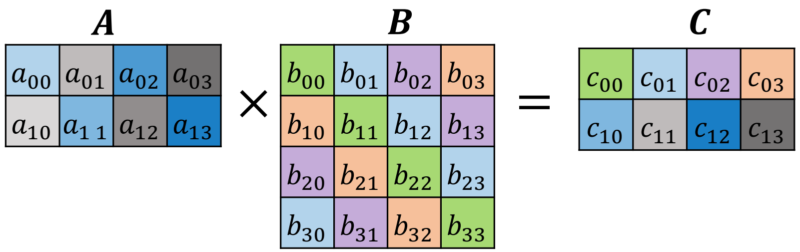


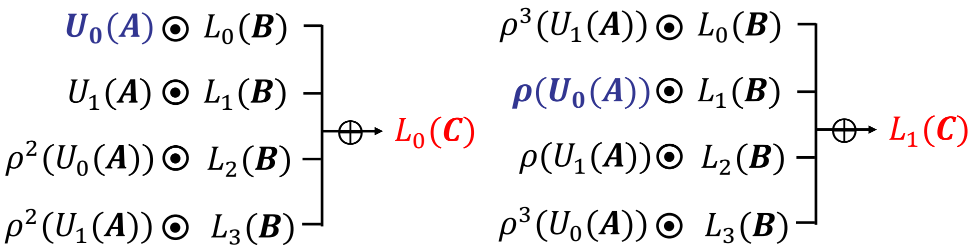
Table of Contents

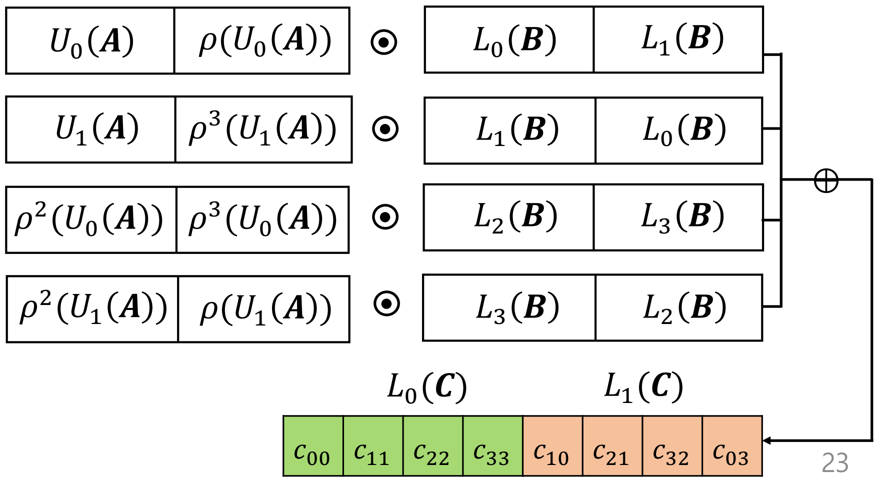
M1. CC-MM: U-L Approach

- Given encrypted $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ with $m \le n$, compute $\boldsymbol{AB}$
  - Each ciphertext of $\boldsymbol{A}$ contains the *same diagonal* of $\boldsymbol{A}$ but *rotated differently* (need mn rotations)



$(m+1)n$ rotations
for $\boldsymbol{A}^{(i)} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B}^{(i)} \in \mathbb{R}^{n\times n}$

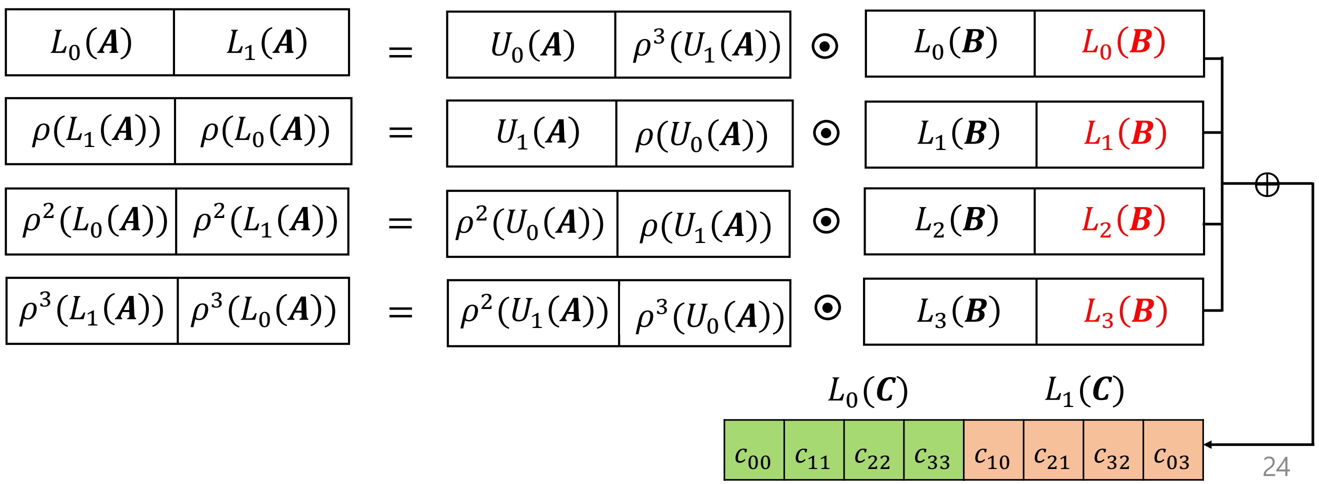




M2. CC-MM: L-L Approach

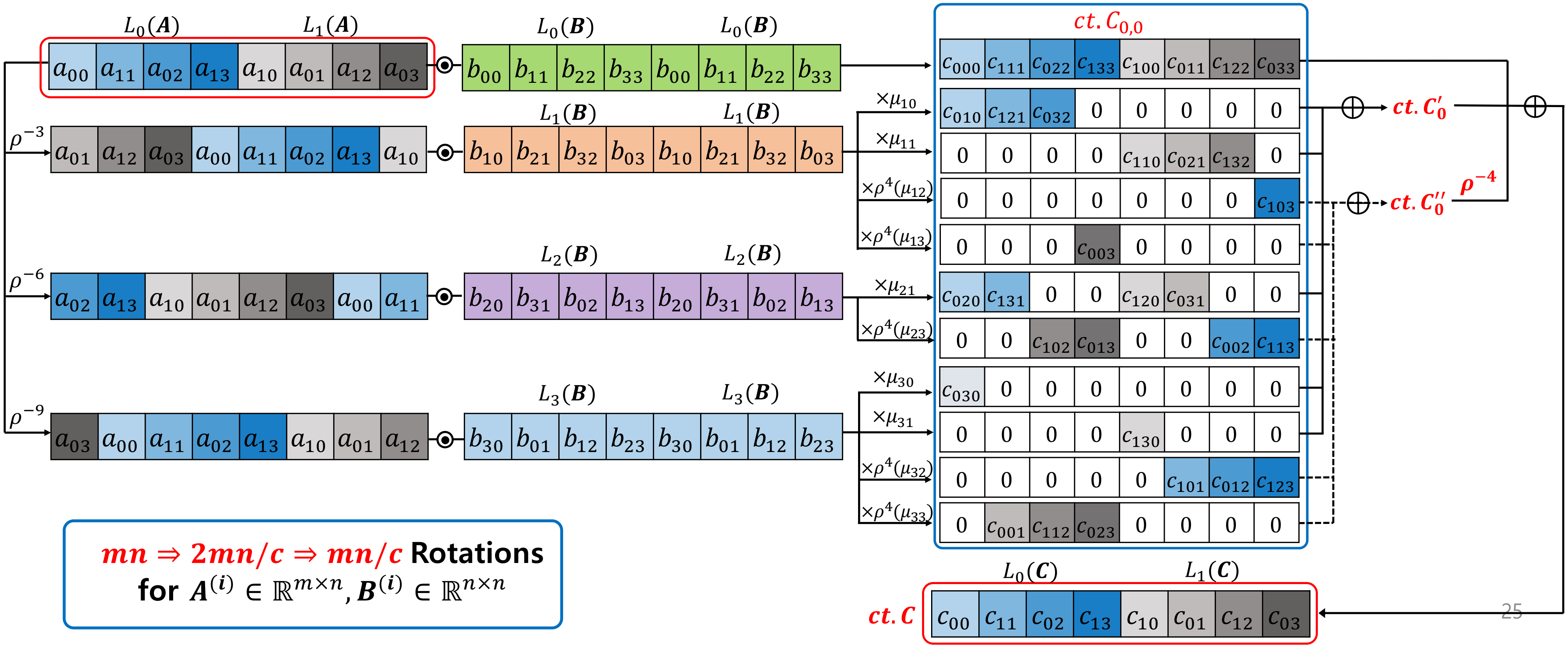
- Given encrypted $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ with $m \le n$, compute $\boldsymbol{AB}$
  - How about packing *replicated* diagonals of B?
  - Use $U_0(\boldsymbol{A}) = L_0(\boldsymbol{A})$, $U_1(\boldsymbol{A}) = \rho\big(L_1(\boldsymbol{A})\big)$
  - Each ciphertext of $\boldsymbol{A}$ contains the *different lower diagonal* but rotated by *the same amount*.

    $\Rightarrow$ $\mathbf{2}\boldsymbol{mn/c}$ rotations for $\left(\boldsymbol{A}^{(i)}\boldsymbol{B}^{(i)}\right)_{1\le i\le H}$ (where $c = s/nH$).



M2. CC-MM: BSGS-integrated L-L Approach

- Rotate the ciphertexts of $\boldsymbol{A}$ internally ⇒ rotate them at first, and apply *only a single rotation* to the summed intermediate result.



$mn \Rightarrow 2mn/c \Rightarrow mn/c$ **Rotations**
for $\boldsymbol{A}^{(i)} \in \mathbb{R}^{m\times n}, \boldsymbol{B}^{(i)} \in \mathbb{R}^{n\times n}$

Comparison

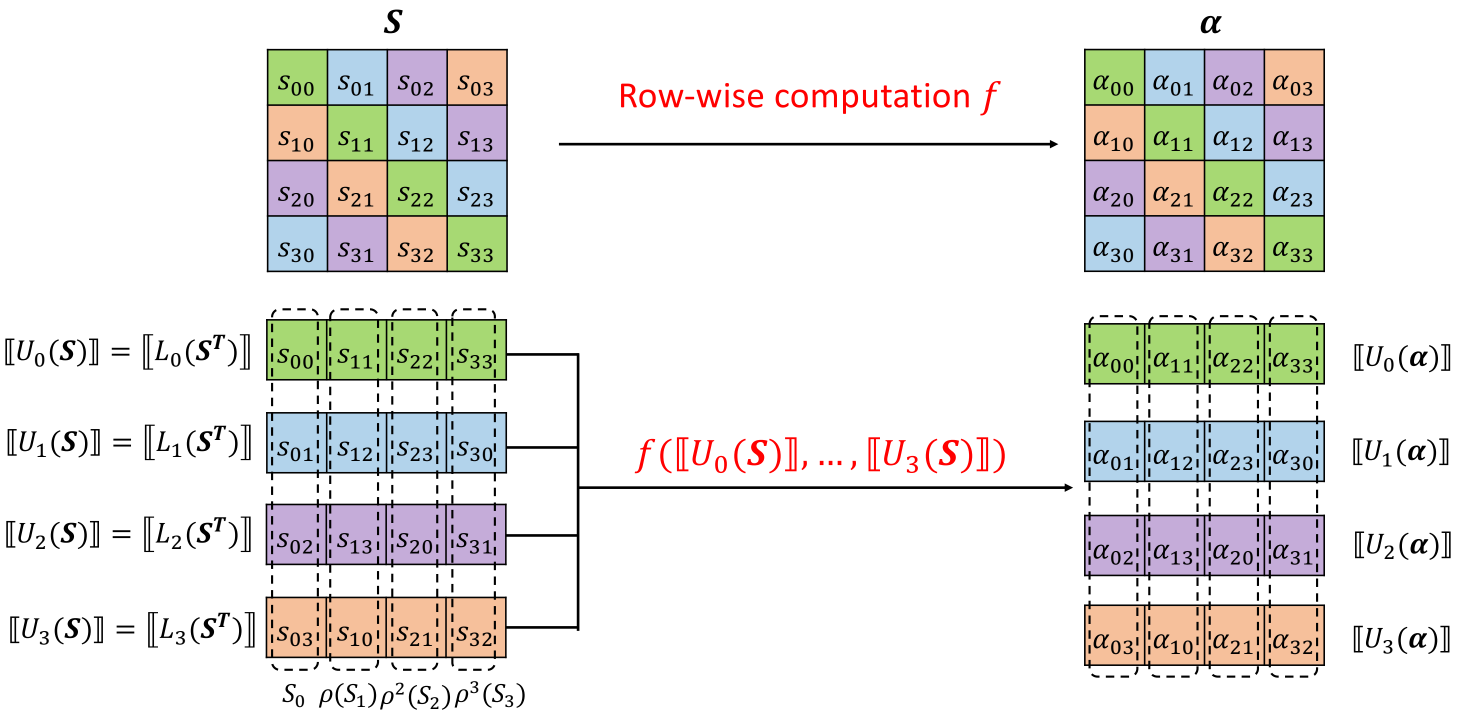
	JKLS'18	Our work
Encoding format	Row-major	Diagonal-major
Parallel large-scale matrix multiplication	✗	✓
PC-MM	Inefficient	Efficient
Transposition	Non-trivial	For free; but a different format

	Equation	H				
		1	2	4	8	16
JKLS'18	$H(5\sqrt{n}+4n)$	572	1144	2272	4576	9152
Our work	$H(n/2+3)+n/4(\log(n/H)+3)$	387	422	520	760	1264

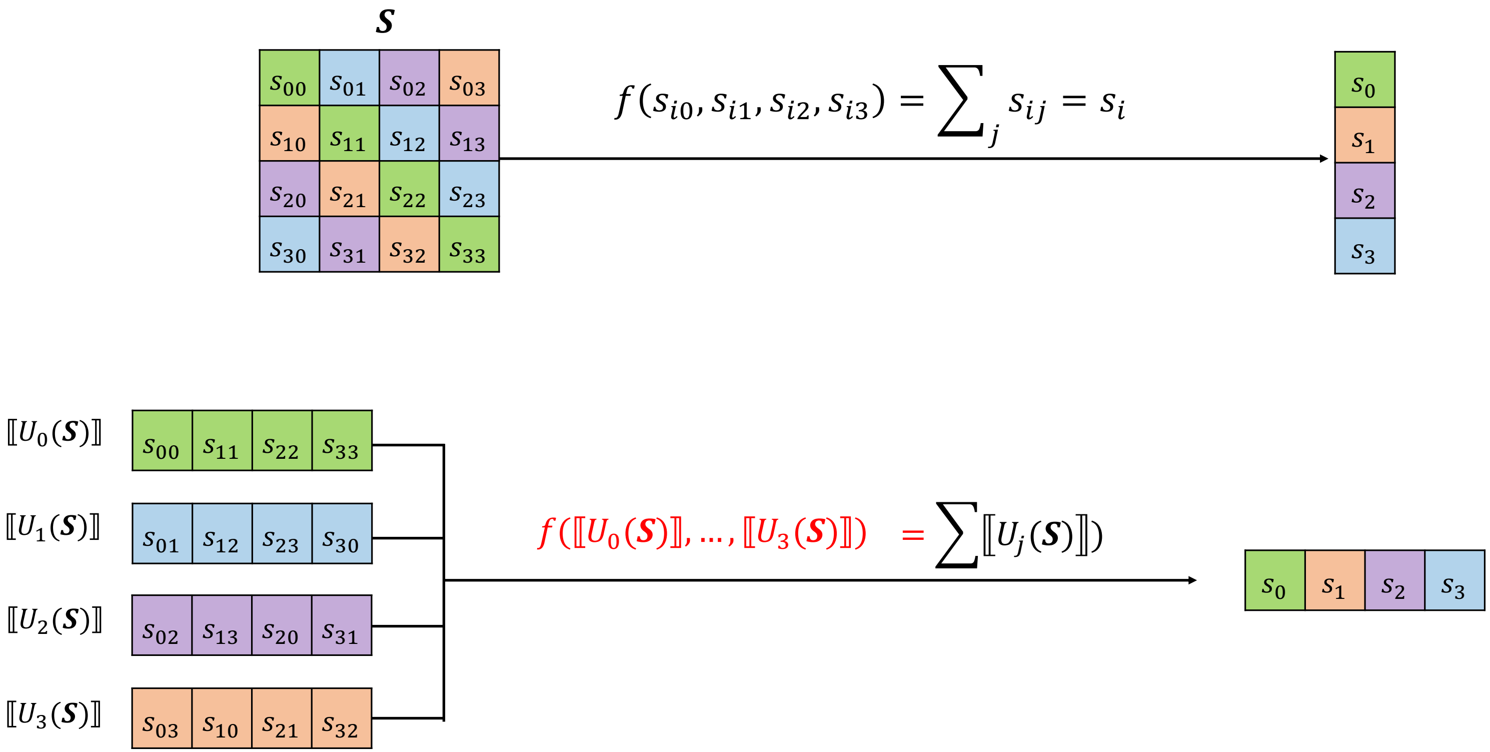
CC-MM computation of $A^{(z)}B^{(z)}$ for $1 \leq z \leq H$,
where $A^{(z)}, B^{(z)} \in \mathbb{R}^{n\times n}$ and $s = n^2$
$(n = 2^7, s = 2^{14})$

Row-wise Computation over Diagonals

- If the upper diagonals of $\boldsymbol{S}$ are provided (= lower diagonals of $\boldsymbol{S}^{\boldsymbol{T}}$)
  - Each row of $\boldsymbol{S}$ is distributed across the same position in different slots.
  - Apply f row-wise on $\boldsymbol{S}$ $\Rightarrow$ apply f directly to ciphertexts (no computation is needed between slots)

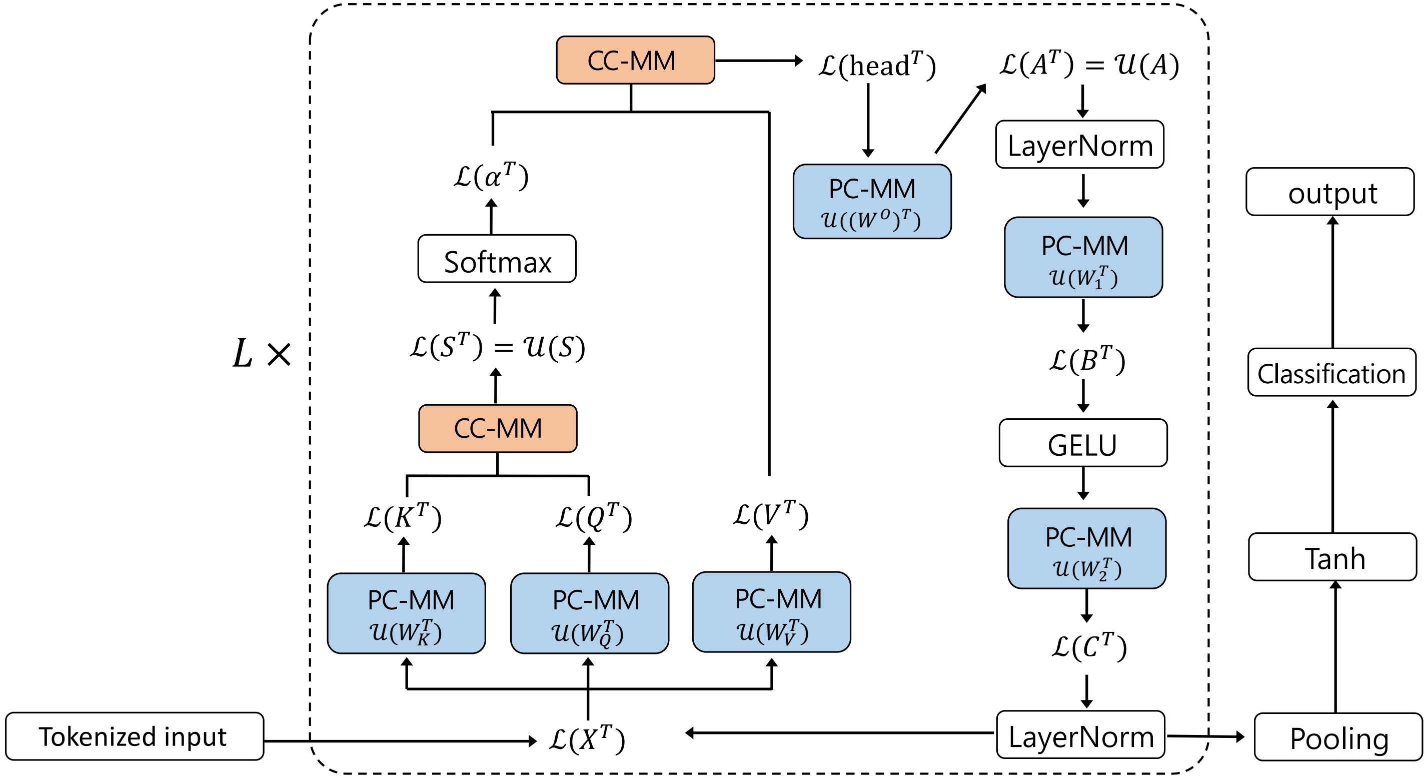


Example


$\boldsymbol{S}$
s_{00} s_{01} s_{02} s_{03}
s_{10} s_{11} s_{12} s_{13}
s_{20} s_{21} s_{22} s_{23}
s_{30} s_{31} s_{32} s_{33}
$f(s_{i0}, s_{i1}, s_{i2}, s_{i3}) = \sum_j s_{ij} = s_i$
s_0 s_1 s_2 s_3
$[\![U_0(\boldsymbol{S})]\!]$ s_{00} s_{11} s_{22} s_{33}
$[\![U_1(\boldsymbol{S})]\!]$ s_{01} s_{12} s_{23} s_{30}
$[\![U_2(\boldsymbol{S})]\!]$ s_{02} s_{13} s_{20} s_{31}
$[\![U_3(\boldsymbol{S})]\!]$ s_{03} s_{10} s_{21} s_{32}
$f([\![U_0(\boldsymbol{S})]\!], \ldots, [\![U_3(\boldsymbol{S})]\!]) = \sum [\![U_j(\boldsymbol{S})]\!])$
s_0 s_1 s_2 s_3

Application to Transformer-based Inference

- THOR: secure Transformer inference framework with HomomOrphic encRyption



Experimental Results

- BERT-base model (L =12 layers, 110M parameters, n =128 tokens)
- **10 minutes** with only a 0.8% accuracy drop (2.7hours in NEXUS): **16x** improvement
  - Softmax: square-and-normalization (see paper)
  - Use the Goldschmidt's algorithm + Adaptive Successive over-relaxation method (aSOR)

Intel Xeon Platinum 8462 at 2.8GHz,
A100 GPU (DESILO FHE library)

Operation	Input	Time (sec)
Attention layer	$3 \times (\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times64})$	49.77
Attention score	$12 \times (\mathbb{R}^{128\times64} \times \mathbb{R}^{64\times128})$	16.25
Softmax	$12 \times (\mathbb{R}^{128\times128})$	15.53
Attention head	$12 \times (\mathbb{R}^{128\times128} \times \mathbb{R}^{128\times64})$	13.08
Multi-head attention	$\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times768}$	27.43
LayerNorm1	$\mathbb{R}^{128\times768}$	7.13
FC1	$\mathbb{R}^{128\times768} \times \mathbb{R}^{768\times3072}$	49.80
GELU	$\mathbb{R}^{128\times3072}$	29.42
FC2	$\mathbb{R}^{128\times3072} \times \mathbb{R}^{3072\times768}$	49.19
LayerNorm2	$\mathbb{R}^{128\times768}$	4.10
Pooler & Classification	$\mathbb{R}^{128\times768}$	2.70
Bootstrappings	-	337.86
Total	-	**602.26**

Dataset	#Test	Metric	Unencrypted				Encrypted
			Baseline	G	G-LN	G-LN-S	
MPRC	408	Accuracy	85.29	85.54	85.54	85.78	**84.80**
		F1-score	89.90	90.05	90.05	90.24	**89.49**
RTE	277	Accuracy	72.20	71.48	71.84	72.20	**71.12**
SST-2	872	Accuracy	91.51	91.40	91.40	91.63	**90.71**

MRPC: Sentence pair 2-class paraphrase

RTE: Premise/hypothesis 2-class entailment

SST-2: Single-sentence 2-class paraphrase

[Zhang+25] "Secure transformer inference made noninteractive", NDSS'25

Table of Contents

Summary

- **New Efficient parallel matrix computation**
  - Block-wise PC-MM
  - Optimized CC-MM using the BSGS strategy
  - Row-wise computation over diagonals

- **Question.** Can the proposed matrix computation be further optimized?
  - Level consumption (vs. 3 levels in JKLS18)
    - PC-MM: 2 levels (one for masking and one for plaintext mult)
    - CC-MM: 2 levels (one for making and one for ciphertext mult)
  - Need *replication* during CC-MM
    - Generate replicated (batched) lower diagonals
    - $\frac{n}{2}(\log c + 1)$ rotations for $\boldsymbol{B}^{(i)} \in \mathbb{R}^{n \times n}$ for $1 \leq i \leq H$ with one level ($c = s/nH$)

Future Works

- Extension
  - Low Latency vs *high-throughput* batch processing
  - Extend secure inference to larger parameter LLMs (e.g., LLaMA)
  - Privacy-preserving training for LLMs

- Potential applications beyond LLM
  - Genomic analysis (e.g., genotype imputation)
  - Machine learning (e.g., neural networks, transfer learning)

Implementation available at:
`https://github.com/crypto-starlab/THOR`

ia.cr/2024/1881