Encrypted matrix operations via bicyclic encoding

陈经纬

中国科学院重庆绿色智能技术研究院
ChongQing Institute of Green and Intelligent Technology,Chinese Academy of Sciences

Joint work with Linhan Yang, Wenyuan Wu, Yang Liu, and Yong Feng

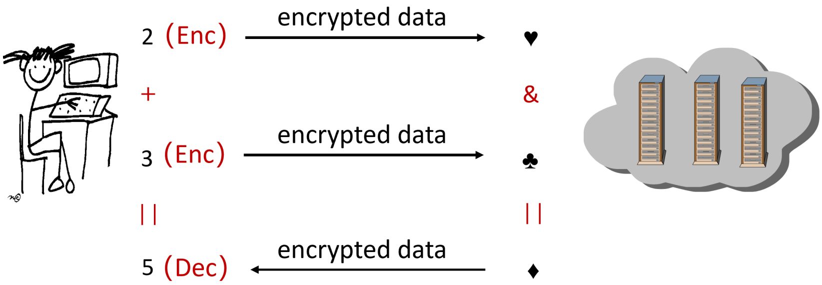
Dec. 18, 2024

Roadmap

1. **Homomorphic encryption**
   - Basics
2. **Plaintext matrix–ciphertext vector multiplication**
   - Fast linear transformation by Halevi-Shoup (2014, 2015)
3. **Ciphertext matrix–ciphertext matrix multiplication**
   - HE-friendly expression by Jiang et al. (2018)
4. **Bicyclic encoding and matrix multiplication**
   - Ciphertext – ciphertext matrix multiplication
   - Performance

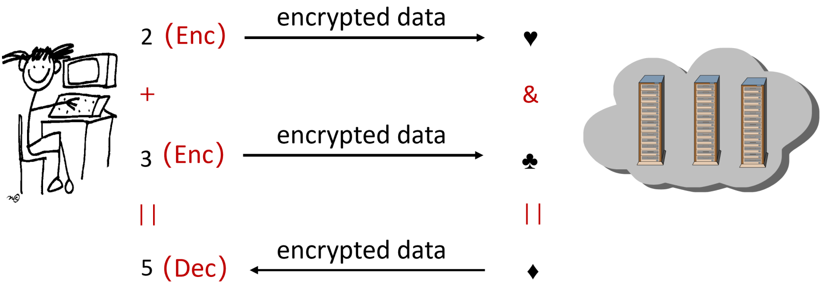
1. **Homomorphic encryption**
   - Basics
2. **Plaintext matrix–ciphertext vector multiplication**
   - Fast linear transformation by Halevi-Shoup (2014, 2015)
3. **Ciphertext matrix–ciphertext matrix multiplication**
   - HE-friendly expression by Jiang et al. (2018)
4. **Bicyclic encoding and matrix multiplication**
   - Ciphertext – ciphertext matrix multiplication
   - Performance

Homomorphic encryption (HE)



- ✓ Communication security
- ✓ Storage security
- ✓ Computation security
- ✓ Post-quantum security

Homomorphic encryption (HE)



- ✓ Communication security
- ✓ Storage security
- ✓ Computation security
- ✓ Post-quantum security

Applications

- Secure outsourcing
- Non-interactive computation
- Privacy-preserving ML

A very brief history on HE

- 1^{st} generation (2009-2012)
  - Plausibility: Gentry, DGHV

A very brief history on HE

- 1^{st} generation (2009-2012)
  - Plausibility: Gentry, DGHV
- 2^{nd} generation (2012-2015)
  - Optimizations: BGV, B/FV
  - Proof of concept implementations

A very brief history on HE

- 1^{st} generation (2009-2012)
  - Plausibility: Gentry, DGHV
- 2^{nd} generation (2012-2015)
  - Optimizations: BGV, B/FV
  - Proof of concept implementations
- 3^{rd} and 4^{th} generation, and more (2015-present)
  - Fast bootstrapping: DM, CGGI, ...
  - Real number arithmetic: CKKS
  - Further optimizations:
    - $\rightsquigarrow$ LW '23, XZDDF '23, MHWW '24, WWLLWLW '24, ...
  - Public libraries: HElib, SEAL, OpenFHE, TFHE, Lattigo, ...
  - Practical applications: iDASH competition, Standardization, ...

Functionality of SIMD-supported HE schemes

- Packing method
  - Enc: $\boldsymbol{x} \mapsto \mathtt{ct} = \mathtt{Enc}(\boldsymbol{x})$, where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$

Functionality of SIMD-supported HE schemes

- Packing method
  - Enc: $\boldsymbol{x} \mapsto \mathtt{ct} = \mathtt{Enc}(\boldsymbol{x})$, where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$
- Homomorphic arithmetic (SIMD)
  - Add: $\mathtt{Enc}(\boldsymbol{x}) + \mathtt{Enc}(\boldsymbol{y}) = \mathtt{Enc}(\boldsymbol{x} + \boldsymbol{y})$
  - Mul: $\mathtt{Enc}(\boldsymbol{x}) \times \mathtt{Enc}(\boldsymbol{y}) = \mathtt{Enc}(\boldsymbol{x} \circ \boldsymbol{y})$
    - ⇝ $\circ$: Hadamard product, i.e., element-wise multiplication
  - CMul: $\boldsymbol{a} \times \mathtt{Enc}(\boldsymbol{x}) = \mathtt{Enc}(\boldsymbol{a} \circ \boldsymbol{x})$

Functionality of SIMD-supported HE schemes

- Packing method
  - $\texttt{Enc}$: $\boldsymbol{x} \mapsto \texttt{ct} = \texttt{Enc}(\boldsymbol{x})$, where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$
- Homomorphic arithmetic (SIMD)
  - $\texttt{Add}$: $\texttt{Enc}(\boldsymbol{x}) + \texttt{Enc}(\boldsymbol{y}) = \texttt{Enc}(\boldsymbol{x} + \boldsymbol{y})$
  - $\texttt{Mul}$: $\texttt{Enc}(\boldsymbol{x}) \times \texttt{Enc}(\boldsymbol{y}) = \texttt{Enc}(\boldsymbol{x} \circ \boldsymbol{y})$
    - $\rightsquigarrow$ $\circ$: Hadamard product, i.e., element-wise multiplication
  - $\texttt{CMul}$: $\boldsymbol{a} \times \texttt{Enc}(\boldsymbol{x}) = \texttt{Enc}(\boldsymbol{a} \circ \boldsymbol{x})$
- Rotation
  - $\texttt{Rot}_i$: $\texttt{Enc}(x_0, \ldots, x_{n-1}) \mapsto \texttt{Enc}(x_i, x_{n-1}, x_0, \ldots, x_{i-1})$

Functionality of SIMD-supported HE schemes

- Packing method
  - $\texttt{Enc}$: $\boldsymbol{x} \mapsto \texttt{ct} = \texttt{Enc}(\boldsymbol{x})$, where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$
- Homomorphic arithmetic (SIMD)
  - $\texttt{Add}$: $\texttt{Enc}(\boldsymbol{x}) + \texttt{Enc}(\boldsymbol{y}) = \texttt{Enc}(\boldsymbol{x} + \boldsymbol{y})$
  - $\texttt{Mul}$: $\texttt{Enc}(\boldsymbol{x}) \times \texttt{Enc}(\boldsymbol{y}) = \texttt{Enc}(\boldsymbol{x} \circ \boldsymbol{y})$
    - $\rightsquigarrow$ $\circ$: Hadamard product, i.e., element-wise multiplication
  - $\texttt{CMul}$: $\boldsymbol{a} \times \texttt{Enc}(\boldsymbol{x}) = \texttt{Enc}(\boldsymbol{a} \circ \boldsymbol{x})$
- Rotation
  - $\texttt{Rot}_i$: $\texttt{Enc}(x_0, \ldots, x_{n-1}) \mapsto \texttt{Enc}(x_i, x_{n-1}, x_0, \ldots, x_{i-1})$
  - Cost: no multiplicative depth but a key switching

Functionality of SIMD-supported HE schemes

- Packing method
  - Enc: $\boldsymbol{x} \mapsto \mathtt{ct} = \mathtt{Enc}(\boldsymbol{x})$, where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$
- Homomorphic arithmetic (SIMD)
  - Add: $\mathtt{Enc}(\boldsymbol{x}) + \mathtt{Enc}(\boldsymbol{y}) = \mathtt{Enc}(\boldsymbol{x} + \boldsymbol{y})$
  - Mul: $\mathtt{Enc}(\boldsymbol{x}) \times \mathtt{Enc}(\boldsymbol{y}) = \mathtt{Enc}(\boldsymbol{x} \circ \boldsymbol{y})$
    - $\rightsquigarrow$ $\circ$: Hadamard product, i.e., element-wise multiplication
  - CMul: $\boldsymbol{a} \times \mathtt{Enc}(\boldsymbol{x}) = \mathtt{Enc}(\boldsymbol{a} \circ \boldsymbol{x})$
- Rotation
  - $\mathtt{Rot}_i$: $\mathtt{Enc}(x_0, \ldots, x_{n-1}) \mapsto \mathtt{Enc}(x_i, x_{n-1}, x_0, \ldots, x_{i-1})$
  - Cost: no multiplicative depth but a key switching
- Compositions of the basic operations
  - TSum: $\mathtt{Enc}(x_0, \ldots, x_{n-1}) \mapsto \mathtt{Enc}(s, \ldots, s)$ with $s = x_0 + \cdots + x_{n-1}$
  - Cost of TSum: $\#\mathtt{Rot} = \log n$

Roadmap

1. **Homomorphic encryption**
   - Basics
2. **Plaintext matrix–ciphertext vector multiplication**
   - Fast linear transformation by Halevi-Shoup (2014, 2015)
3. **Ciphertext matrix–ciphertext matrix multiplication**
   - HE-friendly expression by Jiang et al. (2018)
4. **Bicyclic encoding and matrix multiplication**
   - Ciphertext – ciphertext matrix multiplication
   - Performance

Naïve method

Linear transformation on a ciphertext

- A plain $n \times n$ matrix $\boldsymbol{A}$
- A ciphertext $\mathtt{ct} = \mathtt{Enc}(\boldsymbol{x})$ of an n-dimensional vector $\boldsymbol{x}$
- Goal: Compute an encrypted form of $\boldsymbol{Ax}$

Naïve method

Linear transformation on a ciphertext

- A plain $n \times n$ matrix $\boldsymbol{A}$
- A ciphertext $\texttt{ct} = \texttt{Enc}(\boldsymbol{x})$ of an n-dimensional vector $\boldsymbol{x}$
- Goal: Compute an encrypted form of $\boldsymbol{Ax}$

A naïve algorithm

For $i = 0, 1, \ldots, n-1$ do the following

- Compute $\texttt{ct}_i = \texttt{CMul}(\boldsymbol{a}_i, \texttt{ct})$ // $\boldsymbol{a}_i$: the i-th row of $\boldsymbol{A}$; $\texttt{ct}_i = \texttt{Enc}(\boldsymbol{a}_i \circ \boldsymbol{x})$
- Compute $\texttt{ct}_i = \texttt{TSum}(\texttt{ct}_i)$ // $\texttt{ct}_i = \texttt{Enc}(\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle, \ldots, \langle \boldsymbol{a}_i, \boldsymbol{x} \rangle)$

Naïve method

Linear transformation on a ciphertext

- A plain $n \times n$ matrix $\boldsymbol{A}$
- A ciphertext $\texttt{ct} = \texttt{Enc}(\boldsymbol{x})$ of an n-dimensional vector $\boldsymbol{x}$
- Goal: Compute an encrypted form of $\boldsymbol{Ax}$

A naïve algorithm

For $i = 0, 1, \ldots, n-1$ do the following

- Compute $\texttt{ct}_i = \texttt{CMul}(\boldsymbol{a}_i, \texttt{ct})$ // $\boldsymbol{a}_i$: the i-th row of $\boldsymbol{A}$; $\texttt{ct}_i = \texttt{Enc}(\boldsymbol{a}_i \circ \boldsymbol{x})$
- Compute $\texttt{ct}_i = \texttt{TSum}(\texttt{ct}_i)$ // $\texttt{ct}_i = \texttt{Enc}(\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle, \ldots, \langle \boldsymbol{a}_i, \boldsymbol{x} \rangle)$

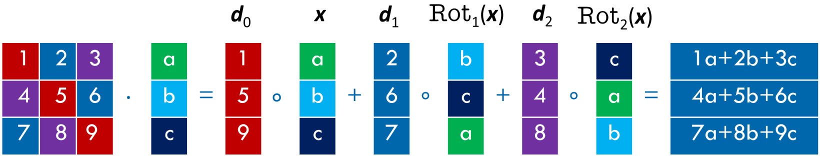
- Cost of the naïve algorithm
  - #resulting ciphertexts: n
  - #CMul: n
  - #Rot: $n \log n$

Diagonal vectors of a matrix and linear transformation*

$$\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix} \cdot \begin{pmatrix}a\\ b\\ c\end{pmatrix} = \underbrace{\begin{pmatrix}1\\ 5\\ 9\end{pmatrix}}_{\boldsymbol{d}_0} \circ \underbrace{\begin{pmatrix}a\\ b\\ c\end{pmatrix}}_{\boldsymbol{x}} + \underbrace{\begin{pmatrix}2\\ 6\\ 7\end{pmatrix}}_{\boldsymbol{d}_1} \circ \underbrace{\begin{pmatrix}b\\ c\\ a\end{pmatrix}}_{\mathrm{Rot}_1(\boldsymbol{x})} + \underbrace{\begin{pmatrix}3\\ 4\\ 8\end{pmatrix}}_{\boldsymbol{d}_2} \circ \underbrace{\begin{pmatrix}c\\ a\\ b\end{pmatrix}}_{\mathrm{Rot}_2(\boldsymbol{x})} = \begin{pmatrix}1a+2b+3c\\ 4a+5b+6c\\ 7a+8b+9c\end{pmatrix}$$

*S. Halevi, V. Shoup, Algorithms in HElib. CRYPTO 2014.

Diagonal vectors of a matrix and linear transformation*



- Diagonal vectors of an $n \times n$ matrix $\boldsymbol{A}$: $\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_{n-1}$
- Now we have

$$\boldsymbol{A}\boldsymbol{x} = \sum_{0 \le i < n} \boldsymbol{d}_i \circ \mathtt{Rot}_i(\boldsymbol{x})$$

*S. Halevi, V. Shoup, Algorithms in HElib. CRYPTO 2014.

Diagonal vectors of a matrix and linear transformation*

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \underbrace{\begin{pmatrix} 1 \\ 5 \\ 9 \end{pmatrix}}_{\boldsymbol{d}_0} \circ \underbrace{\begin{pmatrix} a \\ b \\ c \end{pmatrix}}_{\boldsymbol{x}} + \underbrace{\begin{pmatrix} 2 \\ 6 \\ 7 \end{pmatrix}}_{\boldsymbol{d}_1} \circ \underbrace{\begin{pmatrix} b \\ c \\ a \end{pmatrix}}_{\mathrm{Rot}_1(\boldsymbol{x})} + \underbrace{\begin{pmatrix} 3 \\ 4 \\ 8 \end{pmatrix}}_{\boldsymbol{d}_2} \circ \underbrace{\begin{pmatrix} c \\ a \\ b \end{pmatrix}}_{\mathrm{Rot}_2(\boldsymbol{x})} = \begin{pmatrix} 1a+2b+3c \\ 4a+5b+6c \\ 7a+8b+9c \end{pmatrix}$$

- Diagonal vectors of an $n \times n$ matrix $\boldsymbol{A}$: $\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_{n-1}$
- Now we have

$$\boldsymbol{A}\boldsymbol{x} = \sum_{0 \le i < n} \boldsymbol{d}_i \circ \mathtt{Rot}_i(\boldsymbol{x})$$

- Cost of the fast linear transformation on a ciphertext
  - #resulting ciphertexts: 1
  - #`CMul`: n
  - #`Rot`: n

*S. Halevi, V. Shoup, Algorithms in HElib. CRYPTO 2014.

The baby-step-giant-step (BSGS) strategy[†]

- Reducing $n = \ell \cdot k$ rotations to $\ell + k$

[†] S. Halevi, V. Shoup, Bootstrapping for HElib. EUROCRYPT 2015.

The baby-step-giant-step (BSGS) strategy[†]

- Reducing $n = \ell \cdot k$ rotations to $\ell + k$

$$
\begin{aligned}
\boldsymbol{Ax} &= \sum_{0 \le i < n} \boldsymbol{d}_i \circ \mathtt{Rot}_i(\boldsymbol{x}) \\
&= \sum_{0 \le i < \ell} \sum_{0 \le j < k} \boldsymbol{d}_{ki+j} \circ \mathtt{Rot}_{ki+j}(\boldsymbol{x}) \\
&= \sum_{0 \le i < \ell} \mathtt{Rot}_{ki} \left(\sum_{0 \le j < k} \mathtt{Rot}_{-ki}(\boldsymbol{d}_{ki+j}) \circ \mathtt{Rot}_j(\boldsymbol{x}) \right)
\end{aligned}
$$

plaintext

[†] S. Halevi, V. Shoup, Bootstrapping for HElib. EUROCRYPT 2015.

The baby-step-giant-step (BSGS) strategy[†]

- Reducing $n = \ell \cdot k$ rotations to $\ell + k$

$$
\begin{aligned}
\boldsymbol{Ax} &= \sum_{0 \le i < n} \boldsymbol{d}_i \circ \texttt{Rot}_i(\boldsymbol{x}) \\
&= \sum_{0 \le i < \ell} \sum_{0 \le j < k} \boldsymbol{d}_{ki+j} \circ \texttt{Rot}_{ki+j}(\boldsymbol{x}) \\
&= \sum_{0 \le i < \ell} \texttt{Rot}_{ki} \left(\sum_{0 \le j < k} \texttt{Rot}_{-ki}(\boldsymbol{d}_{ki+j}) \circ \texttt{Rot}_j(\boldsymbol{x}) \right)
\end{aligned}
$$

plaintext

- Cost of the even faster linear transformation on a ciphertext
  - #resulting ciphertexts: 1
  - #CMul: n
  - #Rot: $2\sqrt{n}$

[†] S. Halevi, V. Shoup, Bootstrapping for HElib. EUROCRYPT 2015.

Some sparse linear transformations[‡]

- Assume only $r = \ell \cdot k$ non-zero diag. vectors with indices satisfying
  - $g(ki + j) = g(ki) + g(j)$ for all $0 \leq i < \ell$ and $0 \leq j < k$
- Then we have

$$\boldsymbol{Ax} = \sum_{0 \leq i < \ell} \texttt{Rot}_{g(ki)} \left(\sum_{0 \leq j < k} \texttt{Rot}_{-g(ki)} \left(\boldsymbol{d}_{g(ki+j)} \right) \circ \texttt{Rot}_{g(j)}(\boldsymbol{x}) \right)$$

[‡]X. Jiang, M. Kim, K. Lauter, Y. Song. Secure outsourced matrix computation and application to neural networks, CCS 2018.

Some sparse linear transformations[‡]

- Assume only $r = \ell \cdot k$ non-zero diag. vectors with indices satisfying
  - $g(ki + j) = g(ki) + g(j)$ for all $0 \leq i < \ell$ and $0 \leq j < k$
- Then we have

$$\boldsymbol{A}\boldsymbol{x} = \sum_{0 \leq i < \ell} \mathtt{Rot}_{g(ki)} \left(\sum_{0 \leq j < k} \mathtt{Rot}_{-g(ki)} \left(\boldsymbol{d}_{g(ki+j)} \right) \circ \mathtt{Rot}_{g(j)}(\boldsymbol{x}) \right)$$

- Cost of such a linear transformation
  - #resulting ciphertexts: 1
  - #CMul: r
  - #Rot: $2\sqrt{r}$

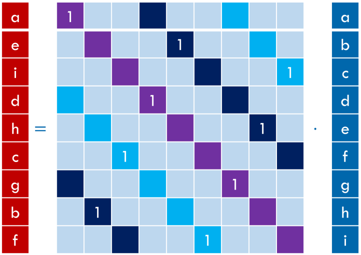
[‡] X. Jiang, M. Kim, K. Lauter, Y. Song. Secure outsourced matrix computation and application to neural networks, CCS 2018.

Some sparse linear transformations‡

- Assume only $r = \ell \cdot k$ non-zero diag. vectors with indices satisfying
  - $g(ki + j) = g(ki) + g(j)$ for all $0 \le i < \ell$ and $0 \le j < k$
- Then we have

$$\boldsymbol{A}\boldsymbol{x} = \sum_{0 \le i < \ell} \texttt{Rot}_{g(ki)} \left(\sum_{0 \le j < k} \texttt{Rot}_{-g(ki)} \left(\boldsymbol{d}_{g(ki+j)} \right) \circ \texttt{Rot}_{g(j)}(\boldsymbol{x}) \right)$$

- Cost of such a linear transformation
  - #resulting ciphertexts: 1
  - #`CMul`: r
  - #`Rot`: $2\sqrt{r}$
- E.g., useful for permutation



‡X. Jiang, M. Kim, K. Lauter, Y. Song. Secure outsourced matrix computation and application to neural networks, CCS 2018.

Roadmap

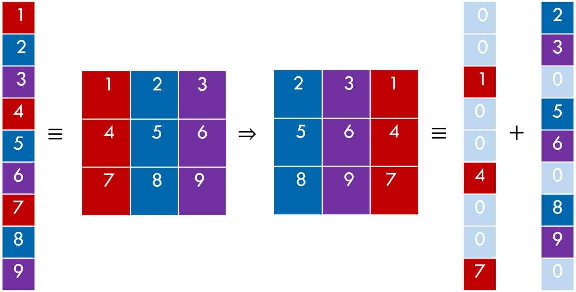
1. **Homomorphic encryption**
   - Basics
2. **Plaintext matrix–ciphertext vector multiplication**
   - Fast linear transformation by Halevi-Shoup (2014, 2015)
3. **Ciphertext matrix–ciphertext matrix multiplication**
   - HE-friendly expression by Jiang et al. (2018)
4. **Bicyclic encoding and matrix multiplication**
   - Ciphertext – ciphertext matrix multiplication
   - Performance

Matrix encoding

- Identify a $d \times d$ matrix as a vector of dimension $n = d^2$ (row by row)
- Cost of some basic matrix operations
  - Matrix addition: $\texttt{\#Add} = 1$
  - Rotation among rows: $\texttt{\#Rot} = 1$

Matrix encoding

- Identify a $d \times d$ matrix as a vector of dimension $n = d^2$ (row by row)
- Cost of some basic matrix operations
  - Matrix addition: `#Add` $= 1$
  - Rotation among rows: `#Rot` $= 1$
  - Rotation among columns: `#Rot` $= 2$, `#CMul` $= 2$

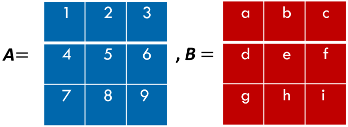


Matrix multiplication (d, d, d)[‡]

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \boldsymbol{B} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

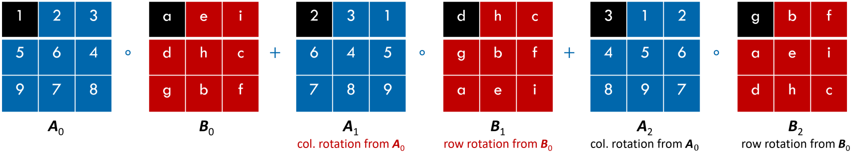
[‡] X. Jiang, M. Kim, K. Lauter, Y. Song. Secure outsourced matrix computation and application to neural networks, CCS 2018.

Matrix multiplication (d, d, d)[‡]



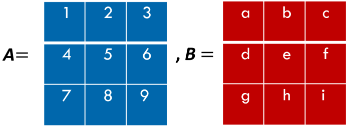
- For $d \times d$ **SQUARE** matrices $\boldsymbol{A}$ and $\boldsymbol{B}$

$$\boldsymbol{A} \cdot \boldsymbol{B} = \boldsymbol{A}_0 \circ \boldsymbol{B}_0 + \cdots + \boldsymbol{A}_{d-1} \circ \boldsymbol{B}_{d-1}$$



‡X. Jiang, M. Kim, K. Lauter, Y. Song. Secure outsourced matrix computation and application to neural networks, CCS 2018.

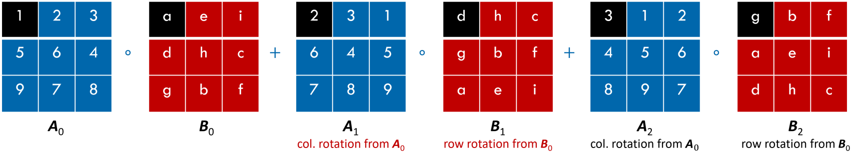
Matrix multiplication (d, d, d)[‡]



- For $d \times d$ **SQUARE** matrices $\boldsymbol{A}$ and $\boldsymbol{B}$

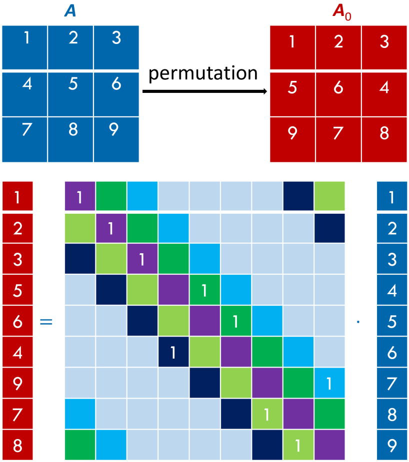
$$\boldsymbol{A} \cdot \boldsymbol{B} = \boldsymbol{A}_0 \circ \boldsymbol{B}_0 + \cdots + \boldsymbol{A}_{d-1} \circ \boldsymbol{B}_{d-1}$$

- Cost of a square matrix multiplication
  - Generation of $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$
  - `#Mul`: d; `#Add`: $d-1$

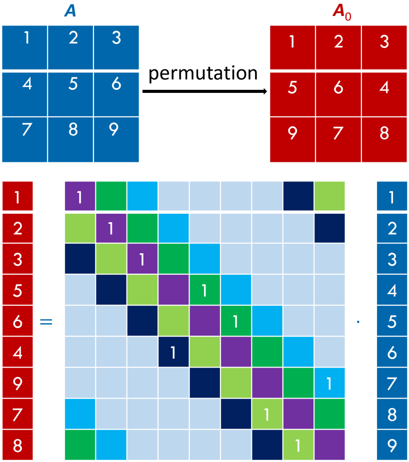


[‡]X. Jiang, M. Kim, K. Lauter, Y. Song. Secure outsourced matrix computation and application to neural networks, CCS 2018.

Generation of A_i and B_i



Generation of A_i and B_i



- Cost for $\boldsymbol{A}_0$ (permutation)
  - #Non-zero diag. vectors: $2d-1$
  - #CMul: $2d-1$
  - #Rot: $2\sqrt{2d-1} \leq 3\sqrt{d}$
- Cost for $\boldsymbol{A}_i$ from $\boldsymbol{A}_0$ (column rotation)
  - #CMul: $2d$
  - #Rot: $2d$
- Cost for $\boldsymbol{B}_0$ (permutation)
  - #Non-zero diag. vectors: d
  - #CMul: d
  - #Rot: $2\sqrt{d}$
- Cost for $\boldsymbol{B}_i$ from $\boldsymbol{B}_0$ (row rotation)
  - #Rot: d

Ciphertext matrix–ciphertext matrix multiplication (d, d, d)

Method	#Ctxts	#Mul	#CMul	#Rot	Mult. depth
Naïve	d^2	d^2	0	$d^2 \log d$	$1\,\texttt{Mul}$
Halevi-Shoup	d	d^2	0	$2d\sqrt{d}$	$1\,\texttt{Mul}$
JKLS	1	d	$5d$	$3d + 5\sqrt{d}$	$1\,\texttt{Mul} + 2\,\texttt{CMul}$

Ciphertext matrix–ciphertext matrix multiplication (d, d, d)

Method	#Ctxts	#Mul	#CMul	#Rot	Mult. depth
Naïve	d^2	d^2	0	$d^2 \log d$	1 Mul
Halevi-Shoup	d	d^2	0	$2d\sqrt{d}$	1 Mul
JKLS	1	d	$5d$	$3d + 5\sqrt{d}$	1 Mul + 2 CMul

- Can we do better?

Ciphertext matrix–ciphertext matrix multiplication (d, d, d)

Method	#Ctxts	#Mul	#CMul	#Rot	Mult. depth
Naïve	d^2	d^2	0	$d^2 \log d$	$1\,\texttt{Mul}$
Halevi-Shoup	d	d^2	0	$2d\sqrt{d}$	$1\,\texttt{Mul}$
JKLS	1	d	$5d$	$3d + 5\sqrt{d}$	$1\,\texttt{Mul} + 2\,\texttt{CMul}$

- Can we do better?
  - Yes.

Roadmap

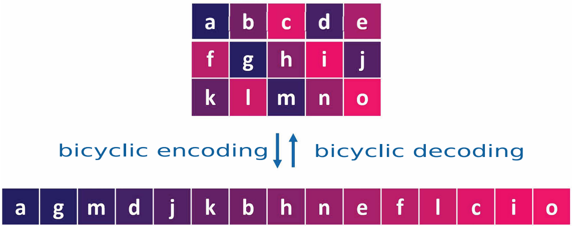
1. **Homomorphic encryption**
   - Basics
2. **Plaintext matrix–ciphertext vector multiplication**
   - Fast linear transformation by Halevi-Shoup (2014, 2015)
3. **Ciphertext matrix–ciphertext matrix multiplication**
   - HE-friendly expression by Jiang et al. (2018)
4. **Bicyclic encoding and matrix multiplication**
   - Ciphertext – ciphertext matrix multiplication
   - Performance

Bicyclic encoding of a matrix

- Assume that $\boldsymbol{A}$ is an $n \times m$ matrix with $\gcd(n, m) = 1$.
- Identify $\boldsymbol{A}$ as a vector of dim mn: a generalization of the diag vector

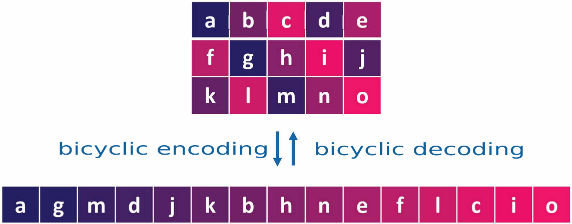
Bicyclic encoding of a matrix

- Assume that $\boldsymbol{A}$ is an $n \times m$ matrix with $\gcd(n, m) = 1$.
- Identify $\boldsymbol{A}$ as a vector of dim mn: a generalization of the diag vector



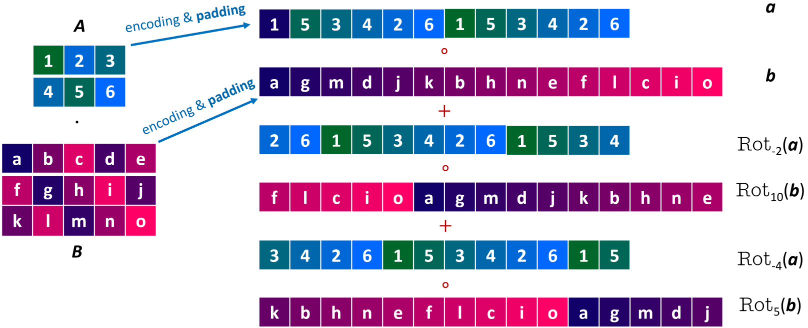
Bicyclic encoding of a matrix

- Assume that $\boldsymbol{A}$ is an $n \times m$ matrix with $\gcd(n, m) = 1$.
- Identify $\boldsymbol{A}$ as a vector of dim mn: a generalization of the diag vector

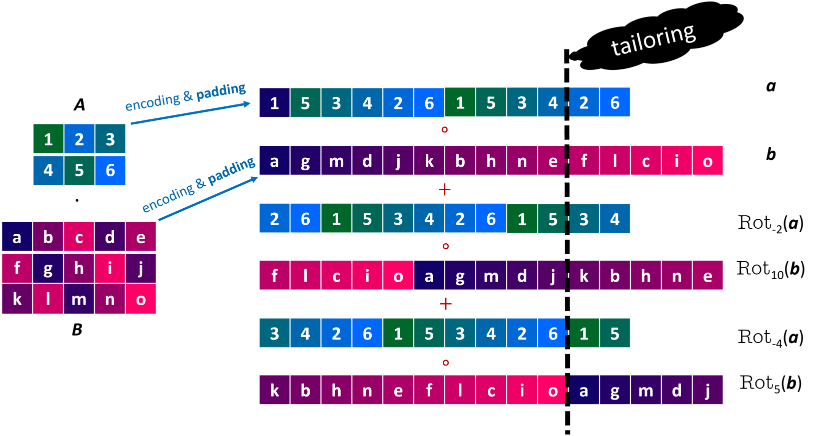


- Basic idea: $\mathbb{Z}/(mn)\mathbb{Z} \cong \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$

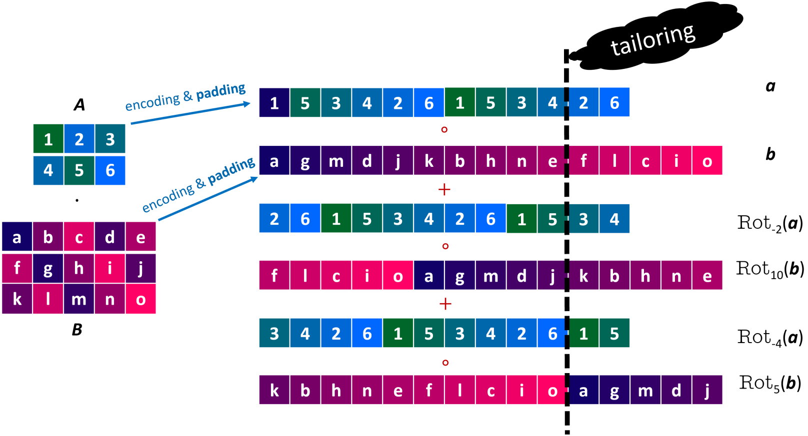
- Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be $n \times m$ and $m \times p$ matrices with n, m, p coprime.



Matrix multiplication $(n, m, p) = (2, 3, 5)$



Matrix multiplication $(n, m, p) = (2, 3, 5)$



- Cost (n, m, p are coprime)
  - #Mul: m
  - #Rot: $2m + \left\lceil \frac{n}{m} \right\rceil + \left\lceil \frac{p}{m} \right\rceil$
  - #CMul: 1
  - Mult. depth: $1\,\mathtt{Mul} + 1\,\mathtt{CMul}$

Encrypted matrix multiplication (n, m, p)

Method	#Ctxts	#Mul	#CMul	#Rot	Mult. depth
Naïve	np	np	0	$np \log m$	1 Mul
Halevi-Shoup*	p	pd	0	$2p\sqrt{d}$	1 Mul
JKLS†	1	d	$5d$	$3d + 5\sqrt{d}$	1 Mul + 2 CMul
BMM-I‡	1	m	0	$2m + 2$	1 Mul

*$d = \max(n, m)$
†$d = \max(n, m, p)$
‡$n,\ m,\ p$ coprime, $\max(n, p) < m$
§$n,\ m,\ p$ coprime, m is a power-of-2 integer

Encrypted matrix multiplication (n, m, p)

Method	#Ctxts	#Mul	#CMul	#Rot	Mult. depth
Naïve	np	np	0	$np \log m$	1 Mul
Halevi-Shoup*	p	pd	0	$2p\sqrt{d}$	1 Mul
JKLS†	1	d	$5d$	$3d + 5\sqrt{d}$	1 Mul + 2 CMul
BMM-I‡	1	m	0	$2m + 2$	1 Mul
BMM-II†§	1	1	0	$3 \log d$	1 Mul

* $d = \max(n, m)$

† $d = \max(n, m, p)$

‡ n, m, p coprime, $\max(n, p) < m$

§ n, m, p coprime, m is a power-of-2 integer

Higher dimension: block versus segment

Given #slots ℓ, a high dim. matrix will be encrypted into **many** ciphertexts.

- Block matrix multiplication
  - E.g., Strassen algorithm $O(d^{\log_2 7})$
    - ⇝ Recursive: Large memory cost
    - ⇝ Loop: Large multiplicative depth

Higher dimension: block versus segment

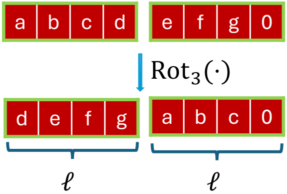
Given #slots ℓ, a high dim. matrix will be encrypted into **many** ciphertexts.

- Block matrix multiplication
  - E.g., Strassen algorithm $O(d^{\log_2 7})$
    - ⇝ Recursive: Large memory cost
    - ⇝ Loop: Large multiplicative depth
- **BMM-III**: Segment matrix multiplication
  - Depends on `Segment` version operations

Higher dimension: block versus segment

Given #slots ℓ, a high dim. matrix will be encrypted into **many** ciphertexts.

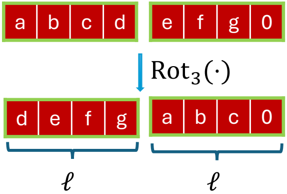
- Block matrix multiplication
  - E.g., Strassen algorithm $O(d^{\log_2 7})$
    - ⇝ Recursive: Large memory cost
    - ⇝ Loop: Large multiplicative depth
- **BMM-III**: Segment matrix multiplication
  - Depends on `Segment` version operations



Higher dimension: block versus segment

Given #slots ℓ, a high dim. matrix will be encrypted into **many** ciphertexts.

- Block matrix multiplication
  - E.g., Strassen algorithm $O(d^{\log_2 7})$
    - ⇝ Recursive: Large memory cost
    - ⇝ Loop: Large multiplicative depth
- **BMM-III**: Segment matrix multiplication
  - Depends on `Segment` version operations
  - #`Mul`: $m \cdot \lceil np/\ell \rceil$
  - #`CMul`: $(4\lceil np/\ell \rceil + 2)m + n + p$
  - #`Rot`: $2m \cdot \lceil np/\ell \rceil$
  - Mult. depth: 1 `CMul` + 1 `Mul`



Roadmap

1. **Homomorphic encryption**
   - Basics
2. **Plaintext matrix–ciphertext vector multiplication**
   - Fast linear transformation by Halevi-Shoup (2014, 2015)
3. **Ciphertext matrix–ciphertext matrix multiplication**
   - HE-friendly expression by Jiang et al. (2018)
4. **Bicyclic encoding and matrix multiplication**
   - Ciphertext – ciphertext matrix multiplication
   - Performance

Implementation and setup

- BMM-I, BMM-II
- Strassen block version of BMM-I, BMM-II
- BMM-III
- Based on CKKS in Microsoft SEAL
  - Ciphertext space: $\mathbb{Z}[X]/\langle X^N + 1, q\rangle$
  - $\log q \approx 50 + L \cdot \log \Delta + 60$
  - L: the number of multiplicative depth
- Matrix entries: $\texttt{pow}(-1, \texttt{i} + \texttt{j}) * \texttt{rand}()/\texttt{pow}(2, 30)$
- Security: ≥ 128 bits
- Error: $\leq 10^{-2}$

Small square matrix multiplication

Table: Performance comparison with the R.-T. algorithm for small-dimensional matrices. $N = 8192$ and $\log \Delta = 30$.

Method	$\log q$	Dimension	Time (ms)	Speedup
R.-T.*	170	$(16, 16, 16)$	199	1.0x
BMM-I	170	$(16, 19, 17)$	130	1.5x
BMM-I	140	$(16, 19, 17)$	82	2.4x
BMM-II	140	$(15, 16, 17)$	**13**	16.6x

*P. Rizomiliotis and A. Triakosia, On matrix multiplication with homomorphic encryption, in Proc. 2022 on Cloud Computing Security Workshop

Square matrix multiplication

Table: Performance comparison for $(128, 128, 128)$ matrix multiplication. $\log \Delta = 30$ except for BMM-III with $\log \Delta = 40$.

Method	$\log q$	$N = 8192$		$N = 32768$	
		Basic block	Time (s)	Basic block	Time (s)
Naïve block JKLS *	200	$(64, 64, 64)$	11.34	$(128, 128, 128)$	14.17
Strassen + JKLS *	200	$(64, 64, 64)$	10.34	$(128, 128, 128)$	14.17
Naïve block BMM-I	140	$(43, 45, 44)$	7.59	$(86, 89, 87)$	13.32
Strassen + BMM-I	140	$(32, 35, 33)$	8.31	$(64, 67, 65)$	11.99
Naïve block BMM-II	140	$(15, 16, 17)$	**4.40**	$(21, 32, 23)$	8.32
Strassen + BMM-II	140	$(11, 8, 9)$	84.89	$(17, 16, 19)$	127.69
BMM-III	190	$(128, 131, 129)$	11.05	$(128, 131, 129)$	41.60

*X. Jiang, M. Kim, K. Lauter, and Y. Song, Secure outsourced matrix computation and application to neural networks, CCS '18

Square matrix multiplication

Table: Performance comparison for $(256, 256, 256)$ matrix multiplication, $\log \Delta = 40$.

Method	$\log q$	$N = 8192$		$N = 32768$	
		Basic block	Time (s)	Basic block	Time (s)
Naïve block JKLS *	200	$(64, 64, 64)$	89.33	$(128, 128, 128)$	112.01
Strassen JKLS *	200	$(64, 64, 64)$	71.54	$(128, 128, 128)$	97.90
Naïve block BMM-I	140	$(43, 45, 44)$	59.57	$(86, 89, 87)$	73.35
Strassen + BMM-I	140	$(32, 35, 33)$	56.98	$(64, 67, 65)$	80.99
Naïve block BMM-I	140	$(15, 16, 17)$	76.60	$(21, 32, 23)$	76.19
BMM-III	190	$(256, 259, 257)$	**42.90**	$(256, 259, 257)$	110.99

*X. Jiang, M. Kim, K. Lauter, and Y. Song, Secure outsourced matrix computation and application to neural networks, CCS '18

Square matrix multiplication

Table: Performance comparison for matrix multiplication of dimension 512 and 1024 with $N = 8192$.

Method	$\log q$	(512, 512, 512) Basic block	(512, 512, 512) Time (s)	(1024, 1024, 1024) Basic block	(1024, 1024, 1024) Time (s)
Naïve block JKLS *	200	(64, 64, 64)	728	(64, 64, 64)	6028
Strassen JKLS *	200	(64, 64, 64)	479	(64, 64, 64)	3514
Naïve block BMM-I	140	(43, 45, 44)	490	(43, 45, 44)	4240
Strassen + BMM-I	140	(32, 35, 33)	390	(32, 35, 33)	2757 [†]
Naïve block BMM-II	140	(15, 16, 17)	1766	(15, 16, 17)	–
BMM-III	190	(512, 515, 513)	**181** [†]	(1024, 1027, 1025)	**1200** [†]

[†] By default, $\log \Delta = 30$ except for these with $\log \Delta = 40$.

*X. Jiang, M. Kim, K. Lauter, and Y. Song, Secure outsourced matrix computation and application to neural networks, CCS '18

Rectangular matrix multiplication

Table: Performance for rectangular matrix multiplication (I).

Huang et al.'s algorithm *†		BMM-III‡			
Dimension	Time (s)	Dimension	KeyGen (s)	Total (s)	Speedup
$(256, 256, 16)$	**6.19**	$(256, 257, 17)$	16.15	23.60	
$(256, 16, 256)$	**6.23**	$(256, 17, 257)$	14.69	19.37	
$(1024, 1024, 16)$	108.22	$(1024, 1025, 17)$	14.68	**55.79**	1.9x
$(1024, 16, 1024)$	108.31	$(1024, 17, 1025)$	14.47	**43.64**	2.4x
$(2048, 2048, 8)$	218.09	$(2048, 2049, 11)$	14.49	**98.01**	2.2x
$(2048, 8, 2048)$	218.09	$(2049, 8, 2051)$	14.63	**81.09**	2.6x

† For Huang et al.'s algorithm, N is set as the same as theirs and $\log \Delta = 30$.

‡ For BMM-III, $N = 8192$ and $\log \Delta = 40$.

*Z. Huang, C. Hong, C. Weng, W.-j. Lu, and H. Qu, More efficient secure matrix multiplication for unbalanced recommender systems, IEEE TDSC, 2023

Rectangular matrix multiplication

Table: Performance (in sec.) for rectangular matrix multiplication (II). $N = 8192$, $\log \Delta = 30$.

Dimension	Naïve block JKLS *	Naïve block BMM-I	Speedup
$(4, 1636, 5)$	36.92	**9.76**	3.7x
$(8, 3405, 9)$	78.58	**19.92**	3.9x
$(16, 6903, 17)$	157.00	**38.57**	4.0x
$(32, 13847, 33)$	317.31	**76.73**	4.1x

*X. Jiang, M. Kim, K. Lauter, and Y. Song, Secure outsourced matrix computation and application to neural networks, CCS '18

Rectangular matrix multiplication

Table: Performance (in sec.) for rectangular matrix multiplication (III). $N = 8192$, $\log\Delta = 30$.

Dimension	Naïve block JKLS *	Naïve block BMM-I	LKS* + Segment + Opt.
$(4, 5, 1636)$	36.31	**0.10**	0.25
$(8, 9, 3405)$	77.56	0.65	**0.63**
$(16, 17, 6903)$	157.32	4.96	**2.23**
$(32, 33, 13847)$	316.76	38.82	**8.24**

*W. Lu, S. Kawasaki, and J. Sakuma, Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data, NDSS '17

Conclusion

- Encrypted matrix multiplication (n, m, p pairwise coprime)
- More flexible and more efficient
- Paper: https://doi.org/10.1109/TIFS.2024.3490862
- Code: https://github.com/hangenba/bicyclic_mat_mul
- Application: privacy-preserving LLM inference, e.g., iDASH - HE track

Conclusion

- Encrypted matrix multiplication (n, m, p pairwise coprime)
- More flexible and more efficient
- Paper: https://doi.org/10.1109/TIFS.2024.3490862
- Code: https://github.com/hangenba/bicyclic_mat_mul
- Application: privacy-preserving LLM inference, e.g., iDASH - HE track
- We did not mention the coefficient encoding method
  - ePrint 2023/1649, 2023/1678, ...
- We did not mention plaintext-ciphertext matrix multiplication
  - ePrint 2024/1284, ...

Conclusion

- Encrypted matrix multiplication (n, m, p pairwise coprime)
- More flexible and more efficient
- Paper: https://doi.org/10.1109/TIFS.2024.3490862
- Code: https://github.com/hangenba/bicyclic_mat_mul
- Application: privacy-preserving LLM inference, e.g., iDASH - HE track
- We did not mention the coefficient encoding method
  - ePrint 2023/1649, 2023/1678, ...
- We did not mention plaintext-ciphertext matrix multiplication
  - ePrint 2024/1284, ...

Open problems

- How to remove the coprime condition?
- Encrypted BLAS? Fast vector/matrix/tensor ops on encrypted data