Private Iris Recognition with High-Performance FHE

Jincheol Ha[1] Guillaume Hanrot[2] Taeyeong Noh[3]
Jung Hee Cheon[3,4] Jung Woo Kim[3] Damien Stehlé[5]

[1]Soongsil University, Seoul, Korea
[2]CryptoLab, Inc., Lyon, France
[3]CryptoLab, Inc., Seoul, Korea
[4]Seoul National University, Seoul, Korea
[5]PQShield, France

PACOH 26 – Homomorphic Encryption Workshop

* This work was done while J. Ha and D. Stehlé were at CryptoLab.

Outline

Introduction

Iris as a biometric authentication

- Among biometric primitives – *face*, *fingerprint*, *heart rhythm*, ... – the iris stands out:
  - **convenient capture**, *and*
  - **high accuracy even at the scale of billions** of users.
- These two strengths are exactly why **World ID** chose the iris to build an authentication system "for all humans" – doing **massive** $1:N$ **identification** (matching one query against a whole database of millions of enrolled irises).

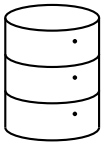
World ID infrastructure

Orb
(capture device)

Feature Extraction

Iris code $c \in \{0,1\}^\ell$
Iris mask $m \in \{0,1\}^\ell$

Private DB Search



Database
of millions
of users

** Images from the WorldCoin blog post.*

World ID's privacy issue

- Storing and computing on billions of biometric templates is a serious privacy risk – need to protect both query and database.
- Two leading cryptographic solutions:
  - **MPC** – multiparty computation,
  - **FHE** – fully homomorphic encryption.
- World ID's approach: secret-sharing multi-party computation (SS-MPC)

Why did World ID pick MPC and not FHE?

"MPC's role in advancing World ID privacy" by **R. Walch and D. Kales** [WK24]

- A blog entry insisting weakness of FHE compared to MPC
- **What they insist:** FHE is not a viable alternative to MPC — on three counts:
  1. security still "relies on **no collusion** between the database holder and the key holder";
  2. compute is prohibitive – **17 ms** for one 16-bit addition (GPU);
  3. comm/storage are massive – **37 MB**/query, **3.5 TB** for 100k entries.

Private Iris Recognition by SS-MPC [BGK+24]

World ID's **deployed** solution (Bloemen, Gillespie, **Kales**, Sippl, **Walch**, 2024):

- **2-out-of-3 MPC**: 3 parties hold database shares; any 2 collaborate to compute (the 3rd is for redundancy).
- Security: 1 party learns nothing; 2 colluding parties reconstruct everything.
- Performance: a batch of **32 users** vs. a $\mathbf{2^{22}}$ database in $\approx \mathbf{1.9\,s}$, on **24**$\times$ **H100** GPUs (3 parties $\times$ 8 GPUs).

Impressive raw speed — but at what deployment cost?

Two *different* kinds of drawback

(A) Inherent to SS-MPC as a technique

- **Poor scaling** with the t-out-of-n access pattern (large t, n are desirable).
- **Latency-bound**: matching needs *tens of communication rounds*.

(B) Specific to the Worldcoin solution [BGK+24]

- Chose **2-of-3** $\Rightarrow$ *no* security against collusion.
- The ≈ 1.9 s headline rests on an **ultra-fast network with co-located parties**.

Drawback (A) is the price of the paradigm; (B) is a choice of this particular deployment.

Why that assumption is self-defeating

- A $\sim$3 Tb/s link with GPU-direct networking $\Rightarrow$ the parties must sit in the **same data-center**, physically together (co-located).
- But co-location undermines the very no-collusion assumption that 2-of-3 security depends on.

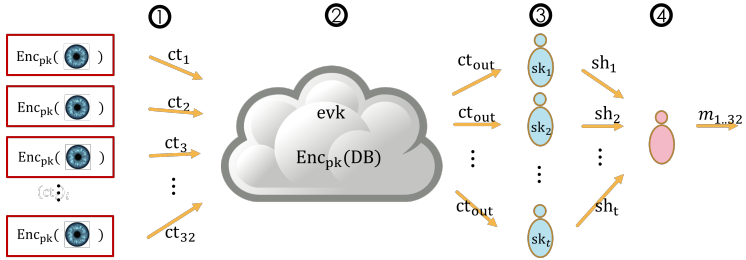
[WK24]'s case against FHE is now out of date

With **ThFHE** and the recent **HEaaN2** library, each one falls:

- **No-collusion / single key holder:**
  ThFHE secret-shares the decryption key among n decryptors $\Rightarrow$ no single key holder to collude with.
- **Heavy computation, communication, and storage cost:**
  High-performance FHE library/protocol close the gap $\Rightarrow$ comparable to SS-MPC.

Let's take a look on ThFHE protocol

The overall system (ThFHE protocol)



- **Queriers** encrypt iris codes under the public key.
- **Computing servers** batch queries and run the homomorphic matching – the heavy, *public* step.
- **Decryptors** run partial decryption with their key shares.
- **Receiver** runs final decryption to recover the output bits.

Advantage of ThFHE deployment

- The encrypted query and database can be public
  $\Rightarrow$ no privacy leakage on the computing server
- **Communication-light**: 2–3 rounds, hundreds of KB/party
  $\Rightarrow$ deployable over a WAN (no co-location needed).
- Computational cost is **almost agnostic to** (t, n).
- Active security adds only ZK proofs on the *light* decryption step.
  - For the correctness of the computing server, we may add redundancy as all the inputs are public.

ThFHE vs SS-MPC: the big picture

	SS-MPC [BGK+24]	**ThFHE (ours)**
Hardware	24× H100	8× RTX-5090
Model	2-of-3, no collusion	t-of-n, tunable
Network	co-located, ∼3 Tb/s	WAN-deployable
Scaling (t, n)	costly	nearly free

Security and deployment already favour ThFHE — the one open question is performance.

The bottom line, in numbers

We build high performance iris matching protocol via HEaaN2:

	SS-MPC [BGK+24]	**ThFHE (ours)**
Hardware	24× H100	8× RTX-5090
Query batch	64 eyes	32 eyes
Database N	$2^{22} \approx 4.19$ M	$7 \cdot 2^{14} \approx 115$ k
Total time	$\approx$ **1.9 s**	$\approx$ **1.8 s**

- SS-MPC handles a larger workload, yet per-GPU and cost-adjusted the two are comparable.
- That headline ≈ 1.9 s assumes a $\sim$3 Tb/s co-located network — a realistic network speed will degrade its performance.

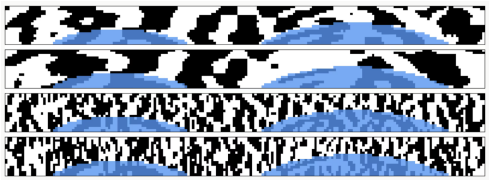
Background

Plain iris recognition: from eye to iris code

Each eye image is turned into iris template:

- **Iris code** $c \in \{0,1\}^\ell$: the binary vector encoding the iris texture.
- **Mask** $m \in \{0,1\}^\ell$: flags bits hidden by eyelids, eyelashes, or reflections.

($\ell = 12800$ in [BGK+24] and ours)



** Images from the WorldCoin blog post.*

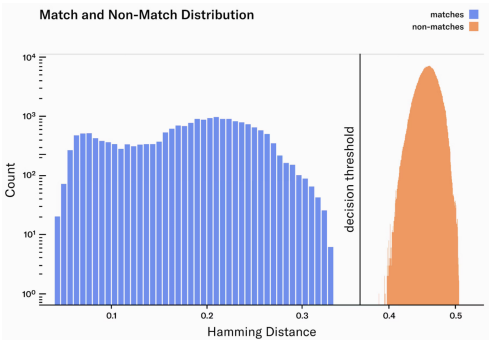
After extracting iris templates, compute the distance for matching.

Plain iris recognition: matching two codes

Compare Hamming distance:

$$\mathrm{HD} = \frac{\left\|(c_1 \oplus c_2) \wedge m_1 \wedge m_2\right\|_1}{\|m_1 \wedge m_2\|_1}.$$

- What we expect:
  - **Same eye** $\Rightarrow \mathrm{HD} \approx 0$;
  - **Different eyes** $\Rightarrow \mathrm{HD} \approx 0.5$.
- We cannot capture ideally the same image in practice.
  - Use ρ **rotated** copies of the query and check the minimum HD.
  - $\rho = 31$ in [BGK+24] and ours.



** Image from the WorldCoin blog post.*

From Hamming distance to inner products [BGK+24]

Bitmask representation. Fold each template (c_i, m_i) into one signed vector $c'_i \in \{-1, 0, +1\}^n$:

$$c'_i \;=\; m_i - 2\,(m_i \wedge c_i) \;=\; \begin{cases} +1, & m_i = 1,\ c_i = 0, \\ -1, & m_i = 1,\ c_i = 1, \\ \;\;0, & m_i = 0 \quad \text{(masked)}. \end{cases}$$

Then the masked XOR count in the numerator collapses to a single inner product,

$$\langle c'_1, c'_2 \rangle \;=\; \|m_1 \wedge m_2\|_1 \;-\; 2\big\|(c_1 \oplus c_2) \wedge m_1 \wedge m_2\big\|_1$$

From Hamming distance to inner products [BGK+24]

So matching is decided by a **normalized inner product**:

$$\text{score} \;=\; \frac{\langle c_1', c_2' \rangle}{\|m_1 \wedge m_2\|_1} \;=\; 1 - 2\,\text{HD}.$$

- Assume the mask data is given in plain ⇒ score computation is an inner product
- One query vs. the database ⇒ **matrix–vector**
- Batched queries vs. the database ⇒ **matrix–matrix**

The core task reduces to two steps

(A) **Score computation** = matrix multiplication (query vectors × database).

(B) **Compare** each score to the cutoff τ.

Polynomial products as matrices: the Toeplitz trick

Multiplying by a fixed polynomial $m = \sum_i m_i X^i$ over the ring $\mathbb{Z}[X]/(X^N + 1)$ is a **linear map** on coefficient vectors, given by its negacyclic Toeplitz matrix:

$$\mathrm{Toep}(m) = \begin{bmatrix} m_0 & -m_{N-1} & \cdots & -m_1 \\ m_1 & m_0 & \cdots & -m_2 \\ \vdots & \vdots & \ddots & \vdots \\ m_{N-1} & m_{N-2} & \cdots & m_0 \end{bmatrix}$$

- **RLWE decryption becomes linear algebra.**
  - A ciphertext (a, b) of m satisfies $m \approx s \cdot a + b$.
  - Writing $\mathbf{S} = \mathrm{Toep}(s)$, this is $\mathbf{S}\,\mathbf{a} + \mathbf{b} \approx \mathbf{m} \pmod q$.
- Stack multiple ciphertexts as columns $\Rightarrow \mathbf{S}\,\mathbf{A} + \mathbf{B} \approx \mathbf{M} \pmod q$.

PCMM: plaintext $\times$ ciphertext as linear algebra [BCH+24]

Goal. Multiply an *encrypted* matrix $\mathbf{M}$ by a *plaintext* matrix $\mathbf{U}$:

$$\mathbf{S}\,\mathbf{A} + \mathbf{B} \approx \mathbf{M} \xrightarrow{\cdot\,\mathbf{U}} \mathbf{S}\,\underbrace{(\mathbf{AU})}_{\mathbf{A}'} + \underbrace{(\mathbf{BU})}_{\mathbf{B}'} \approx \mathbf{MU}.$$

- $(\mathbf{A}', \mathbf{B}') = (\mathbf{AU}, \mathbf{BU})$ is directly an encryption of $\mathbf{MU}$ — the secret part $\mathbf{S}$ is never touched.
- Encrypted matmul $\Rightarrow$ **two plaintext matmuls** (PPMMs) $\mathbf{AU}$ and $\mathbf{BU}$ modulo q

RGSW-based CCMM: ciphertext $\times$ ciphertext as linear algebra [BCH+25]

Can we use a similar idea for CCMM?

- Assume $\mathbf{S}_1\mathbf{A}_1 + \mathbf{B}_1 = \mathbf{M}_1$ and $\mathbf{S}_2\mathbf{A}_2 + \mathbf{B}_2 = \mathbf{M}_2$. Then,

$$\begin{aligned}\mathbf{M}_1\mathbf{M}_2 &= (\mathbf{S}_1\mathbf{A}_1 + \mathbf{B}_1)(\mathbf{S}_2\mathbf{A}_2 + \mathbf{B}_2) \\ &= (\mathbf{S}_1 \cdot \underline{\mathbf{A}_1\mathbf{S}_2}\mathbf{A}_2 + \underline{\mathbf{B}_1\mathbf{S}_2}\mathbf{A}_2) + (\mathbf{S}_1 \cdot \mathbf{A}_1\mathbf{B}_2 + \mathbf{B}_1\mathbf{B}_2) \\ &= \mathbf{S}_1 \cdot (\mathbf{A}'_1\mathbf{A}_2 + \mathbf{A}_1\mathbf{B}_2) + (\mathbf{B}'_1\mathbf{A}_2 + \mathbf{B}_1\mathbf{B}_2)\end{aligned}$$

  where $\mathbf{A}'_1 = \mathbf{A}_1\mathbf{S}_2$ and $\mathbf{B}'_1 = \mathbf{B}_1\mathbf{S}_2$.

- If we know $\mathbf{A}'_1$ and $\mathbf{B}'_1$, then CCMM becomes 4 PPMM:

$$\mathbf{A}'_1\mathbf{A}_2,\ \mathbf{B}'_1\mathbf{A}_2,\ \mathbf{A}_1\mathbf{B}_2,\ \mathbf{B}_1\mathbf{B}_2$$

But giving $\mathbf{A}_1\mathbf{S}_2$ and $\mathbf{B}_1\mathbf{S}_2$ together with $\mathbf{A}_1$ and $\mathbf{B}_1$ is insecure.

RGSW-based CCMM: ciphertext x ciphertext as linear algebra [BCH+25]

Use **RGSW encryption** of $\mathbf{M}_1$: $\mathbf{S}_1\mathbf{A}_1 + \mathbf{B}_1 = \mathbf{M}_1$ and $\mathbf{S}_1\mathbf{A}'_1 + \mathbf{B}'_1 = \mathbf{S}_2\mathbf{M}_1$.

- Now can we do $\mathbf{M}_1\mathbf{M}_2 = \mathbf{S}_1(\mathbf{A}'_1\mathbf{A}_2 + \mathbf{A}_1\mathbf{B}_2) + (\mathbf{B}'_1\mathbf{A}_2 + \mathbf{B}_1\mathbf{B}_2)$?
  $\Rightarrow$ No, we should consider the error inside ciphertexts.
- This is exactly the same issue in keyswitching.
- Use auxiliary modulus p for the RGSW ciphertext:

$$\mathbf{S}_1\mathbf{A}_1 + \mathbf{B}_1 \approx p\mathbf{M}_1 \pmod{pq}, \quad \mathbf{S}_1\mathbf{A}'_1 + \mathbf{B}'_1 \approx p\mathbf{M}_1\mathbf{S}_2 \pmod{pq}$$

$$\mathbf{S}_1\mathbf{A}_2 + \mathbf{B}_2 \approx \mathbf{M}_2 \pmod{q}$$

Store database in RGSW ciphertexts and send queries in RLWE ciphertexts to do RGSW-based CCMM.

Two-way classification

Two-way classification

A polynomial p is an (I_0, I_1, ε)-**classification function** if it separates two input intervals into the labels 0 and 1:

$$|p(x)| \leq \varepsilon \quad (x \in I_0), \qquad |p(x) - 1| \leq \varepsilon \quad (x \in I_1).$$

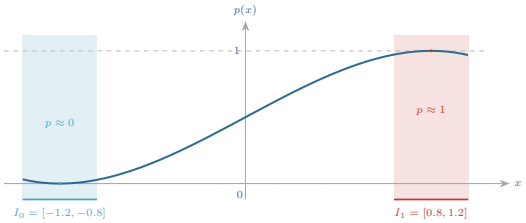
In iris recognition. The two labels are the matching decision:

- I_1: **same-eye** pairs $\Rightarrow$ high score $\rightarrow$ **match**.
- I_0: **different-eye** pairs $\Rightarrow$ low score $\rightarrow$ **non-match**.
- The threshold τ lives in the *empty gap* between the two clusters
  - *Remark.* No constraint outside $I_0 \cup I_1$

Two-way classification

Toy example. The shifted cubic

$$p(x) = \frac{1}{2} + \frac{3x - x^3}{4}$$



$I_0 = [-1.2, -0.8], I_1 = [0.8, 1.2], \varepsilon = 0.032$

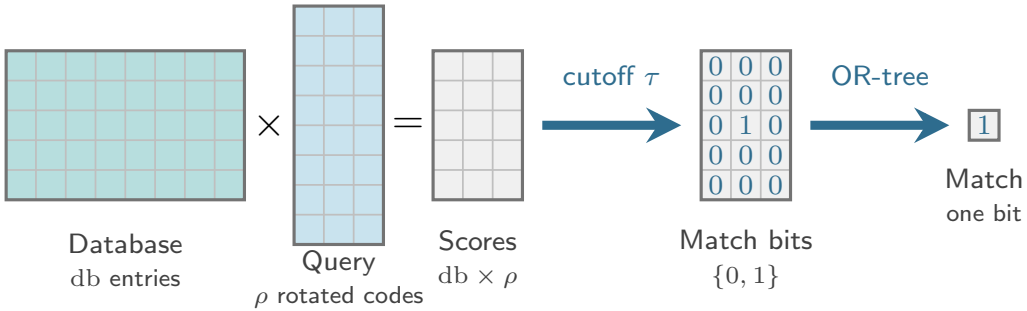
In practice.

- Use multi-interval Remez algorithm on $I_0 \cup I_1$.
- Decompose into a few polynomials of smaller degrees.
- Paterson–Stockmeyer evaluation method to keep the multiplicative depth low.
- Classify with moderate ε and then clean to high-precision [CKSS25].

A First Approach

The matchDB primitive

- **Input:** a query of ρ rotated iris codes; a database of db entries.
- **Output:** one bit per query eye – match or no-match.



The 5-step pipeline

1. **Query preprocessing** – homomorphically convert the transmitted query (communication format) into the format CCMM can consume.
2. **Batched inner products (CCMM)** – the score numerators.
3. **Normalize** – divide by denominators (plaintext masks).
4. **Classify** – map scores into $\{0, 1\}$ (possible intermediate outputs for the input between I_0 and I_1).
5. **Post-process** – remove intermediate result and OR them into a single match bit.

In this talk, we concentrate on the core circuit: Step 2-4.

Core circuit

First approach: minimize the linear-algebra cost

Run the **CCMM at the smallest possible modulus**, so the score computation is as cheap as possible.

CCMM at low level $\rightarrow$ BTS $\rightarrow$ Classification

Drawback:

- Need to bootstrap all the scores: $\rho \cdot (\#\,\mathrm{db})/N$ ciphertexts
- Bootstrapping cost dominates others, far exceeding the linear algebra cost.

How to balance BTS cost and linear-algebra cost?

Folding: Reducing the Number of Bootstraps

Initial idea of folding

- Current workflow: CCMM $\rightarrow$ BTS $\rightarrow$ Classification
  - BTS before classification is to recover computation level
  - One BTS per classification in this setting
- **What do we really need for classification?**
  - Clearly separated input domain (e.g., threshold τ)
  - Finally output a single binary matching result over DB
- **Initial Idea**: classify with low precision before BTS, add them, and do BTS

  - Fold: add k partially classified scores into one ciphertext.
  - Can we further generalize this?

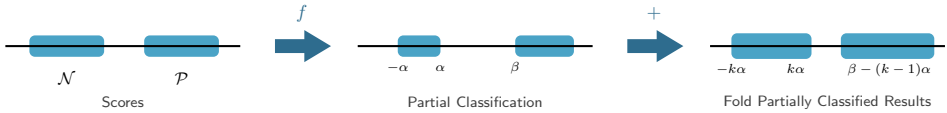
Generalizing the folding idea

What we have: no duplicate users $\Rightarrow$ at most one match per query eye

Folding assumption

Among the k folded scores, at most one is a true match.

- Keeping this assumption, we can reduce the number of BTS in the core circuit by a factor of k.
- In this concept, we do not need binary outputs in the folding steps.



Deterministic folding

Two input intervals: $\mathcal{N}$ (non-match scores), $\mathcal{P}$ (match scores).

- Pick a polynomial f that pushes them apart:

$$f(\mathcal{N}) \subset [-\alpha, \alpha], \qquad f(\mathcal{P}) \geq \beta, \qquad (\alpha \ll \beta).$$

- Fold k scores $x_1, \ldots, x_k \in \mathcal{N} \cup \mathcal{P}$ with at most one in $\mathcal{P}$, and sum $S = \sum_{i=1}^{k} f(x_i)$:

$$\text{all non-match: } S \leq k\alpha, \qquad \text{one match: } S \geq \beta - (k-1)\alpha.$$

- If $k\alpha < \beta - (k-1)\alpha$ then we can classify two cases based on S.

Probabilistic folding

- The deterministic bound $\alpha < \beta/(2k-1)$ is too strict for large k.
- Probabilistic approach is possible if we know the input distribution for the non-match score.

Definition (folding polynomial)

Given k and closed intervals $\mathcal{N}_f, \mathcal{P}_f$ such that $\max \mathcal{N}_f < \min \mathcal{P}_f$ and $p_1, p_2 \ll 1$, a $(k, \mathcal{N}_f, \mathcal{P}_f)$-**folding polynomial** is a polynomial f with

$$\Pr_{(x_0,\ldots,x_{k-1}) \leftarrow \mathcal{D}^k} \left[\sum_{i=0}^{k-1} f(x_i) \notin \mathcal{N}_f \right] \leq p_1,$$

$$\Pr_{(x_1,\ldots,x_{k-1}) \leftarrow \mathcal{D}^{k-1}} \left[\min_{x_0 \in \mathcal{P}} f(x_0) + \sum_{i=1}^{k-1} f(x_i) \notin \mathcal{P}_f \right] \leq p_2.$$

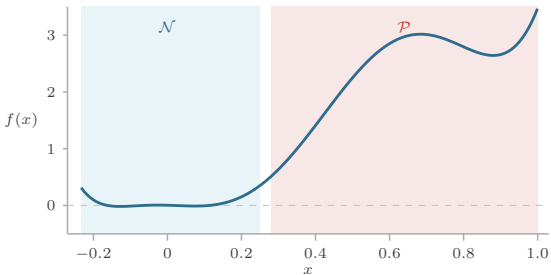
Our folding polynomial

A degree-7 polynomial f with folding capability $k = 16$:

- Non-match scores modeled by $\mathcal{D} \approx \mathcal{N}(0.008,\, 0.034^2)$;[1] positive interval $\mathcal{P} = [0.28, 1.0]$.
- Folding output intervals

  $$\mathcal{N}_f = [-0.13, 0.33], \quad \mathcal{P}_f = [0.4, 3.8].$$

- $p_1 \approx 5.4 \cdot 10^{-10}$, $p_2 \approx 5.3 \cdot 10^{-11}$
  $\Rightarrow$ negligible impact on FA/FR.

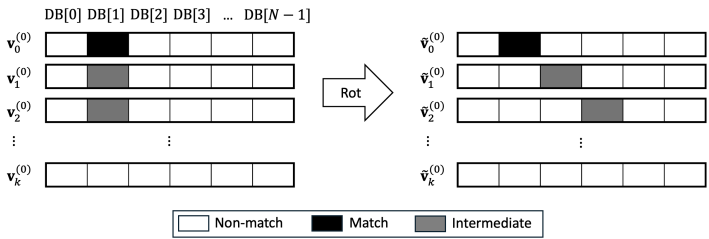


[1] Obtained from ND-IRIS-0405 dataset with Worldcoin's open-source iris template extractor

Which scores can we fold?

Which slots can we fold?

- **Different query eyes?** No.
  - Easy to achieve folding assumption, but cannot distinguish which query hits.
- We can fold only **within a single query eye**: among ρ rotated iris templates.
  - But we cannot keep the folding assumption among the rotations
    $\Rightarrow$ use rotation trick



New workflow with folding

$$\text{CCMM} \rightarrow \text{Fold} \rightarrow \text{BTS} \rightarrow \text{Classify}.$$

The number of BTS in the core circuit reduced by a factor k.

Tradeoff of folding:

- Folding requires the CCMM at a higher modulus.
- Larger linear algebra cost and encrypted DB size.
  - Bootstrapping drops far more, so better performance.
  - Larger DB size make its deployment harder.

Implementation and Results

Implementation

Setup

- HEaaN2 library supporting GPU implementation of CKKS
- Require 8 RTX-5090; PoC implementation on a single RTX-5090

Optimization: speed + memory

- RTX-5090 only has 32GB memory
- Data movement between RAM and GPU is heavy
  - Need to pre-load all the DB and BTS keys
- Distribute data over multiple GPUs;
  - Maximize parallelism over GPUs
  - Minimize communication cost between GPUs

Optimizing CCMM: int8 cuBLAS

RGSW-based CCMM consists of 4 PPMM. How to optimize PPMM?

- Reduce PPMM to **int8 tensor-core matmul** using the cuBLAS library:
  - 8-bit input matrices $\rightarrow$ 32-bit output matrix
- 8-bit RNS
  - Switch modulus to $q = \prod_i q_i$ where $q_i's$ are co-prime and $q_i \leq 2^8$.
  - Compute matmul over each q_i using RNS
- 8-bit square RNS
  - Limited choice for q_i's but the target modulus is about 357 bits.
  - Use $q = \prod_i q_i^2$:

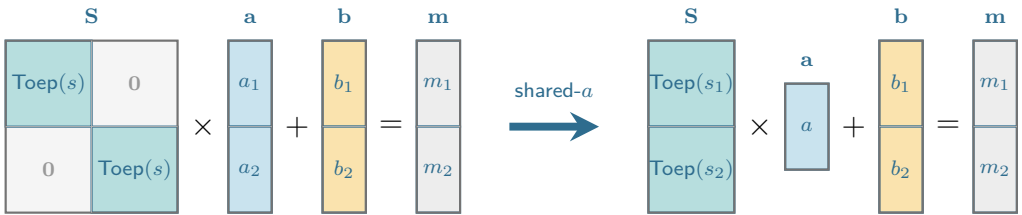
$$(a_1 q_i + a_0)(b_1 q_i + b_0) = (a_1 b_0 + a_0 b_1 + \lfloor \tfrac{a_0 b_0}{q_i} \rfloor) + (a_0 b_0 \bmod q_i) \pmod{q_i^2}$$

  - Smaller number of words $\Rightarrow$ reduce encrypted DB size

Optimizing CCMM: MSRLWE "shared-a" format [BCH+24]

Shared-a format can save the computation and memory for the a-part of the ciphertext:

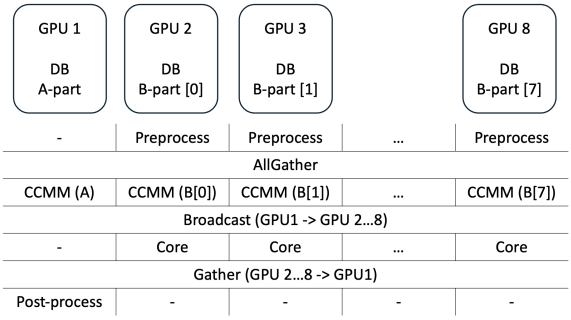
$$\begin{cases} s \cdot a_1 + b_1 = m_1 \\ s \cdot a_2 + b_2 = m_2 \end{cases} \quad \text{vs.} \quad \begin{cases} s_1 \cdot a + b_1 = m_1 \\ s_2 \cdot a + b_2 = m_2 \end{cases}$$



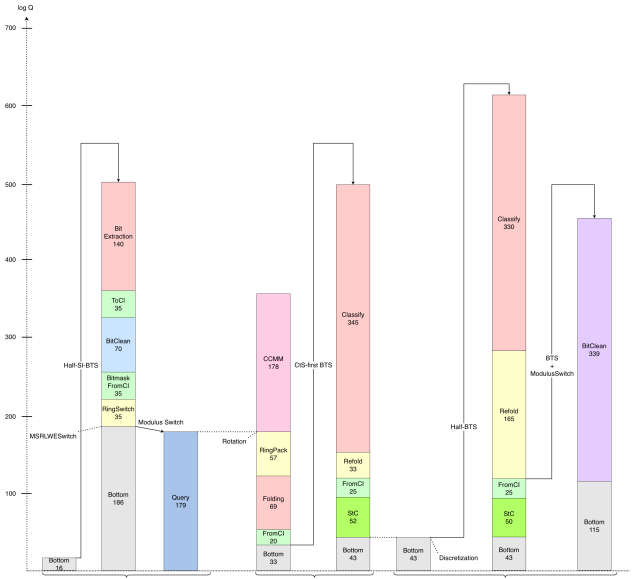
block-diagonal square $\mathbf{S}$

stacked Toeplitz $\mathbf{S}$, smaller $\mathbf{a}$

Workload over 8 GPUs



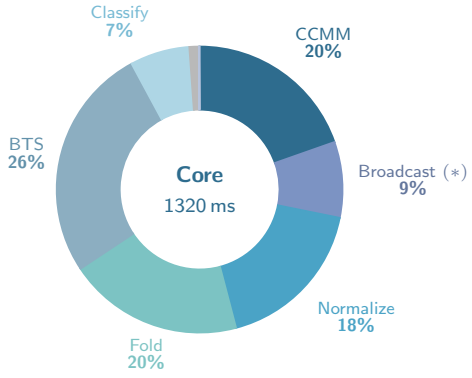
- 7×2^{14} DB entries over 8 GPUs
- (Shared) A-part of CCMM is computed in a single GPU
  - $\mathbf{S}_1(\mathbf{A}_1'\mathbf{A}_2 + \mathbf{A}_1\mathbf{B}_2) + (\mathbf{B}_1'\mathbf{A}_2 + \mathbf{B}_1\mathbf{B}_2)$
- Use the same BTS key for preprocessing and core circuit


log Q
700
600
500
400
300
200
100
Bit Extraction 140
ToCI 35
BitClean 70
Bitmask FromCI 35
RingSwitch 35
Bottom 180
Bottom 16
Half-SI-BTS
MSRLWESwitch
Modulus Switch
Query 179
CCMM 178
RingPack 57
Folding 69
FromCI 20
Bottom 33
Rotation
CtS-first BTS
Classify 345
Refold 33
FromCI 25
StC 52
Bottom 43
Bottom 43
Half-BTS
Discretization
Classify 330
Refold 165
FromCI 25
StC 50
Bottom 43
BTS + ModulusSwitch
BitClean 339
Bottom 115
Preprocessing
Core Circuit
Post-processing

Results

	Step	Runtime (ms)
	CCMM	259
	Broadcast $(*)$	113
	Normalize	234
	Fold	261
	BTS	349
	Classify	89
	Refold	14
	Gather $(*)$	1
	Core total	**1320**

$(*)$ estimated communication cost.



Preprocessing in 487 ms + Post-processing in 30 ms $\Rightarrow$ **1.8 sec in total**

Comparison to SS-MPC

	SS-MPC [BGK+24]	**ThFHE (ours)**
Hardware	24× H100	8× RTX-5090
DB Size	2^{22}	$7 \cdot 2^{14}$
Query Batches	64 eyes	32 eyes
Latency	1.9 sec	1.8 sec
Communication	∼81 GB/party	∼100s KB/party
Rounds	∼40	3
Network	> 3 Tb/s, co-located	WAN

- **Comparable** time and cost (adjusting for GPU prices).
- Far less communication; do not assume co-located network

Comparison to the first approach

Folding trades linear-algebra cost for far fewer bootstraps

$16\times$ fewer BTS, at a moderately higher CCMM modulus.

The folding approach achieves higher throughput:

- Estimate at the same scale, counting only the BTS cost of the first approach
- For the core circuit, folding enjoys at least $1.7\times$ larger throughput

But the first approach is easier to deploy:

- CCMM at the **smallest modulus** $\Rightarrow$ smaller encrypted DB, lower memory
- Needs **fewer GPUs** for the same size of DB

Conclusion

Conclusion

- **ThFHE** is a realistic alternative to SS-MPC for large-scale private biometrics: similar speed, mild communication, distributable security.

	[WK24]	ThFHE (ours)
Query (1 iris code)	37 MB	512 KB ($74\times$)
Database (100k)	3.5 TB	123 GB ($30\times$)

- **Folding** is a general tool to shrink encrypted data early in large-scale FHE computations.
- Linear algebra is now fast enough that *trading it against bootstrapping pays off*.

Thank you for listening!

arXiv:2601.17561

References I

[BCH+24] Y. Bae et al. “Plaintext-ciphertext matrix multiplication and bootstrapping: fast and fused”. In: *CRYPTO*. 2024.

[BCH+25] Y. Bae et al. “Fast Homomorphic Linear Algebra with BLAS”. In: (2025). Available at https://arxiv.org/abs/2307.09288.

[BGK+24] R. Bloemen et al. *Large-Scale MPC: Scaling Private Iris Code Uniqueness Checks to Millions of Users*. IACR eprint. Available at https://eprint.iacr.org/2024/705. 2024.

References II

[WK24] R. Walch and D. Kales. *MPC's role in advancing World ID privacy features*. https://core.taceo.io/articles/mpc-for-iris-code-uniqueness/. Published on October 16, 2024, retrieved on May 27th, 2026. 2024.