Scaling Private Iris Code Uniqueness Checks to Millions of Users

Remco Bloemen, Daniel Kales, Philipp Sippl, **Roman Walch**

July 23rd, 2024

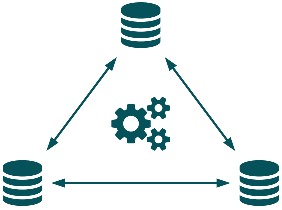
TACEO and Me

- Roman Walch
    - PhD from IAIK, TU Graz, Austria
        - MPC, FHE, ZK, and symmetric ciphers/hash functions
        - finished January 2024
    - Co-founder and lead cryptographer at TACEO
- TACEO
    - Spinoff of TU Graz
    - Currently 11 people
    - Goal is to build the encrypted compute layer
        - Allow to compute on a private shared state using MPC and ZK

Introduction

Secure Multiparty Computation (MPC)

- MPC allows mutually untrusting parties to compute functions on combined input
  - Inputs stay private
- Flexible technology
  - Many protocols and different security levels
    - Semi-honest vs. malicious security
    - Honest vs. dishonest majority
- Potential to bring privacy to many use cases!
  - Privacy-preserving data analysis
  - Threshold signatures and wallets
  - This project: Decentralization



World ID Infrastructure

- World ID
  - Digital identity linked to individuals
  - Unique identifier for each individual
  - Only humans, no AI
  - Authentication via zero-knowledge proofs
- Setup phase for new identifiers
  - Uniqueness enforced via iris scans
  - Compare new iris scan to database
  - Iris scans only used during signup

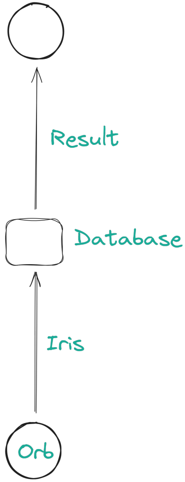
Figure: © Worldcoin

Database Check

- Database previously hosted by Worldcoin Foundation
- Centralized database has privacy concerns and potentially allows misuse
  - Database full with biometric data
  - Partial information about iris can be reconstructed from code
  - Deny giving out an ID for specific individuals

$\Rightarrow$ Decentralize iris code database

  - Split database amongst multiple organizations securely using MPC

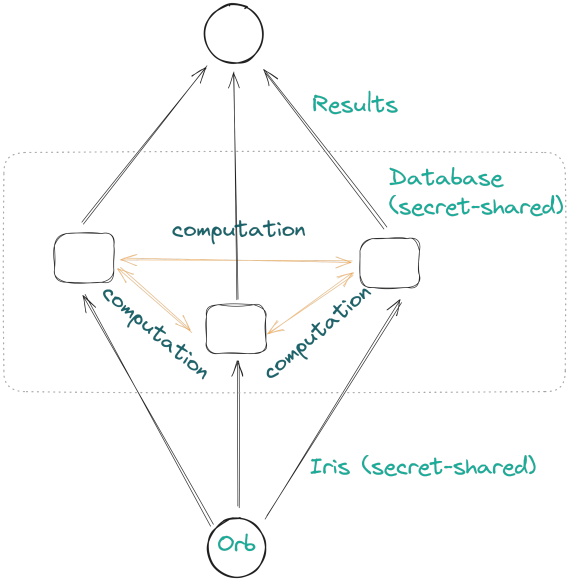


Decentralized Iris Database using MPC

- MPC-shared database
  - Parties have random secret-shares
- Orb secret-shares new iris code
- Compute similarity check protocol in MPC

$\Rightarrow$ No database holder learns database content or new iris code

- But: Overhead of MPC protocols



The Protocol

Iris Similarity Check Protocol

- Iris code $\vec{c}$ with mask $\vec{m}$
  - Mask hides faulty bits
- Match new iris code against whole database
  - Comparison of two iris codes via fractional hamming distance

$$
\begin{aligned}
\vec{m} &= \vec{m}_1 \wedge \vec{m}_2 \\
\texttt{ml} &= \texttt{CountOnes}(\vec{m}) \\
\texttt{hd} &= \texttt{CountOnes}((\vec{c}_1 \oplus \vec{c}_2) \wedge \vec{m}) \\
\texttt{hd}/\texttt{ml} &< t \in \mathbb{R}
\end{aligned}
$$

$\Rightarrow$ Simple protocol, but difficult to do efficiently in MPC

MPC-Problems

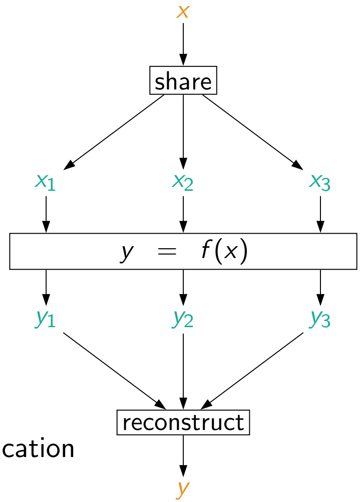
- Mixed operations
  - Hamming-distance: XOR (boolean), Sum (16-bit integer)
  - Comparison and aggregation (boolean)
- Data sizes:
  - 1 iris code $\equiv$ 12 800 bits
  - Current database size: $\sim$6 million iris codes
- Communication overhead
  - Parties exchange randomized data for each multiplication/AND gate
  - Problem for huge database!

$\Rightarrow$ Janus [ELS+24]: $\sim$2k iris code comparisons per minute

Introducing MPC

Additive Secret Sharing

- Share x for n parties: $[x] = (x_1, x_2, \ldots, x_n)$
  - Sample $n-1$ random elements $x_1, \ldots, x_{n-1}$
  - Last share: $x_n = x - \sum_{i=1}^{n-1} x_i$
  - $\Rightarrow$ Reconstruct: $x = \sum_{i=1}^{n} x_i$
- Properties:
  - $n-1$ shares have no information on x
  - All shares required for reconstruction
  - Scheme is linear!
  - Share addition, constant addition, constant multiplication can be computed without interaction
  - Share multiplication requires party-interaction



3-Party Replicated Sharing

- Additive sharing, where each party has two shares
  - Share $[x] = (x_1, x_2, x_3)$
  - Party i has (x_i, x_{i-1})
- Linear operations can be computed without interaction
- Only 2 out of 3 parties required to reconstruct secret (honest majority)
- Multiplication $[z] = [x] \cdot [y]$:
  - Local part: $z_i = x_i \cdot y_i + x_{i-1} \cdot y_i + x_i \cdot y_{i-1} + r_i$ ... with random zero share r_i
  - Transform additive share z_i to replicated share by sending z_i to party $i + 1$
- Dot product $[z] = \sum_i [x_i] \cdot [y_i]$
  - Compute local parts of all multiplications
  - Reshare the sum

3-Party Replicated Sharing

TACEO

- Additive sharing, where each party has two shares
  - Share $[x] = (x_1, x_2, x_3)$
  - Party i has (x_i, x_{i-1})
- Linear operations can be computed without interaction
- Only 2 out of 3 parties required to reconstruct secret (honest majority)
- Multiplication $[z] = [x] \cdot [y]$:
  - Local part: $z_i = x_i \cdot y_i + x_{i-1} \cdot y_i + x_i \cdot y_{i-1} + r_i$ ... with random zero share r_i
  - Transform additive share z_i to replicated share by sending z_i to party $i + 1$
- Dot product $[z] = \sum_i [x_i] \cdot [y_i]$
  - Compute local parts of all multiplications
  - Reshare the sum

3-Party Replicated Sharing

- Additive sharing, where each party has two shares
  - Share $[x] = (x_1, x_2, x_3)$
  - Party i has (x_i, x_{i-1})
- Linear operations can be computed without interaction
- Only 2 out of 3 parties required to reconstruct secret (honest majority)
- Multiplication $[z] = [x] \cdot [y]$:
  - Local part: $z_i = x_i \cdot y_i + x_{i-1} \cdot y_i + x_i \cdot y_{i-1} + r_i$ ... with random zero share r_i
  - Transform additive share z_i to replicated share by sending z_i to party $i + 1$
- Dot product $[z] = \sum_i [x_i] \cdot [y_i]$
  - Compute local parts of all multiplications
  - Reshare the sum

3-Party Replicated Sharing

- Additive sharing, where each party has two shares
  - Share $[x] = (x_1, x_2, x_3)$
  - Party i has (x_i, x_{i-1})
- Linear operations can be computed without interaction
- Only 2 out of 3 parties required to reconstruct secret (honest majority)
- Multiplication $[z] = [x] \cdot [y]$:
  - Local part: $z_i = x_i \cdot y_i + x_{i-1} \cdot y_i + x_i \cdot y_{i-1} + r_i$ ... with random zero share r_i
  - Transform additive share z_i to replicated share by sending z_i to party $i + 1$
- Dot product $[z] = \sum_i [x_i] \cdot [y_i]$
  - Compute local parts of all multiplications
  - Reshare the sum

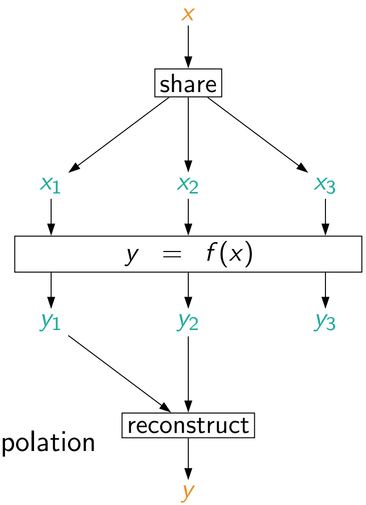
Shamir Sharing

- Different approach to secret sharing over field $\mathbb{F}_p$:
  - Threshold sharing (k-out-of-n)
- Random polynomial with x in constant term

$$p(X) = x + a_1 \cdot X + \ldots + a_t \cdot X^t$$

...with random a_i

- Share $[x] = (p(1), p(2), \ldots, p(n))$
- Reconstruct from $k = t + 1$ shares using Lagrange interpolation



Shamir sharing (cont.)

- Linear operations can be computed locally on shares
- Multiplications:
  - $z_i = x_i \cdot y_i$ is valid share of $z = x \cdot y$
  - But: Polynomial degree doubles
- Our case:
  - $n = 3$ parties, $t = 1$ (honest majority)
- Multiply share with Lagrange coefficient:
  - Shamir with $t = 1 \Rightarrow$ 2-party additive
  - Shamir with $t = 2$ (e.g., after multiplication) $\Rightarrow$ 3-party additive

$\Rightarrow$ Dot-product to replicated sharing: Only communicate result

First Experiments

Efficient Hamming Distance

- Biggest Factor in communication
- Idea: Rewrite to dot product:

$$\begin{aligned}\mathtt{hd}([\vec{a}], [\vec{b}]) &= \mathtt{CountOnes}([\vec{a}] \oplus [\vec{b}]) \\ &= \sum_i [a_i] + \sum_i [b_i] - 2 \cdot \langle [\vec{a}], [\vec{b}] \rangle\end{aligned}$$

  - Linear operations require no communication (sums, multiply by 2, etc.)
  - 1 dot product:
    - Communication equal to one multiplication in replicated sharing or Shamir
- Optimized MPC protocol:
  - Orb shares bits over larger ring $\mathbb{Z}_t$, s.t. computation does not overflow
  - Use replicated sharing or Shamir sharing
  - Public masks $\vec{m}$

  $\Rightarrow$ Communication independent to vector sizes

Efficient Hamming Distance

TACEO

- Biggest Factor in communication
- Idea: Rewrite to dot product:

$$\begin{aligned}\mathtt{hd}([\vec{a}],[\vec{b}]) &= \mathtt{CountOnes}([\vec{a}]\oplus[\vec{b}])\\ &= \sum_i [a_i] + \sum_i [b_i] - 2\cdot\langle[\vec{a}],[\vec{b}]\rangle\end{aligned}$$

  - Linear operations require no communication (sums, multiply by 2, etc.)
  - 1 dot product:
    - Communication equal to one multiplication in replicated sharing or Shamir
- Optimized MPC protocol:
  - Orb shares bits over larger ring $\mathbb{Z}_t$, s.t. computation does not overflow
  - Use replicated sharing or Shamir sharing
  - Public masks $\vec{m}$

  $\Rightarrow$ Communication independent to vector sizes

Threshold comparison

- What about share comparison $[a] < [b]$?
- If subtraction does not overflow, then rewrite to MSB extraction:

$$[a] < [b] \Leftrightarrow \mathtt{MSB}([a-b])$$

$\Rightarrow$ Arithmetic to binary conversion:

$$x_1 + x_2 + x_3 = x \quad \Rightarrow \quad x_1' \oplus x_2' \oplus x_3' = x$$

- In 3-party replicated sharing over rings $\mathbb{Z}_{2^k}$:
  - Split shares $[x_1] = (x_1, 0, 0)$, $[x_2] = (0, x_2, 0)$, $[x_3] = (0, 0, x_3)$
  - Add $[x_1], [x_2], [x_3]$ in MPC using binary addition circuit
- More complicated in prime field $\mathbb{F}_p$

Security Models

- Two security models:
  - Semi-honest version of ABY3 [MR18]
  - Extension for malicious security
- Options for malicious security:
  - Triple-sacrificing (e.g., with cut-and-choose [ABF+17])
  - Distributed zero-knowledge proofs (e.g., SWIFT [KPPS21])
  - SPDZWise MACs (e.g., Fantastic Four [DEK21])
- Our Experiments:
  - Arithmetic: SPDZWise MACs

  $\Rightarrow$ Preserves communication being independent of vector sizes in dot products
  - Binary: Cut-and-choose based triple sacrificing

  $\Rightarrow$ Smallest overhead for AND gates

Experiments

TACEO

Protocol	Runtime (*ms*)	Data (MB)
Plain	134	-
Semi-honest	426	0.598
Malicious	2 900	4.643

Table: Singlethreaded benchmark for DB with 100 000 iris codes.

- Low communication!
- Throughput:
  - Semi-honest: ∼230k iris code comparisons per second
  - Malicious: ∼34k iris code comparisons per second

First Results

- Experiments (including report):
  https://github.com/TaceoLabs/worldcoin-experiments
- Conclusion:
  - Focus on high-performance

  $\Rightarrow$ Focus on semi-honest version
- Lots of ideas for improvement

Improvements

Masked Bitvectors

- Idea: Encode mask in iris code:

c	m	c'
1	1	1
0	1	-1
?	0	0

- We show in paper:

$$\texttt{CountOnes}((\vec{c}_1 \oplus \vec{c}_2) \wedge \vec{m}') < t \cdot \texttt{ml}$$

becomes

$$\langle \vec{c}_1', \vec{c}_2' \rangle > (1 - 2 \cdot t) \cdot \texttt{ml}$$

$\Rightarrow$ Saves two sums and masking $\vec{c}_1$ and $\vec{c}_2$ in MPC

Rep3 vs. Shamir

- Private iris codes, public masks

$$\langle [\vec{c}_1'], [\vec{c}_2'] \rangle > (1 - 2 \cdot t) \cdot \texttt{CountOnes}(\vec{m}_1, \vec{m}_2)$$

$\Rightarrow$ 1 dot product and MSB extraction

- Replicated sharing:
  - Store 2 shares
  - 3 multiplications to calculate 1 MPC multiplications
  - Ring $\mathbb{Z}_{2^k}$: Cheaper MSB-extract
- Shamir sharing:
  - Store 1 share
  - 1 multiplication to calculate 1 MPC multiplications
  - Transform to replicated sharing after dot-product
  - Field $\mathbb{F}_p$: More expensive MSB-extract

Hiding Iris Codes and Masks

- Private iris codes, private masks

$$\langle [\vec{c}_1'], [\vec{c}_2'] \rangle > (1 - 2 \cdot t) \cdot \langle [\vec{m}_1], [\vec{m}_2] \rangle$$

- Share multiplied with $v \in \mathbb{R}$ is expensive in MPC

$\Rightarrow$ Approximate $(1 - 2 \cdot t)$ with $\frac{a}{b}$:

$$b \cdot \langle [\vec{c}_1'], [\vec{c}_2'] \rangle > a \cdot \langle [\vec{m}_1], [\vec{m}_2] \rangle$$

- Problem: $a \cdot [x]$ should not overflow
- Tradeoff:
  - Larger ring $\Rightarrow$ Dot product in larger ring
  - Keep ring size $\Rightarrow$ Lift shares to larger ring in MPC

Benchmarks

- So far this is the status of the paper: https://eprint.iacr.org/2024/705.pdf
- Singlethreaded performance (AWS Graviton3), localhost network
- Dot products:
  - ~2M per second
- Threshold comparison (including lifting):
  - ~10M per second

$\Rightarrow$ 2 Dot products + threshold comparison:

  - Throughput: ~900k iris code comparisons per second

Galois Rings and GPU

Shamir over Galois Ring

- Shamir vs. Rep3: Can we get best of both worlds?
  - Shamir sharing helps with RAM size dot-product compute
  - Replicated sharing over $\mathbb{Z}_{2^k}$ is more efficient for bit operations
  - Conversion is complex and expensive

Why not Shamir over $\mathbb{Z}_{2^k}$?

- Problem: Need sequence of exceptional points for Lagrange interpolation

$$\lambda_i = \prod_{j \neq i} \frac{j}{j - i}$$

  - Pairwise differences of exceptional points need to be invertible
  - Largest sequence of exceptional points for $\mathbb{Z}_{2^k}$: 2
  - Cannot even do 2-party Shamir sharing...

Shamir over Galois Ring (cont.)

- Shamir over Galois Ring $\mathbb{Z}_{2^k}[X]/(X^2 - X - 1)$!
  - Degree-1 polynomial with coefficients in $\mathbb{Z}_{2^k}$, operations modulo $(X^2 - X - 1)$.
  - Length of exceptional sequence: $2^d = 4$
  - Can do 3-party Shamir!
- Naive approach: Embed $\mathbb{Z}_{2^k}$ in constant term of $\mathbb{Z}_{2^k}[X]/(X^2 - X - 1)$.
  - Problem: Overhead of 2x, same as replicated sharing

Shamir over Galois Ring (cont.)

- Packing: Embed 2 elements of $\mathbb{Z}_{2^k}$ as $a_0 + a_1 X$ into a single GR element.
- Smart choice of quotient polynomial:

$$(a_0 + a_1 X) \cdot (b_0 + b_1 X) \mod X^2 - X - 1 = (a_0 b_0 + a_1 b_1) + (a_0 b_1 + a_1 b_0 + a_1 b_1)X$$

- Constant term of Galois-Ring multiplication is dot-product of 2 $\mathbb{Z}_{2^k}$ elements.
  - Lagrange coefficients for reconstruction can be multiplied onto a beforehand.
  - Don't even need to compute X term.

$$[c_0 + c_1 X]_{\texttt{Add}} = [c_0]_{\texttt{Add}} + [c_1]_{\texttt{Add}} X = (\lambda \cdot [a_0 + a_1 X]_{\texttt{Shamir}}) \cdot [b_0 + b_1 X]_{\texttt{Shamir}}$$

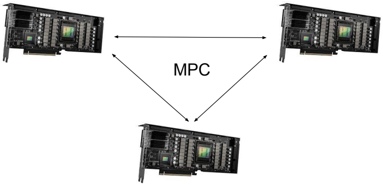
$\Rightarrow$ Store 1 share, 1 multiplication per dot-element, cheap ring-MSB-extract

GPU Implementation

- Dot-product well suited for GPU's
- Nvidia NCCL:
  - GPUs directly communicate over network
  - No GPU $\Leftrightarrow$ CPU data transfer
  - Rust cudarc library

$\Rightarrow$ Execute whole protocol on multiple GPUs

- Result on 3 AWS P5 instances (8x H100 GPUs, 3.2 Tbps)
  - Throughput: $\sim$2.48 billion iris code comparisons per second



Conclusion

- Learnings:
  - Consider GPUs for massively improved throughput
  - Clever protocol optimizations + fast hardware:

  $\Rightarrow$ MPC can be fast enough for real world use cases with millions of users
- Project status:
  - Predecessor (only shared dot-product) deployed
    - Cleartext database is deleted
  - Prototype of full version on GPU done
  - Working on error management, adding new iris to database, tracing info, ...

  $\Rightarrow$ Deployed in the next months

Questions

?

Scaling Private Iris Code Uniqueness Checks to Millions of Users

Remco Bloemen, Daniel Kales, Philipp Sippl, **Roman Walch**

July 23rd, 2024

Bibliography I

[ABF+17] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. ‘‘Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate Per Second Barrier’’. In: IEEE Symposium on Security and Privacy. IEEE Computer Society, 2017, pp. 843–862.

[ADEN21] Mark Abspoel, Anders P. K. Dalskov, Daniel Escudero, and Ariel Nof. ‘‘An Efficient Passive-to-Active Compiler for Honest-Majority MPC over Rings’’. In: ACNS (2). Vol. 12727. LNCS. Springer, 2021, pp. 122–152.

[BGIN19] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. ‘‘Practical Fully Secure Three-Party Computation via Sublinear Distributed Zero-Knowledge Proofs’’. In: CCS. ACM, 2019, pp. 869–886.

[BKSW24] Remco Bloemen, Daniel Kales, Philipp Sippl, and Roman Walch. ‘‘Large-Scale MPC: Scaling Private Iris Code Uniqueness Checks to Millions of Users’’. In: IACR Cryptol. ePrint Arch. (2024), p. 705.

Bibliography II

[DEK21] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. "Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security". In: USENIX Security Symposium. USENIX Association, 2021, pp. 2183–2200.

[ELS+24] Kasra Edalatnejad, Wouter Lueks, Justinas Sukaitis, Vincent Graf Narbel, Massimo Marelli, and Carmela Troncoso. "Janus: Safe Biometric Deduplication for Humanitarian Aid Distribution". In: SP. IEEE, 2024, pp. 115–115.

[KPPS21] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. "SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning". In: USENIX Security Symposium. USENIX Association, 2021, pp. 2651–2668.

[MR18] Payman Mohassel and Peter Rindal. "ABY^3: A Mixed Protocol Framework for Machine Learning". In: CCS. ACM, 2018, pp. 35–52.

[Sha79] Adi Shamir. "How to Share a Secret". In: Commun. ACM 22.11 (1979), pp. 612–613.

Appendix

Malicious Security

- SPDZWise MACs for Arithmetic
  - Extend shares $[x]$ with MAC $[\gamma] = [\alpha \cdot x]$ using MAC-key $[\alpha]$
    - Extend operations to also compute on MACs
  - MAC-check: Ensures correctness for all multiplications
  - Security: Operate on 64-bit shares instead of 16-bit
- Triple sacrificing for Binary:
  - Offline Phase:
    - Precompute m AND triples $([x], [y], [z])$, where $[z] = [x] \wedge [y]$
    - Reduce them to n valid AND triples with cut-and-choose and sacrificing
  - Online phase: Check each AND gate by sacrificing pre-computed triples

Malicious Security

- SPDZWise MACs for Arithmetic
  - Extend shares $[x]$ with MAC $[\gamma] = [\alpha \cdot x]$ using MAC-key $[\alpha]$
    - Extend operations to also compute on MACs
  - MAC-check: Ensures correctness for all multiplications
  - Security: Operate on 64-bit shares instead of 16-bit
- Triple sacrificing for Binary:
  - Offline Phase:
    - Precompute m AND triples $([x], [y], [z])$, where $[z] = [x] \wedge [y]$
    - Reduce them to n valid AND triples with cut-and-choose and sacrificing
  - Online phase: Check each AND gate by sacrificing pre-computed triples

MPC Lifting

TACEO

- We opt for cheaper dot product in $\mathbb{Z}_{2^{16}}$ and 16-bit accuracy for a, b
- Lifting $[x]_{2^{16}}$ to $[x]_{2^{32}}$:
  - Problem: Reconstruction $x_1 + x_2 + x_3 = x \mod 2^{16}$

  $\Rightarrow x_1 + x_2 + x_3 = x + c_1 \cdot 2^{16} + c_2 \cdot 2^{17} \mod 2^{32}$
  - Extract c_1, c_2 using 18-bit binary addition circuit

  $\Rightarrow$ Interpret $[x]_{2^{16}}$ as $[x]_{2^{32}}$ and subtract $c_1 \cdot 2^{16}$ and $c_2 \cdot 2^{17}$
- Trick: $b = 2^{16}$
  - $2^{16} \cdot [x]_{2^{16}} = [2^{16} \cdot x]_{2^{32}}$

  $\Rightarrow$ MPC lifting of $\langle [\vec{c_1}], [\vec{c_2}] \rangle$ is free

MPC Lifting

- We opt for cheaper dot product in $\mathbb{Z}_{2^{16}}$ and 16-bit accuracy for a, b
- Lifting $[x]_{2^{16}}$ to $[x]_{2^{32}}$:
  - Problem: Reconstruction $x_1 + x_2 + x_3 = x \mod 2^{16}$

  $\Rightarrow x_1 + x_2 + x_3 = x + c_1 \cdot 2^{16} + c_2 \cdot 2^{17} \mod 2^{32}$
  - Extract c_1, c_2 using 18-bit binary addition circuit

  $\Rightarrow$ Interpret $[x]_{2^{16}}$ as $[x]_{2^{32}}$ and subtract $c_1 \cdot 2^{16}$ and $c_2 \cdot 2^{17}$
- Trick: $b = 2^{16}$
  - $2^{16} \cdot [x]_{2^{16}} = [2^{16} \cdot x]_{2^{32}}$

  $\Rightarrow$ MPC lifting of $\langle [\vec{c_1'}], [\vec{c_2'}] \rangle$ is free

Security and Comparison to Homomorphic Encryption

- MPC solution:
  - Non-collusion assumption for computing parties
- Homomorphic encryption (HE):
  - HE encrypted database

  ⇒ Key-holder with non-collusion assumption
  - Performance and ciphertext expansion
    - Addition of encrypted 16-bit integer: 100 *ms*
    - Encryption of 1 iris code: 37 MB

  ⇒ Slower, high communication, and larger database size expansion