To Search and Protect:

An MPC/FHE Approach to Encrypted Search

Shai Halevi - AWS

Theme: FHE Is Never Alone

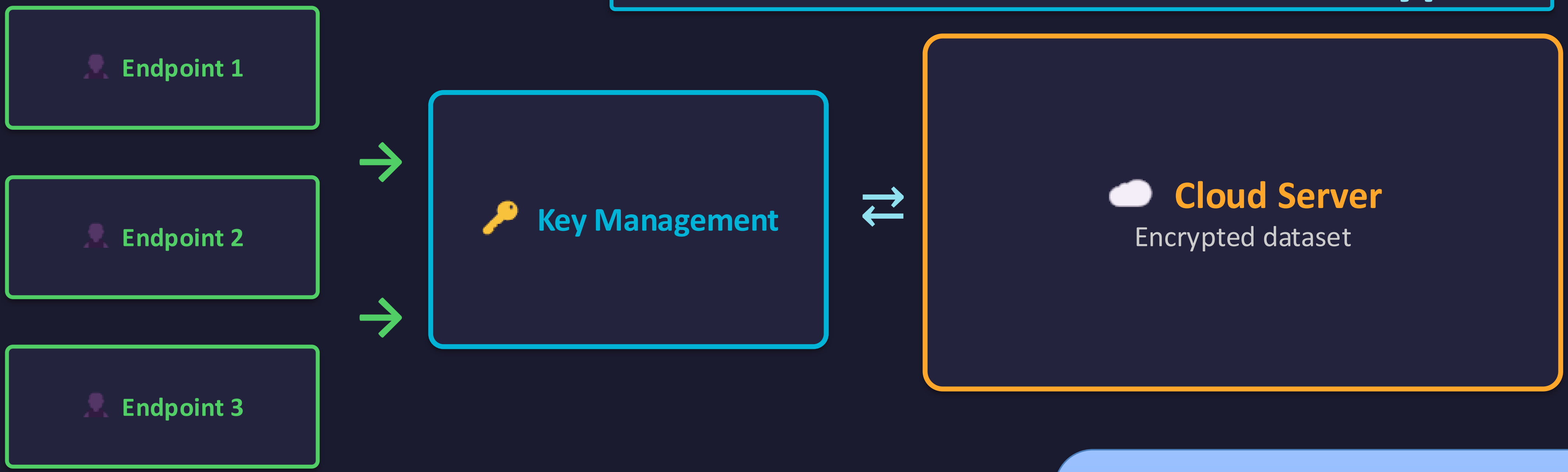
FHE enables computation on encrypted data without interaction...

...but someone still needs to encrypt the question and decrypt the answer

FHE is always a component within a distributed system

This Talk: Looking at Encrypted Search

Both Data and Queries are Encrypted



1. Cloud server — stores encrypted dataset, performs computation
2. End-points — many users/devices issuing search queries
3. Key Management in between — at least for decrypting the answers

Similar to Searchable Symmetric Encryption (SSE), but we don't want the leakage

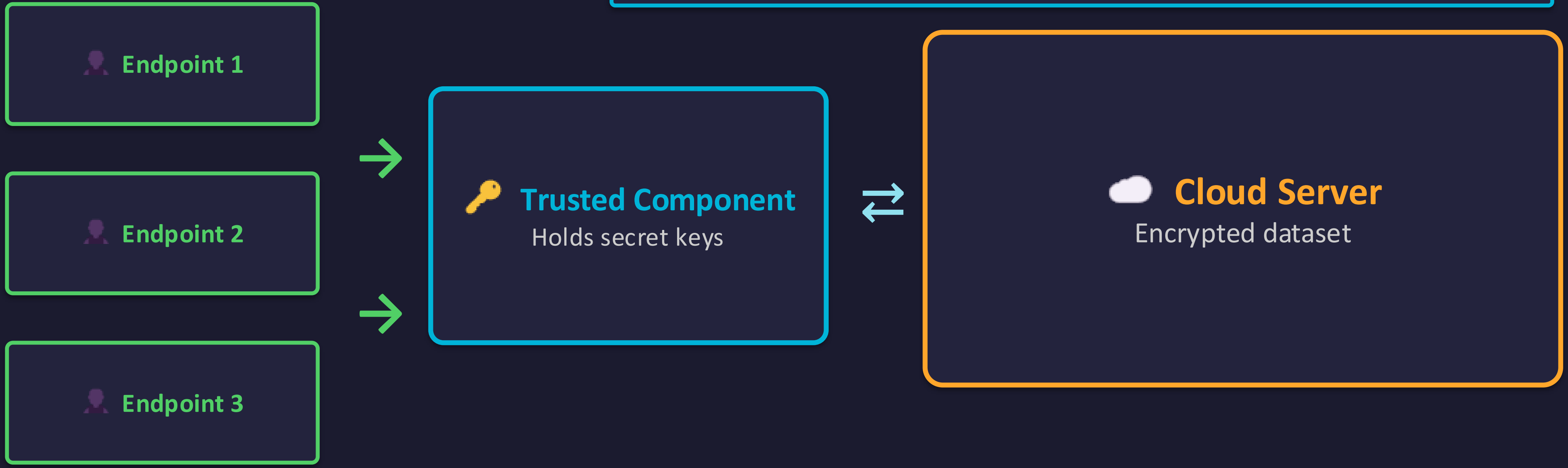
Aside: A Different Encrypted-Search Problem

Only Queries are Encrypted

Also interesting, but not our focus today

This Talk: Looking at Encrypted Search

Both Data and Queries are Encrypted



1. Cloud server — stores encrypted dataset, performs computation
2. End-points — many users/devices issuing search queries
3. Trusted component — holds secret keys, must be available to the endpoints

Why Should Anyone Trust This MORE?

> *"You've replaced a cloud that sees my data with a cloud that does encrypted computation plus a trusted component that holds keys. Why is this better?"*

The trusted component sits on my premises? (maybe, but unlikely at scale)

It's simpler, so easier to audit and harden?

It's distributed across multiple nodes?

It's unaware of data semantics?

It's stateless — no evolving state?

It doesn't store the actual data, so breach damage is limited?

Cryptographic solution must block realistic attack vectors that the system faces

Scenario A — "Keep It Out of the Cloud"

🔑 On-Premises
Trusted Component

⇄
High latency?
Costly bandwidth

☁️ Cloud
Server

Who

Regulated enterprise (finance, healthcare, government)

Why

Compliance requires data never leave customer infrastructure in the clear

Key property

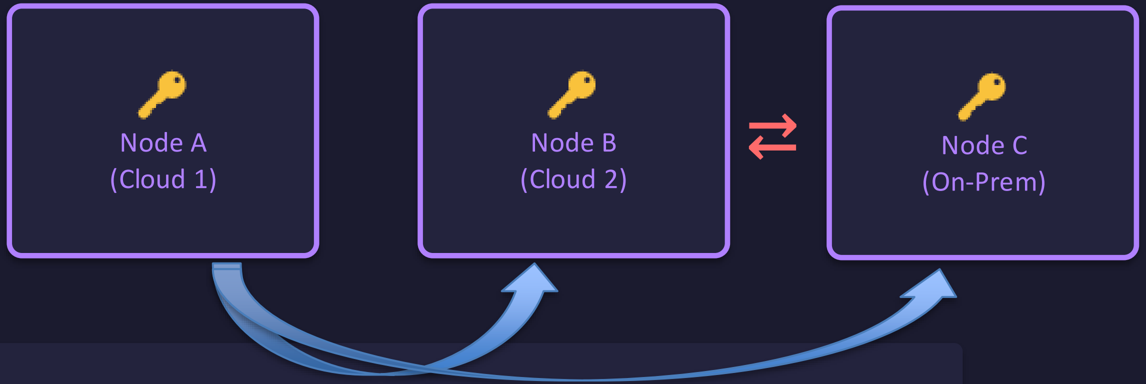
Geographic & administrative separation between keys and data

Implications for technology:

- 🔴 Bandwidth & round trips are expensive → interactive protocols less appealing
- 🟢 Favor non-interactive or low-round solutions → FHE/SSE shine here
- 🟠 Compressing the answer may be needed, but could be computationally expensive

Scenario B — Distributed Trust (2-out-of-3)

- Eliminates a single point of compromise, maybe multi-cloud?
  - Note: Distributed trusted component can also encrypt (not just decrypt)
- They need to communicate with the server, with each other
  - Maybe then can all be near each other?
- Added Complexity, requires a non-collusion assumption
  - Why is non-collusion reasonable to expect?



- An appealing option: one of these parties deals correlated randomness to the other two

Scenario C — Reducing the Attack Surface

Who

Cloud provider or large platform operator

Why

Not selling "encrypted search" to customers — using encryption internally to contain blast radius

Complex Search System

- 10M lines of code
- 200 dependencies
- Daily deploys
- ML models, indices, caches
- Never sees plaintext or keys

🔑 Trusted Component

- Simple, Hardened

Beneficiary: internal product team, not end customer

What Makes "Simple" Simple?

Desirable properties of the trusted component:

Small code base — auditable, formally verifiable

Minimal dependencies — small supply-chain surface

Stateless — no evolving state across queries

Not performance critical — can run in a TEE

Implications for technology:

⚠️ **FHE decryption in trusted component?**

Requires full FHE library — complex?, large?
Side-channel considerations. Is that "simple"?

- **Can we get Keygen/Encrypt/Decrypt formally verified?**

🟢 **Information-theoretic protocols**

Typically simpler, easier to formally verify.

⚠️ May need correlated randomness (where is it coming from?)

A Design Space, Not a Single Answer

Start with the deployment scenario, not the technology.

Let's Get Technical

A concrete encrypted search problem:

k-Nearest Neighbors / Search by Similarity

Useful for semantic search, biometrics

Dataset: N records, each keyed by a vector

Query: a vector

Result: payload of the k records with highest ~~cosine similarity~~ inner product

Two phases, two very different problems:

Phase 1
Encrypted matrix-vector product (EMVP)
(Bi)Linear • Massive data • Structured

We'll talk about this part

Phase 2
Post-processing: extract top-k
Nonlinear • Smaller data

Encrypted Matrix-Vector Product

1. Setup: $\widehat{M} = MatEnc_{sk}(M)$

Encrypted Matrix-Vector Product

1. Setup: $\widehat{M} = MatEnc_{sk}(M)$
2. Query: $\hat{q} = QryEnc_{sk}(q)$

Encrypted Matrix-Vector Product

1. Setup: $\widehat{M} = MatEnc_{sk}(M)$
2. Query: $\hat{q} = QryEnc_{sk}(q)$
3. Answer: $\hat{a} = Mult(\widehat{M}, \hat{q})$

Encrypted Matrix-Vector Product

1. Setup: $\widehat{M} = MatEnc_{sk}(M)$
2. Query: $\hat{q} = QryEnc_{sk}(q)$
3. Answer: $\hat{a} = Mult(\widehat{M}, \hat{q})$
4. Decrypt: $a = Dec_{sk}(\hat{a}) = M \cdot q^T \in F^N$

Deployed EMVP for Encrypted Search

In the context of "proof of humanity", $\text{dim} = 10000$

From HE & Garbed-Circuits: *Janus: Safe Biometric Deduplication for Humanitarian Aid Distribution*, EdalatNejad, Lueks, Sukaitis, Narbel, Marelli, Troncoso, S&P 2024

Using 3PC (much faster): *Large-Scale MPC: Scaling Private Iris Code Uniqueness Checks to Millions of Users*, Bloemen, Gillespie, Sippl, Walch, ePrint /2024

Many Solutions in the Literature

Using degree-2 HE (duh) [BGN05,Gen09,...]

Linear HE when the matrix is public [Tiptoe - HDCZ23]

But actually, linear HE is enough also for secret matrix:

- Run public-matrix EMVP with $M' = M + R$ $\left(R = PRG(sk)\right)$
  - Yields $a' = M' \cdot q^T$
- Client recovers $M \cdot q^T = a' - R \cdot q^T$

Client keeps state (q)

- Can reduce client's work using trapdoored matrix R
  [Braverman-Newman 25, Vaikuntanathan-Zamir 25]
  - pseudorandom R, trapdoor used to speed up multiply-by-R
  - Can be built from LPN, MQ-style assumptions (more if time permits)

New Approach: EMVP via Secret Dual Codes

[Benhamouda-Chen-H-Ishai-Krawczyk-Mour-Rabin-Rosen 25]

- Symmetric Encryption: Client uses secret key to encrypt dataset, queries (as well as for decryption)

- Field-agnostic: Server/client do matrix-vector products over F
  – Somewhat higher-dimension encoded matrix/vectors
- Server work $\approx Nw$ (see next slide)
- Client work $o(Nw)$ (using trapdoored matrices, as low as $\tilde{O}(w)$)
  – Field-agnostic makes it easy to distributed the client (no noise handling)

New Approach: EMVP via Secret Dual Codes

Some variants based on ring-LPN (over-F)

– Everything "optimal upto polylog" (but not concretely efficient)

Other variants based on new hardness assumption

– Variants of "Learning Subspace with Noise" [Dodis-Kalai-Lovett'09]

– Asymptotic overhead over cleartext as small as $(1 + o(1))$

- Can get $<$ 2x for matrix dimensions as small as ~200

Technical Approach: Secret Dual Codes

Pseudorandom codes over $\mathbb{F}$ determined by secret key sk

Optionally: structured codes for better efficiency

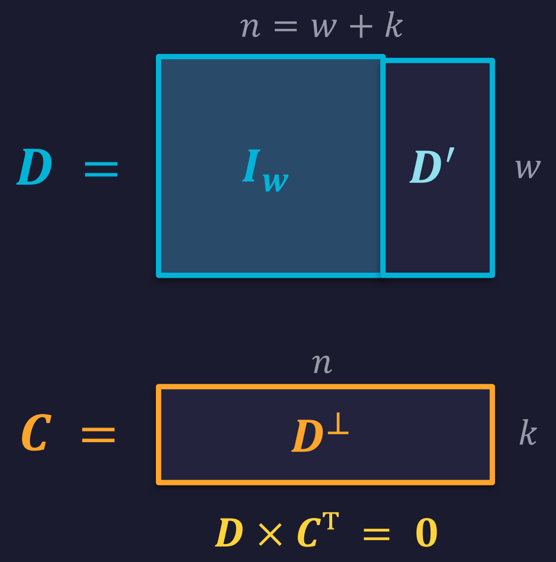
Data Code

Spanned by the rows of $D \in \mathbb{F}^{w \times n}$

D derived from sk, represented by a systematic generating matrix

Query Code

Spanned by rows of $C = D^{\perp} \in \mathbb{F}^{k \times n}$



EMVP Protocol, First Try

Setup: $\widehat{M} = M \times D + R$

$M \in \mathbb{F}^{N \times w}$ — the data matrix (N records, w-dim vectors)

R — pseudorandom matrix determined by sk

$\widehat{M}$ is stored at the server

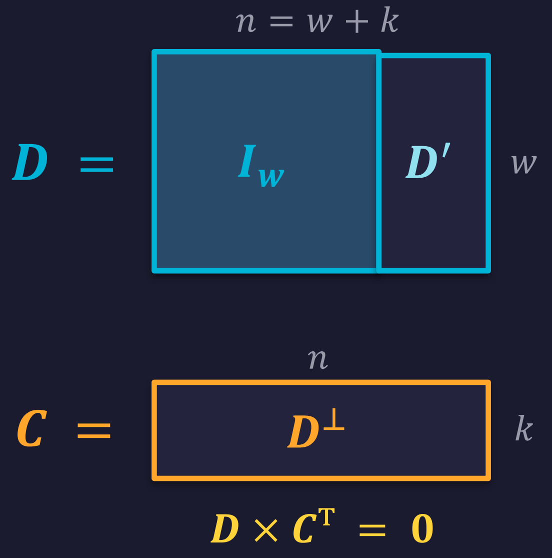
Query: Send to server $\tilde{q}_i = (q_i|0) + c_i$

$c_i = r_i^T C$ is a random codeword in C

Answer: Server returns $\tilde{a}_i = \widehat{M} \cdot \tilde{q}_i^T$

$$\tilde{a}_i = M \cdot q^T + R \cdot \tilde{q}^T$$

Decryption: Output $a = \tilde{a}_i - R \cdot \tilde{q}_i = M\, q^T$



EMVP Protocol, First Try: Efficiency

Setup: $\widehat{\boldsymbol{M}} = \boldsymbol{M} \times \boldsymbol{D} + \boldsymbol{R}$

Storage overhead: $^{n}/_{w} = 1 + {}^{k}/_{w}$

$M \in \mathbb{F}^{N \times w}$ — the data matrix (N records, w-dim vectors)

R — pseudorandom matrix determined by sk

$\widehat{\boldsymbol{M}}$ is stored at the server

Query: **Send to server** $\tilde{\boldsymbol{q}}_i = (\boldsymbol{q}_i | \boldsymbol{0}) + \boldsymbol{c}_i$

Upload overhead: $^{n}/_{w} = 1 + {}^{k}/_{w}$

$c_i = r_i^T C$ is a random codeword in C

Answer: **Server returns** $\tilde{\boldsymbol{a}}_i = \widehat{\boldsymbol{M}} \cdot \tilde{\boldsymbol{q}}_i^T$

No download overhead

$$\tilde{a}_i = M \cdot q^T + R \cdot \tilde{q}^T$$

Decryption: **Output** $\boldsymbol{a} = \tilde{\boldsymbol{a}}_i - \boldsymbol{R} \cdot \tilde{\boldsymbol{q}}_i = \boldsymbol{M}\, \boldsymbol{q}^T$

EMVP Protocol, First Try: Security?

Data encryption: $\widehat{M} = M \times D + R$, R is pseudorandom so M is hidden

Query encryption: $\tilde{q}_i = (q_i|0) + r_i^T C$

All the q_i's are the same → All $\tilde{q}_i$'s live in a coset of C

→ Rank$(\tilde{q}_1, \tilde{q}_2, \ldots) \leq k + 1$

If the qi's are random → Rank$(\tilde{q}_1, \tilde{q}_2, \ldots) \sim n$

Detectable after enough queries observed!

We want the $\tilde{q}_i$'s to be pseudorandom

EMVP Protocol, With Better Security

Idea: Make $\tilde{q}_i$ pseudorandom by adding "noise"

- But standard LPN or LWE noise hurts correctness

Instead:

- Mixture noise: w/ prob $\mu < 1$, replace $\tilde{q}_i$ by a sample from a different distribution (e.g., uniform)
  - *Repeat/encode-across-columns to correct failures*
- Planting noise: reveal a random low-dimensional subspace $\tilde{Q}_i$ containing $\tilde{q}_i$

Query: **Compute $\tilde{q}_i$ as before. Send to server $\tilde{Q}_i = [q_{i,1}^T | \dots | q_{i,s}^T]$**

such that $\tilde{q}_i = \sum_j \alpha_{i,j} \cdot q_{i,j}$ for random scalars $\alpha_{i,j}$

Additional $s \times$ overhead

Answer: **Send back $A_i = \widehat{M} \cdot \tilde{Q}_i$**

Decrypt: **Output $a = A_i \cdot (\alpha_{i,1}, \dots, \alpha_{i,s})^T - R \cdot \tilde{q}_i$**

Security related to "Learning Subspace with Noise" (LNS), see next slide

LSN Assumptions (Learning Subspace with Noise)

Identify "perturbed" vectors from a secret linear/affine subspace

New(ish) type of hardness assumptions:

- Originated for leakage-resilience
  [Dodis–Kalai–Lovett 09]
- Structured variants implicit in doubly-efficient secret-key PIR
  [Boyle–Ishai–Pass–Wootters 17, Canetti–Holmgren–Richelson 17]
- Search-LSN studied by learning theory community
  [Chen–De–Vijayaraghavan 21]
- Usage for PIR, similar to ours [Chen-Ishai-Mour-Rosen 25]
- Equivalent to LPN when $\mathbb{F}$ is small and $n = k + 1$

EMVP Protocol, Improving Efficiency

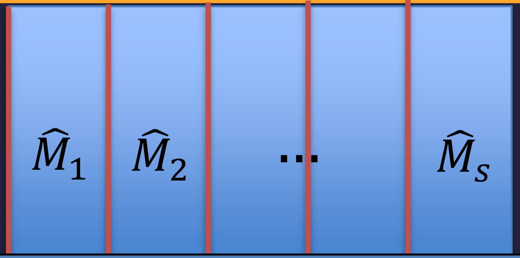
Query:

Upload overhead: ${}^{n}/_{w} = 1 + {}^{k}/_{w}$

- Compute $\tilde{q}_i$ as before
- Break $\tilde{q}_i$ into s blocks, multiply j'th block by a random scalar $\alpha_{i,j}^{-1}$

- Concatenate and send to server: $\hat{q}_i = \left(\hat{q}_{i,1} = \alpha_{i,1}^{-1} \cdot block_{i,1},\ \dots, \hat{q}_{i,s} = \alpha_{i,s}^{-1} \cdot block_{i,s}\right)$

Answer



- Break $\widehat{M}$ into column-blocks $\left[\widehat{M}_1 | \cdots | \widehat{M}_s\right]$
- Return $A_i = \left[\widehat{M}_1 \cdot \hat{q}_{i,1}^T | \cdots | \widehat{M}_s \cdot \hat{q}_{i,s}^T\right]$

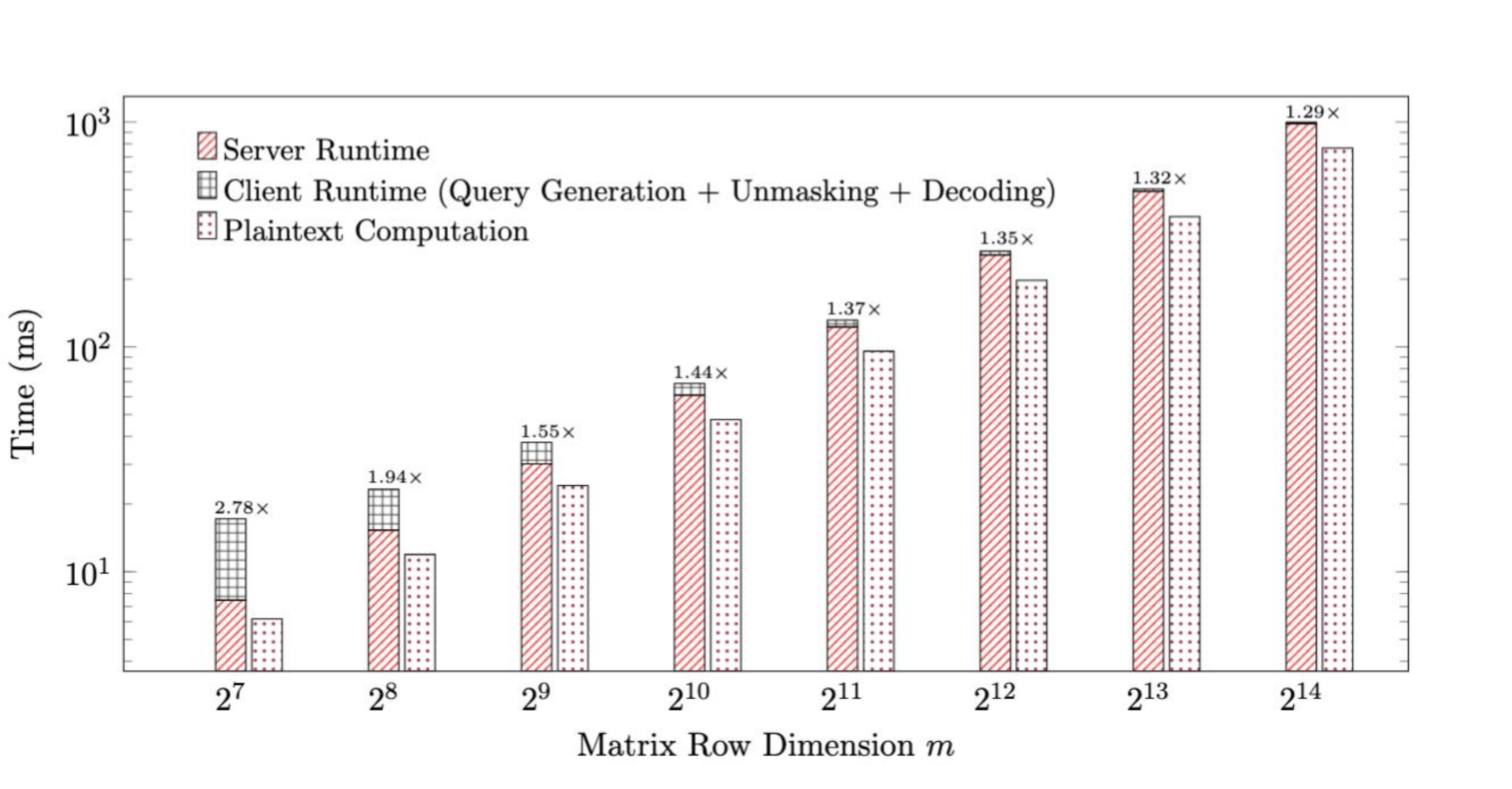
Download overhead $s \times$

Decrypt: Output $a = A_i \cdot \left(\alpha_{i,1}, \dots, \alpha_{i,s}\right)^T - R \cdot \tilde{q}_i$

Client multiplies $N \times s$ matrix by a vector

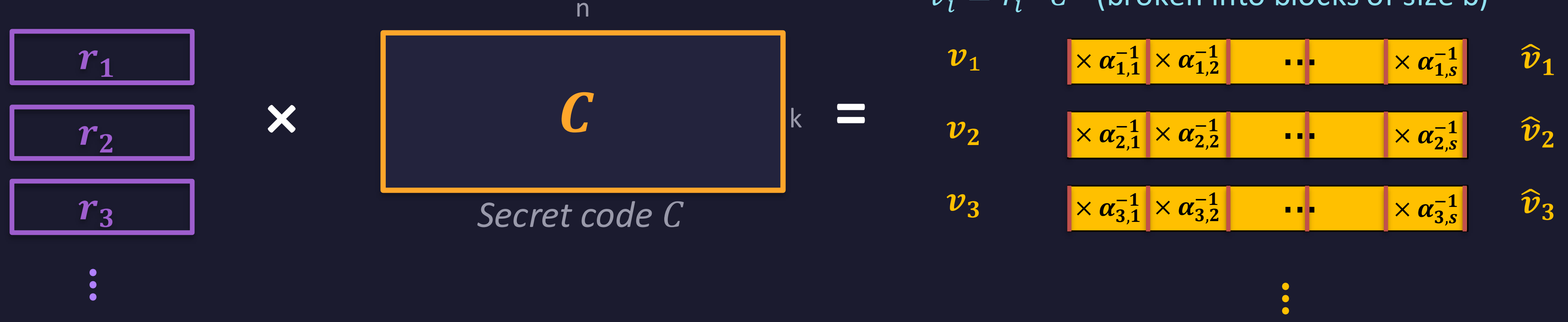
Some Timing Results

- (w, k, b) = (10000, 2600, 140), over Z_p with $|p| = 32$



Security Under "Split-LSN" Hardness

$v_i = r_i \cdot C^T$ (broken into blocks of size b)



How hard is it to distinguish the vi's from random?

Algebraic attacks (à la [Raz 09, Arora-Ge 11])
$\approx b^{k/b}$ work

Algebraic Attack

Theorem: For any (k, n)-code C and $d < n - k$, $\exists$ a non-zero degree-d polynomial $P_C(\cdot)$ s.t.

$$\text{span}(\vec{v}_1, \ldots, \vec{v}_d) \cap C \neq \{\vec{0}\} \;\Rightarrow\; P_C(\vec{v}_1, \ldots, \vec{v}_d) = 0$$

- For us, each block is a vector, $d = \lceil {}^{k}/_{b} \rceil + 1$
- The monomials are from a universe of size b^d (regardless of C)

The attack:

- From each $\hat{q}_i$, prepare a dimension-b^d vector of potential monomials

$$\vec{u}_i = block_{i,1} \otimes block_{i,2} \otimes \cdots \otimes block_{i,d}$$

- The $\vec{u}_i$'s are all orthogonal to the coefficient vector of P_C
- So they are rank-deficient, observable after seeing b^d of them

Constructing Trapdoored Matrices from LPN

- H is a public code with fast syndrome, for which LPN holds
- E is a secret sparse matrix, derived from the secret key
  - R is pseudorandom under dual-LPN $(\boldsymbol{H}, \boldsymbol{He}) \approx (\boldsymbol{H}, \boldsymbol{r})$
- $\vec{w} = R \times \vec{u}$ is fast because:
  - $\vec{v} = E \times \vec{u}$ is fast since E is very sparse
  - $\vec{w} = H \times \vec{v}$ is fast since H has fast syndrome

Even Faster Trapdoored Matrices

Based on various MQ-style assumptions

$$R = S_L \times \Pi_L \times S \times \Pi_L \times S_R$$

- The S'es are structured secret matrices
  - E.g., S_L multiplies by $(1, r_L)$ over F_{p^k}
- The Π's are public permutation matrices
- Speculative (but plausible) new hardness assumptions

EMVP Protocol: The Moral (1)

- Super-Efficient “poor man’s FHE”
  - Delegating (secret) linear functions, secrecy almost for free
    - Can be made even more efficient by batching many queries together
  - Can be extended to bilinear forms with very little overhead [Joshi-Wagner-H-Mishra 26] (used for lexical search)
  - Very easy to authenticate the server’s answer
    - Using homomorphic MACs (a-la-SPDZ)
    - This “verifiable FHE”, malicious security
  - EMVP is “easier than” (does not imply) linear HE
    - Unlikely to even imply CRH/OT/PKE

EMVP Protocol: The Moral (2)

- LSN-style assumptions require further study, but…
  - Everything is achievable under LPN with polylog overhead
  - Hiding data relies only on standard assumptions
  - Queries are “locally hidden” information-theoretically

Back to Encrypted Search

Two phases, two very different problems:

Phase 1	Phase 2
Encrypted matrix-vector product (EMVP) (Bi)Linear • Massive data • Structured	Post-processing: extract top-k Nonlinear • Smaller data

- Can use the new EMVP Protocol
- Before sending the answer, the server holds an encryption of $M \cdot q^T$
  - Much smaller than the full M, q
  - Dimension $N \times s$ instead $N \times w$

- Here we need “real FHE”/MPC
- Deep (F)HE to do everything on the server
- Or additive-HE to transform it to additive sharing of the result
  - Then run Client-Server 2PC
  - Very efficient aggregation (OR, Max, Histogram) [Benhamouda-H-Ishai-Rathee 26]

Key Takeaways

FHE is never alone

- Thinking of it as part of a larger distributed system may offer simplification options
- Always start from the deployment scenario

EMVP compression + FHE/MPC post-processing is appealing

- Could be effective in many settings

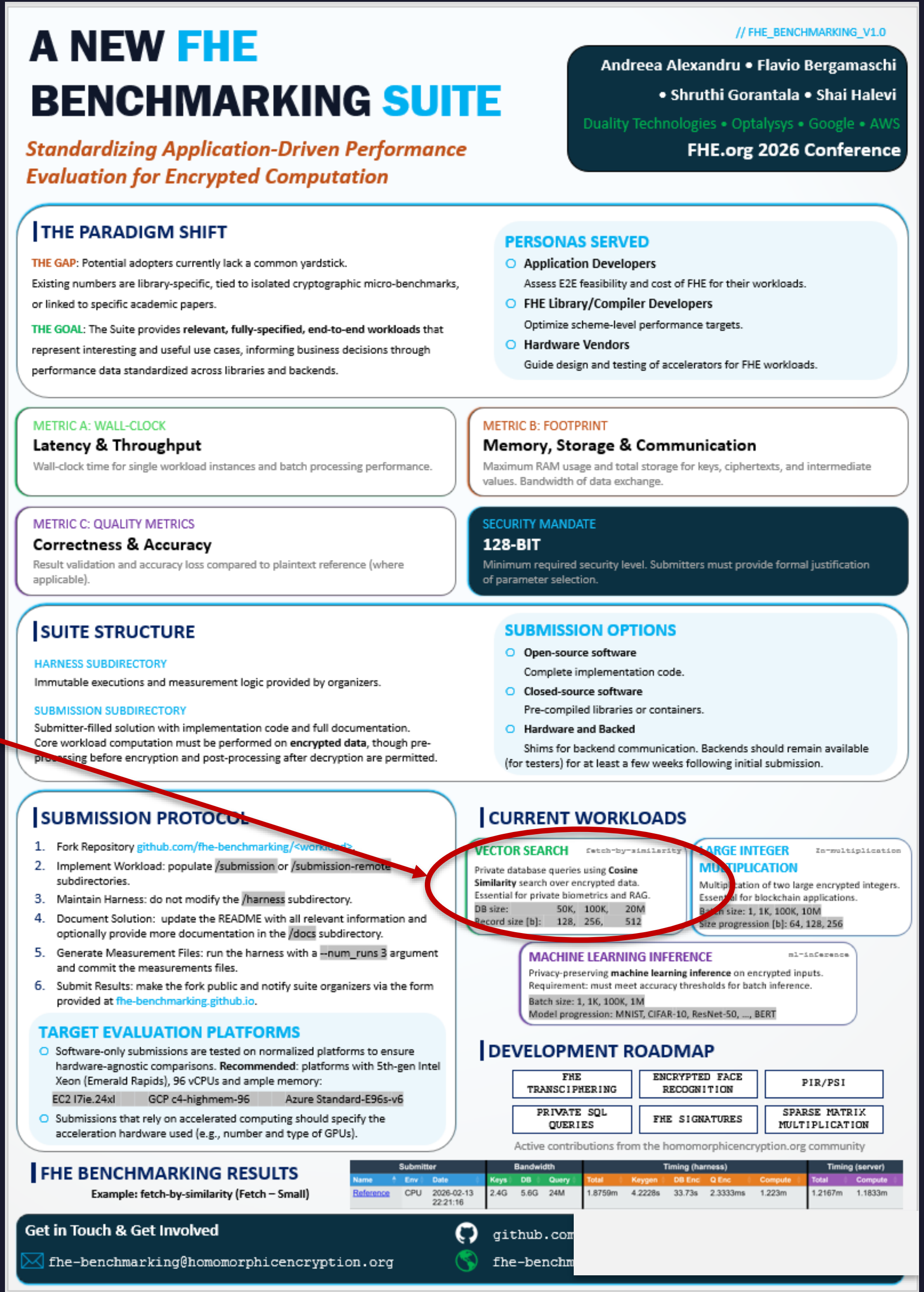
and a Reminder

A new FHE benchmarking suite

https://fhe-benchmarking.org

Exactly the encrypted-kNN search that I described here

Please submit your solutions



// FHE_BENCHMARKING_V1.0

A NEW FHE BENCHMARKING SUITE

Standardizing Application-Driven Performance Evaluation for Encrypted Computation

Andreea Alexandru • Flavio Bergamaschi • Shruthi Gorantala • Shai Halevi
Duality Technologies • Optalysys • Google • AWS
FHE.org 2026 Conference

THE PARADIGM SHIFT

THE GAP: Potential adopters currently lack a common yardstick. Existing numbers are library-specific, tied to isolated cryptographic micro-benchmarks, or linked to specific academic papers.

THE GOAL: The Suite provides **relevant, fully-specified, end-to-end workloads** that represent interesting and useful use cases, informing business decisions through performance data standardized across libraries and backends.

PERSONAS SERVED

- **Application Developers**
  Assess E2E feasibility and cost of FHE for their workloads.
- **FHE Library/Compiler Developers**
  Optimize scheme-level performance targets.
- **Hardware Vendors**
  Guide design and testing of accelerators for FHE workloads.

METRIC A: WALL-CLOCK
Latency & Throughput
Wall-clock time for single workload instances and batch processing performance.

METRIC B: FOOTPRINT
Memory, Storage & Communication
Maximum RAM usage and total storage for keys, ciphertexts, and intermediate values. Bandwidth of data exchange.

METRIC C: QUALITY METRICS
Correctness & Accuracy
Result validation and accuracy loss compared to plaintext reference (where applicable).

SECURITY MANDATE
128-BIT
Minimum required security level. Submitters must provide formal justification of parameter selection.

SUITE STRUCTURE

HARNESS SUBDIRECTORY
Immutable executions and measurement logic provided by organizers.

SUBMISSION SUBDIRECTORY
Submitter-filled solution with implementation code and full documentation. Core workload computation must be performed on **encrypted data**, though pre-processing before encryption and post-processing after decryption are permitted.

SUBMISSION OPTIONS

- **Open-source software**
  Complete implementation code.
- **Closed-source software**
  Pre-compiled libraries or containers.
- **Hardware and Backed**
  Shims for backend communication. Backends should remain available (for testers) for at least a few weeks following initial submission.

SUBMISSION PROTOCOL

1. Fork Repository github.com/fhe-benchmarking/<workload>
2. Implement Workload: populate /submission or /submission-remote subdirectories.
3. Maintain Harness: do not modify the /harness subdirectory.
4. Document Solution: update the README with all relevant information and optionally provide more documentation in the /docs subdirectory.
5. Generate Measurement Files: run the harness with a --num_runs 3 argument and commit the measurements files.
6. Submit Results: make the fork public and notify suite organizers via the form provided at fhe-benchmarking.github.io.

TARGET EVALUATION PLATFORMS

- Software-only submissions are tested on normalized platforms to ensure hardware-agnostic comparisons. **Recommended**: platforms with 5th-gen Intel Xeon (Emerald Rapids), 96 vCPUs and ample memory:
  EC2 I7ie.24xl GCP c4-highmem-96 Azure Standard-E96s-v6
- Submissions that rely on accelerated computing should specify the acceleration hardware used (e.g., number and type of GPUs).

CURRENT WORKLOADS

VECTOR SEARCH `fetch-by-similarity`
Private database queries using **Cosine Similarity** search over encrypted data. Essential for private biometrics and RAG.
DB size: 50K, 100K, 20M
Record size [b]: 128, 256, 512

LARGE INTEGER MULTIPLICATION `Ip-multiplication`
Multiplication of two large encrypted integers. Essential for blockchain applications.
Batch size: 1, 1K, 100K, 10M
Size progression [b]: 64, 128, 256

MACHINE LEARNING INFERENCE `ml-inference`
Privacy-preserving **machine learning inference** on encrypted inputs. Requirement: must meet accuracy thresholds for batch inference.
Batch size: 1, 1K, 100K, 1M
Model progression: MNIST, CIFAR-10, ResNet-50, ..., BERT

DEVELOPMENT ROADMAP

FHE TRANSCIPHERING	ENCRYPTED FACE RECOGNITION	PIR/PSI
PRIVATE SQL QUERIES	FHE SIGNATURES	SPARSE MATRIX MULTIPLICATION

Active contributions from the homomorphicencryption.org community

FHE BENCHMARKING RESULTS

Example: fetch-by-similarity (Fetch – Small)

Submitter			Bandwidth			Timing (harness)					Timing (server)	
Name	Env	Date	Keys	DB	Query	Total	Keygen	DB Enc	Q Enc	Compute	Total	Compute
Reference	CPU	2026-02-13 22:21:16	2.4G	5.6G	24M	1.8759m	4.2228s	33.73s	2.3333ms	1.223m	1.2167m	1.1833m

Get in Touch & Get Involved
fhe-benchmarking@homomorphicencryption.org
github.com
fhe-benchm


Questions?