MUSES: Efficient Multi-User Searchable Encrypted Database

Tung Le, Rouzbeh Behnia, Jorge Guajardo, Thang Hoang

USENIX Security 2024

Philadelphia, Pennsylvania

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

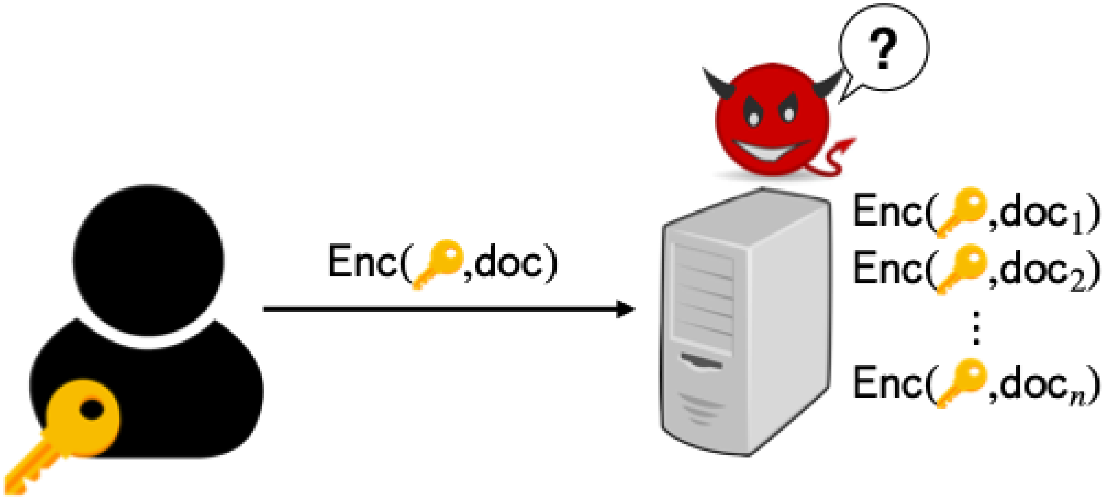
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Overview

End-to-end encrypted systems are increasingly popular

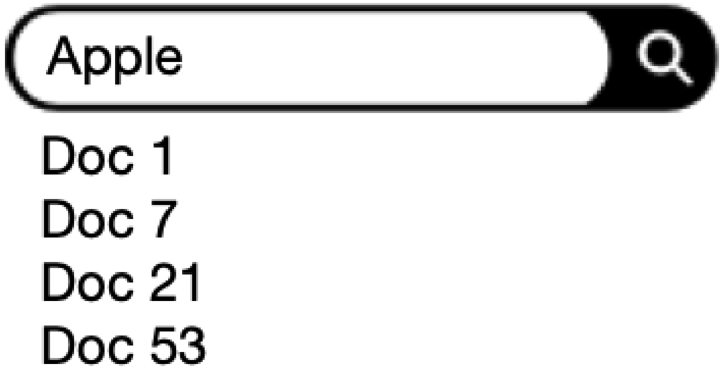
Keybase PREVEIL SpiderOak sync.com tresorit

Provide strong security guarantees if attacker compromises server

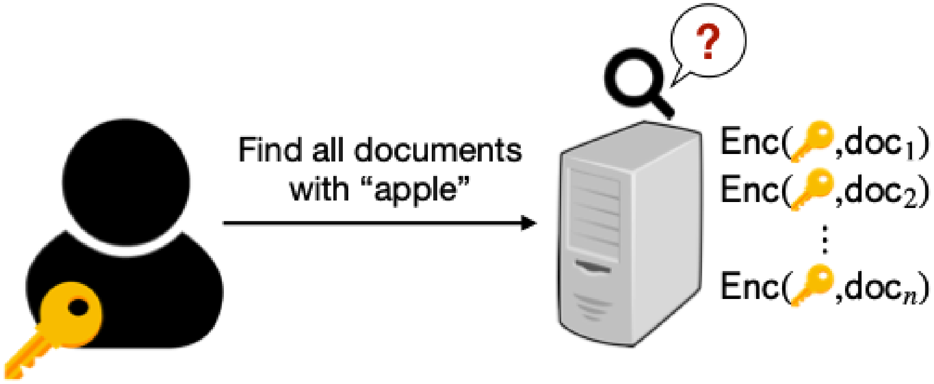


Overview

Users expect the ability to execute search



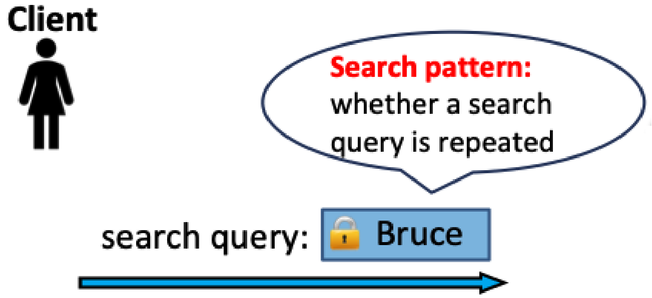
Challenge: server cannot decrypt data to search

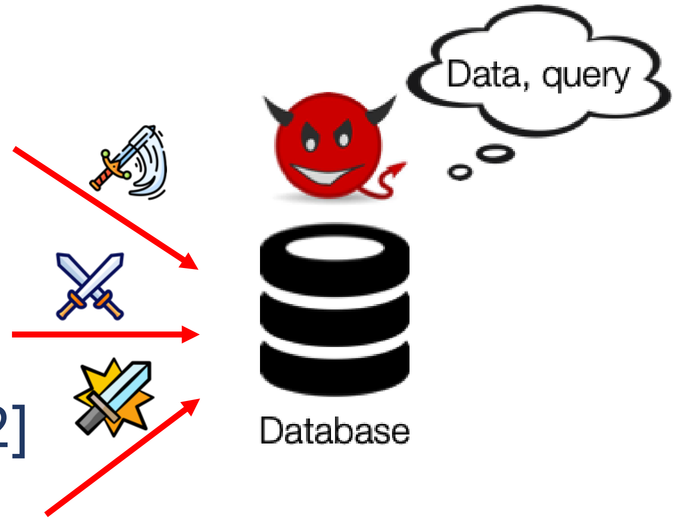


Overview

Leakage-abuse Attacks in Searchable Encryption:

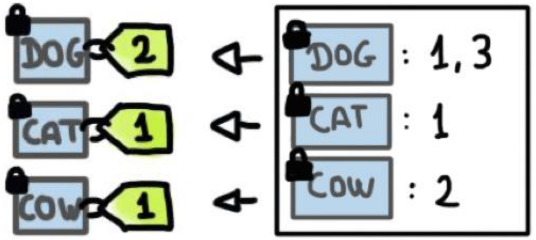
- **Search Pattern:** [IKK’12, LZWT’14, OK’21]





- **Result Pattern:** [IKK’12, CGPR’14, ZKP’16, LCNL’22, OK’22]

Result pattern: Repetition of returned matching documents



Volume pattern: The number of matching documents

- **Volume Pattern:** [BKM’19, LCNL’22, OK’22, ZWXYL’23]

MUSES: Efficient Multi-Writer Encrypted Search

Practical examples of searchable encrypted platforms:

- ✓ Cosmian
- ✓ Amazon AWS Database Encryption SDK
- ✓ MongoDB

We propose **MUSES**, which features by:

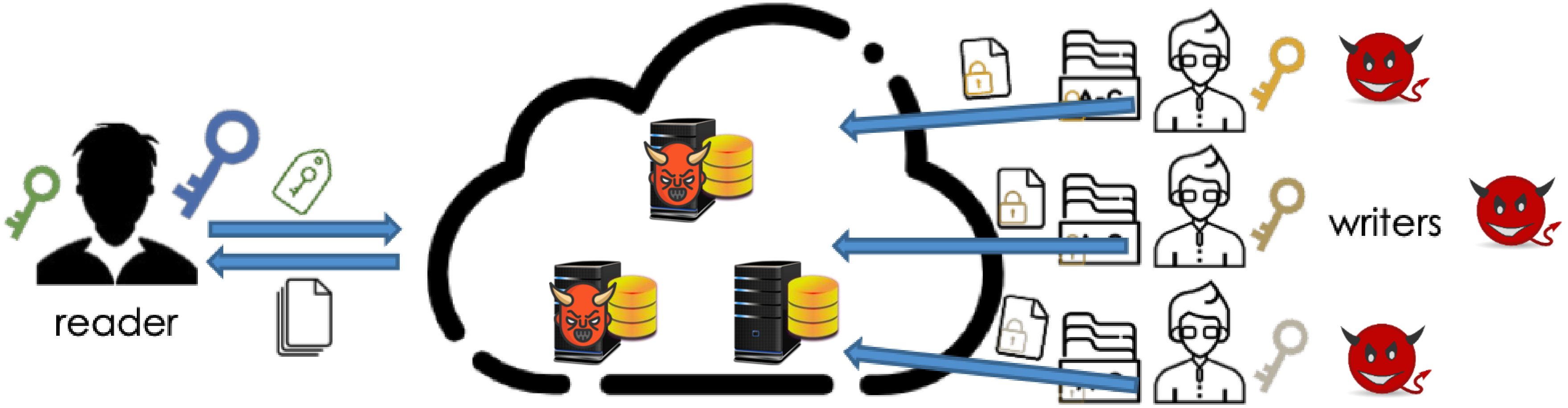
- ✓ Multi-writer support
- ✓ Hide all statistical information, including search, result, and volume patterns
- ✓ Minimal user overhead (regarding computation and communication costs)

System and Threat Model

An honest reader who holds a public/private key pair

Multiple writers, where each owns its independent database

L servers, in which up to $L-1$ servers can be corrupt



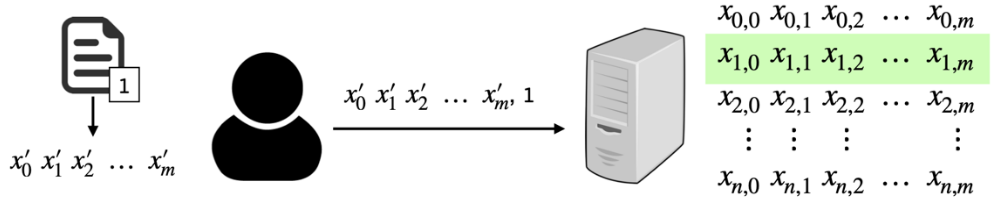
Security against *semi-honest servers* and potentially *corrupt writers*

Search Index

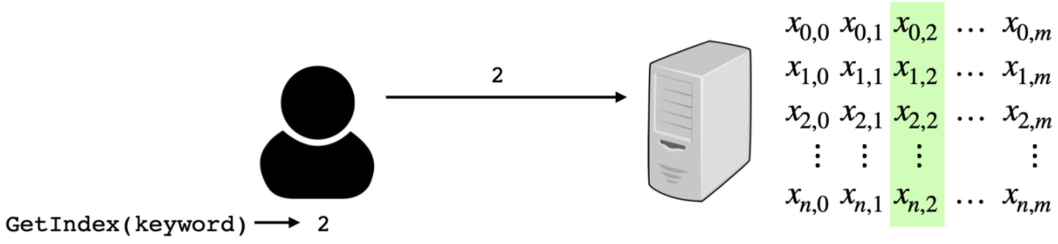
	"amazon"	"google"	"netflix"	...	"apple"
doc 1	1	0	1	0	1
doc 2	0	1	1	0	0
doc 3	1	1	0	0	0
...	0	1	0	1	1
doc N	1	0	0	1	1

Bitmap for keywords in doc 2

Update:

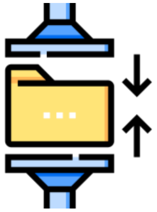


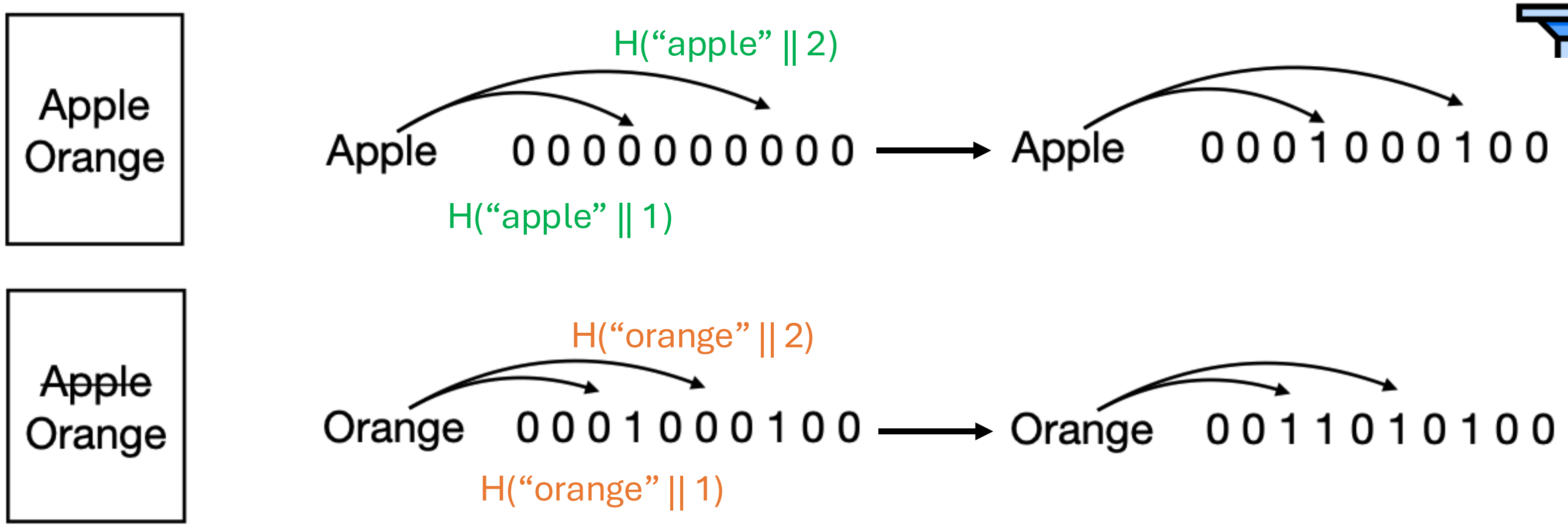
Search:



Compressing Search Index

Using Bloom filter to compress the search index

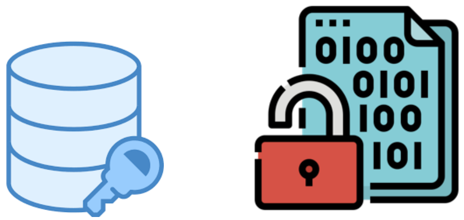




$$
\begin{matrix}
x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\
x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\
x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\
\vdots & \vdots & \vdots & & \vdots \\
x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m}
\end{matrix}
$$

Search index with Bloom filters

Encrypted Search Index



Key-Homomorphic Pseudorandom Function (KH-PRF): [BLMR'13]

Learning with Rounding (LWR):

$$F: \mathbb{Z}_q^n \times \{0,1\}^* \rightarrow \mathbb{Z}_p$$

$$F\big(\underbrace{\boldsymbol{k}^{(1)} + \boldsymbol{k}^{(2)}}_{\boldsymbol{k} = \boldsymbol{k}^{(1)} + \boldsymbol{k}^{(2)}}, s\big) = F\big(\boldsymbol{k}^{(1)}, s\big) + F\big(\boldsymbol{k}^{(2)}, s\big) + e,\ \ e \in \{0,1\} \text{ is a small error}$$

$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix} \longrightarrow \begin{matrix} \mathsf{Enc}_k(x_{0,0}) & \mathsf{Enc}_k(x_{0,1}) & \mathsf{Enc}_k(x_{0,2}) & \ldots & \mathsf{Enc}_k(x_{0,m}) \\ \mathsf{Enc}_k(x_{1,0}) & \mathsf{Enc}_k(x_{1,1}) & \mathsf{Enc}_k(x_{1,2}) & \ldots & \mathsf{Enc}_k(x_{1,m}) \\ \mathsf{Enc}_k(x_{2,0}) & \mathsf{Enc}_k(x_{2,1}) & \mathsf{Enc}_k(x_{2,2}) & \ldots & \mathsf{Enc}_k(x_{2,m}) \\ \vdots & \vdots & \vdots & & \vdots \\ \mathsf{Enc}_k(x_{n,0}) & \mathsf{Enc}_k(x_{n,1}) & \mathsf{Enc}_k(x_{n,2}) & \ldots & \mathsf{Enc}_k(x_{n,m}) \end{matrix}$$

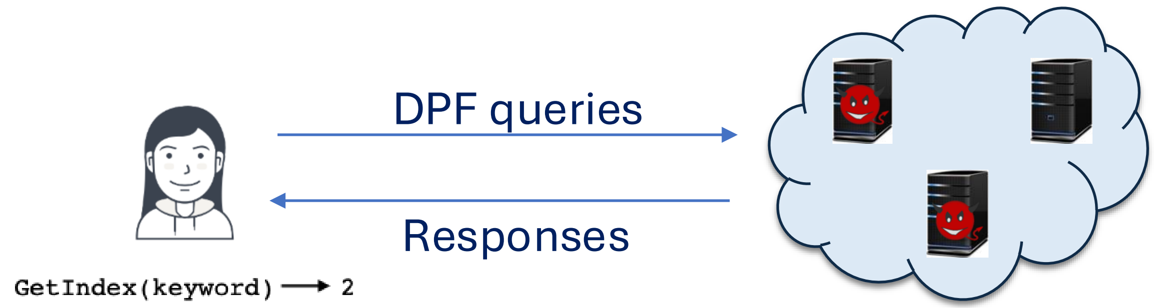
$$\mathrm{Enc}_k(x_{i,j}) = x_{i,j} + F\big(\boldsymbol{k}_j, s_i\big) \pmod p$$

Distributed Point Functions (DPFs)

- Uses multiple servers to hide which element the user is retrieving
- If at least one server is honest, an attacker cannot learn the index requested
- Requires a linear scan over the entire array

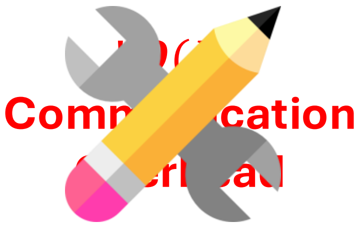
Leveraging DPFs to search

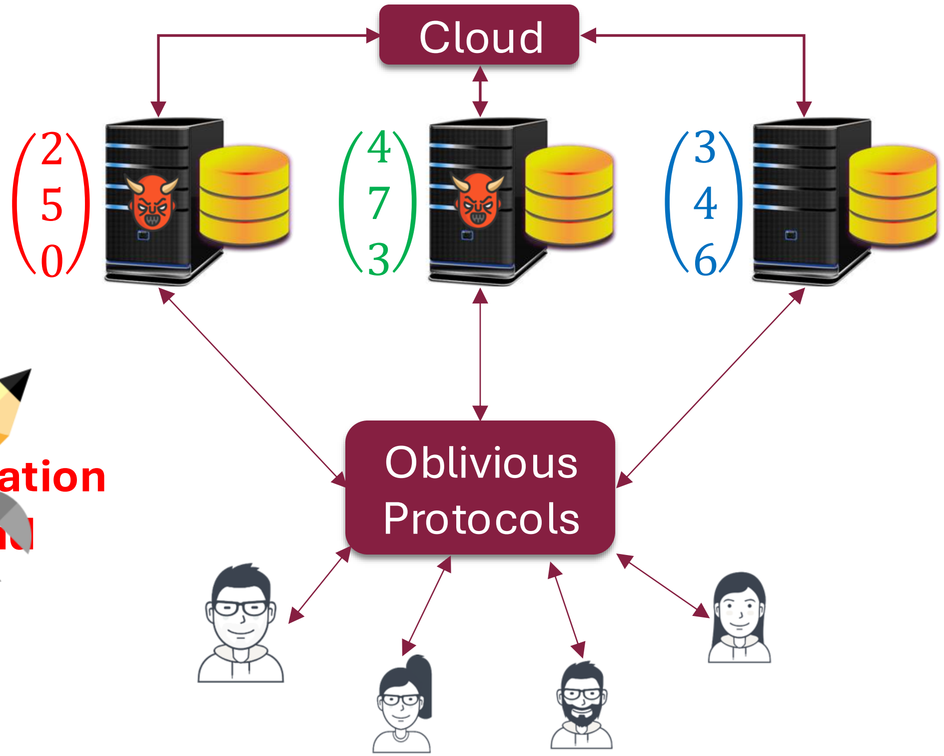
If at least one trust server is honest, MUSES hides search patterns



$$\begin{pmatrix} x_{0,2} \\ x_{1,2} \\ x_{2,2} \\ \vdots \\ x_{n,2} \end{pmatrix}$$

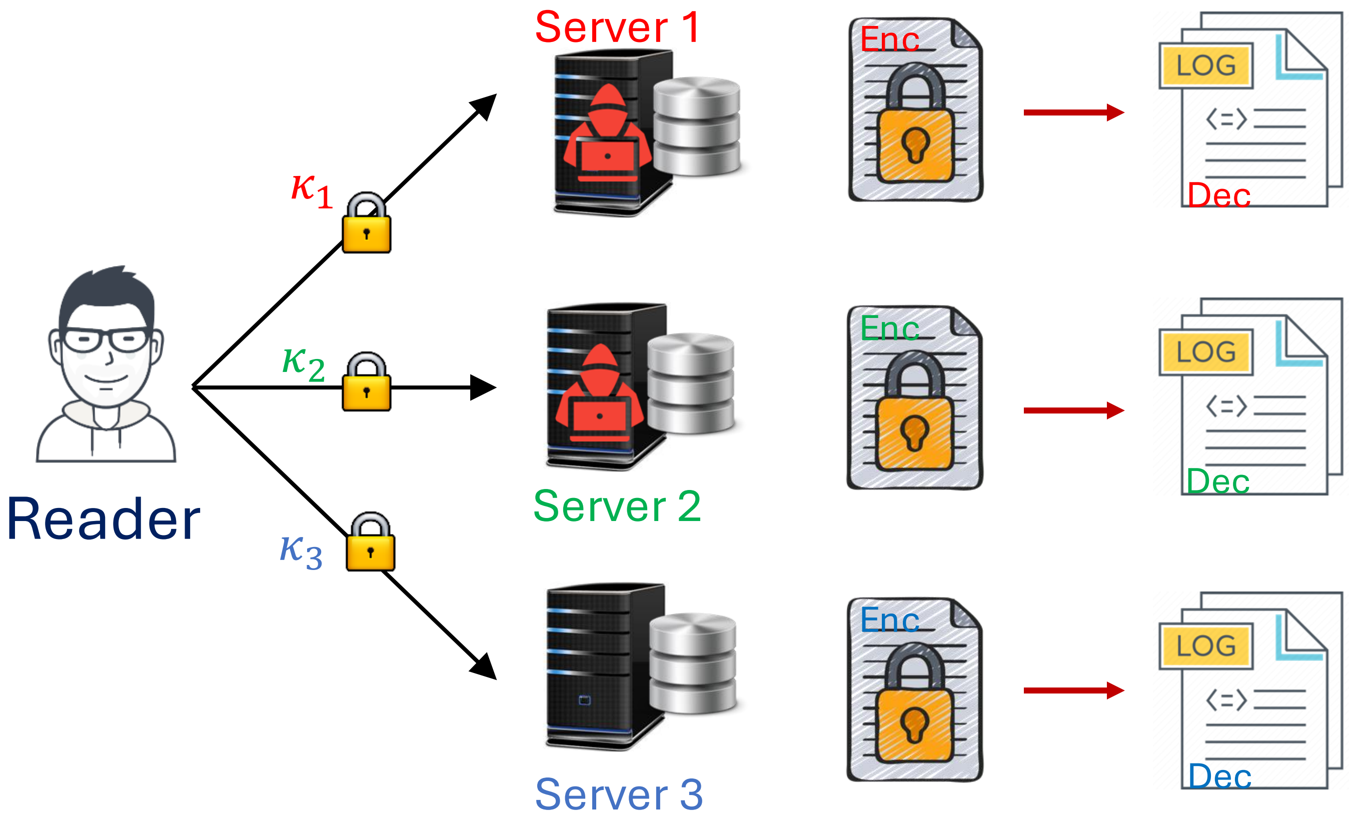
$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$





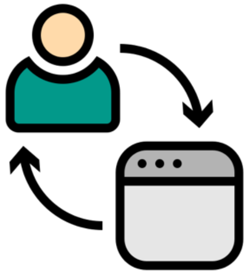
$$\begin{pmatrix} 2 \\ 5 \\ 0 \end{pmatrix} + \begin{pmatrix} 4 \\ 7 \\ 3 \end{pmatrix} + \begin{pmatrix} 3 \\ 4 \\ 6 \end{pmatrix} \bmod 2^3 \underset{\text{Dec}}{=} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Delegating Decryption



Can the servers open secret shares and output documents?

→ No, it reveals result and volume patterns



Key-homomorphic PRFs:

- Keys are secret shared

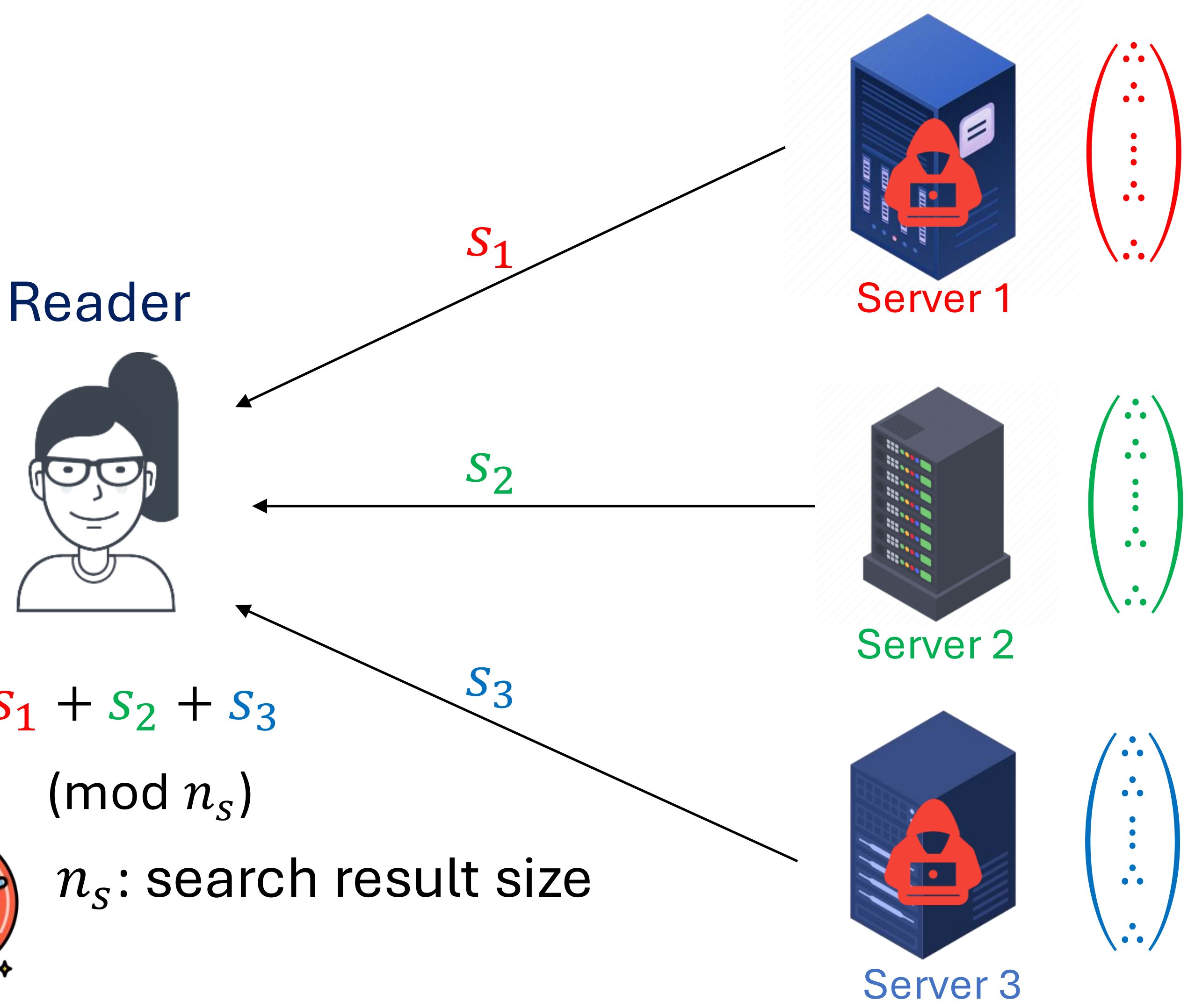
→ No server can learn private data

- Key secret-shares are random

→ Hide search patterns

Our ideas: Oblivious Padding & Shuffling

Multiparty Oblivious Counting

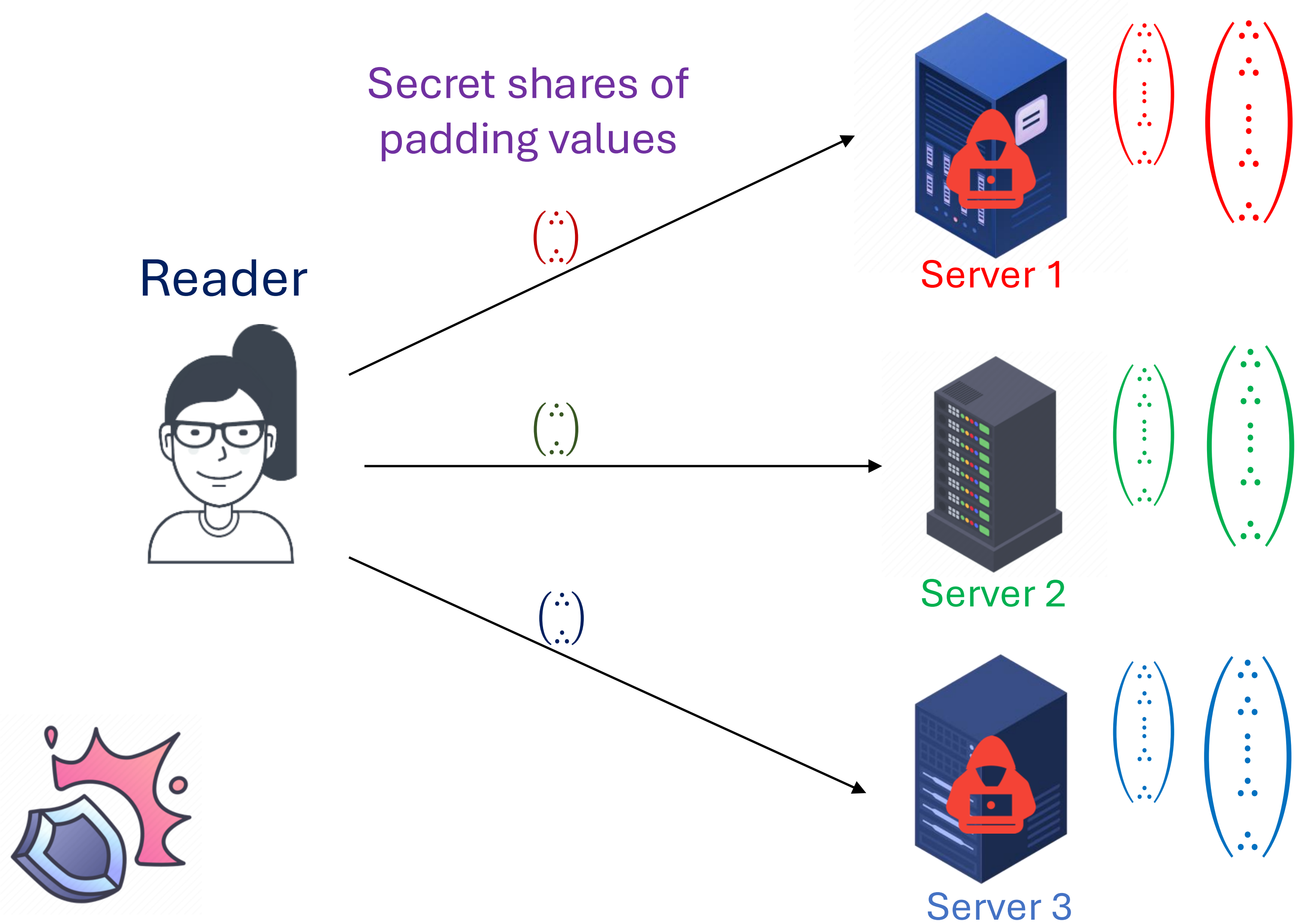


$s = s_1 + s_2 + s_3 \pmod{n_s}$

n_s: search result size

- **Efficiency:**
  - ✓ 1 communication round
  - ✓ Local lightweight operations: circular shifts, additions over small integer numbers
- Most overhead is done in the preprocessing phase
- More efficient than generic MPCs (e.g., Garbled Circuit) when counting values are small (e.g., < 10)

Multiparty Oblivious Padding



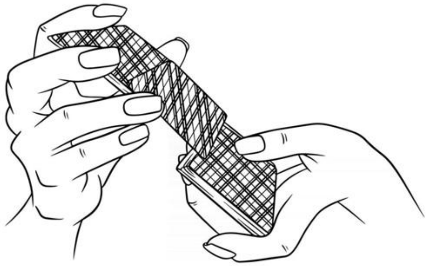
Search result size: n_s

Padding vector size: n_s

Total (padded) size: $N + n_s$

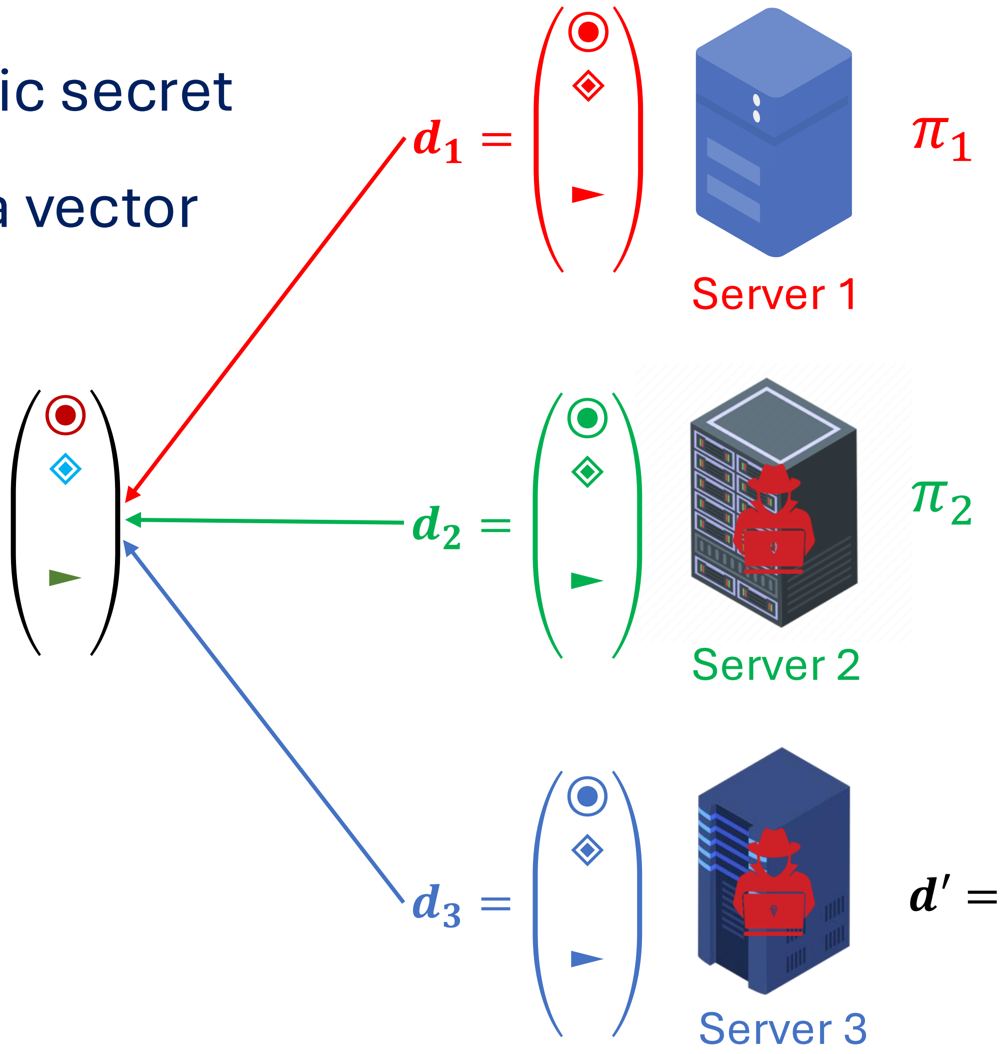
Always the same number of matches is returned for all queries

Multiparty Oblivious Shuffling



Input: Arithmetic secret shares of a data vector

$\boldsymbol{d} = \boldsymbol{d_1} + \boldsymbol{d_2} + \boldsymbol{d_3} =$

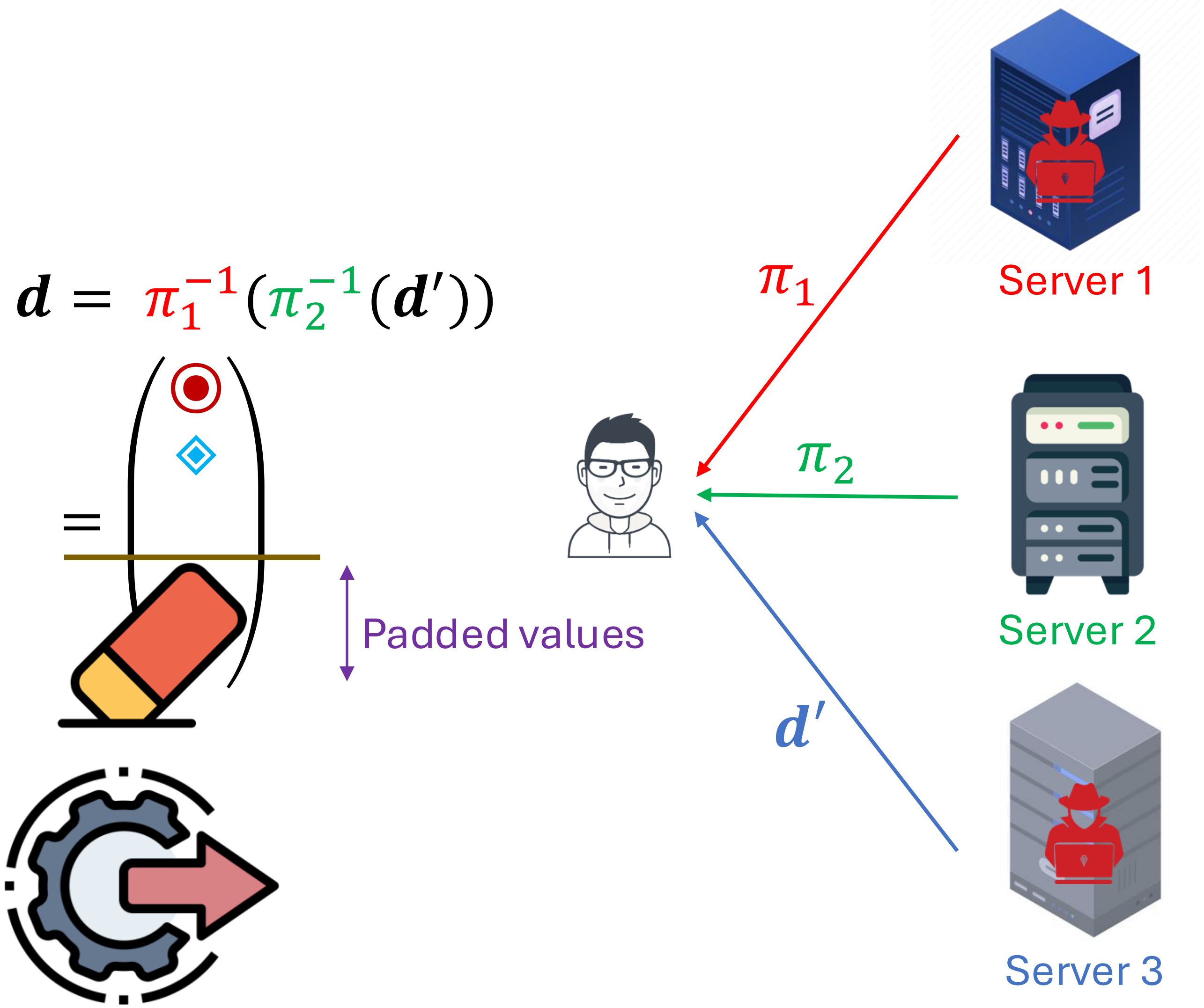


$\boldsymbol{d}' = \pi_2(\pi_1(\boldsymbol{d})) =$

Output:

- A permutation π_i for each server $P_i \in \{P_1, \dots, P_{L-1}\}$
- Opened *shuffled* data vector for server P_L

Final Step – Reverse Shuffling/Output



Search Complexity:

- Reader communication: $O(n_s)$
- Reader computation: $O(N)$
- Server computation: $O(N.m)$, including additions and multiplications over small integer numbers

Parameters:

- n_s: search result size
- m: Bloom filter size
- N: #documents

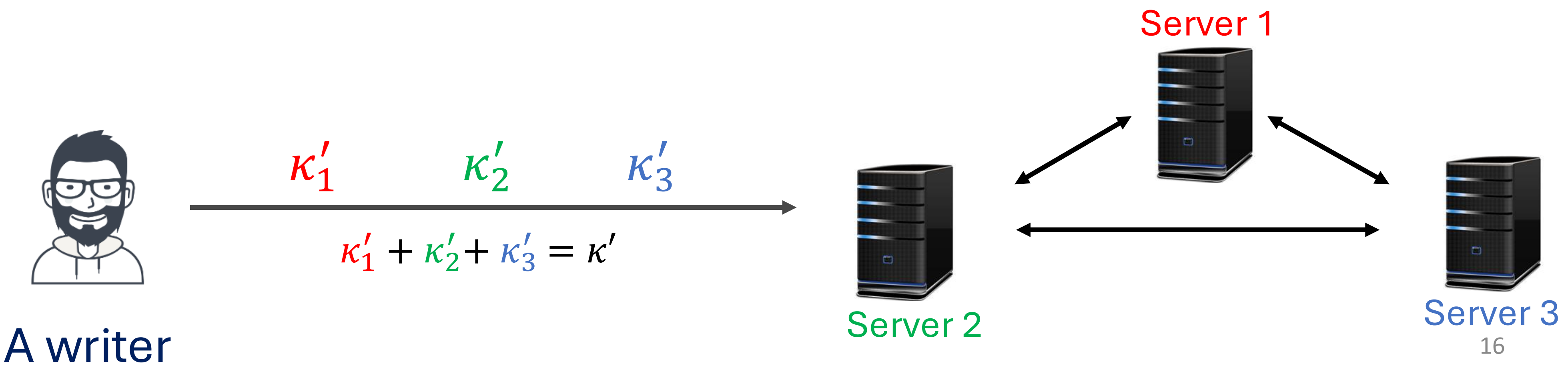
Permission Revocation

A writer needs to revoke the reader's search permission

⟶ Re-encrypt its search index

Key rotation:

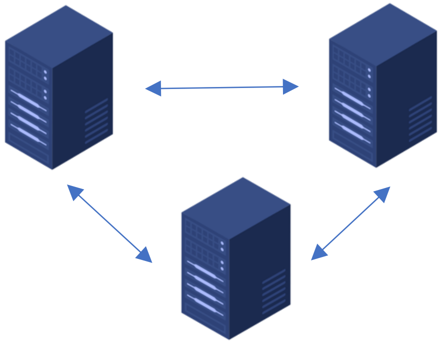
- A good practice recommended by Google, Microsoft, or Amazon
- Mandated by regulations: *NIST.SP.800-57pt1r4*, *PCI-DSS-v4-0*



Evaluation - Configuration

Servers:

- Amazon EC2 r5n.4xlarge instances
- 8-core Intel Xeon Platinum 8375C CPU @ 2.9 GHz, 128 GB RAM



Client:

- Intel i7-6820HQ CPU @ 2.7 GHz, 16 GB RAM

Implementation:

- C++ with ~2,500 LOCs
- Libraries: Secp256k1, OpenSSL, EMP-Toolkit, ZeroMQ

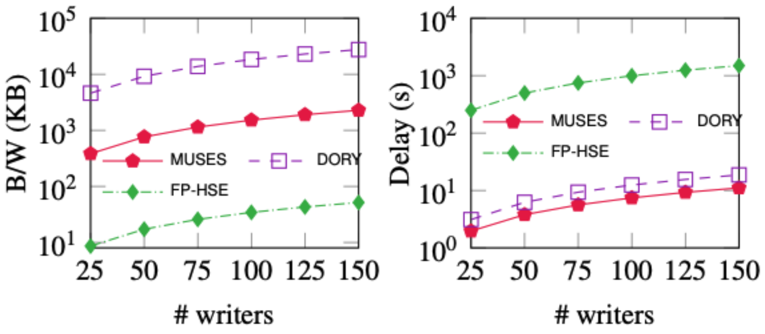
Evaluation – Search

Reader's bandwidth:

$12\times - 97\times$ smaller than DORY (hide patterns), $6\times$ larger than FP-HSE (leak patterns)

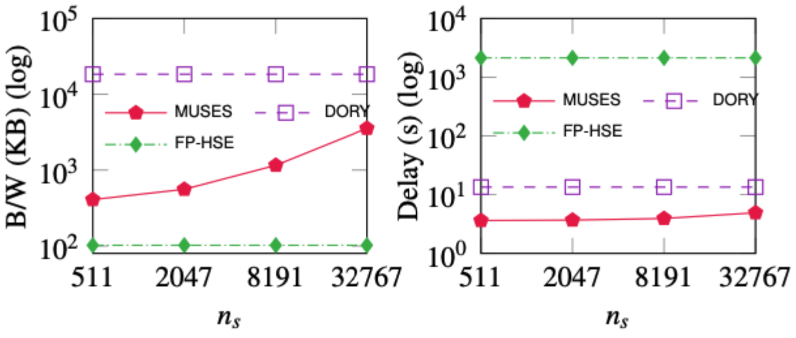
End-to-end latency:

$2\times - 4\times$ faster than DORY, $127\times - 632\times$ faster than FP-HSE



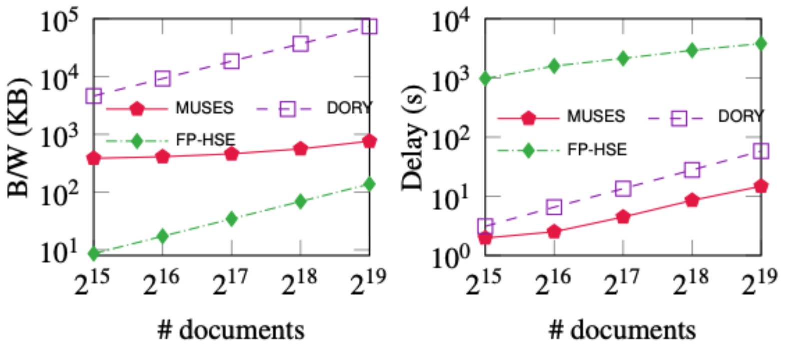
(a) Reader's bandwidth (b) E2E delay

Figure 6: Keyword search performance (log scale on y-axis).



(a) Reader's bandwidth (b) E2E delay

Figure 10: Keyword search performance with varying n_s.



(a) Reader's bandwidth (b) E2E delay

Figure 11: Keyword search performance w/ varying database sizes.

Evaluation – Permission Revocation

Writer's bandwidth:

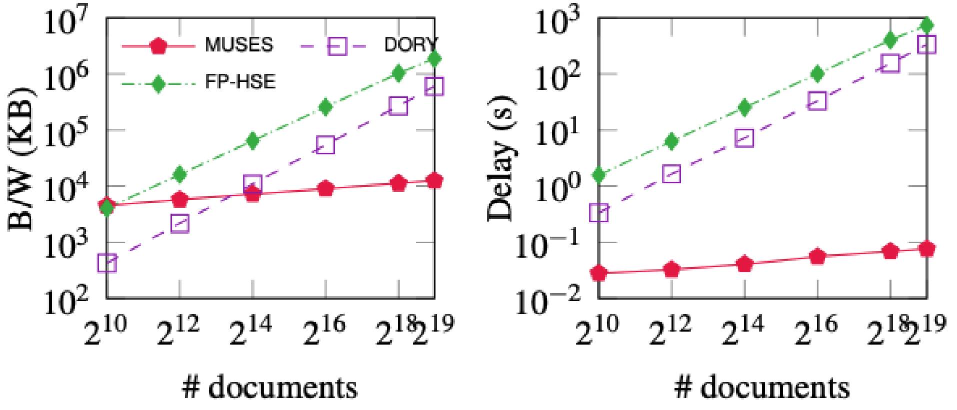
$2\times - 150\times$ smaller than DORY/FP-HSE

Writer's latency:

$12\times - 9600\times$ faster than DORY/FP-HSE

End-to-end latency:

$2\times - 6\times$ faster than DORY/FP-HSE



(a) Writer's bandwidth (b) Writer's latency

Figure 7: Permission revocation performance (log scale on y-axis).

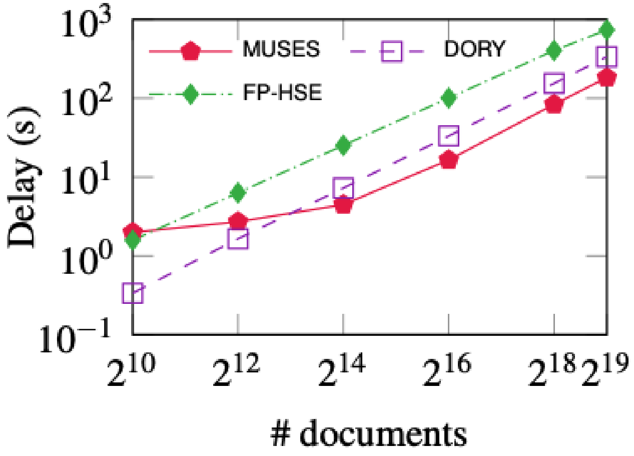
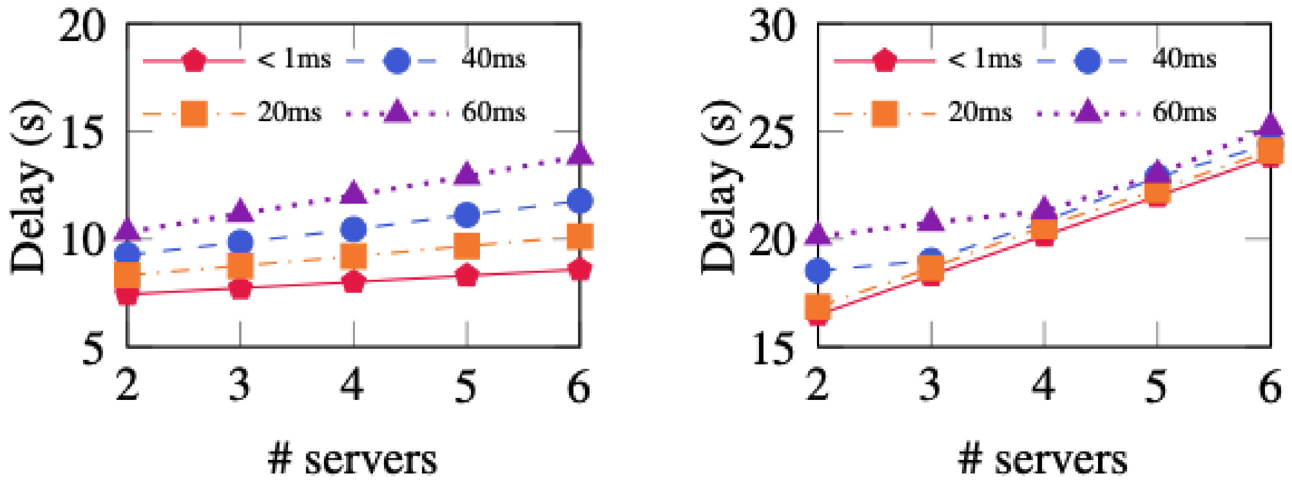


Figure 8: E2E permission revocation delay (log scale on y-axis).

Evaluation – Multiple Servers

Keyword search: 7.4s-8.6s (1ms network latency), 10.3s-13.8s (60ms latency)

Permission revocation: 16.5s-23.8s (1ms latency), 20.1s-25.2s (60ms latency)



(a) Keyword search (b) Permission revocation

Figure 12: MUSES latency with varying numbers of servers.

Conclusion

Our MUSES:

- Hide *all* statistical information: search, result, and volume patterns
- Minimal user overhead for search and permission revocation

Our artifact is available at: github.com/vt-asaplab/MUSES

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

THANK FOR YOUR ATTENTION

Q&A