MAPLE: A METADATA-HIDING POLICY-CONTROLLABLE ENCRYPTED SEARCH PLATFORM WITH MINIMAL TRUST

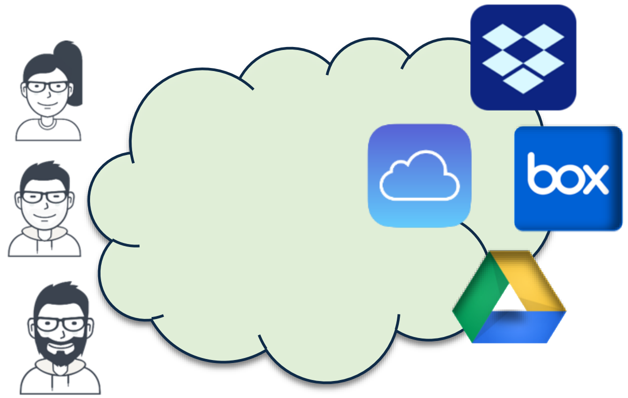
Tung Le, Thang Hoang

Virginia Tech, USA

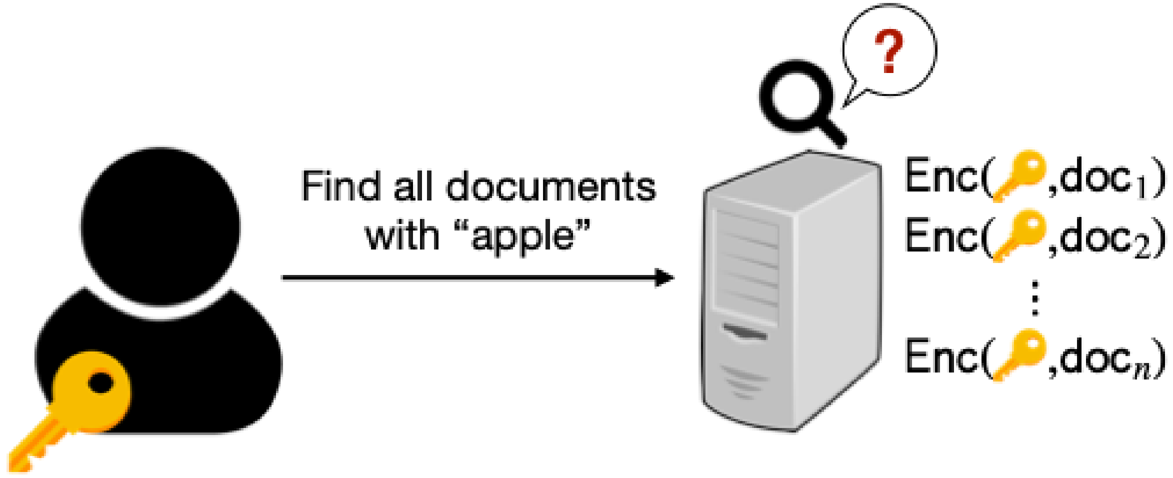
Privacy Enhancing Technologies Symposium (PETs) 2023

Lausanne, Switzerland

Overview



Storage-as-a service (STaaS)



Overview

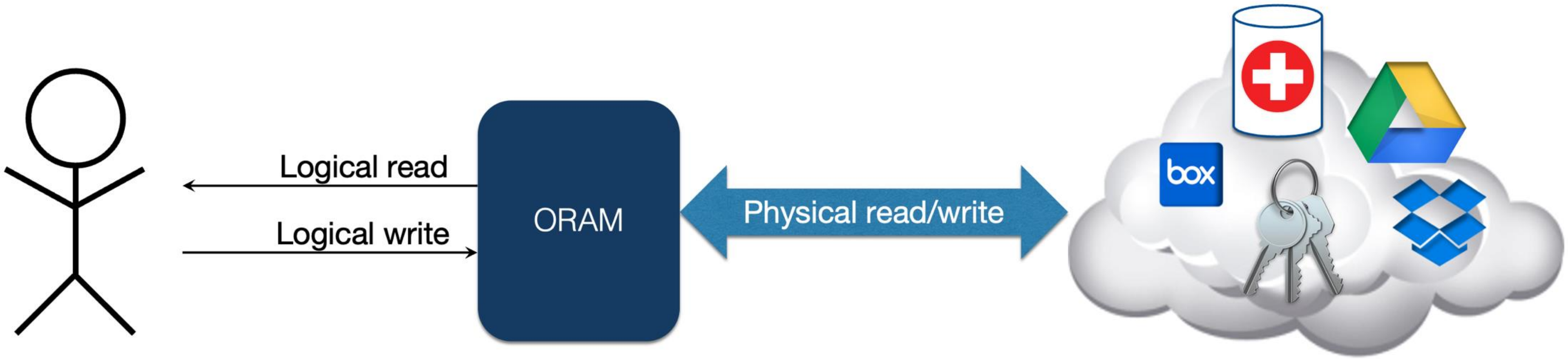
Searchable Encryption (SE)

Previous SEs	Our Work (MAPLE)
• ODSE’19, DORY’20, DURASIFT’20: ▪ Hide Search Access Pattern with Search Complexity $O(N.m)$*. ▪ Limited Multi-user Support.	• Minimal Leakage. • Search Complexity $O(N \log m)$. • Multi-user Support.

* *N: #documents, m: keyword space/keyword representation*

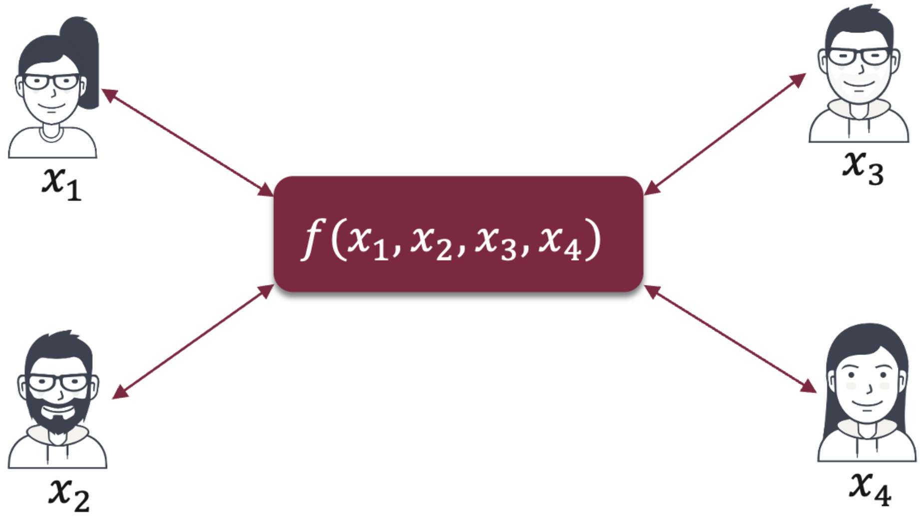
Oblivious RAM (ORAM)

- Oblivious Random Access Machine (ORAM) allows a client to hide the access pattern when accessing data stored on untrusted memory.



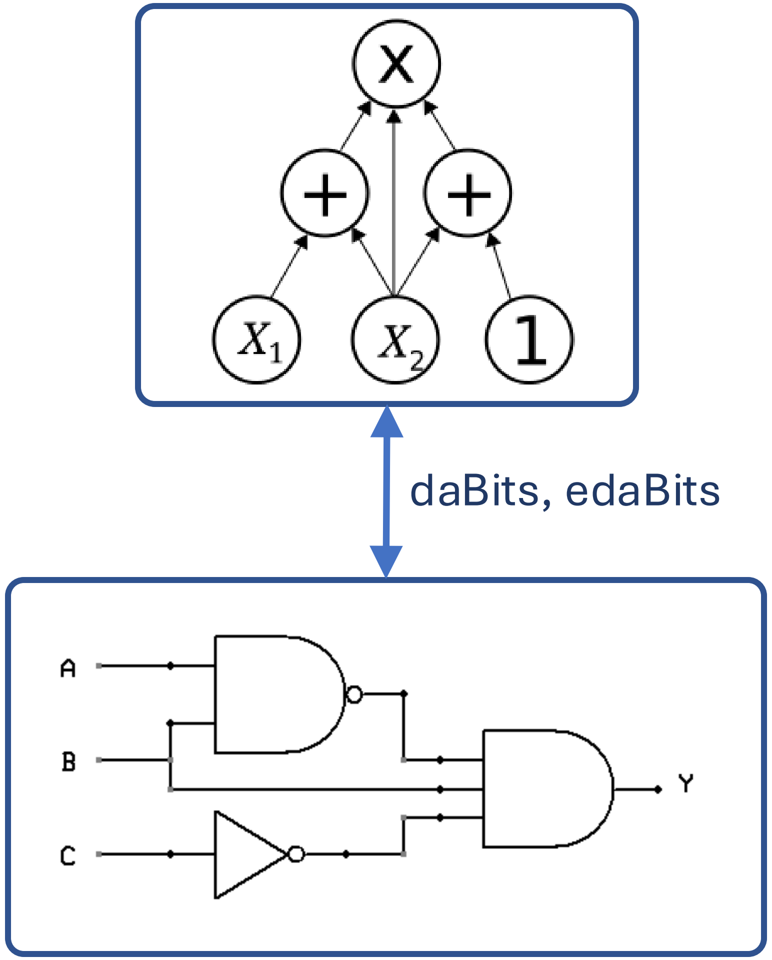
ORAM applications: Cloud storage-as-a-service (personal data storage, health-record database, password management), searchable encryption, secure multiparty computation

Multi-Party Computation (MPC)



MPC permits multiple parties to jointly evaluate a function without revealing private inputs of individuals

Arithmetic MPC
(SPDZ, Shamir SS, replicated SS)



Boolean MPC
(garbled circuits)

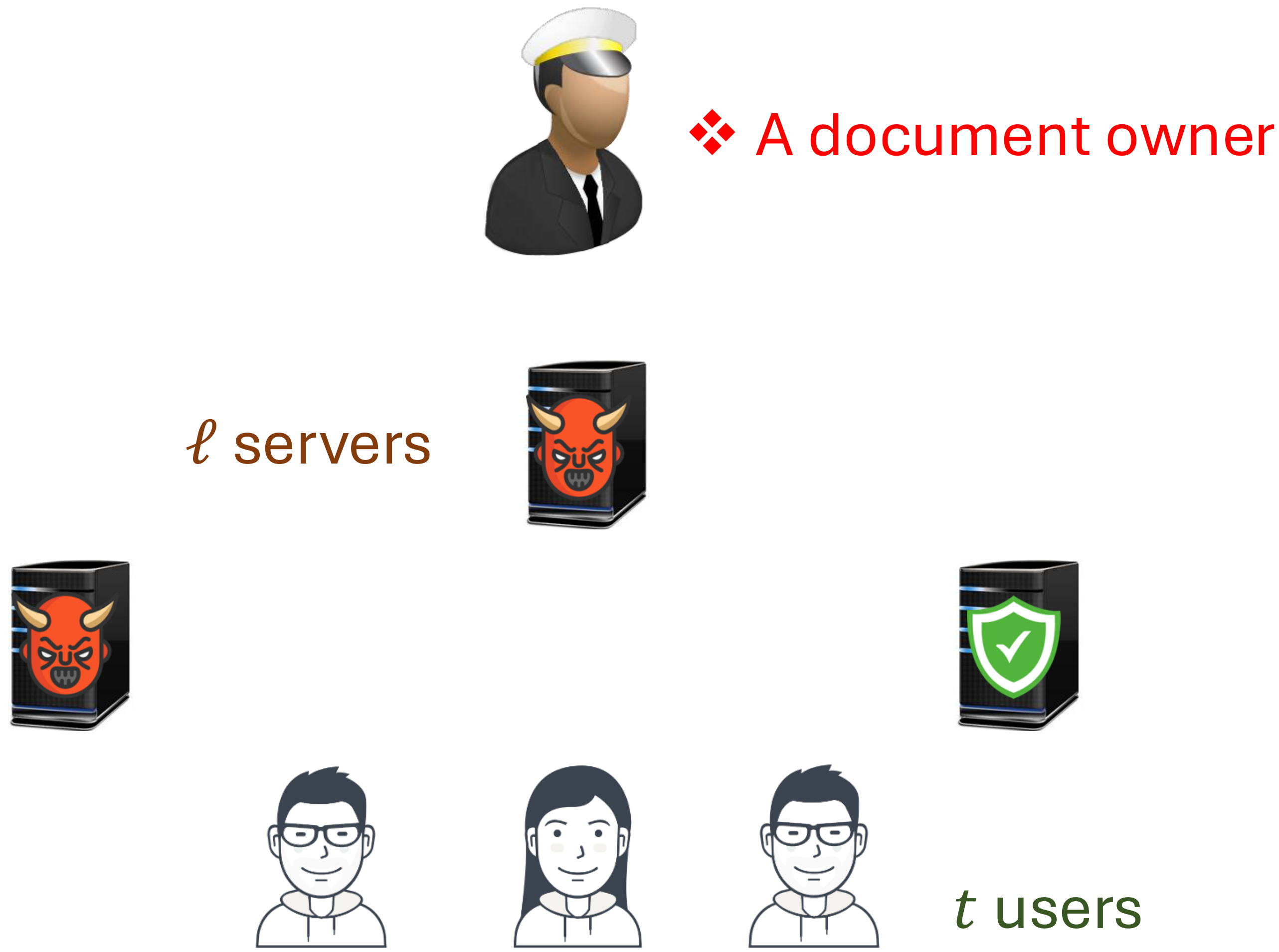
Leakage-abuse Attacks

- **Search Pattern:** [IKK’12, LZWT’14, OK’ 21].
- **Access Pattern:** [IKK’12, CGPR’14, ZKP’16, LCNL’22, OK’22].
- **Volume Pattern:** [BKM’19, LCNL’22, OK’22, ZWXYL’23].
- **Update Pattern:** [ACMR’16, RACM’17].

....[PW’16, KKNO’16, GTS’17, PWLP’20]

- ➢ Discover keywords in queries.
- ➢ Recover document plaintext.

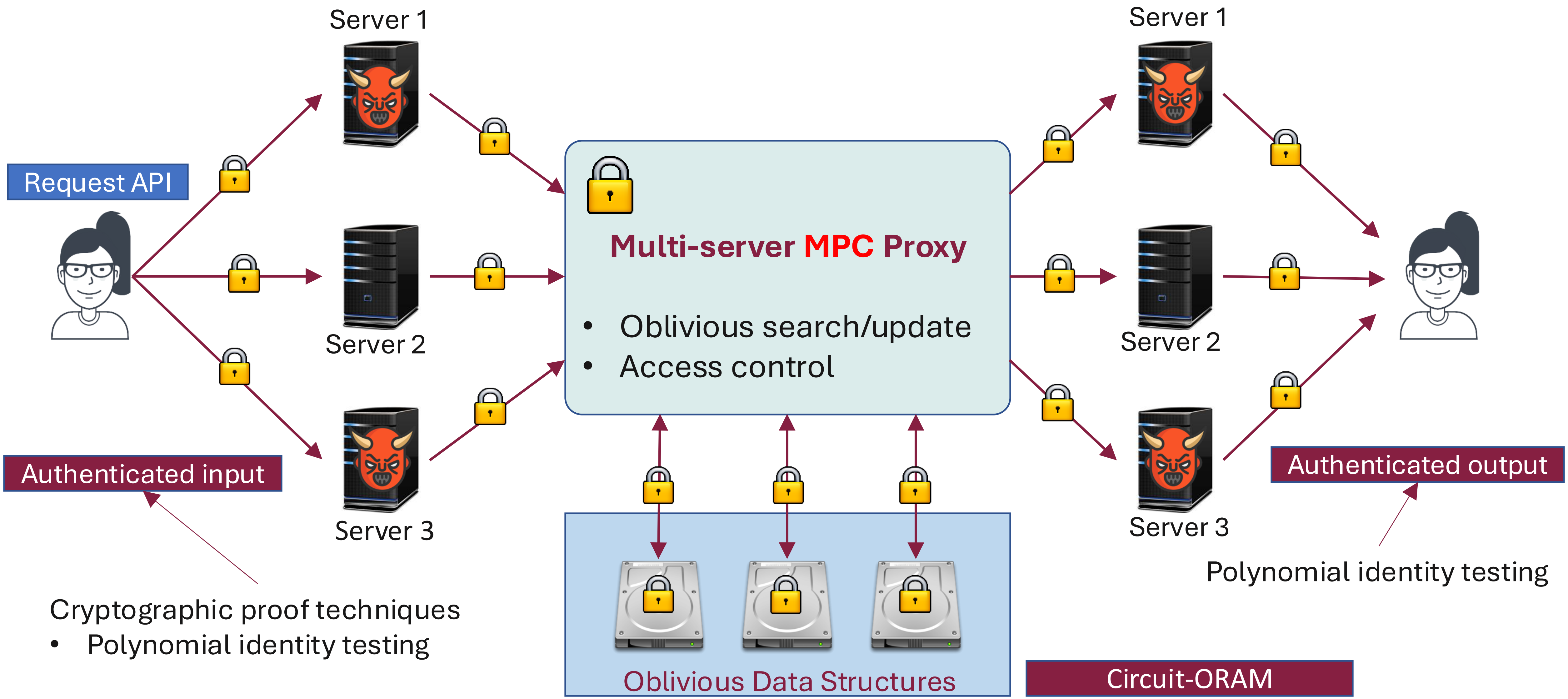
System and Threat Model



Malicious Security:

- Malicious users
- Malicious servers
- Collusion between users & servers

System Design



Search Index Design

Bloom Filter	BF_1	BF_2	BF_3	…	BF_m
	w_1	w_2	w_3	…	w_M
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

Keyword representation

Search Index (IDX)

$\otimes$

0
0
0
1
0

False Positive Rate:

$$\epsilon = \left(1 - \left(\frac{1}{m}\right)^{kn}\right)^{k}$$

- k: # BF indices for each item
- m: BF size
- n: # inserted items

Access Policy Index

	User #1		...		User #t
Doc #1	00	01	10	00	10
Doc #2	11	10	10	11	11
Doc #3	01	10	01	00	00
...	10	11	11	00	11
Doc #N	11	11	00	11	10

Access Control (AC)

IDX

1	0	1	1	0	1	1
1	1	0	1	1	0	1
1	1	0	0	1	0	0
1	0	1	0	1	1	0
0	1	0	1	1	1	0
0	1	0	1	0	0	0
1	0	1	0	0	1	0

AC

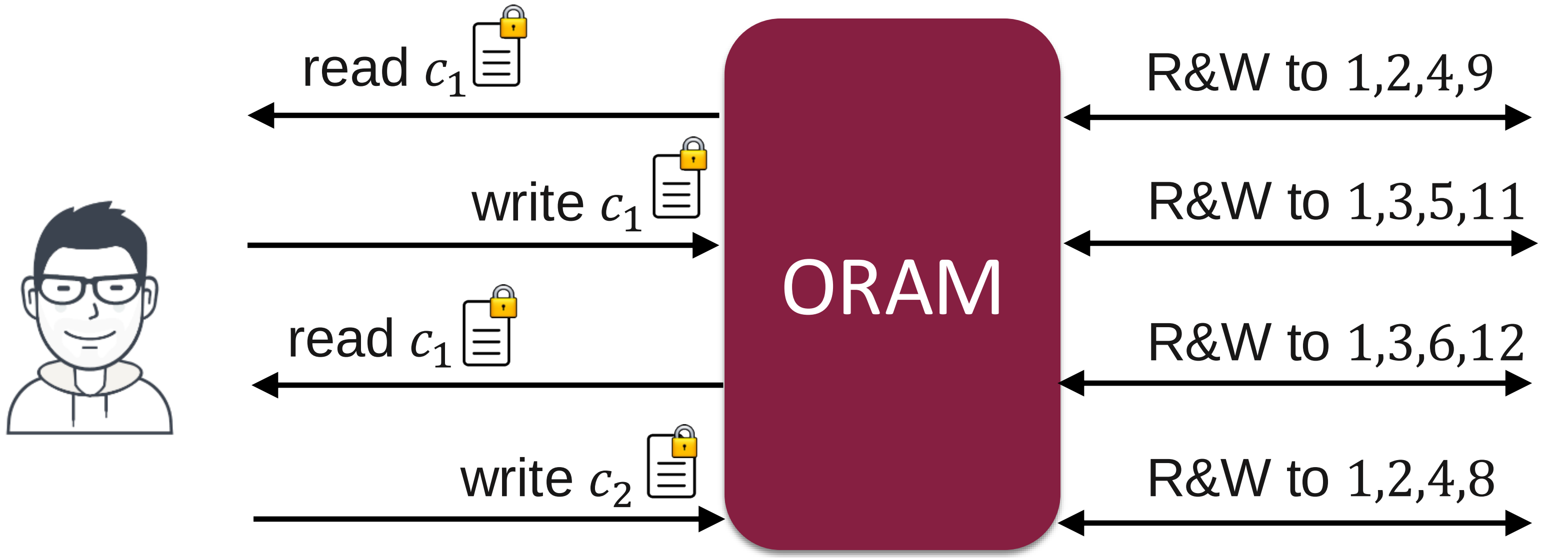
w	w	r	r	.	w	.
w	w	.	r	.	r	.
.	w	w	w	w	.	r
.	w	r	w	.	r	w
r	.	w	r	w	r	w
.	w	r	r	r	.	r
.	r	w	.	w	r	w

$\otimes$

0
1
0
0
0
0
0

Oblivious Table (OTAB)

$1 = 1 \oplus 0 \oplus 0$



IDX

$m \times z$ ($N \times z$)

1	0	1	1	0	1
1	1	0	1	1	0
1	1	0	0	1	0
1	0	1	0	1	1
0	1	0	1	1	1
0	1	0	1	0	0
1	0	1	0	0	1
0	1	1	0	1	0

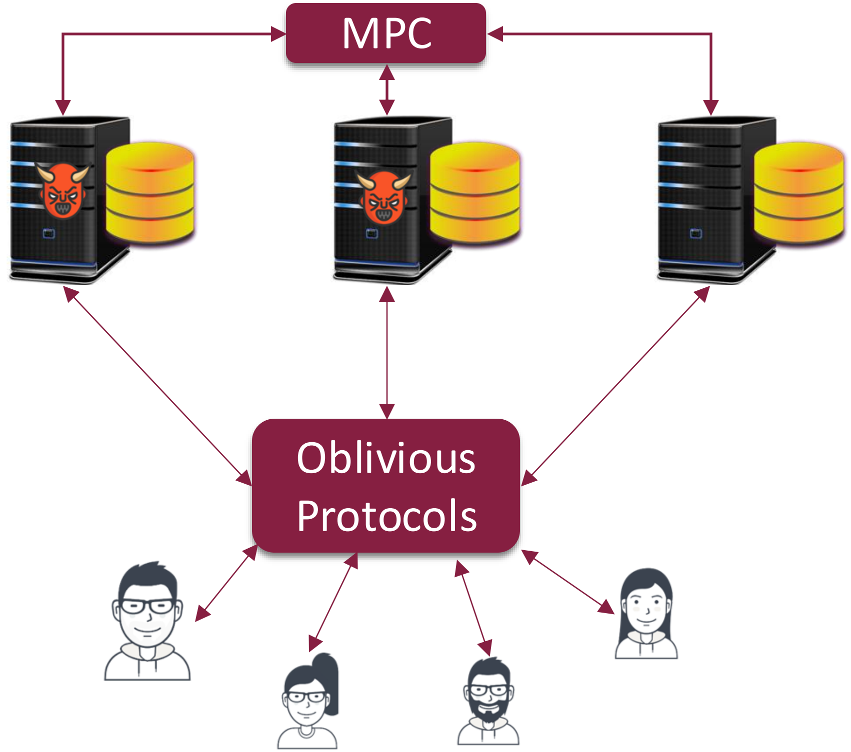
Same order!!!

AC

$t \times z$ ($N \times z$)

w	w	r	r	.	w
w	w	.	r	.	r
.	w	w	w	w	.
.	w	r	w	.	r
r	.	w	r	w	r
.	w	r	r	r	.
.	r	w	.	w	r
r	.	w	w	.	r

Search Operation



Permutation Matrix

$m \times z$

1	0	0	0	0	0
0	0	1	0	0	0
0	0	0	0	0	1
0	0	0	1	0	0

Columns of IDX are shuffled, how to update?

IDX

$m \times z$

1	0	1	1	0	1
1	1	0	1	1	0
1	1	0	0	1	0
1	0	1	0	1	1
0	1	0	1	1	1
0	1	0	1	0	0
1	0	1	0	0	1
0	1	1	0	1	0

$z \times N$

Retrieve k columns

AC

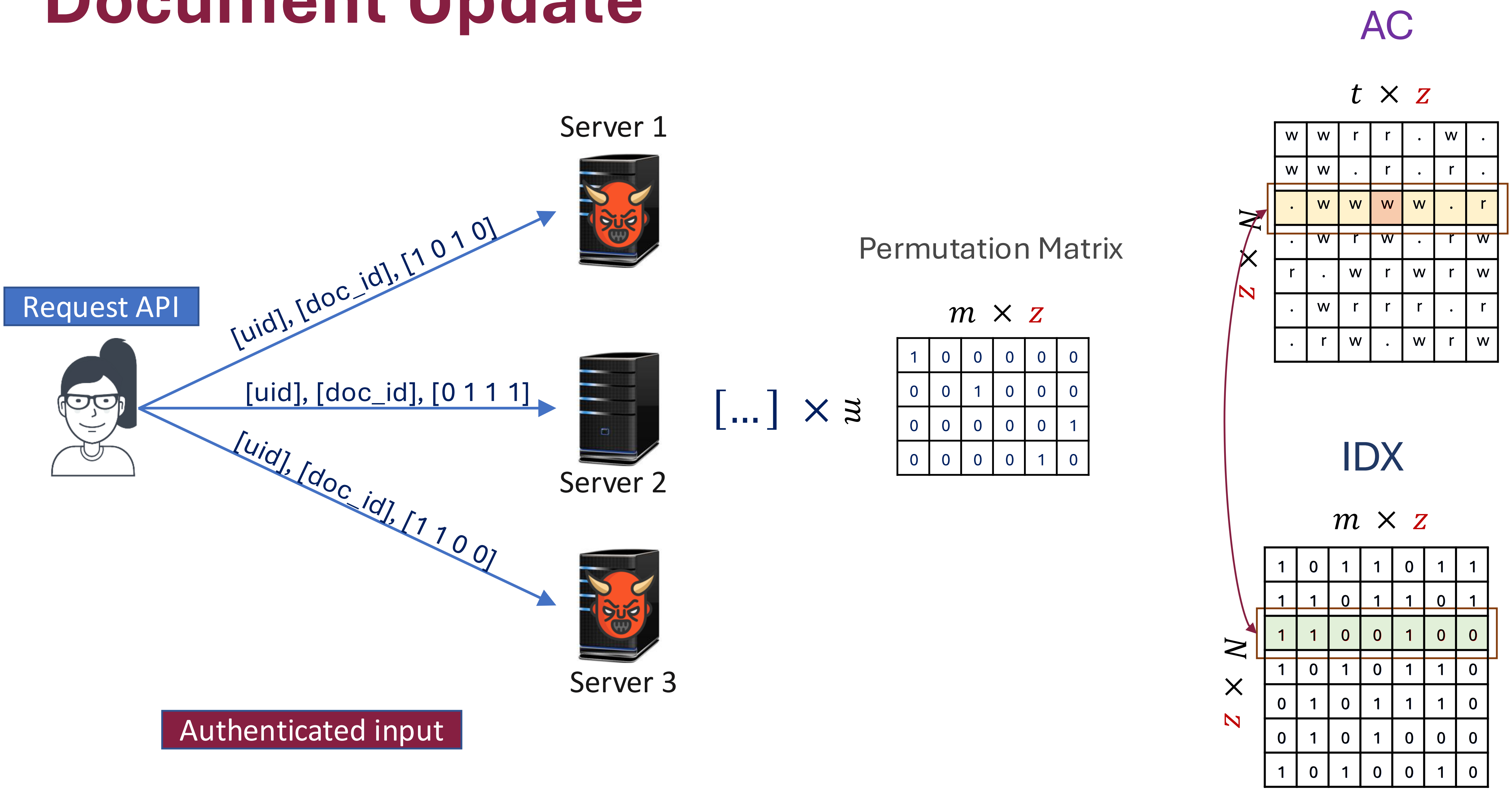
$t \times z$

w	w	r	r	.	w
w	w	.	r	.	r
.	w	w	w	w	.
.	w	r	w	.	r
r	.	w	r	w	r
.	w	r	r	r	.
.	r	w	.	w	r
r	.	w	w	.	r

$z \times N$

Retrieve 1 column

Document Update



Permission Update

[uid], [doc_id], [r/w]

Document owner

Multi-server MPC Proxy

- Oblivious search/update
- Access control

Authenticated input

AC

$t \times z$

$N \times z$

w	w	r	r	.	w	.
w	w	.	r	.	r	.
.	w	w	w	w	.	r
.	w	r	w	.	r	w
r	.	w	r	w	r	w
.	w	r	r	r	.	r
.	r	w	.	w	r	w

Evaluation - Configuration

- **Server:**
  - Amazon EC2 r5n.16xlarge.
  - 32-core Intel Xeon Platinum 8375C CPU @ 2.9 GHz.
  - 512 GB RAM.
- **Client**:
  - Macbook Pro 14 2021 M1-Max.
  - 32 GB RAM.
- **Implementation**:
  - C++ with ~4,000 LOCs.
  - EMP-toolkit, ZeroMQ

Evaluation – Search Delay

- DORY: $O(N.m)$, MAPLE: $O(N.\log m)$.
- $2.6\times - 10.7\times$ slower than DORY with BF size $\leq 2^{14}$, and outperforms when BF size $\geq 2^{16}$.

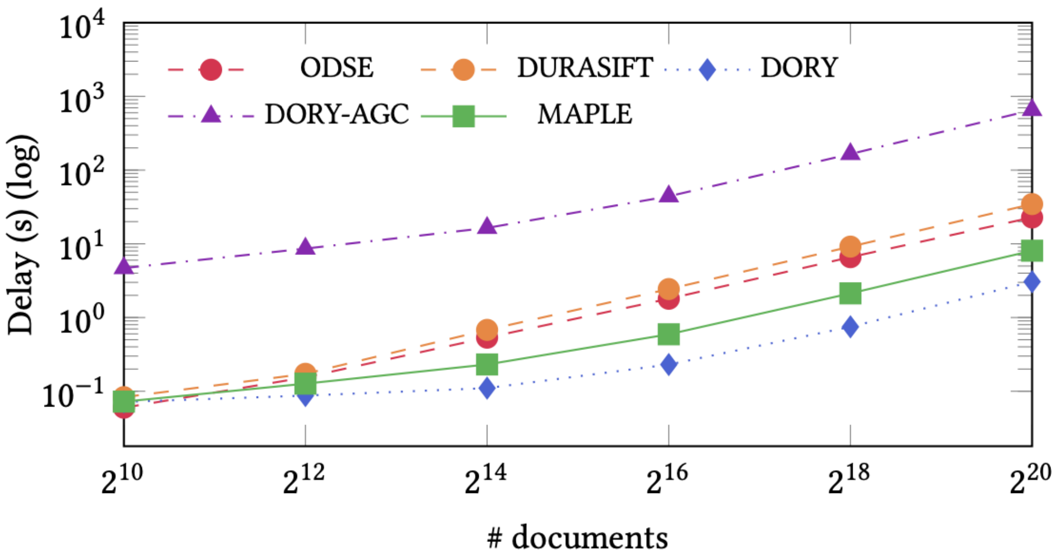


Figure 5: Search delay of MAPLE and its counterparts.

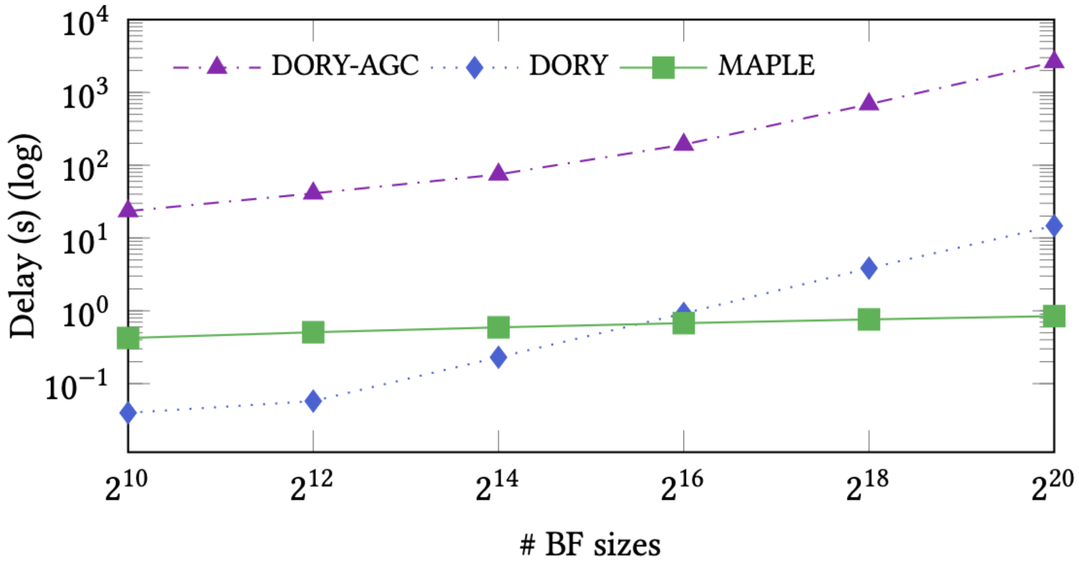
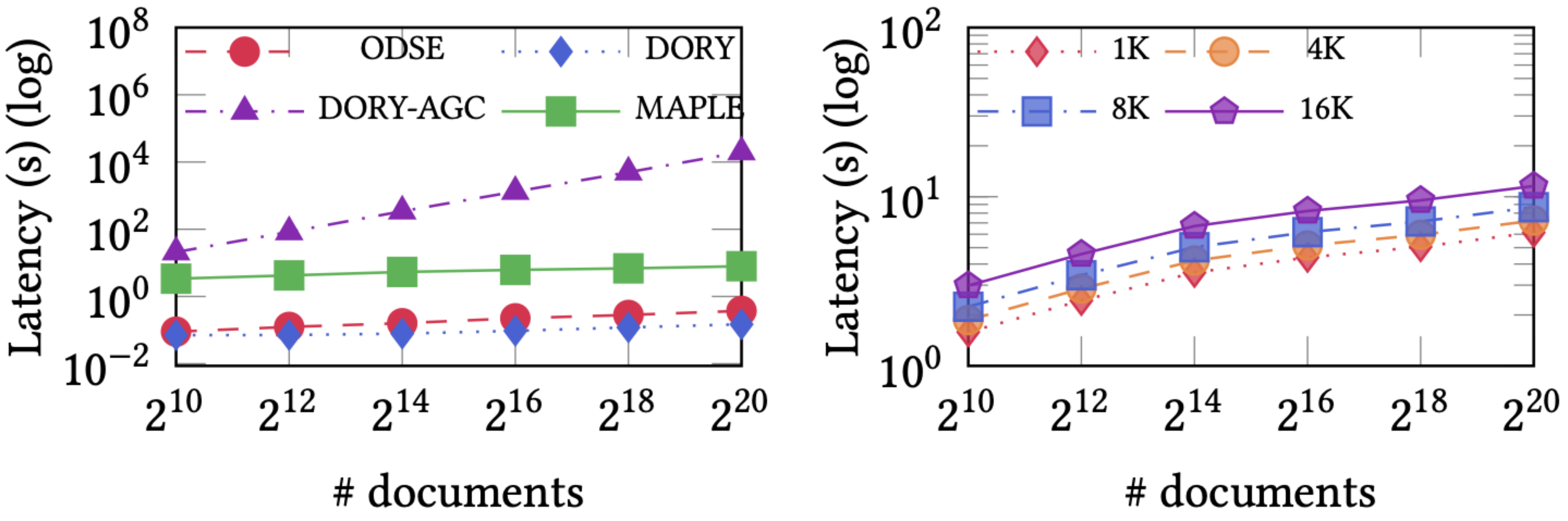


Figure 6: Search delay with varied BF sizes.

Evaluation – Update Delay

- Document update: $O(m \log N + m^2)$
- 3.3s – 7.8s slower to achieve oblivious update



(a) Document update **(b) Permission update**

Figure 8: Update delay of MAPLE and its counterparts.

Conclusion

Our MAPLE:

- Support multi-user with fine-grained access control.
- Oblivious search with better complexity $O(N \log m)$.
- Minimal leakage with malicious security.

Our source code is available at: github.com/vt-asaplab/MAPLE

Thank you for your attention

Q&A

References