Metal

A Metadata-Hiding File-Sharing System

Weikeng Chen Raluca Ada Popa

UC Berkeley

End-to-end encrypted file sharing

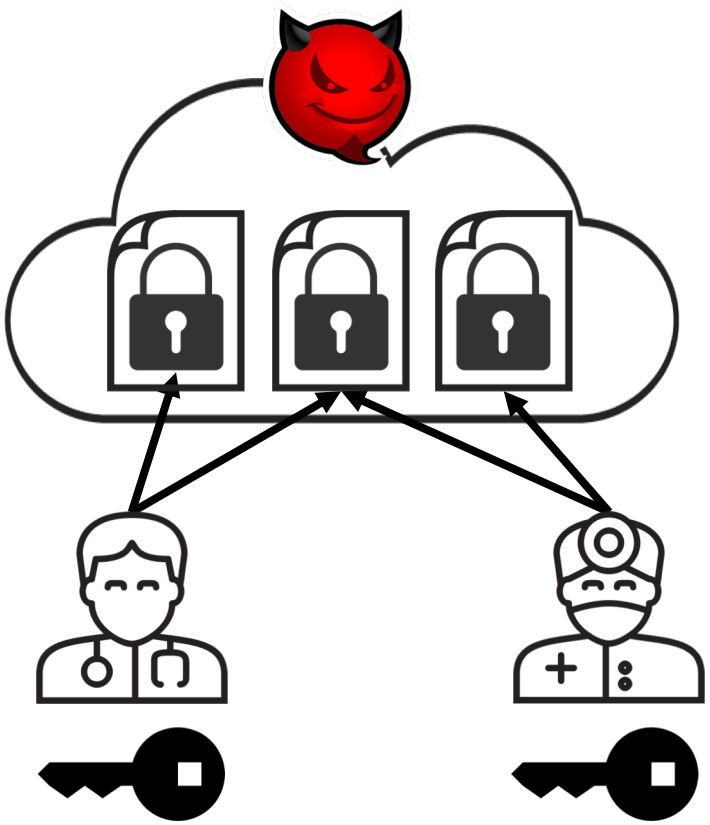
Academia:

DepSky, M-Aegis, Mylar, Plutus, ShadowCrypt, Sieve, SiRiUS

Industry:

PREVEIL

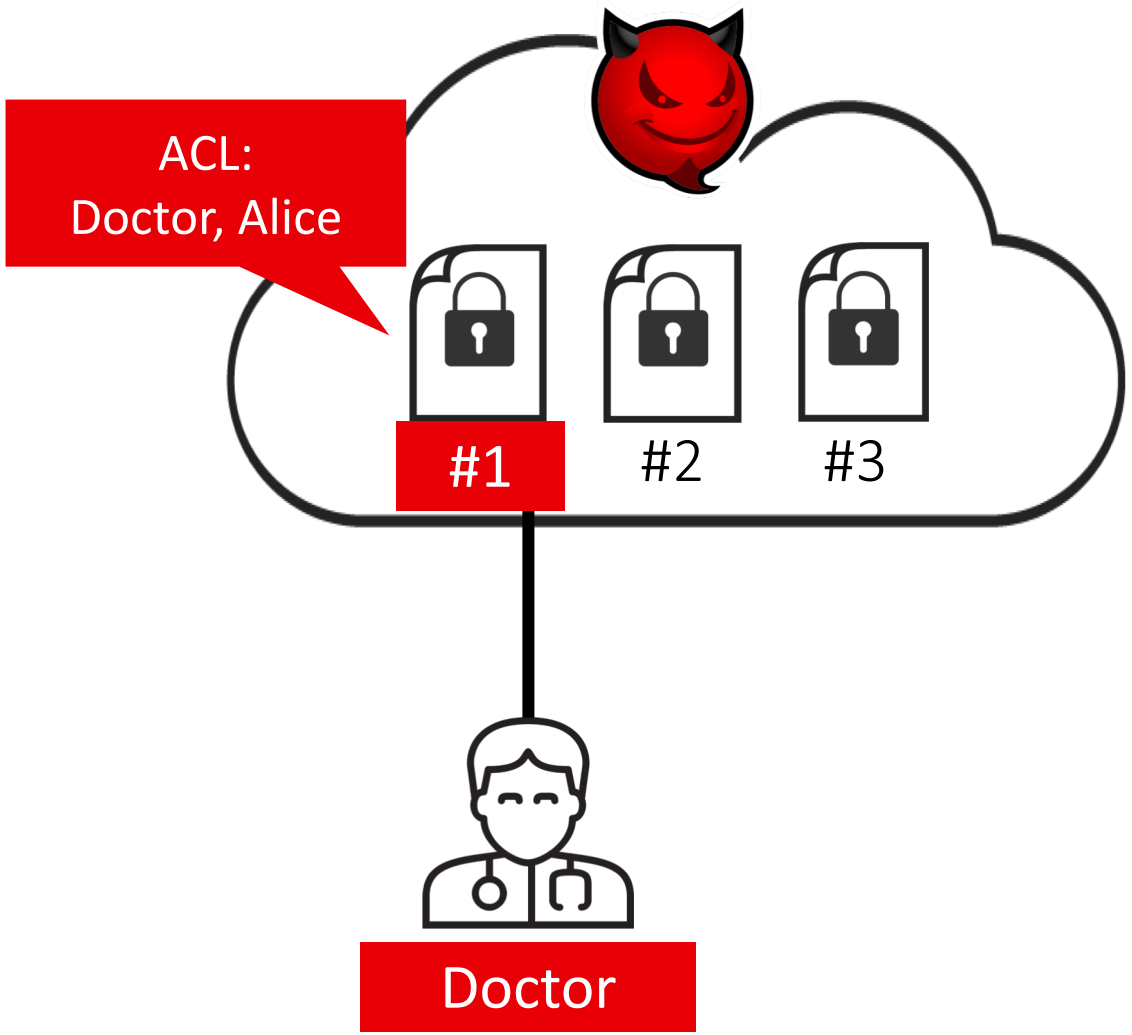
tresorit



Encryption is not enough: Metadata still leaks

- **User identities:** who read or wrote a file
- **File access patterns:** which file is being read or written

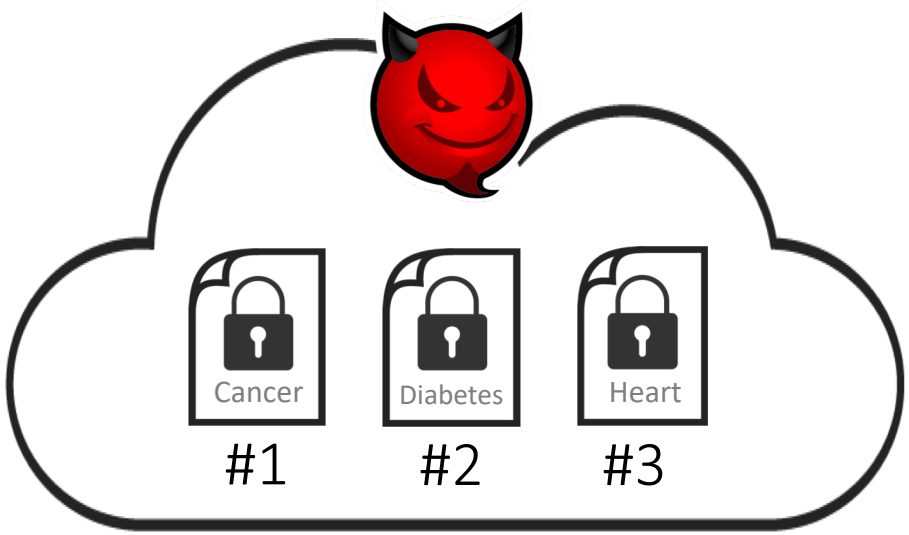
Encryption is not enough: Metadata still leaks



The attacker sees:

- The doctor is accessing a file
- The file is #1
- #1's access control list

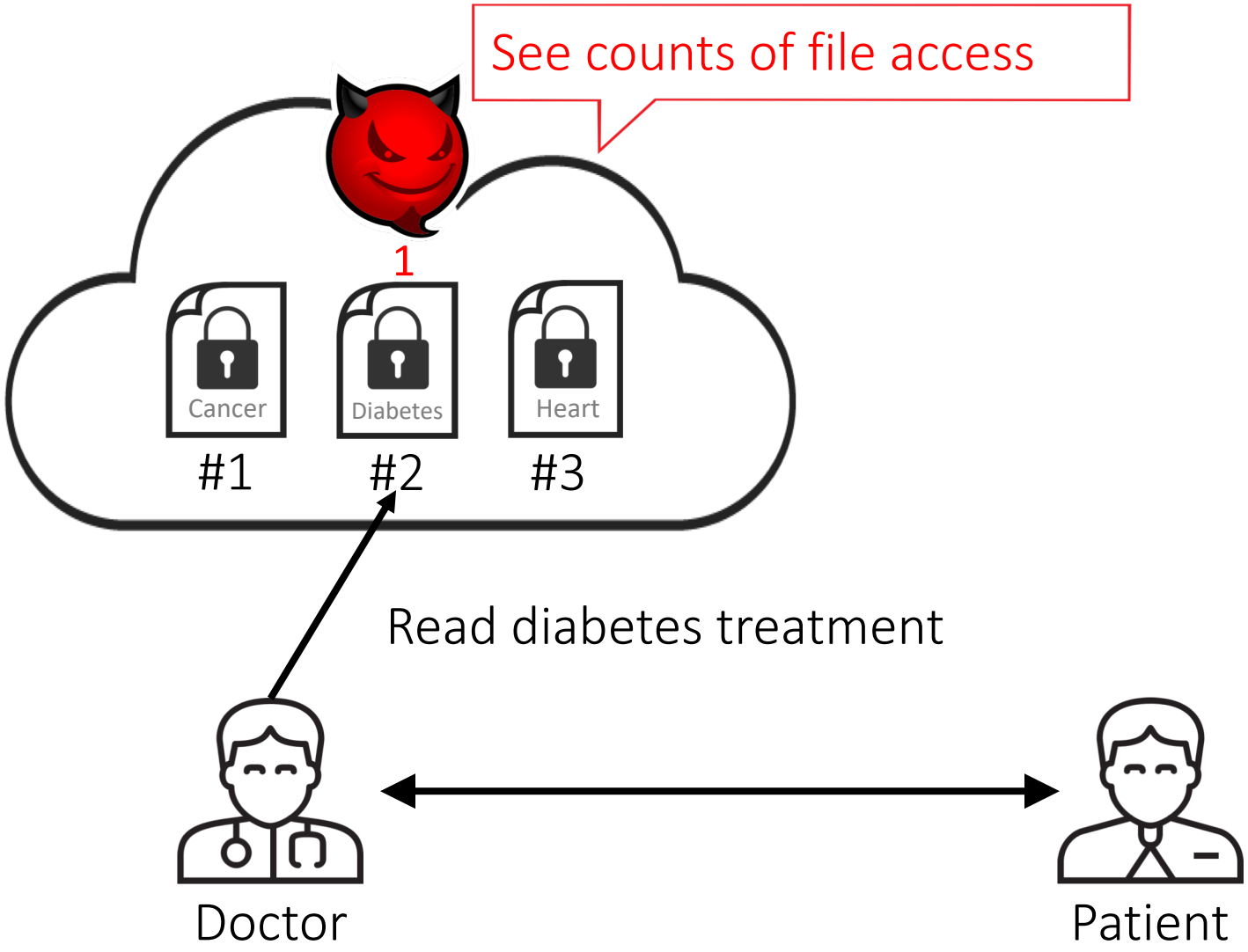
An attack using file access patterns



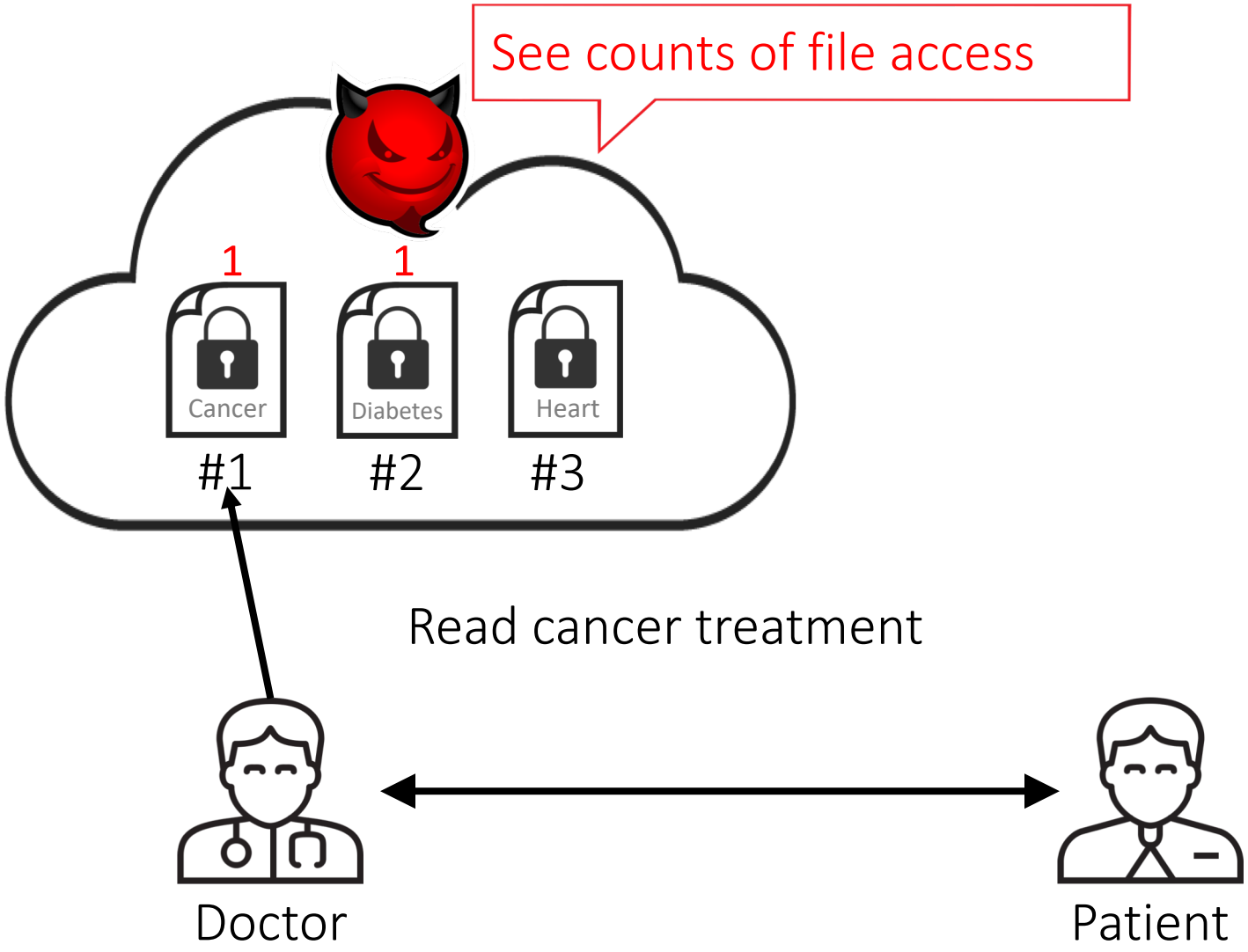
Filenames are also encrypted

Doctor

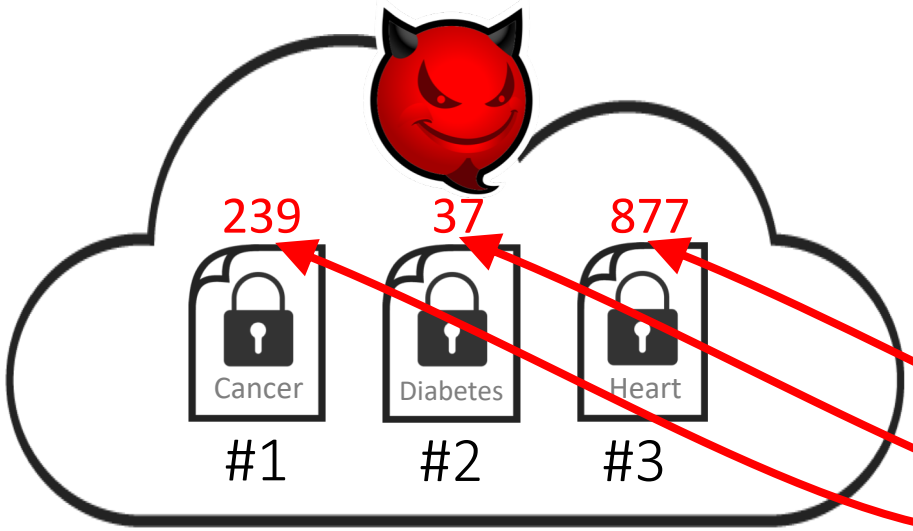
An attack using file access patterns



An attack using file access patterns



An attack using file access patterns

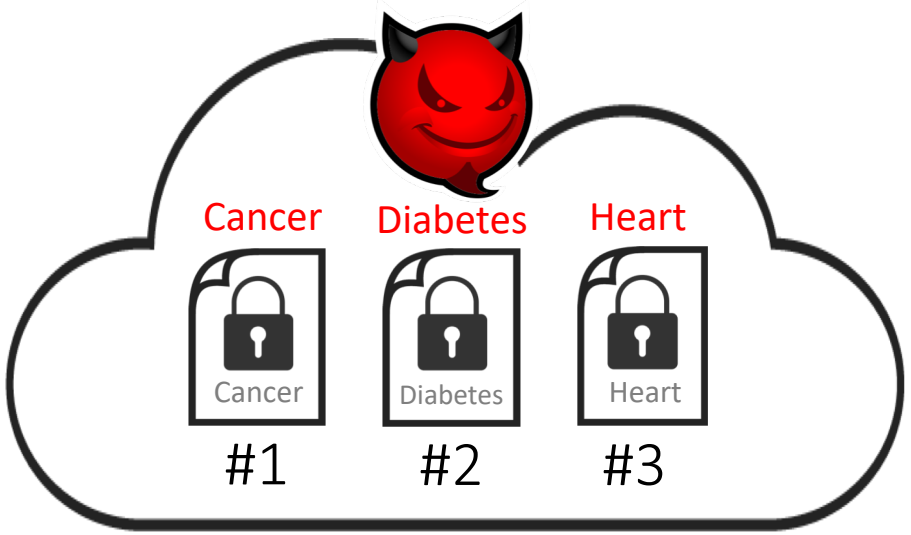


Public database on disease incidence rate

Cancer	Diabetes	Heart
2.49‰	0.39‰	8.61‰

Doctor

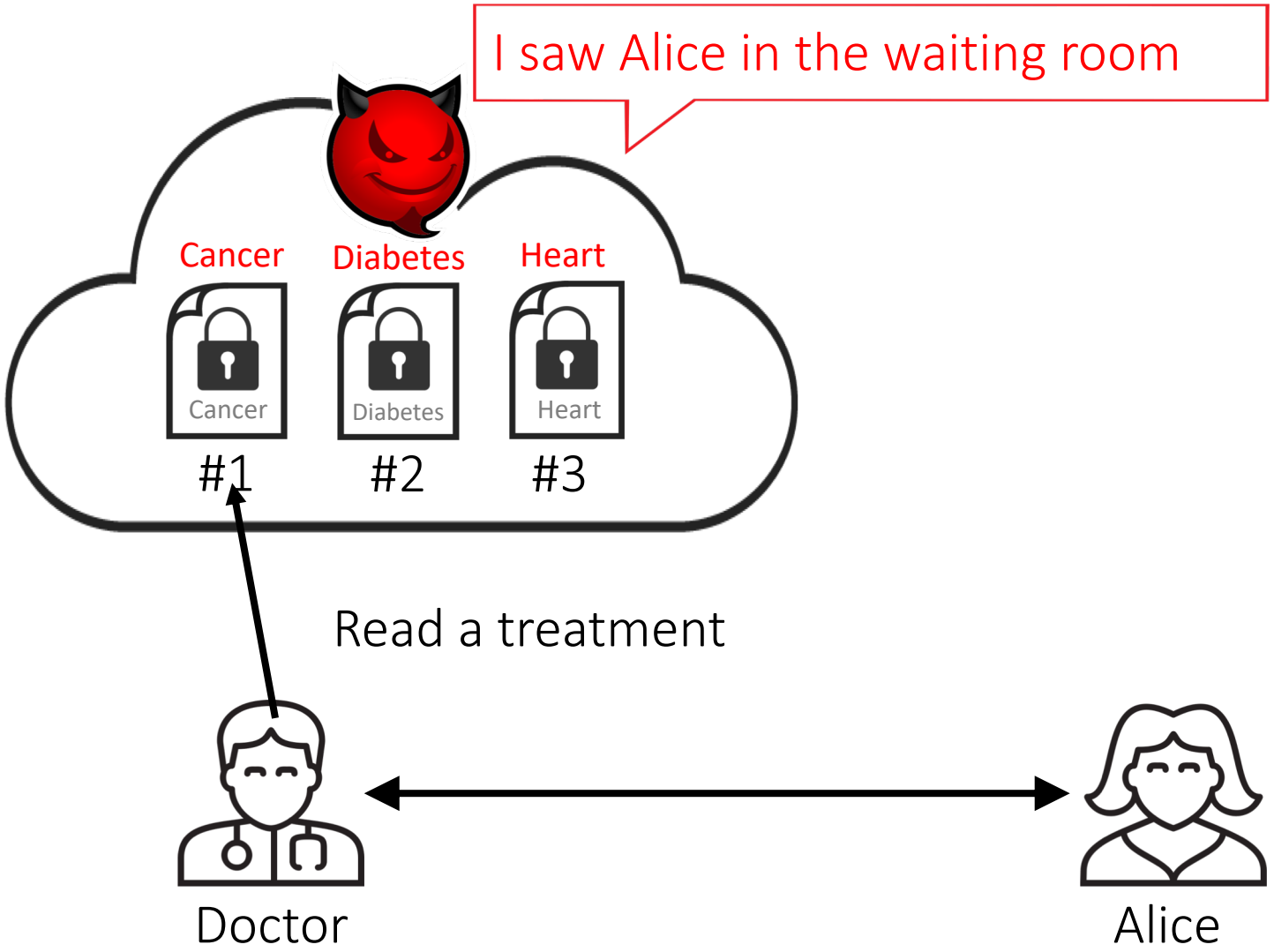
An attack using file access patterns



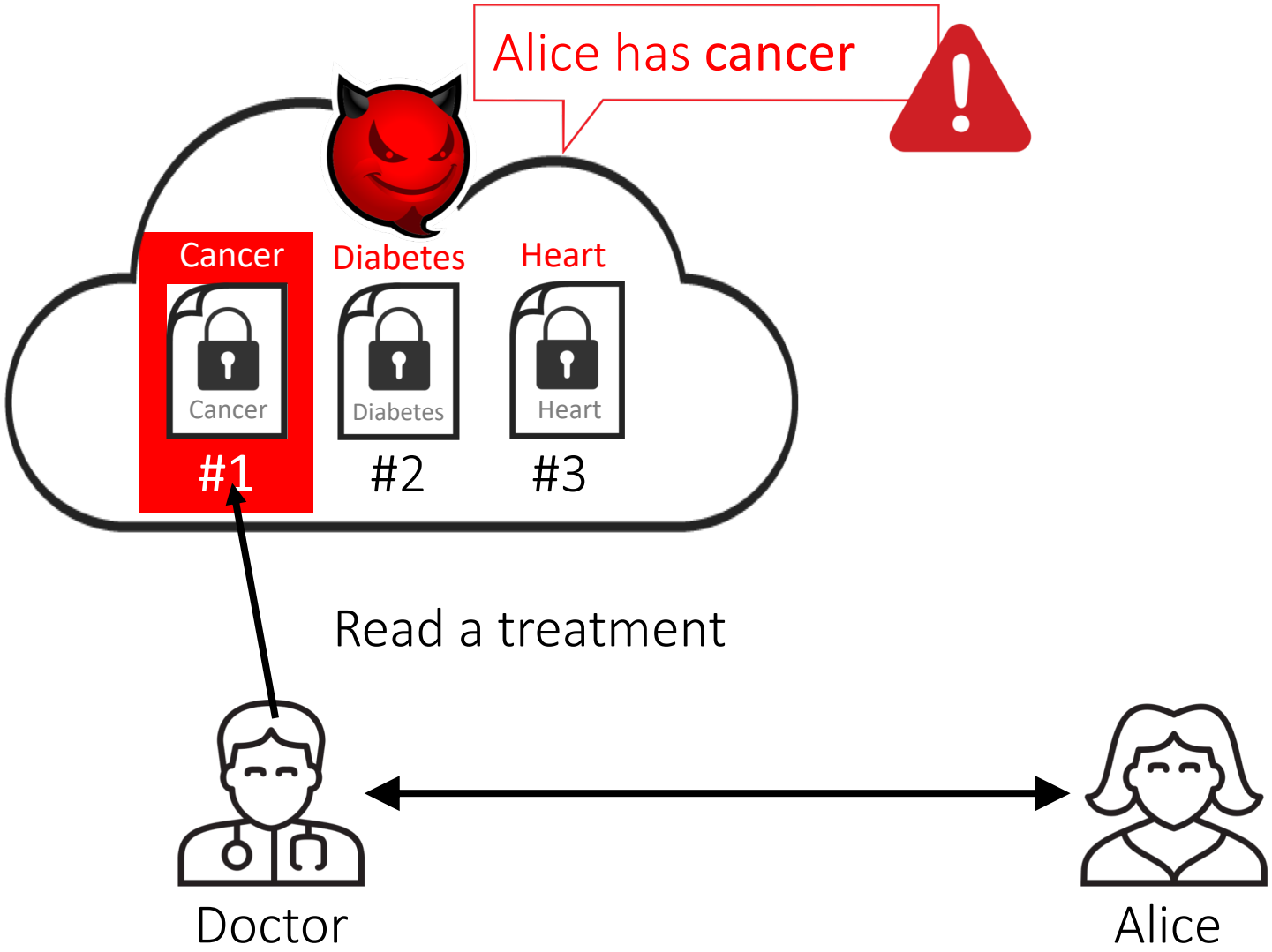
Filenames inferred

Doctor

Reveal a patient's disease



Reveal a patient's disease



Encryption is not enough: Metadata still leaks

- **User identities:** who read/wrote a file
- **File access patterns:** which file is being read/written

Existing solution: PIR-MCORAM [MMRS17]

Single-server, secure against malicious users

Lower bound: Single-server construction has linear server computation in # of files

Example:

Access a 64KB file in a million-file storage

PIR-MCORAM's amortized time $> \mathbf{30}$ min

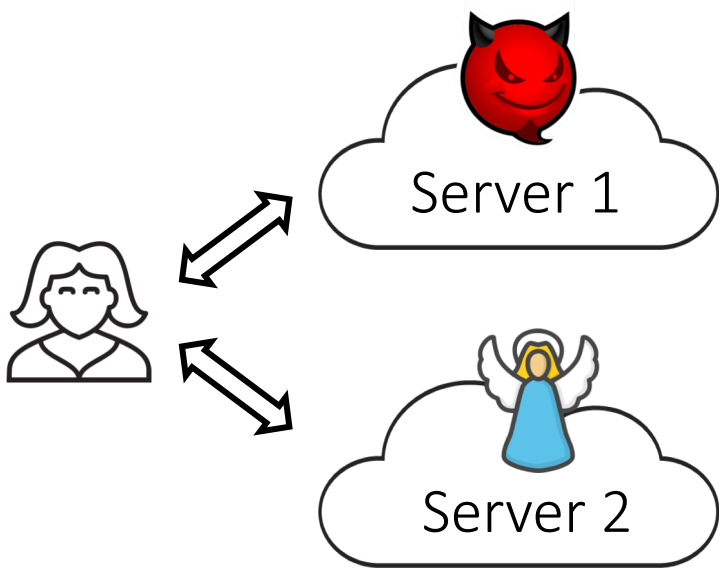
Existing solution: Anonymous RAM [BHKP16]

Assumes **two semi-honest servers** that do not collude

✓ Logarithmic overhead

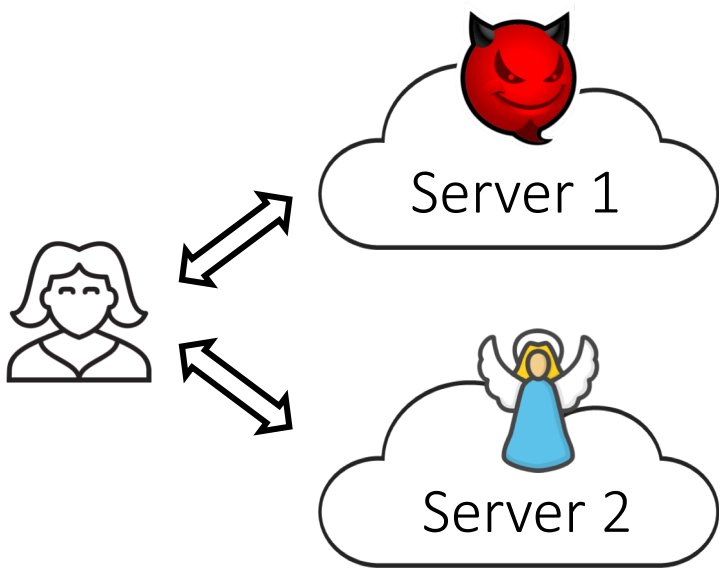
⚠ Does not support file sharing

⚠ Expensive zero-knowledge proofs



Metal

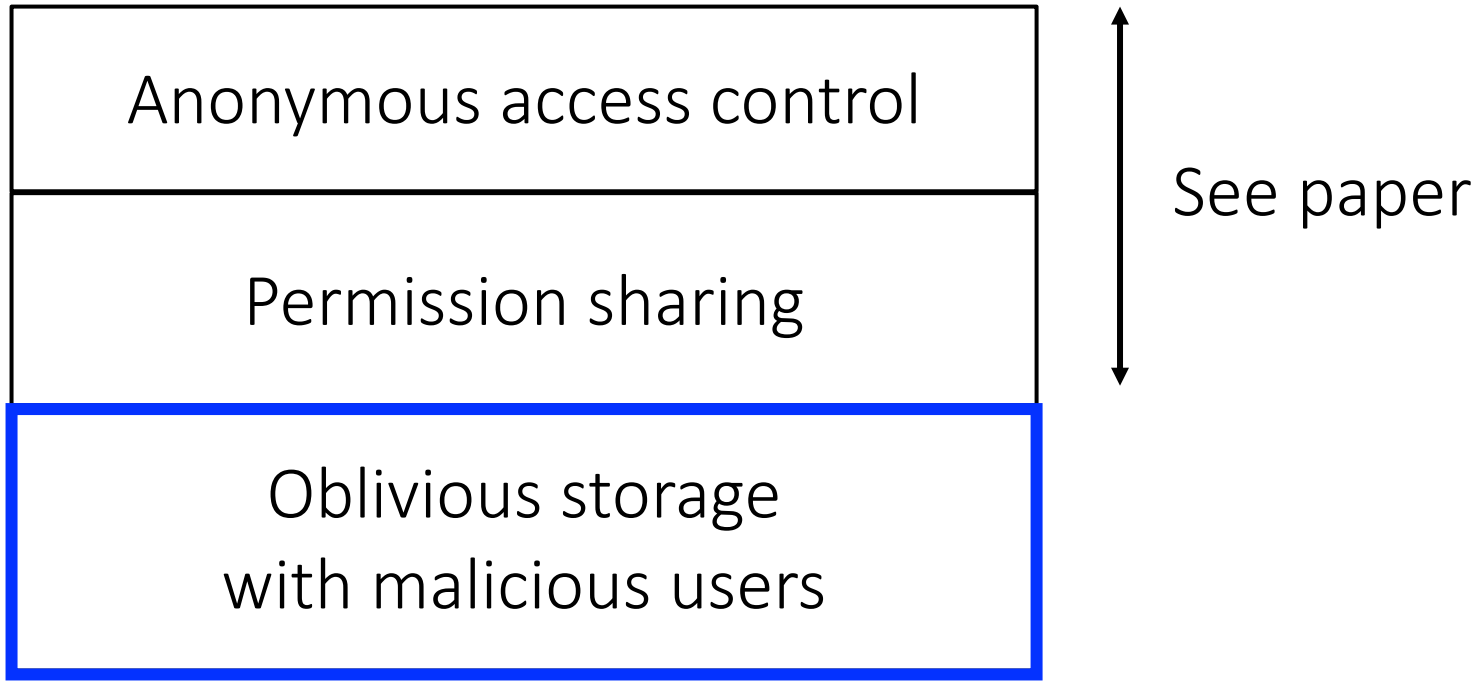
Assumes **two semi-honest servers** that do not collude



- ✓ Logarithmic overhead
- ✓ Support file sharing
- ✓ Much faster

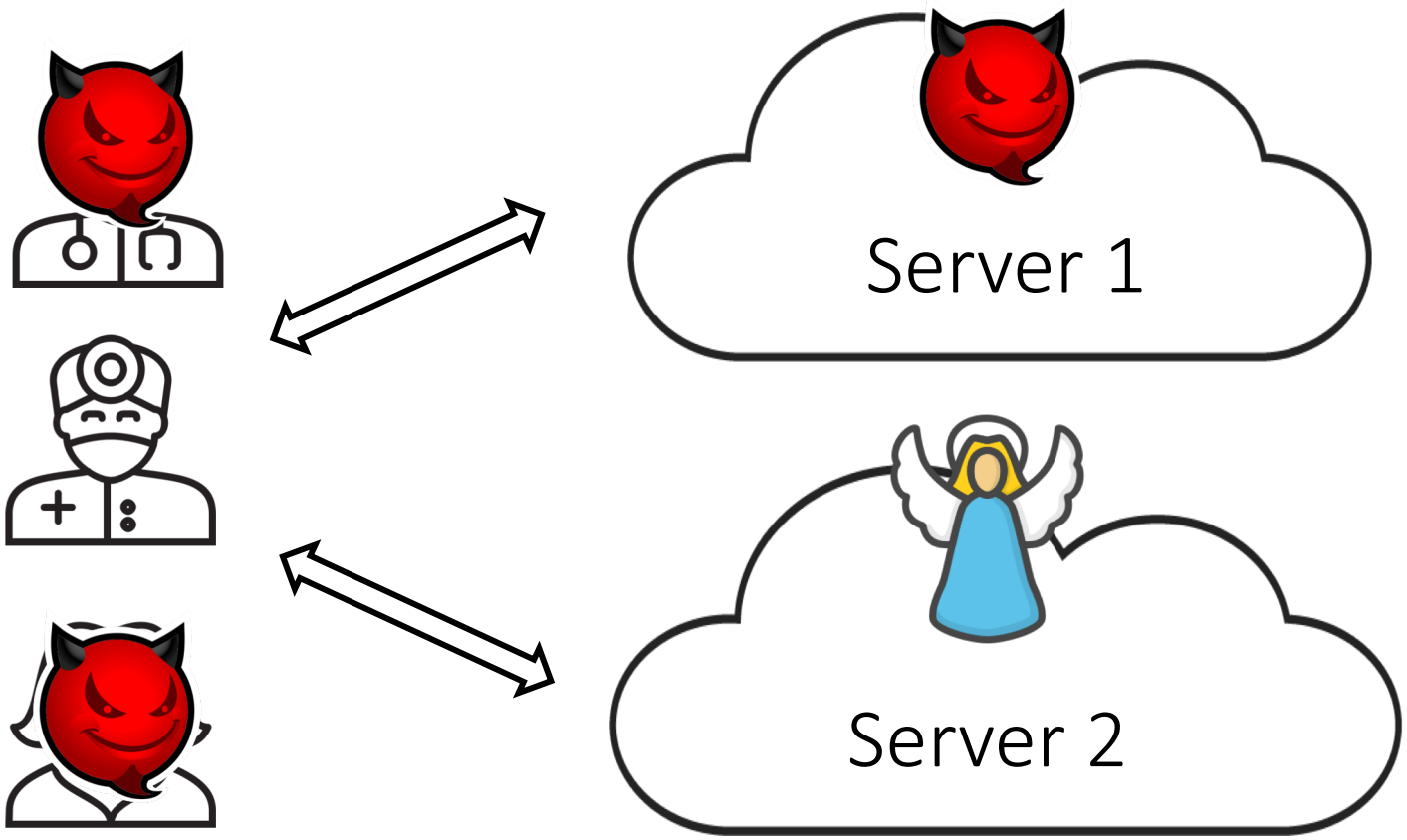
  500× faster than PIR-MCORAM and 20× faster than AnonRAM

Metal's three components



Metal's threat model

Some users
can be malicious

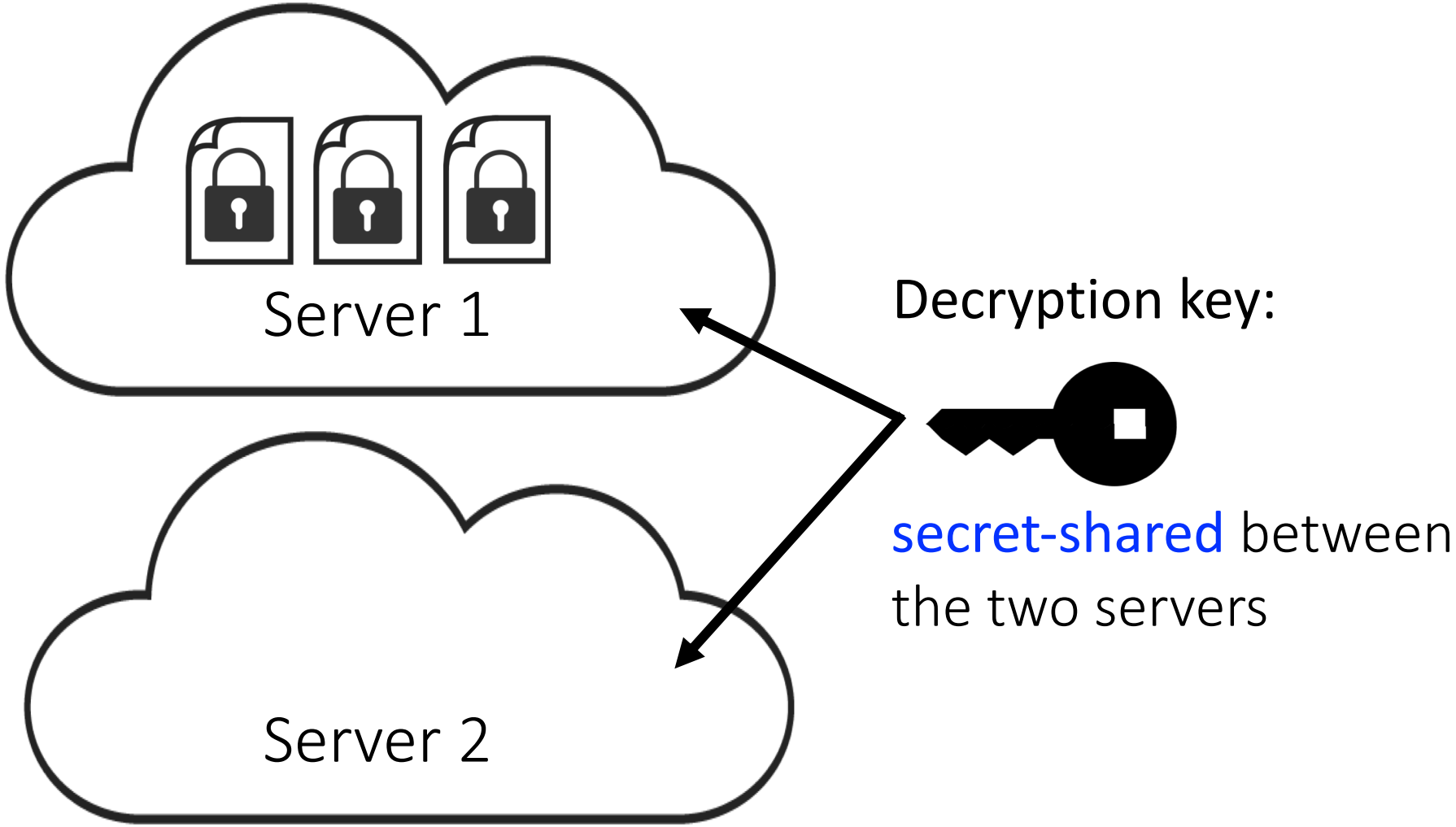


Metal’s goals

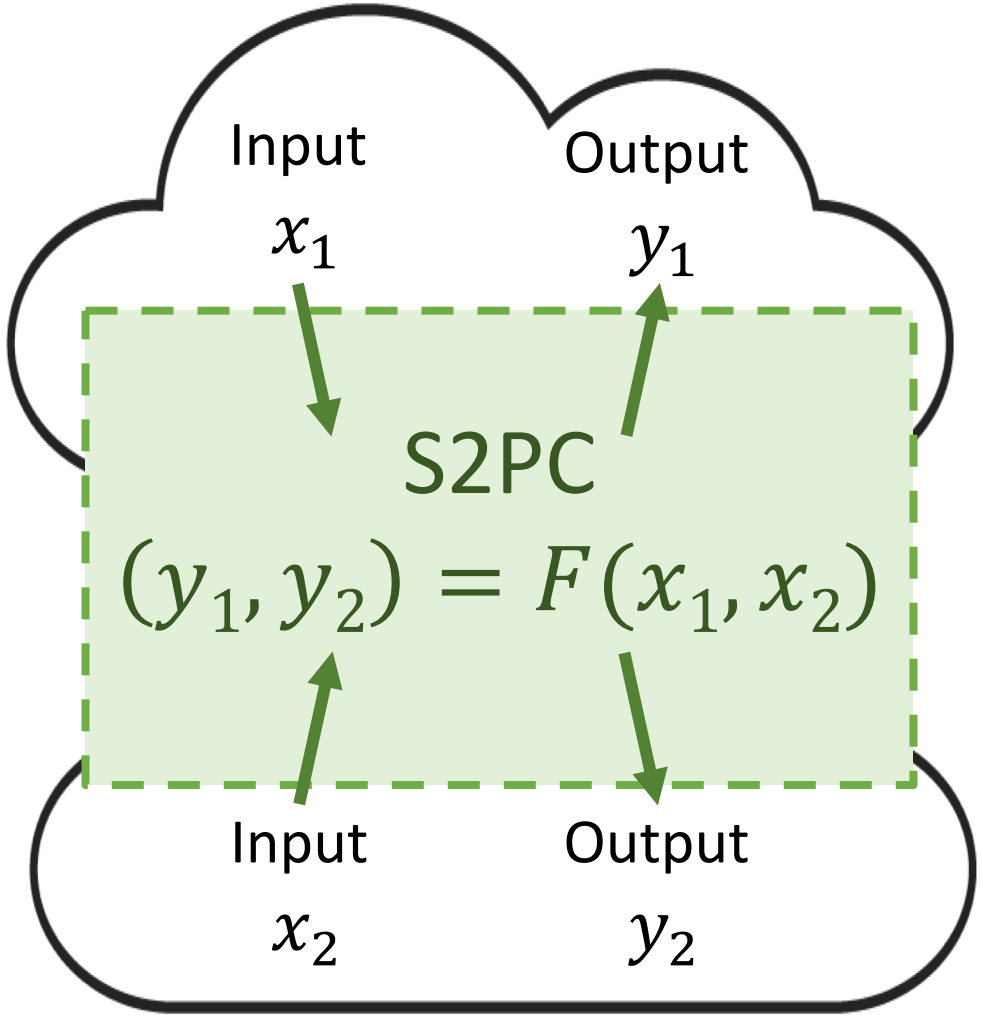
Privacy Any given access should be oblivious among all the files accessible by the honest users

Efficiency

Metal's file layout

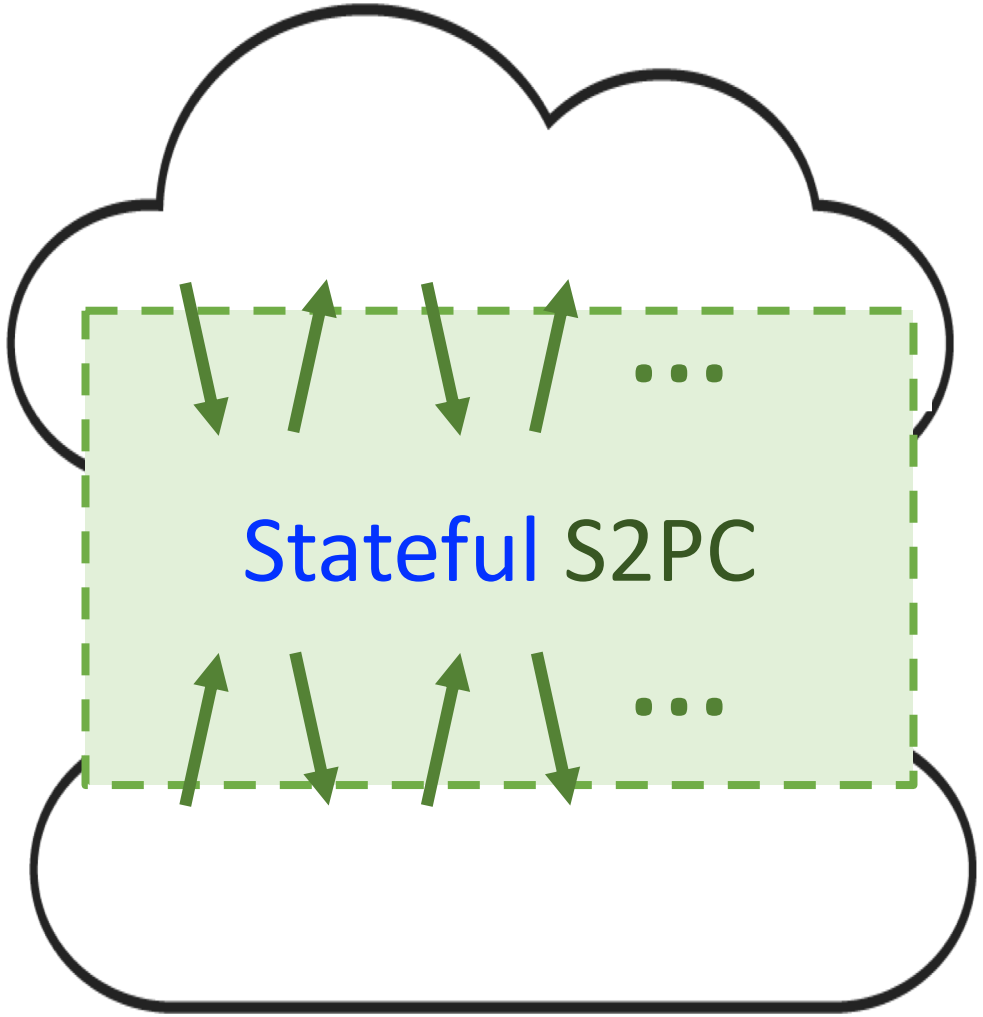


Secure two-party computation (S2PC) [Yao86]



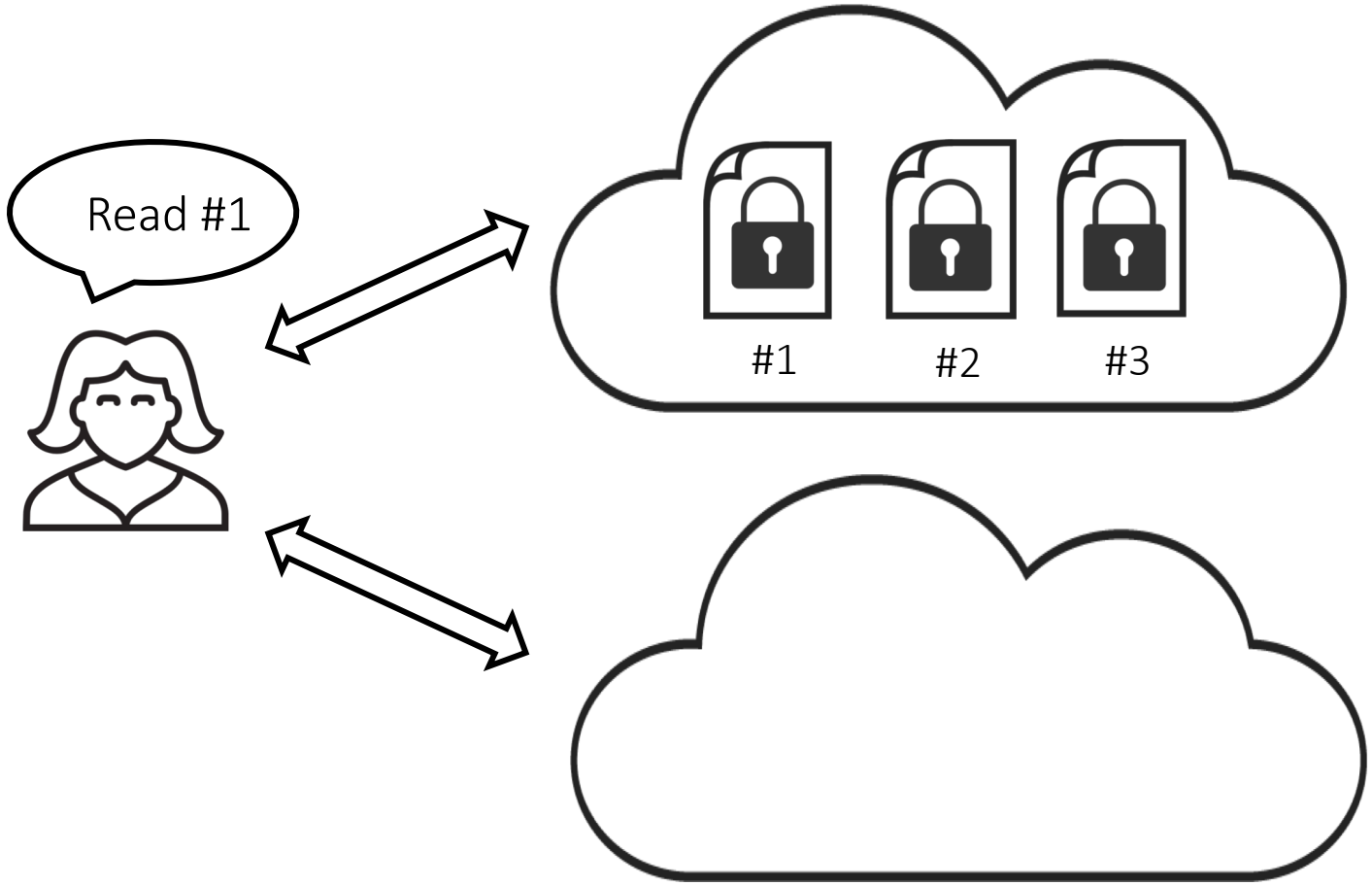
Security guarantee:
Each server only learns its own input and output

Our S2PC uses reactive Yao's protocol



The S2PC is a continuous service, rather than a one-time computation

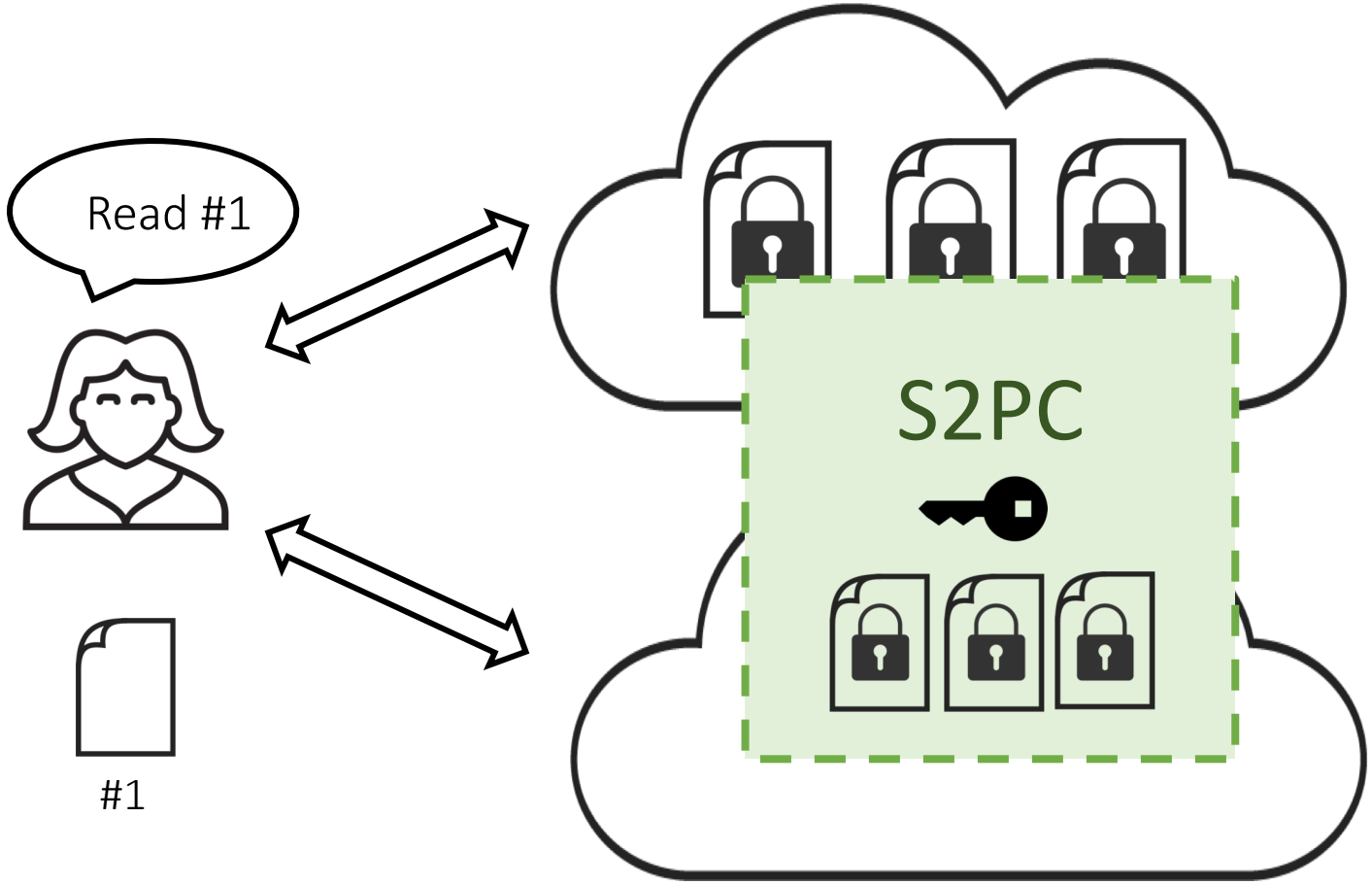
Strawman 1: All files in S2PC



Security goal:

- No server sees the file

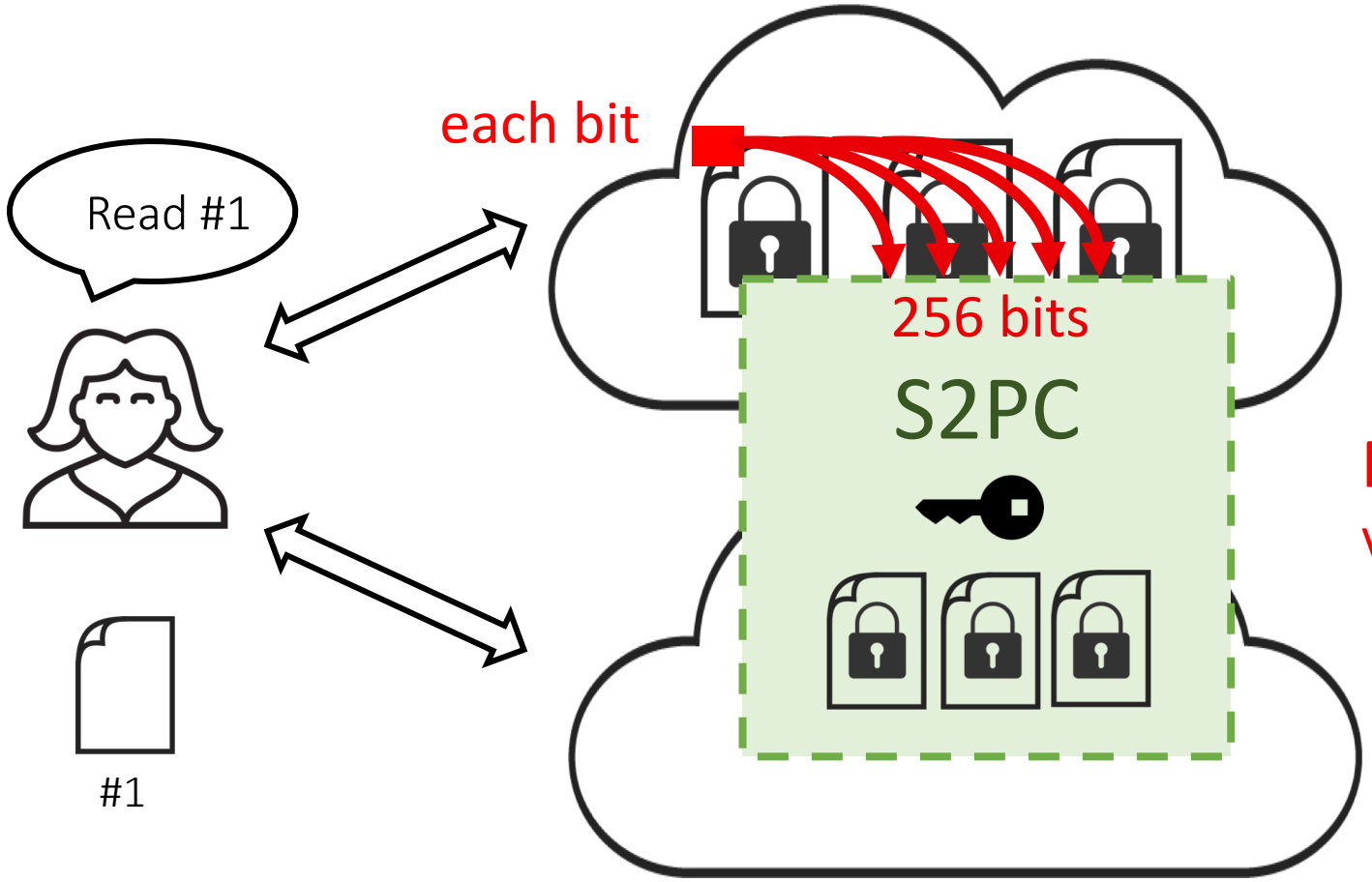
Strawman 1: All files in S2PC



Security goal:

- No server sees the file

Strawman 1: All files in S2PC

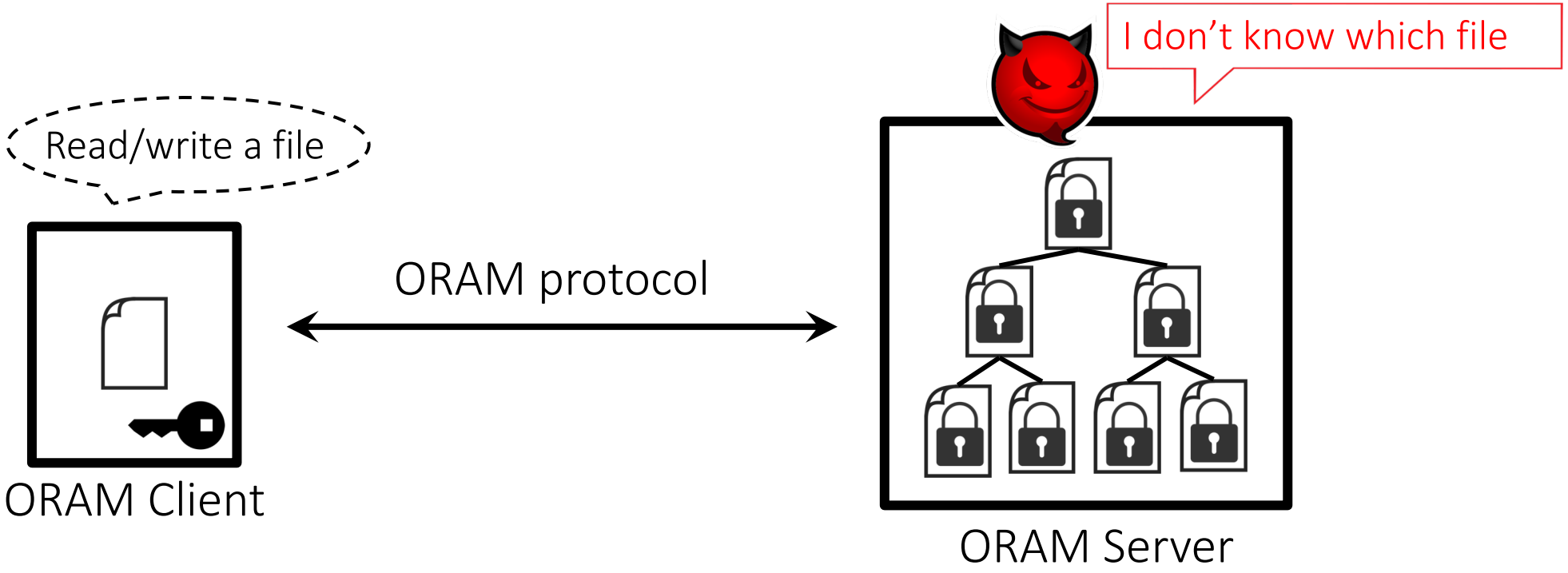


Problem:

Very large S2PC circuit

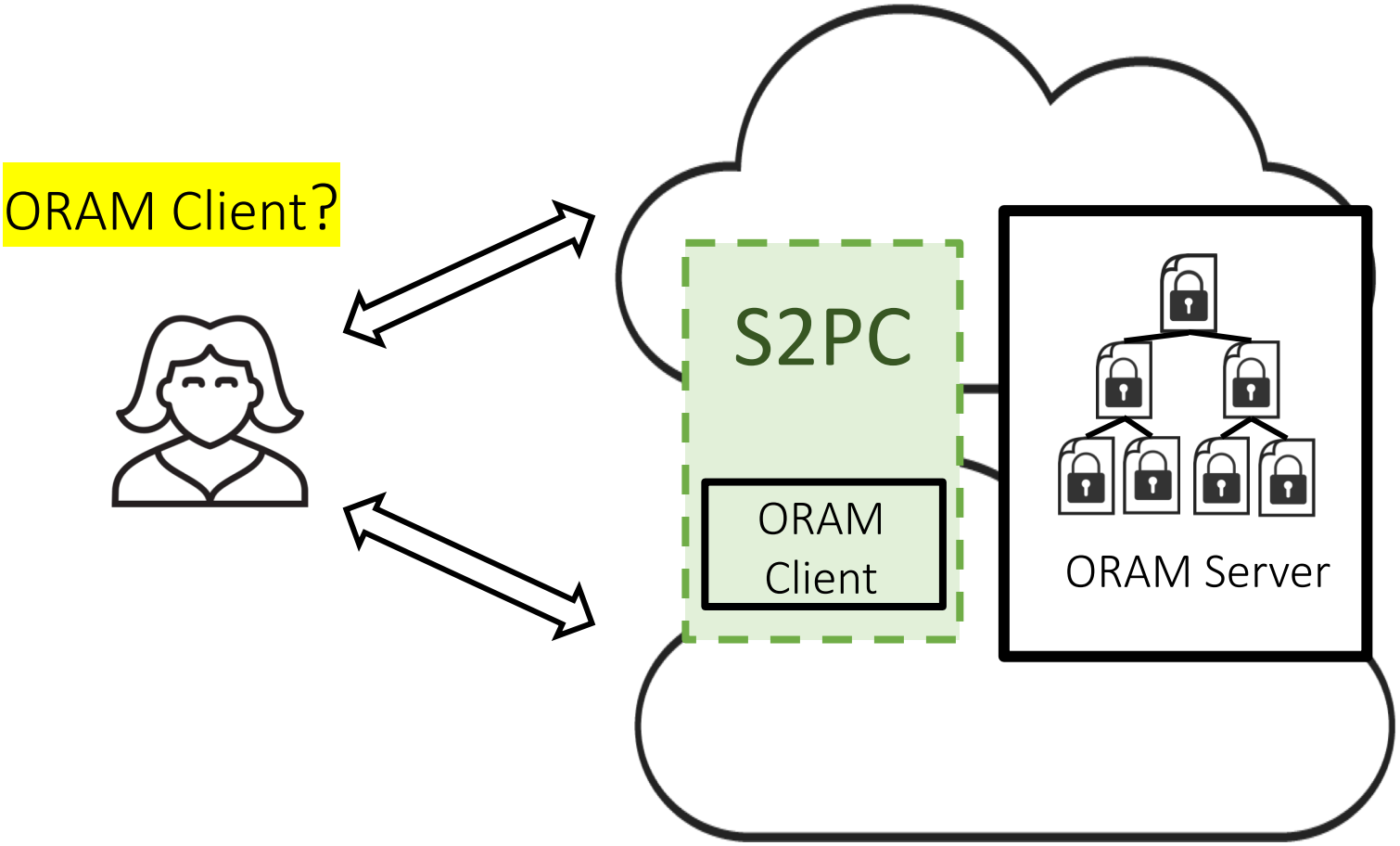
Oblivious RAM [Goldreich86, Ostrovsky90]

Access a file in a way that hides which file and is efficient

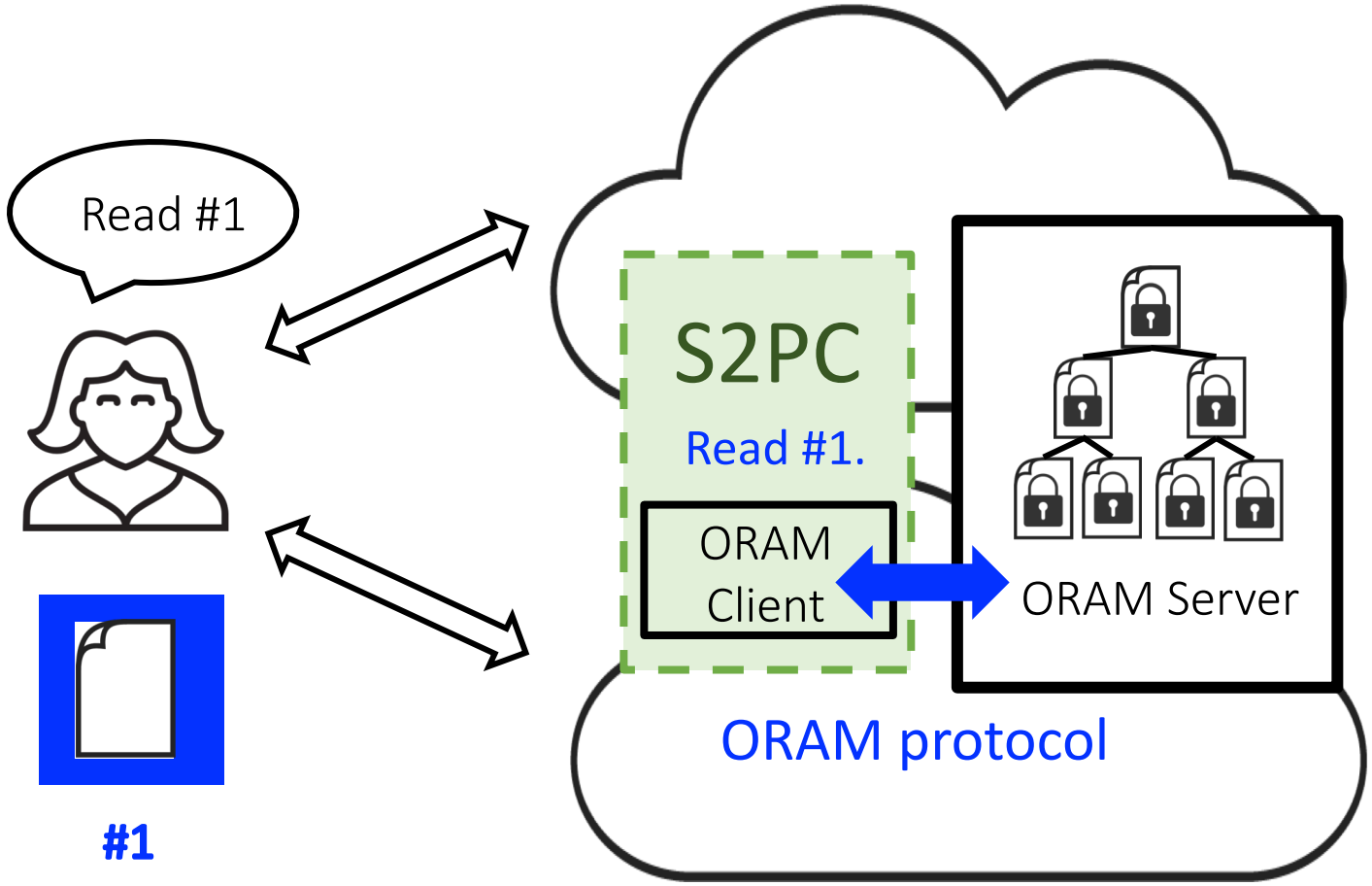


The ORAM server only accesses log(N) files

Strawman 2: S2PC + ORAM



Strawman 2: S2PC + ORAM

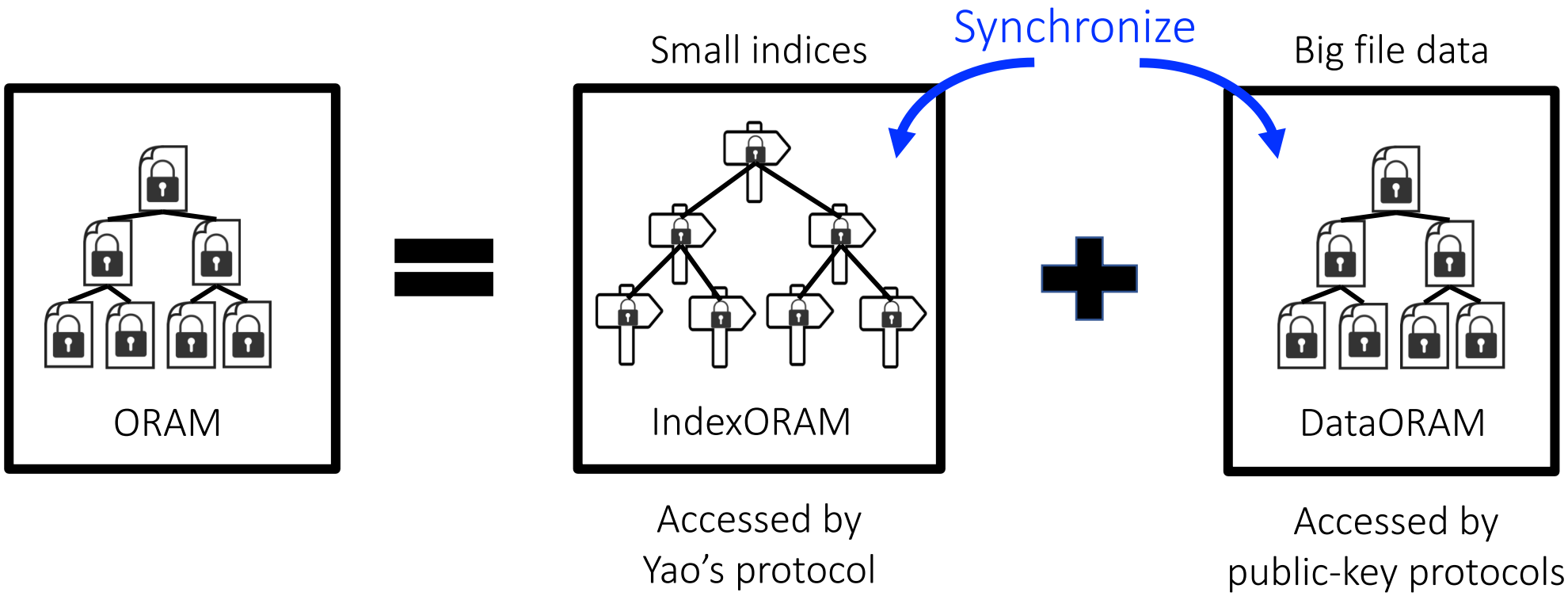


Problem:

Still very large S2PC circuit (e.g., 75 s per access)

Technique: Synchronized inside-outside ORAM

Observation: For efficiency, data should be outside S2PC



Challenge: Synchronizing IndexORAM and DataORAM

Encryption in DataORAM

- Use ElGamal encryption

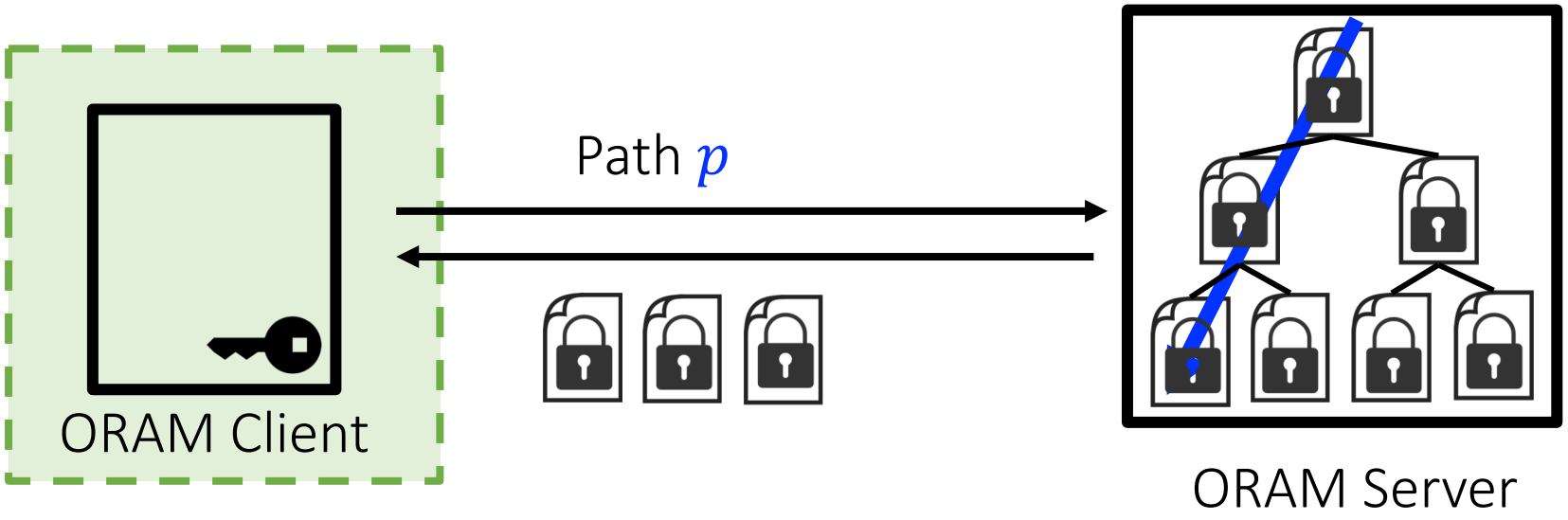
- One can **rerandomize** a ciphertext using the public key

✓ Can be leveraged in designing oblivious protocols

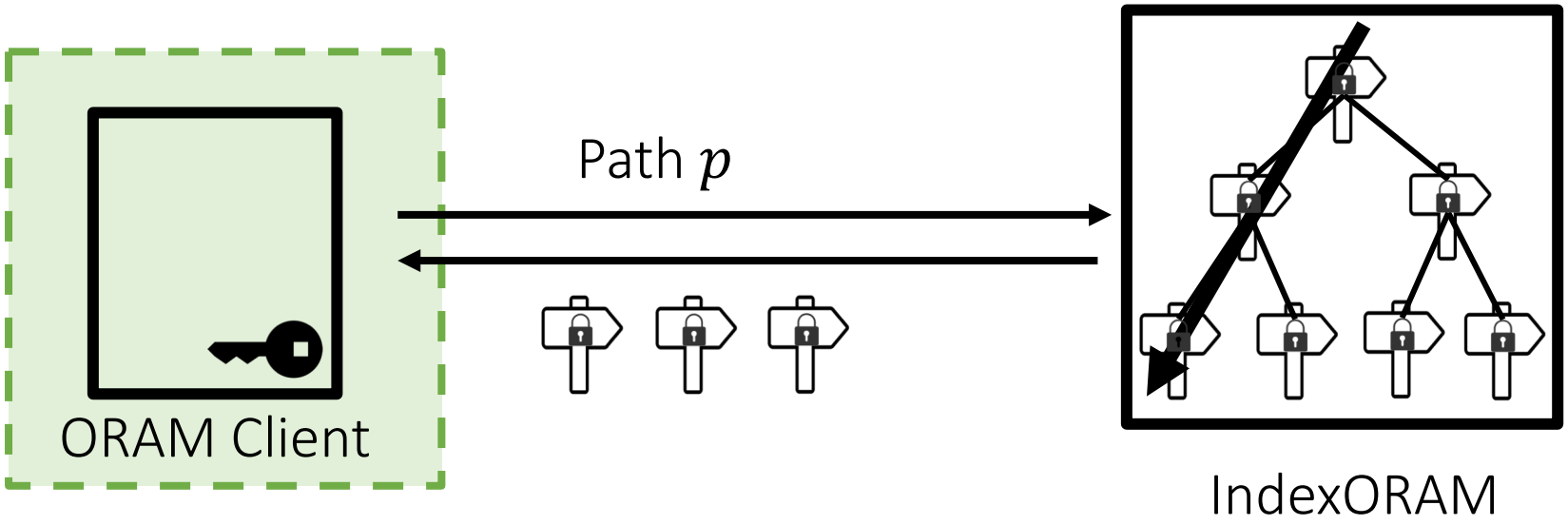
DataORAM

ORAM access

In ORAM, access to a file downloads data on a path

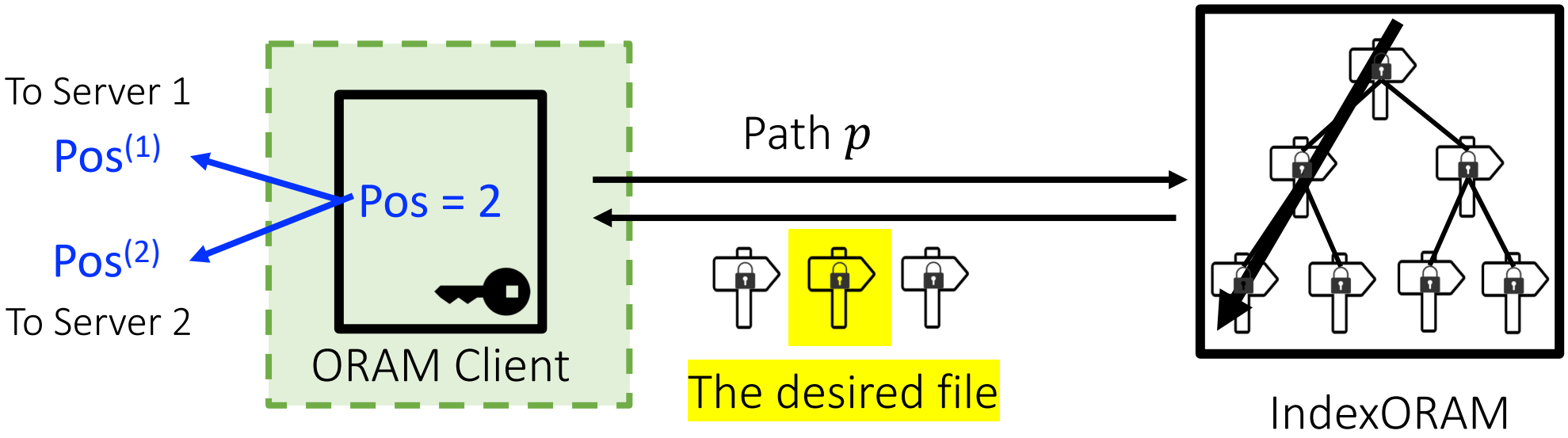


Synchronized ORAM access in Metal

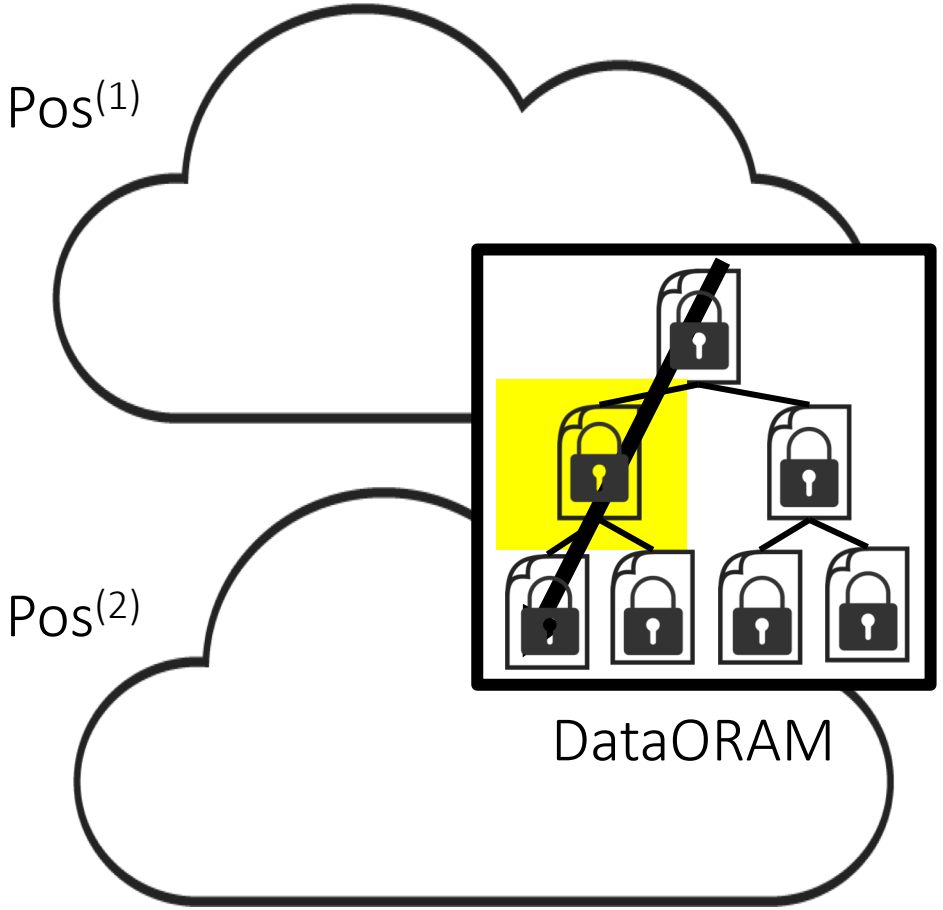


Synchronized ORAM access in Metal

Secret-share the position



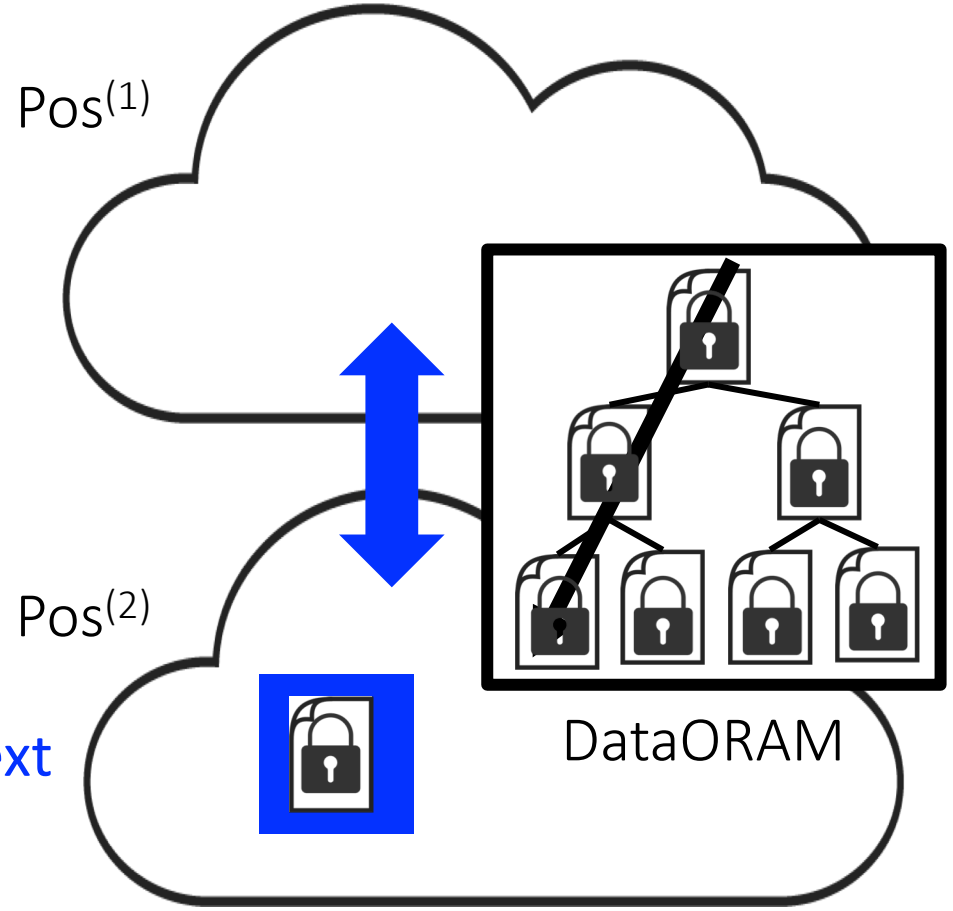
Synchronized ORAM access in Metal



Challenge:

Obtains the data obliviously

Synchronized ORAM access in Metal



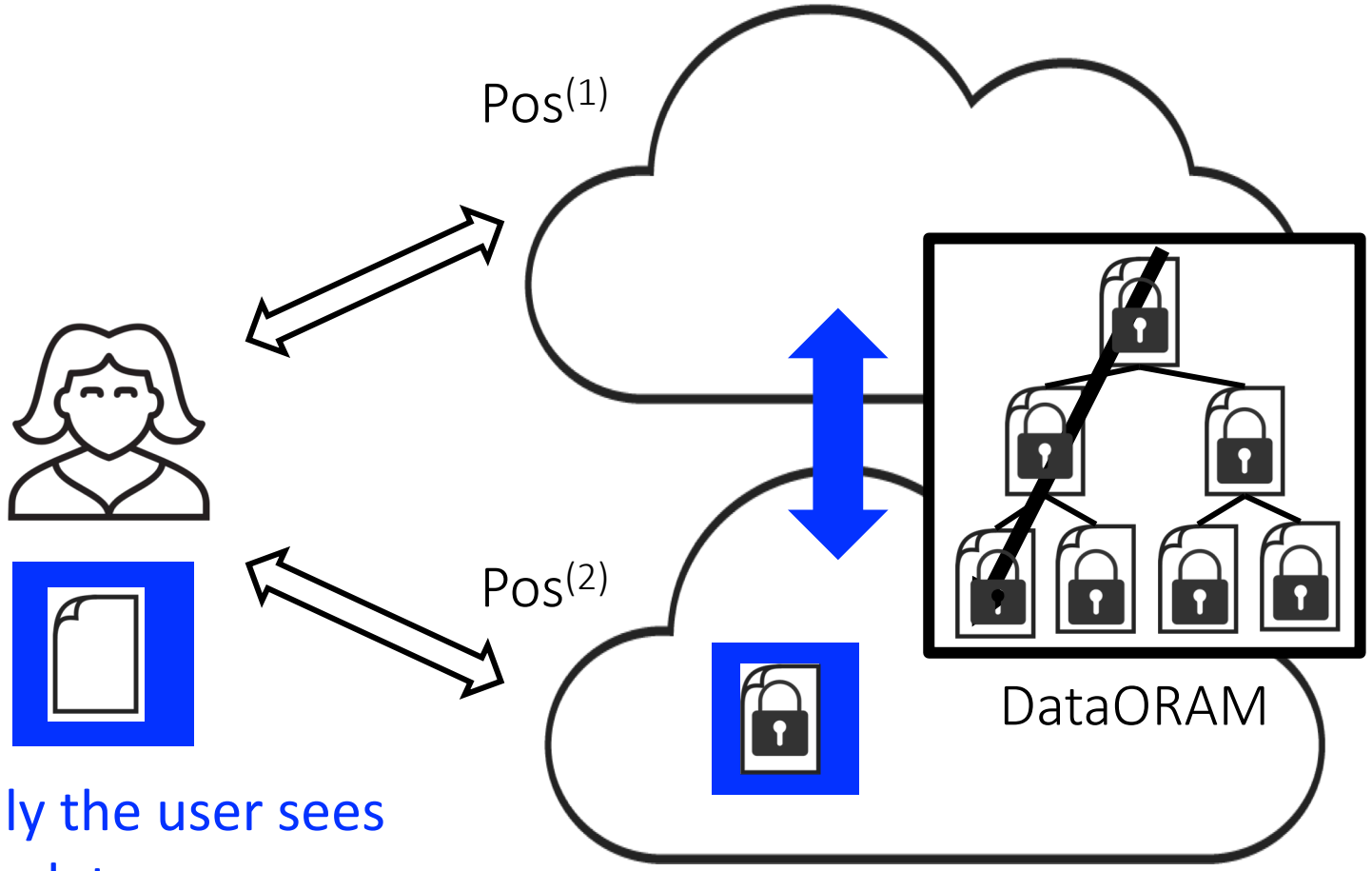
Challenge:

Obtains the data obliviously

Uses secret-shared doubly oblivious transfer

A rerandomized ciphertext of the file

Synchronized ORAM access in Metal



Challenge:

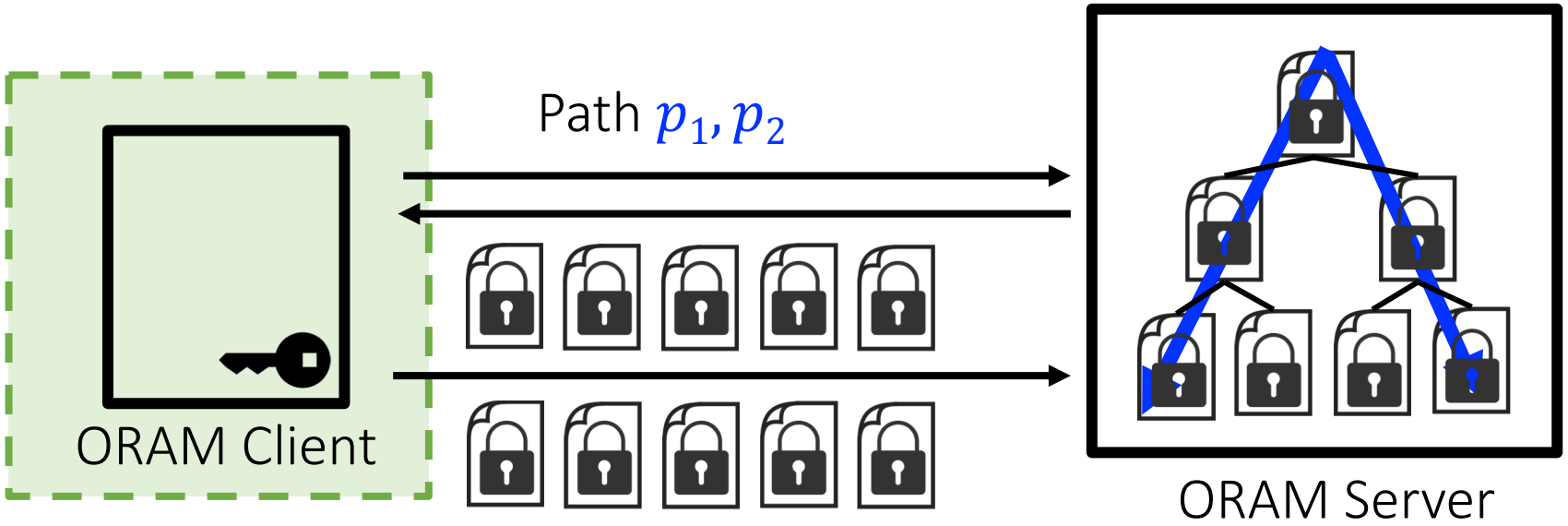
Obtains the data obliviously

The two servers then run threshold decryption to return the plaintext data.

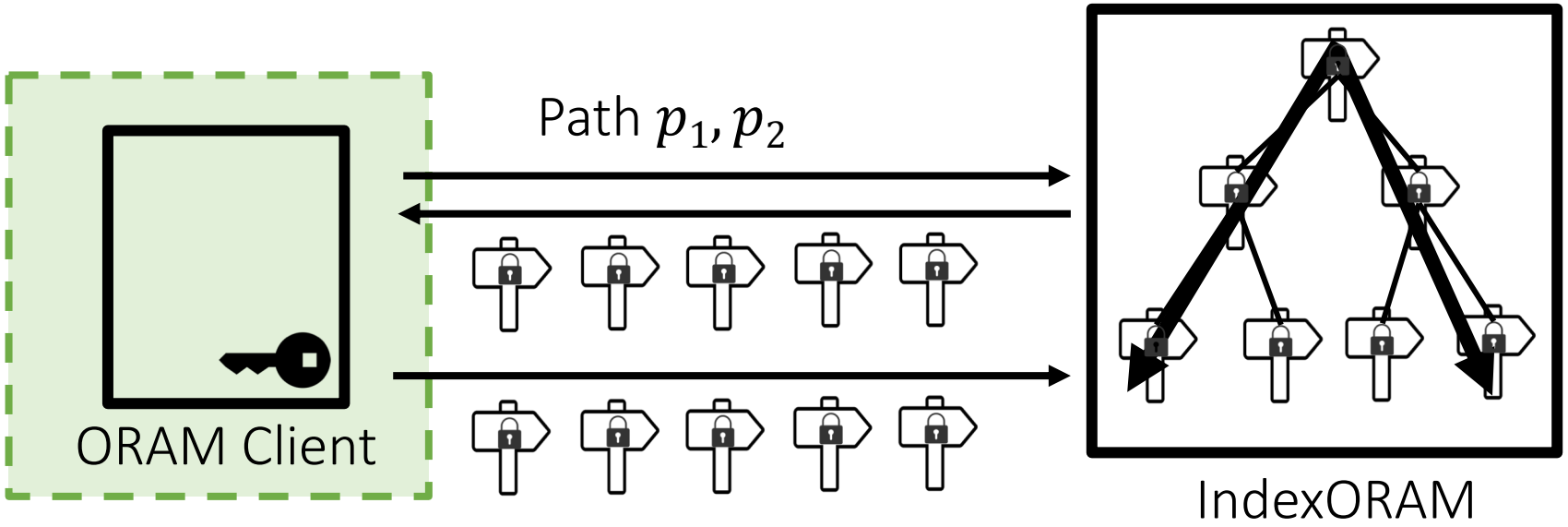
Only the user sees the data

ORAM update

The client updates the ORAM by reorganizing the files



Synchronized ORAM update in Metal

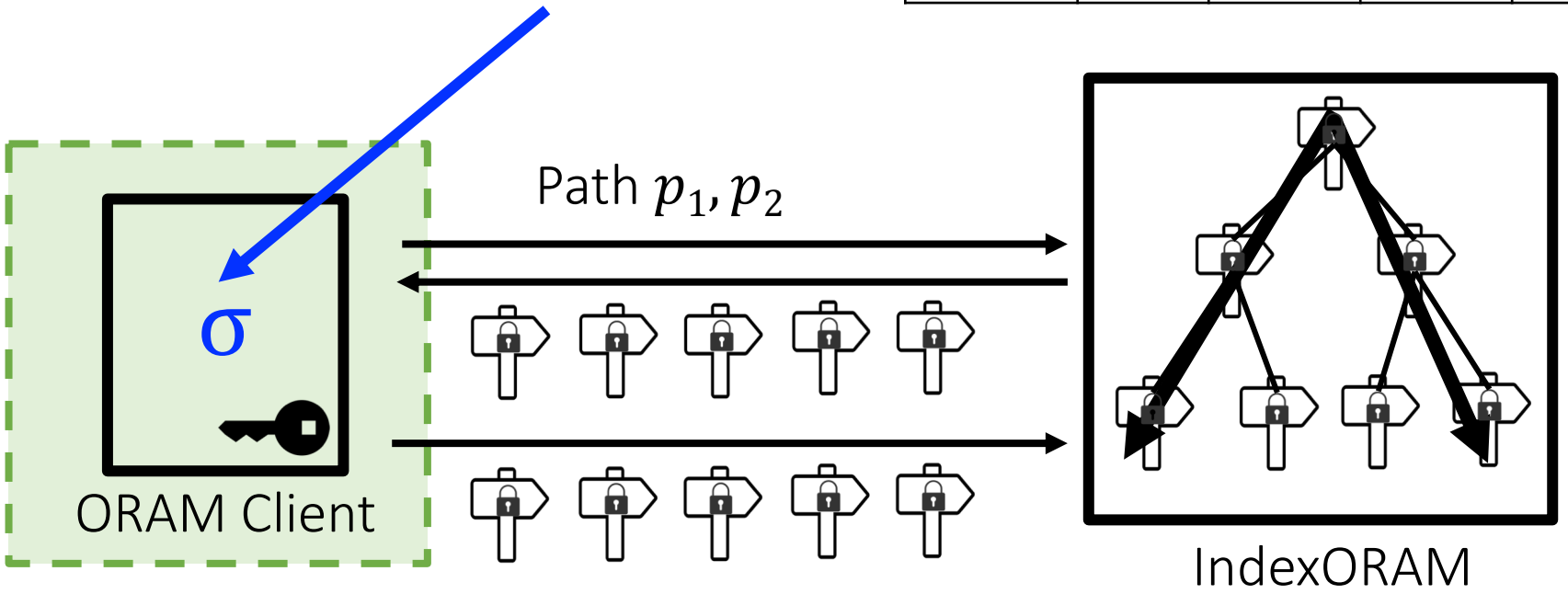


Challenge: securely apply the same update on DataORAM

Technique: Tracking and permutation generation

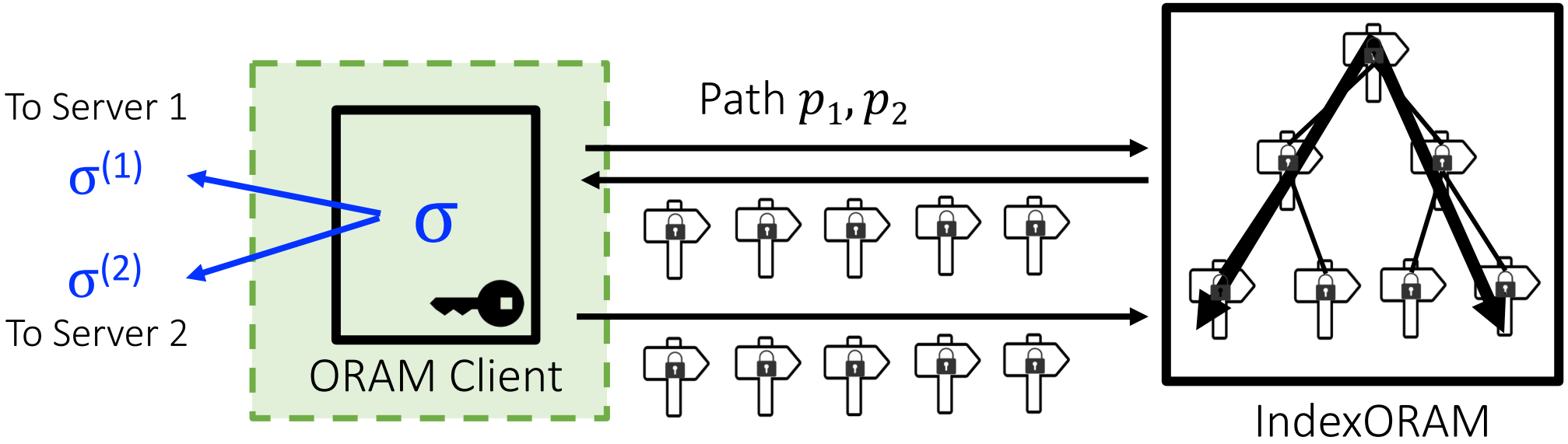
the ORAM update can be expressed as a permutation

Old	A	B	C	---	---
New	A	---	B	---	C

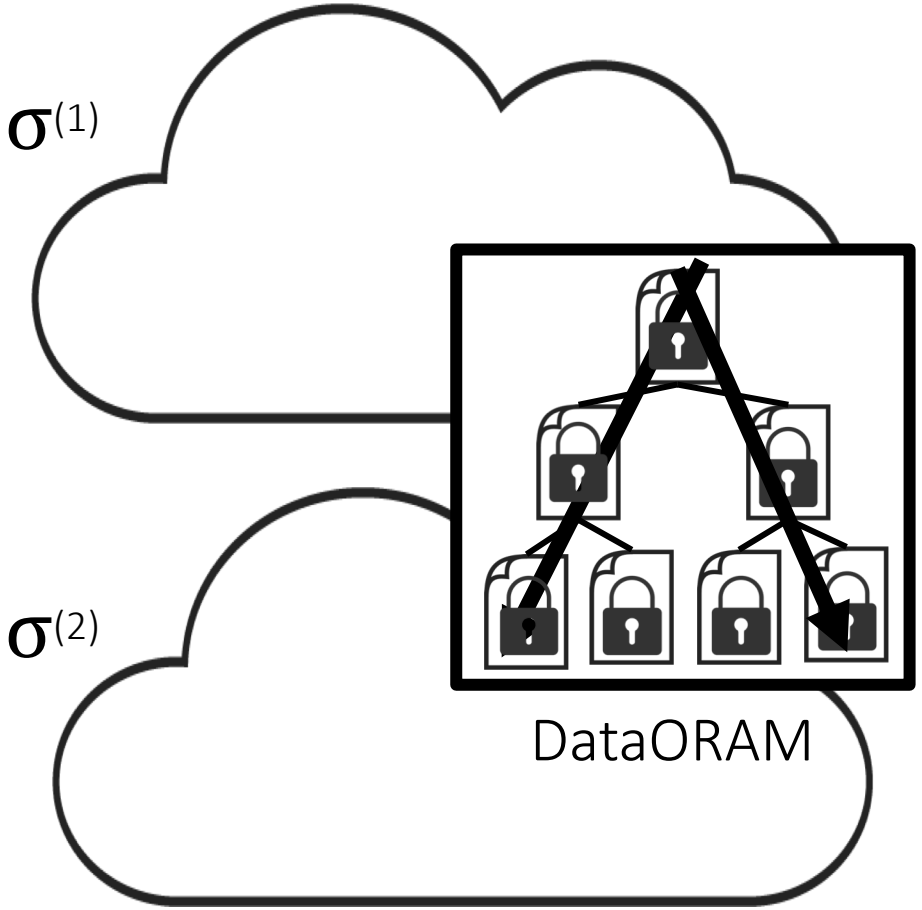


Synchronized ORAM update in Metal

Secret-share the permutation



Synchronized ORAM update in Metal

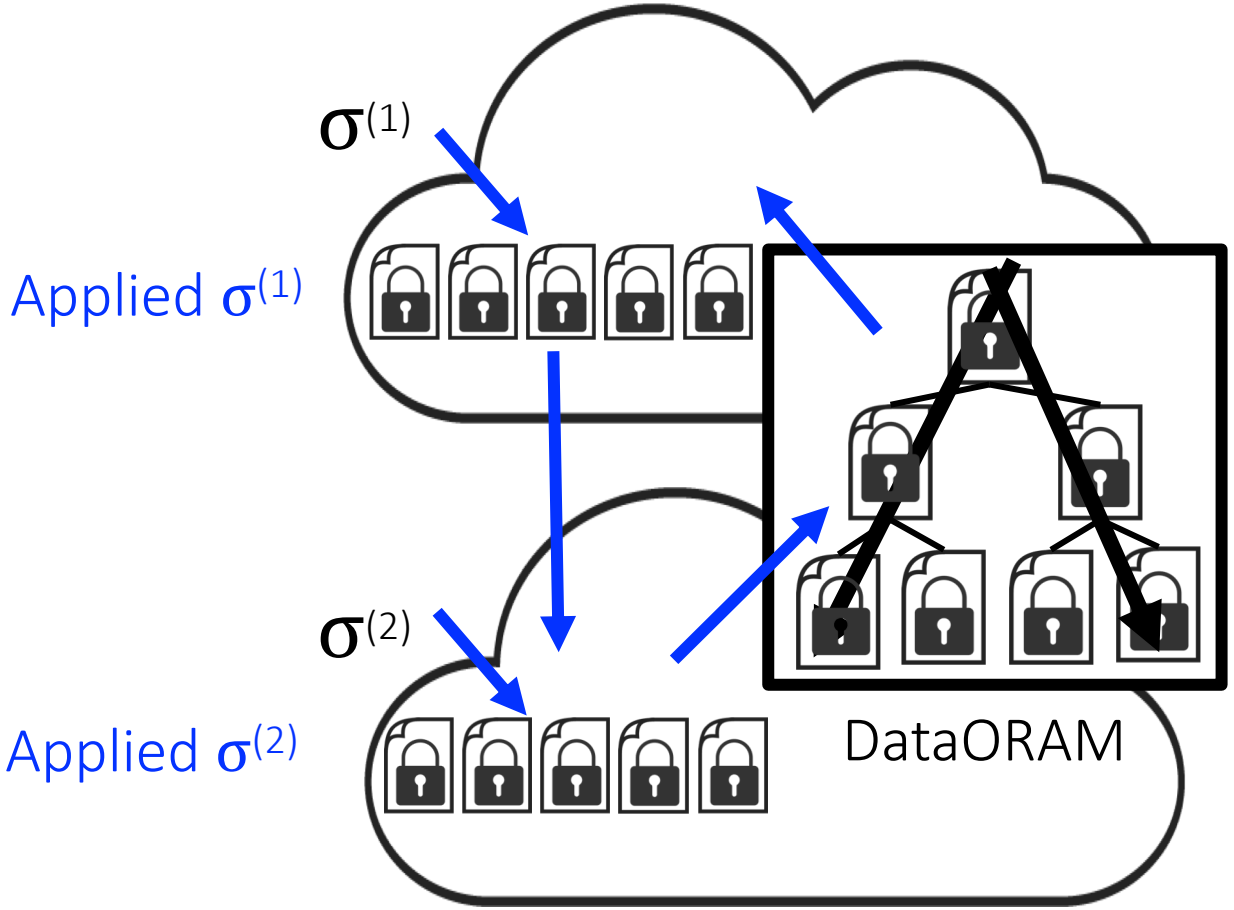


Challenge:

Applies the permutation, but hides the permutation

Each server performs a secret share of permutation in turn

Synchronized ORAM update in Metal



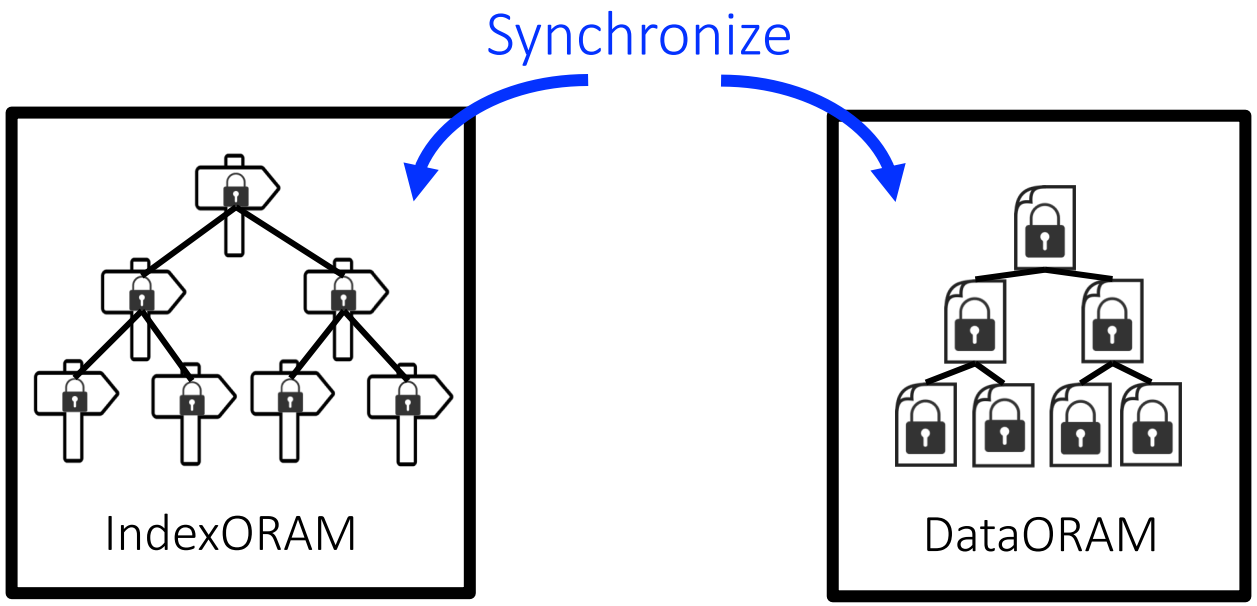
Challenge:

Applies the permutation, but hides the permutation

Each server performs a secret share of permutation in turn

Ciphertexts are rerandomized during the permutation.

Synchronized IndexORAM and DataORAM



✓ The two techniques improve over S2PC + ORAM by 20×

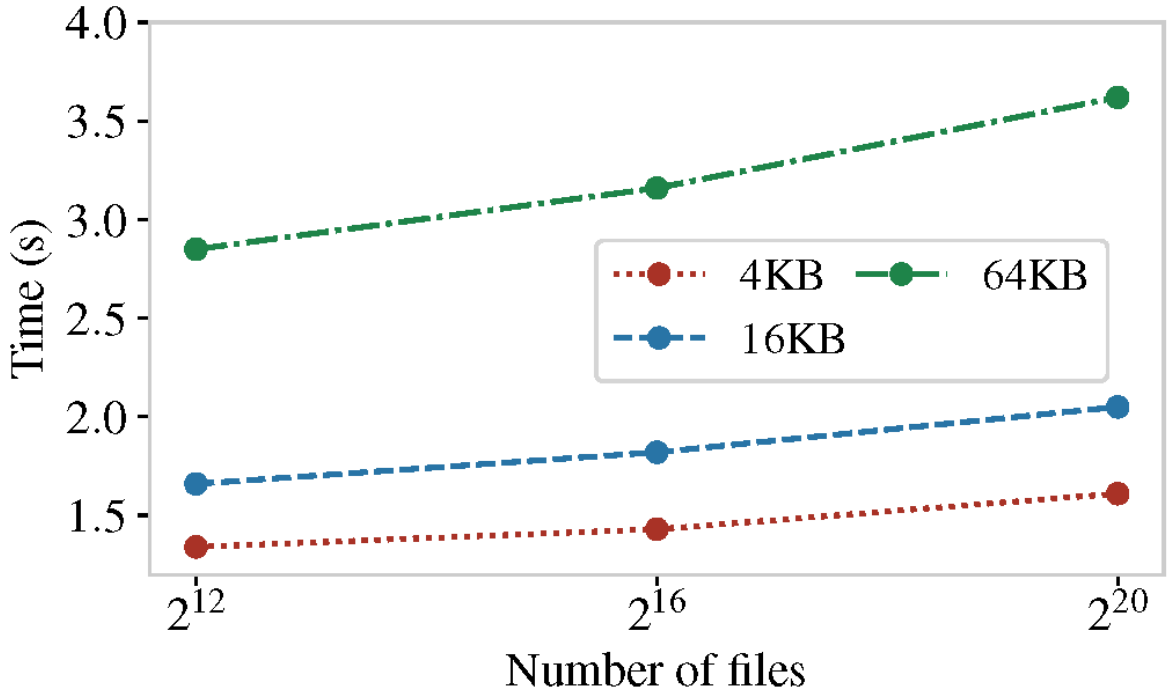
Evaluation setup

Metal is implemented in C/C++ using the Obliv-C platform [ZE15]

Evaluation setup:

- Two servers, one in Northern California, one in Oregon
- One client, in Canada

Metal's file access latency



The file access latency is within a few seconds

500× faster than PIR-MCORAM and 20× faster than AnonRAM

Metal

A Metadata-Hiding File-Sharing System

www.oblivious.app

Thank you!