SAC Summer School 2019

Encrypted Search: Leakage Suppression

Seny Kamara

BROWN

ENCRYPTED SYSTEMS LAB

How Should we Handle Leakage?

- **Approach #1:** ORAM simulation
  - Store and simulate data structure with ORAM
  - General-purpose
  - Zero-leakage (if data is transformed appropriately)
  - polylog overhead per read/write on top of simulation
- **Approach #2:** Custom oblivious structures

How Should we Handle Leakage?

- **Approach #3:** Rebuild [K.14]
  - Rebuild encrypted structure after `t` queries
  - Set `t` using cryptanalysis
  - Open question: can you rebuild encrypted structures?
- **Approach #4:** Leakage suppression
  - Suppression compilers
  - Suppression transforms

Q: can we reduce leakage?

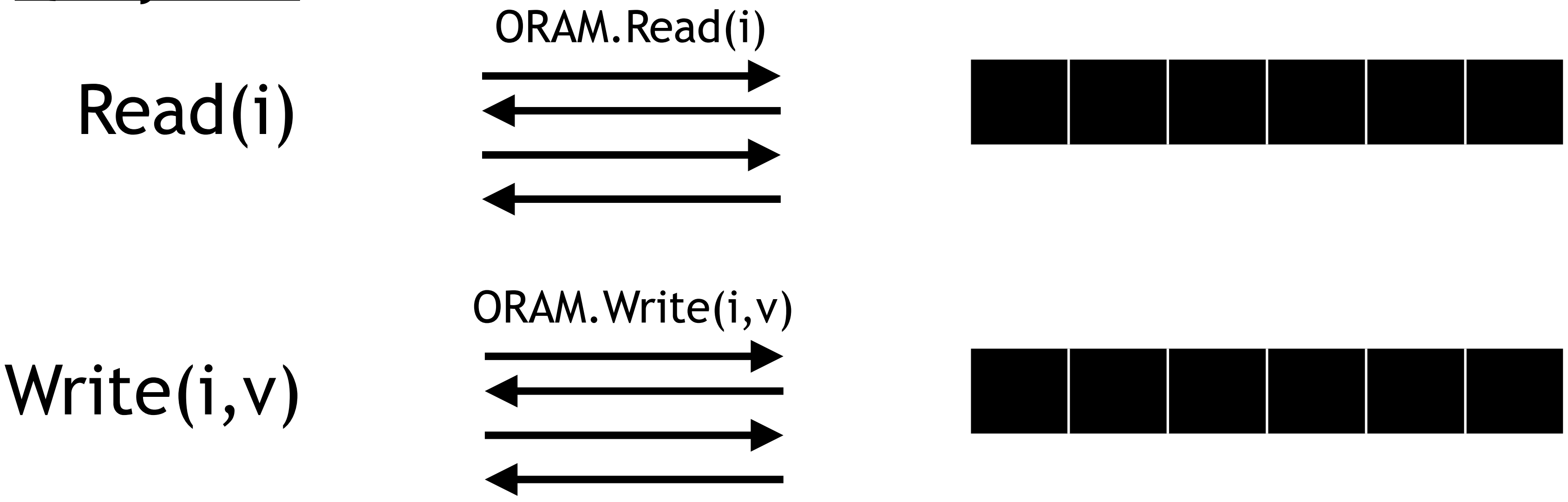
Leakage Suppression via ORAM

- Common answer is “use ORAM!”
  - usually without any details
  - or experiments
- How exactly do we use ORAM to search?

ORAM

<u>Setup time</u>

<u>Query time</u>



Leakage Suppression via ORAM

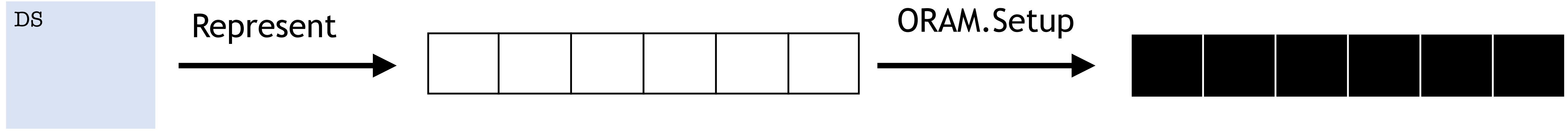
- ORAM supports read & write operations to an array
  - with polylog(n) cost
  - and leakage profile $\mathbf{\Lambda}_{\text{ORAM}} = (\mathscr{L}_{\text{S}}, \mathscr{L}_{\text{Q}}) = (\text{dsize}, \perp)$
- ORAM is a “low-level” primitive
  - designed for read/write operations to an *array*
  - what if we want to query a more complex structure?
- Need to use ORAM simulation

ORAM Simulation

- Represent DS as an array and store in ORAM
- Client simulates `Query(DS,q)` algorithm
  - replaces each `Read(i)` with `ORAM.Read(i)`
  - replaces each `Write(i,v)` with `ORAM.Write(i,v)`

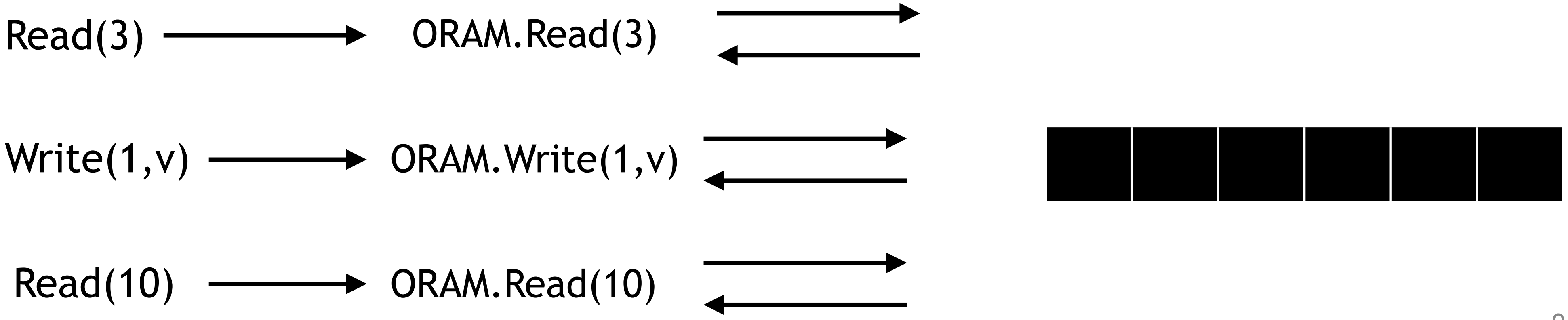
ORAM Simulation

Setup time



Query time

Query(DS,q)

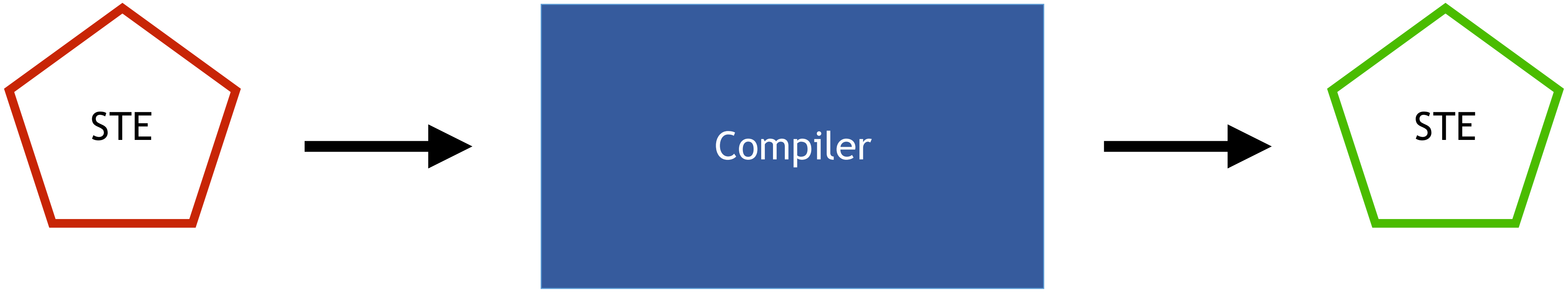


ORAM Simulation

- Costs O(T·polylog(|DS|))
  - where T is runtime of Query(DS,q)
- Leakage profile
  - Λ = (dsize, (runtime, vol))
  - vol: size of response (can be suppressed with padding)
- Can we do better?

Q: can we do better than ORAM simulation?

Suppression Compiler



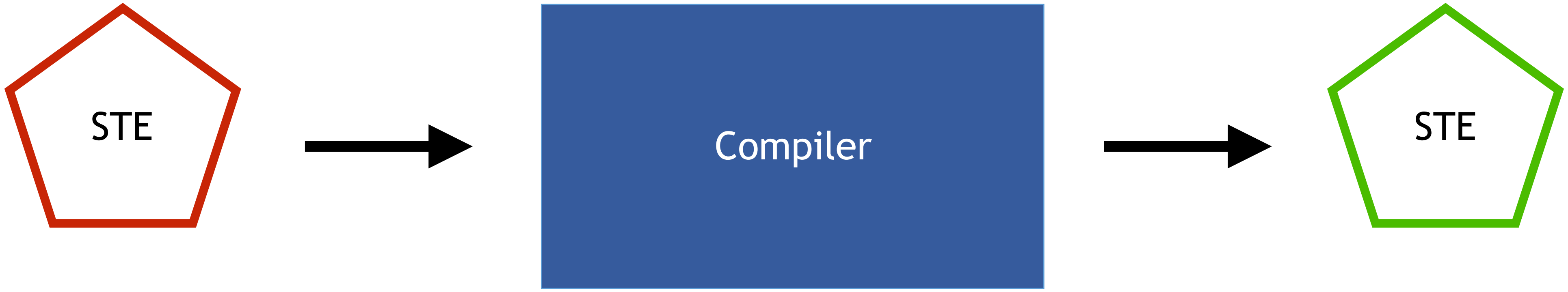
$\Lambda = (\mathscr{L}_{S}, \mathscr{L}_{Q})$

$= (\bigstar, (\text{patt}_1, \text{patt}_2))$

$\Lambda = (\mathscr{L}_{S}, \mathscr{L}_{Q})$

$= (\bigstar, \text{patt}_2)$

Suppression Compiler for Query Equality

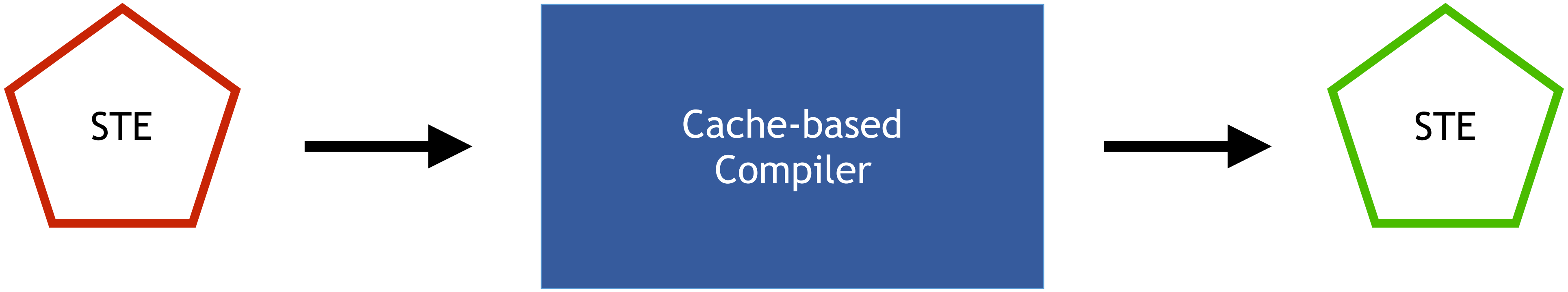


$\Lambda = (\mathscr{L}_{S}, \mathscr{L}_{Q})$
$= (\bigstar, \mathrm{qeq})$

$\Lambda = (\mathscr{L}_{S}, \mathscr{L}_{Q})$
$= (\bigstar, \perp)$

Q: Can we build such a thing?

Suppression Compiler for Query Equality



$\Lambda = (\mathscr{L}_S, \mathscr{L}_Q)$
$= (\bigstar, (\text{qeq}, \text{patt}))$

$\Lambda = (\mathscr{L}_S, \mathscr{L}_Q)$
$= (\bigstar, \text{nrp})$

nrp is the *non-repeating sub-pattern* of patt

Non-Repeating Sub-Patterns

- Leakage patterns can be decomposed into sub-patterns:

$$\mathsf{patt} = \begin{cases} \mathsf{patt}_1 & \text{if ``condition'' is true} \\ \mathsf{patt}_2 & \text{otherwise.} \end{cases}$$

- Non-repeating sub-patterns ≈ leakage on non-repeating queries

$$\mathsf{patt} = \begin{cases} \mathsf{nrp} & \text{if queries are unique} \\ \mathsf{misc} & \text{otherwise.} \end{cases}$$

Suppression Compiler for Query Equality

$\Lambda = (\mathscr{L}_S, \mathscr{L}_Q)$
$= (\bigstar, (\text{qeq}, \text{patt}))$

$\Lambda = (\mathscr{L}_S, \mathscr{L}_Q)$
$= (\bigstar, \text{nrp})$

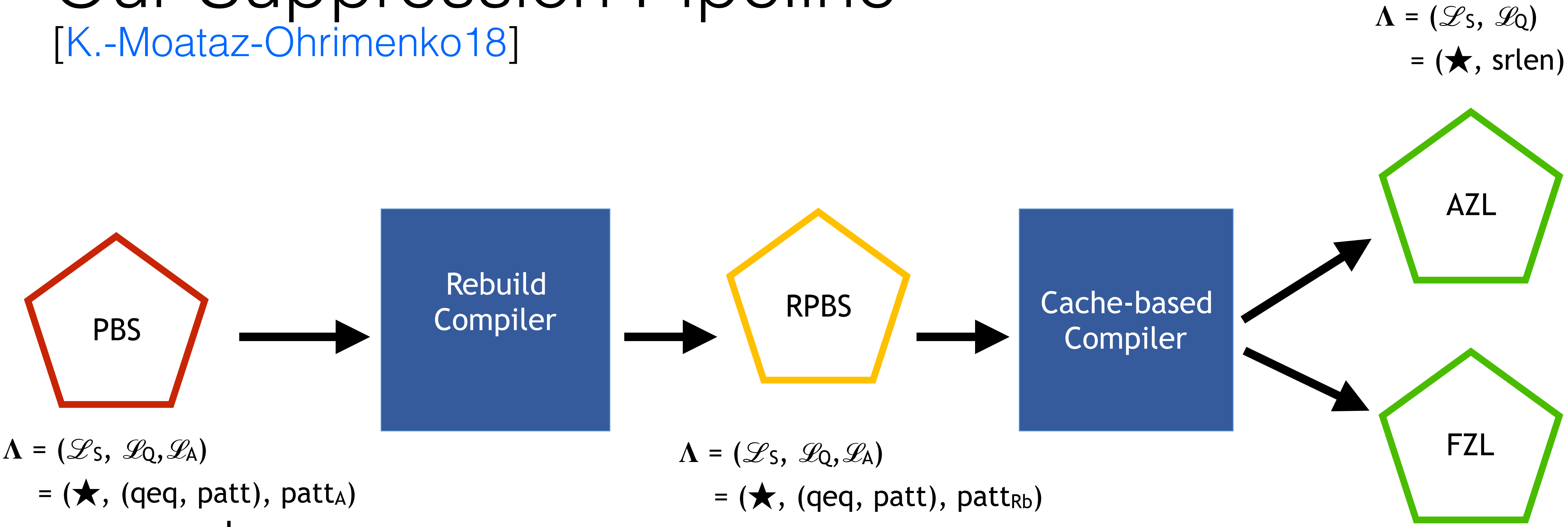
$$\text{patt} = \begin{cases} \text{nrp} & \text{if queries are unique} \\ \text{misc} & \text{otherwise.} \end{cases}$$

Cache-based Compiler and Rebuilding

- Cache-based Compiler
  - needs to rebuild encrypted structure from time to time
- So base STE scheme has to have a Rebuild protocol
- Rebuild protocol must
  - be efficient for server
  - have $O(1)$ client storage
  - be zero-leakage

Our Suppression Pipeline

[K.-Moataz-Ohrimenko18]

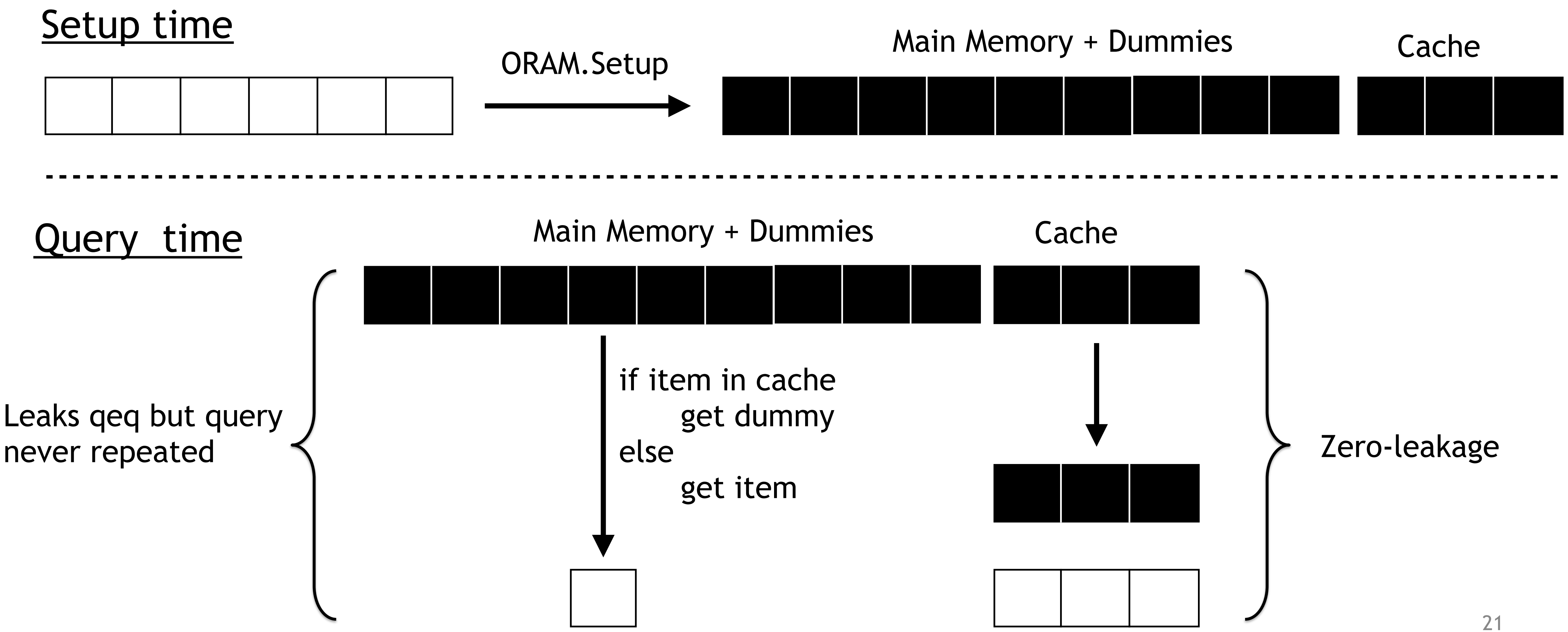


$$\text{patt} = \begin{cases} \text{srlen} & \text{if queries are unique} \\ \text{misc} & \text{otherwise.} \end{cases}$$

Q: How does the CBC work?

Square Root ORAM

[Goldreigh-Ostrovsky92]



Reinterpreting Square Root ORAM

[K.-Moataz-Ohrimenko18]

Setup time

ORAM.Setup →

Main Memory + Dummies

ERAM

$\Lambda_{\text{ERAM}} = (\mathcal{L}_S, \mathcal{L}_Q) = (\bigstar, \text{qeq})$

Cache

EDX

$\Lambda_{\text{EDX}} = (\mathcal{L}_S, \mathcal{L}_Q) = (\bigstar, \perp)$

Query time

Main Memory + Dummies

ERAM

$\Lambda_{\text{ERAM}} = (\mathcal{L}_S, \mathcal{L}_Q) = (\bigstar, \text{qeq})$

if item in cache
    get dummy
else
    get item

Cache

EDX

$\Lambda_{\text{EDX}} = (\mathcal{L}_S, \mathcal{L}_Q) = (\bigstar, \perp)$

Reinterpreting Square Root ORAM

- Square root ORAM ≈
  - "uses a ZL encrypted dictionary…
  - …to suppress the qeq leakage of an encrypted RAM"
- Q: Can we replace the ERAM with another encrypted structure?
  - if yes then no multiplicative **polylog** overhead due to simulation

The Cache-Based Compiler

<u>Setup time</u>

ORAM.Setup

Main Memory + Dummies

EDS

$\Lambda_{EDS} = (\mathscr{L}_S, \mathscr{L}_Q)$
$= (\bigstar, \text{qeq})$

Cache

EDX

$\Lambda_{EDX} = (\mathscr{L}_S, \mathscr{L}_Q)$
$= (\bigstar, \perp)$

<u>Query time</u>

Main Memory + Dummies

EDS

$\Lambda_{EDS} = (\mathscr{L}_S, \mathscr{L}_Q)$
$= (\bigstar, \text{qeq})$

Cache

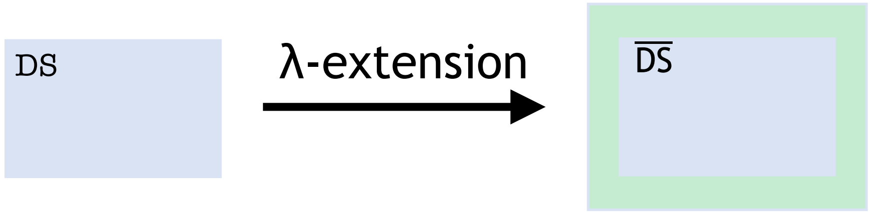
EDX

$\Lambda_{EDX} = (\mathscr{L}_S, \mathscr{L}_Q)$
$= (\bigstar, \perp)$

if item in cache
    get dummy
else
    get item

The Cache-Based Compiler

- EDS has to satisfy certain properties
  - has to be rebuildable
  - has to be "extendable" ≈ can store dummies



  - has to be "safe" ≈ handles dummies securely

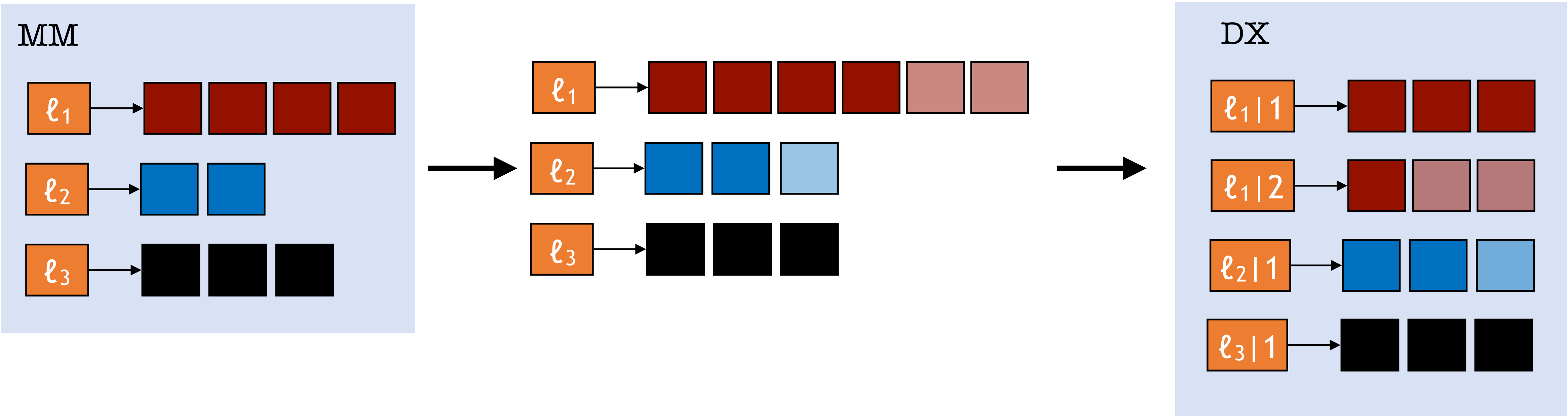
$$\mathscr{L}_S(\overline{DS}) \leq \mathscr{L}_S(DS) \qquad \mathscr{L}_Q(\overline{DS},q) \leq \mathscr{L}_S(DS,q)$$

  - has to have "small" non-repeating sub-pattern

The Piggy Back Scheme (PBS)

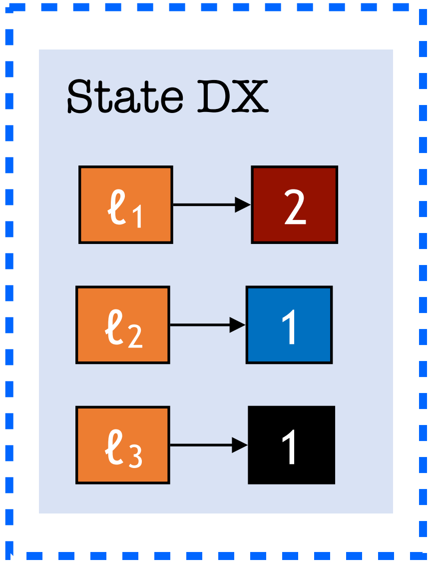
[K.-Moataz-Ohrimenko18]

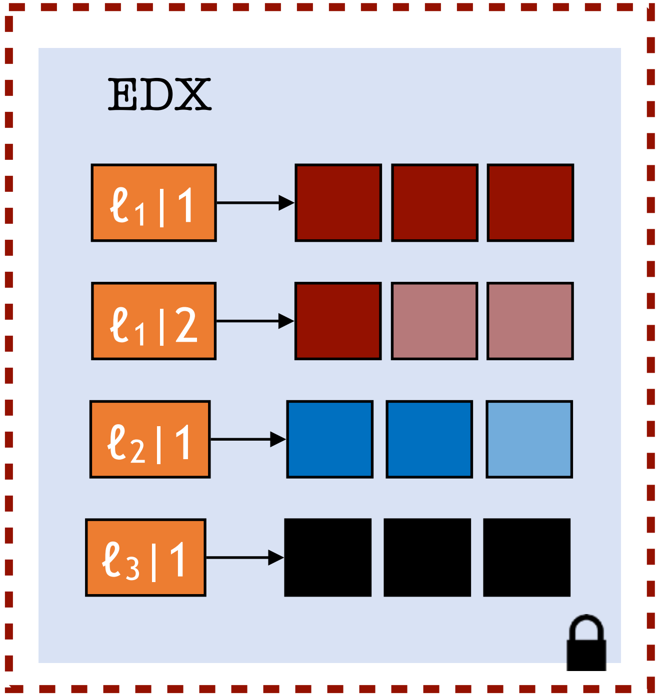
- Data structure transformation
  - pad tuples to multiple of **α** (e.g., **α** = 3)



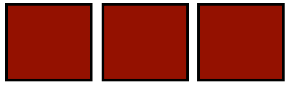
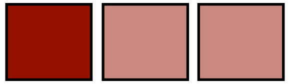
The Piggy Back Scheme (PBS)

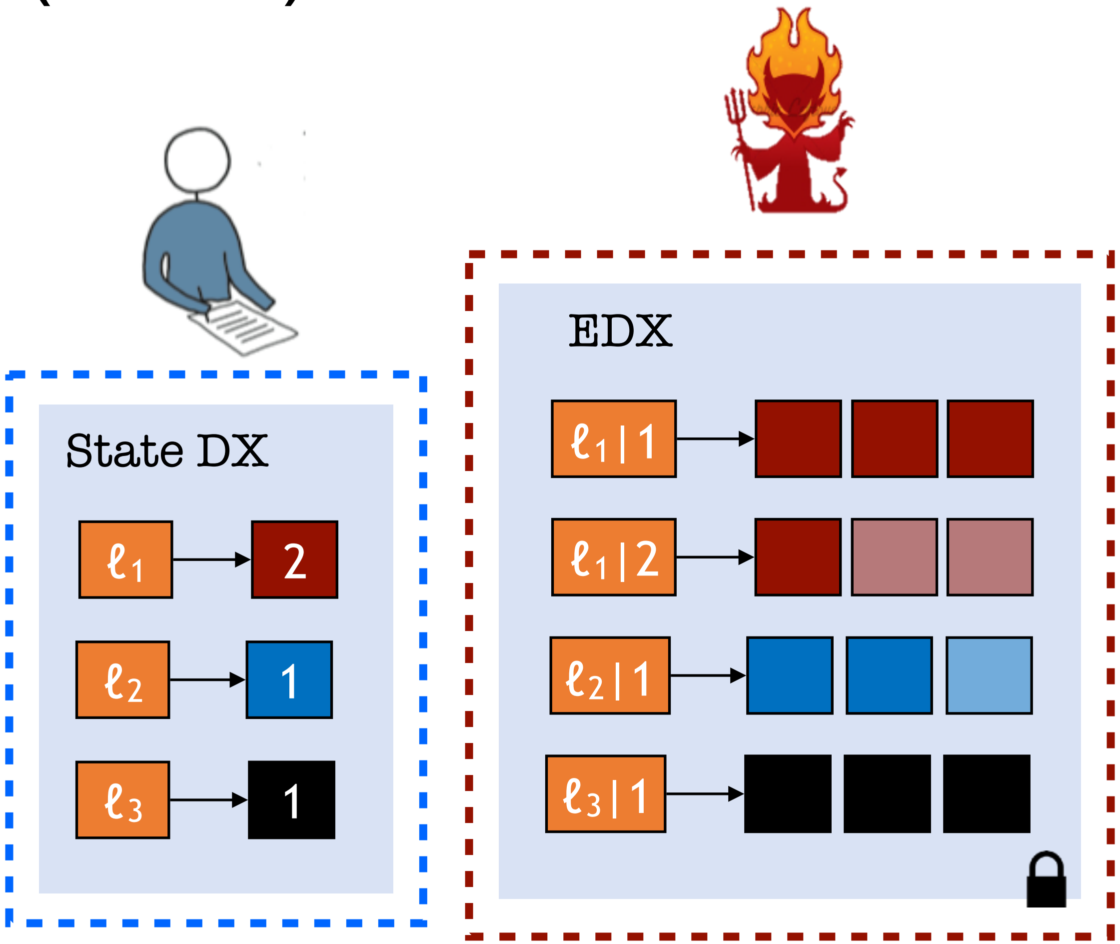
- PBS.Setup(1k, EMM = DX)
  - creates client state that maps labels to number of blocks
  - sends encrypted dictionary EDX to server





The Piggy Back Scheme (PBS)

- Consider sequence **(ℓ_1, ℓ_3, ℓ_2, …)**
- PBS.Get(**K**, **state**, **Q**, **ℓ_1**)
  - **2** := **DX[ℓ_1]**
  - Enqueue **$\ell_1|1$** and **$\ell_1|2$** on **Q**
  - **query** := **Q.dequeue()**
  - send **EDX.Token(K, query)**
  - client only gets back 
- PBS.Get(**K**, **state**, **Q**, **ℓ_3**)
  - …
  - client gets back 



The Piggy Back Scheme (PBS)

- PBS leverages a **new tradeoff**
  - *security vs. latency*
  - hides response length (volume) but response not immediate
- PBS has leakage profile
  - $\boldsymbol{\Lambda} = (\mathscr{L}_{\mathrm{S}}, \mathscr{L}_{\mathrm{Q}}) = (\bigstar, \textbf{rqeq}, \bigstar)$
    - where **rqeq** has non-repeating sub-pattern
      - $\perp$ on all but the last query
      - **srlen** on the last query

Latency Analysis of PBS

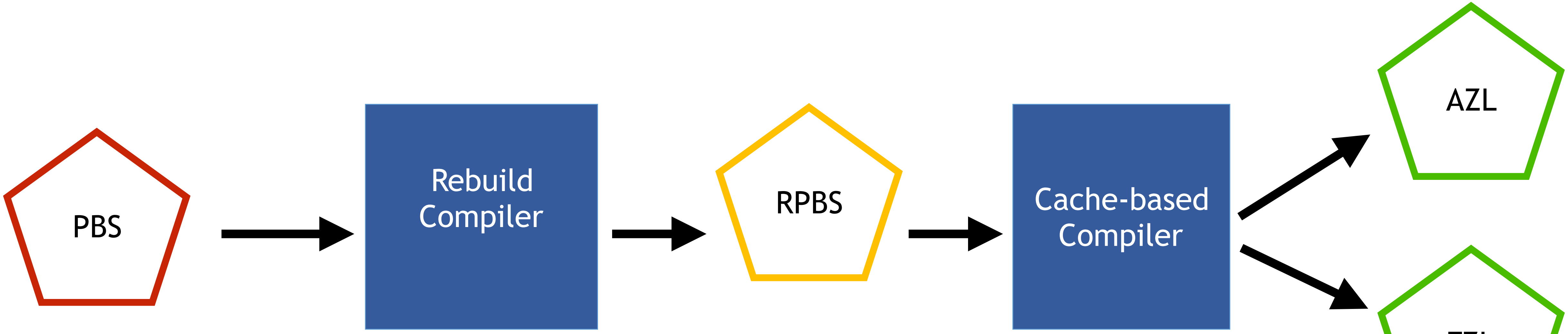
Thm: If queries and responses are Zipf distributed then under the inverted query hypothesis, latency is t + ε·t with probability at least

$$1 - \exp\left(-2t\left(\varepsilon \cdot \frac{\alpha}{\mu} \right)^2 \right)$$

Our Suppression Pipeline

[K.-Moataz-Ohrimenko18]

$\Lambda = (\mathscr{L}_S, \mathscr{L}_Q)$
$= (\bigstar, \text{srlen})$



$\Lambda = (\mathscr{L}_S, \mathscr{L}_Q, \mathscr{L}_A)$
$= (\bigstar, (\text{qeq}, \text{patt}), \text{patt}_A)$

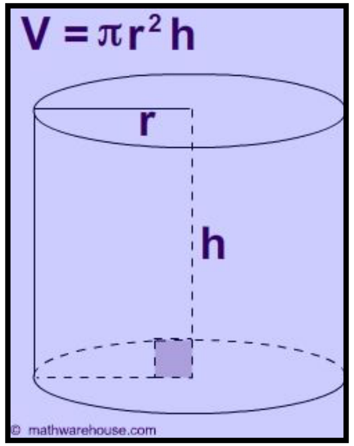
$\Lambda = (\mathscr{L}_S, \mathscr{L}_Q, \mathscr{L}_A)$
$= (\bigstar, (\text{qeq}, \text{patt}), \text{patt}_{Rb})$

$\Lambda = (\mathscr{L}_S, \mathscr{L}_Q)$
$= (\bigstar, \perp)$

$$\text{patt} = \begin{cases} \text{srlen} & \text{if queries are unique} \\ \text{misc} & \text{otherwise.} \end{cases}$$

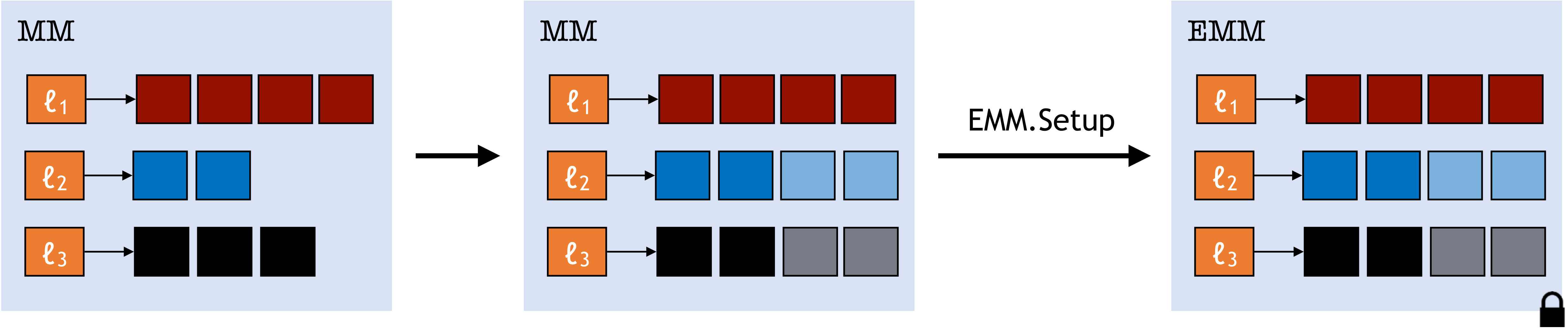
Q: Can we suppress other patterns besides the query equality?

The Volume Pattern



- Volume pattern is the size of a response
  - very common leakage pattern (even ORAM leaks it)
  - hard to suppress without blowup in storage
- [Kellaris-Kollios-Nissim-O'Neill16,…]
  - series of attacks vs. volume pattern of range queries

Suppressing Volume with Naive Padding



- Query complexity $O(\max_{\ell \in \mathbb{L}_{\mathsf{MM}}} \#\mathsf{MM}[\ell])$
- Storage complexity $O(\#\mathbb{L}_{\mathsf{MM}} \cdot \max_{\ell \in \mathbb{L}_{\mathsf{MM}}} \#\mathsf{MM}[\ell])$

Can we do better?

Computationally-Secure Leakage

Unbounded Adversary

vs.

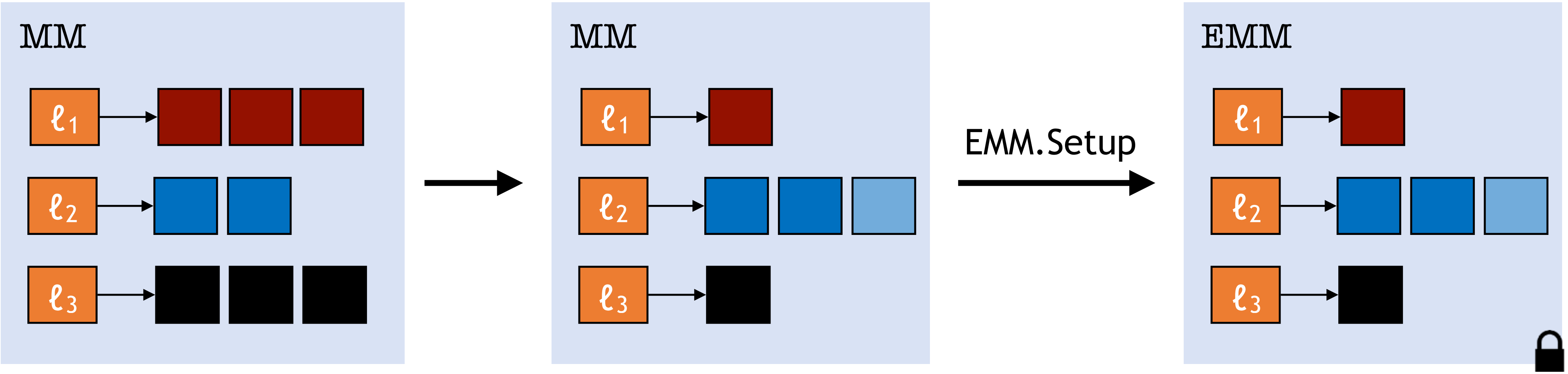
Bounded Adversary

Pseudo-Random Transform (PRT)

- Let $F:\{0,1\}^k \times \{0,1\}^* \longrightarrow \{0,1\}^{\log \mu}$ be a PRF
- Let $\lambda \geq 0$ be a parameter (min. response length)
- For each label ℓ in MM
    - compute $\mathsf{len}(\ell) = \lambda + F_K(\ell \mid \#\mathsf{MM}[\ell])$
    - if $\mathsf{len}(\ell) < \#\mathsf{MM}[\ell]$ truncate ℓ's tuple to length $\mathsf{len}(\ell)$
    - if $\mathsf{len}(\ell) > \#\mathsf{MM}[\ell]$ pad ℓ's tuple to length $\mathsf{len}(\ell)$

Pseudo-Random Transform (PRT)

- Example with $\lambda = 1$ and $\mu = 3$



$\lambda + F_K(\ell_1 | 4) = 1 + 0 = 1$

$\lambda + F_K(\ell_2 | 2) = 1 + 2 = 3$

$\lambda + F_K(\ell_3 | 3) = 1 + 1 = 1$

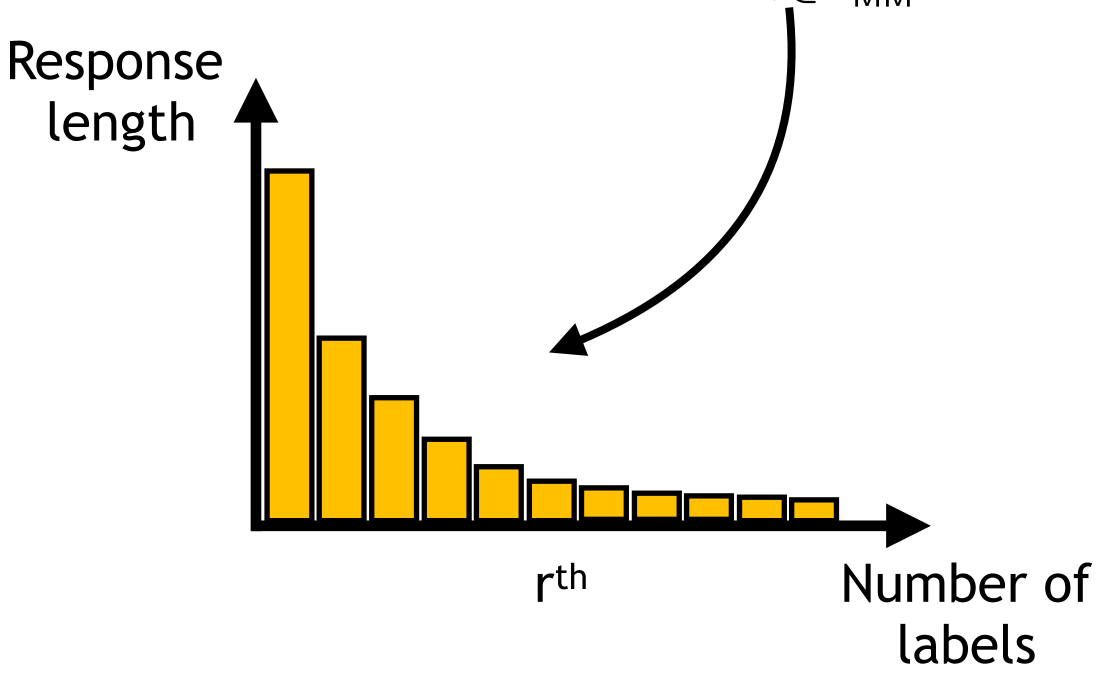
Pseudo-Random Transform (PRT)

- PRT is a “lossy” transformation
- PRT exploits a **new tradeoff**
  - lossiness vs. security
- Volume hiding relies on pseudo-randomness of F
- Need to analyze
  - Number of truncations
  - Storage overhead

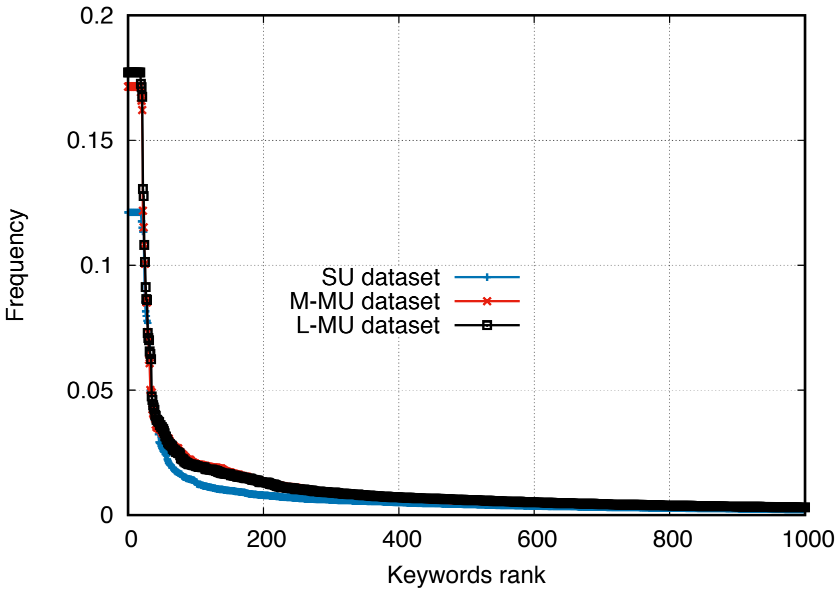
Zipf-Distributed Multi-Maps

- A MM is Zipf-distributed if the rth tuple has length

$$\frac{1}{r \cdot H_{\#\mathbb{L}_{\mathsf{MM}},1}} \cdot \sum_{\ell \in \mathbb{L}_{\mathsf{MM}}} \#\mathsf{MM}[\ell]$$



Enron dataset



Pseudo-Random Transform (PRT)

Thm: Let 1/2 < **α** < 1. If MM is Zipf-distributed, then MM' has size at most

$$\alpha \cdot \#\mathbb{L} \cdot \max_{\ell \in \mathbb{L}} \#\mathsf{MM}[\ell]$$

with probability at least $1 - \exp\Big(- \#\mathbb{L} \cdot (2\alpha - 1)^2 / 8 \Big)$.

Furthermore, it incurs at most

$$\frac{1}{\log(\#\mathbb{L})} \cdot \#\mathbb{L}$$

truncations with probability at least $1 - \exp\Big(- 2 \cdot \#\mathbb{L} \cdot \log^2(\#\mathbb{L}) \Big)$.

Pseudo-Random Transform (PRT)

- PRT has many advantages
  - easy to use and implement 😀
  - doesn't impact query and storage complexity too much 😀
- But it is is lossy 😞
  - for keyword search one can rank results
  - so only low-ranked results are lost

Q: Can we design a non-lossy transformation?

Densest Subgraph Transform

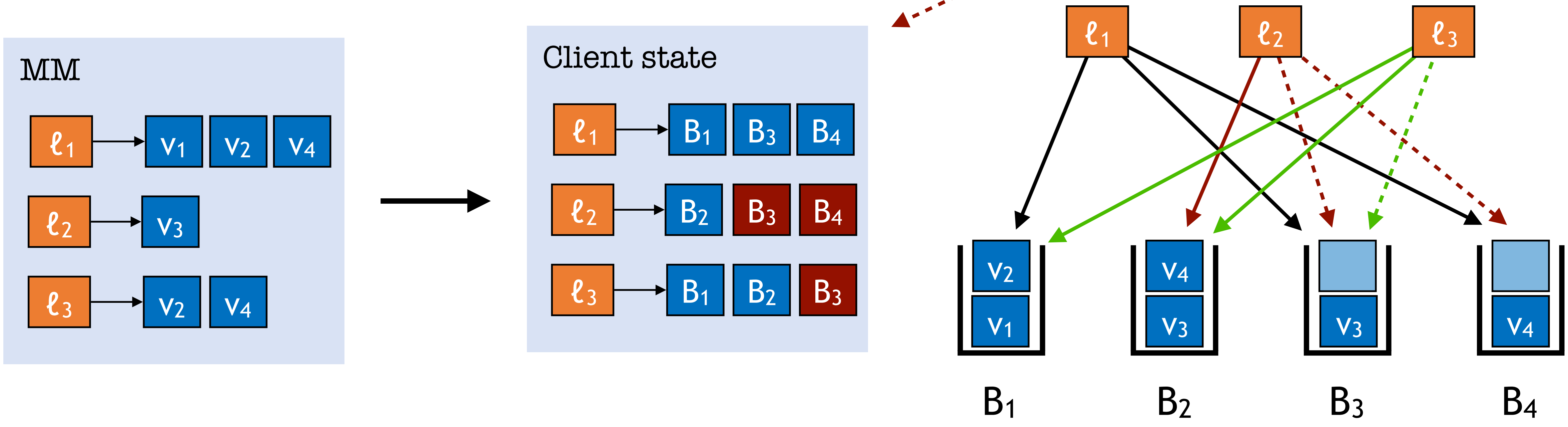
[K.-Moataz19]

- Data structure transformation
  - hides volume 😀
  - query complexity ≈ query complexity of naive padding ☹️
  - storage complexity ≤ storage complexity of naive padding 😀
  - non-lossy 😀
- How is this possible?
  - New EMM design framework
  - Computational assumptions from average-case complexity

Densest Subgraph Transform
[K.-Moataz19]

- For each label pick $\mu = \max_{\ell} \#MM[\ell]$ bins at random
  - store values in bins
  - if $\#MM[\ell] < \mu$ don't store anything in remaining bins
- Pad all bins

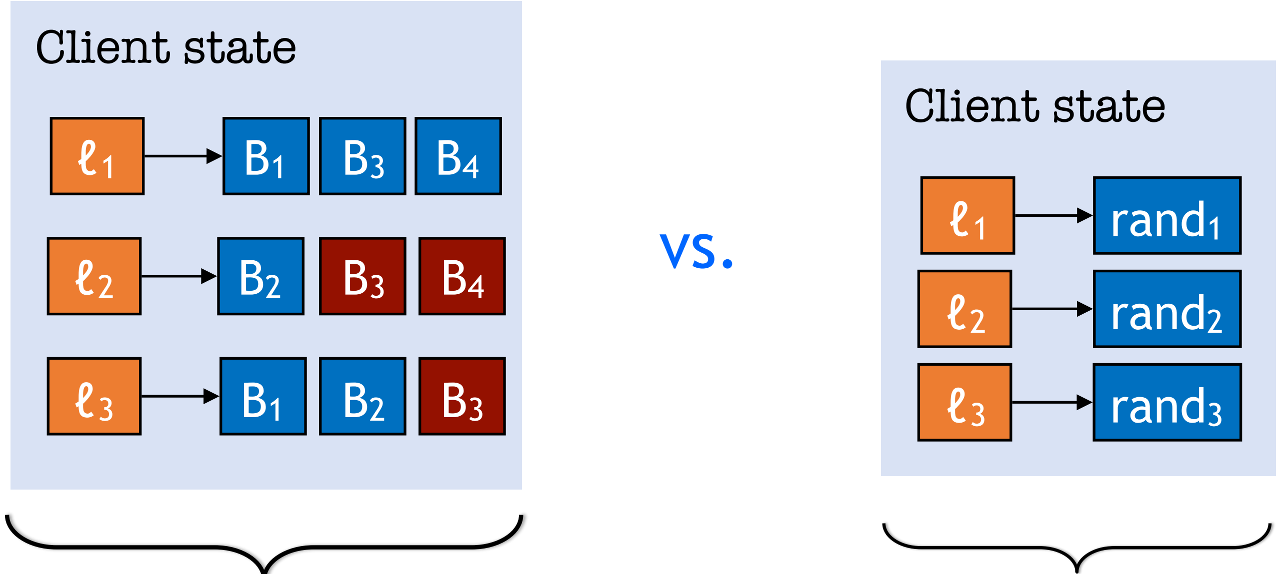
Size of client state ≈ size of MM



Densest Subgraph Transform

[K.-Moataz19]

- Compressing the state
  - instead of choosing edges/bins uniformly at random
  - use a PRF and store key/rand value in state



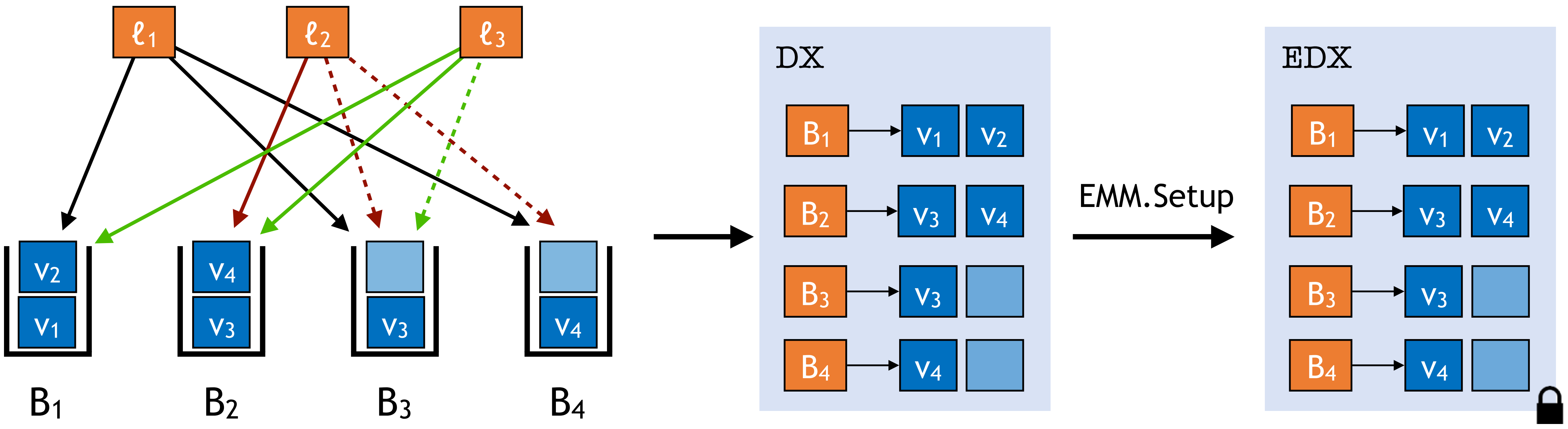
$$O(\#\mathbb{L} \cdot \max_{\ell \in \mathbb{L}} \#\mathrm{MM}[\ell])$$

$$O(\#\mathbb{L})$$

Some PRF seeds can lead to collisions so just pick again until no collisions

Densest Subgraph Transform
[K.-Moataz19]

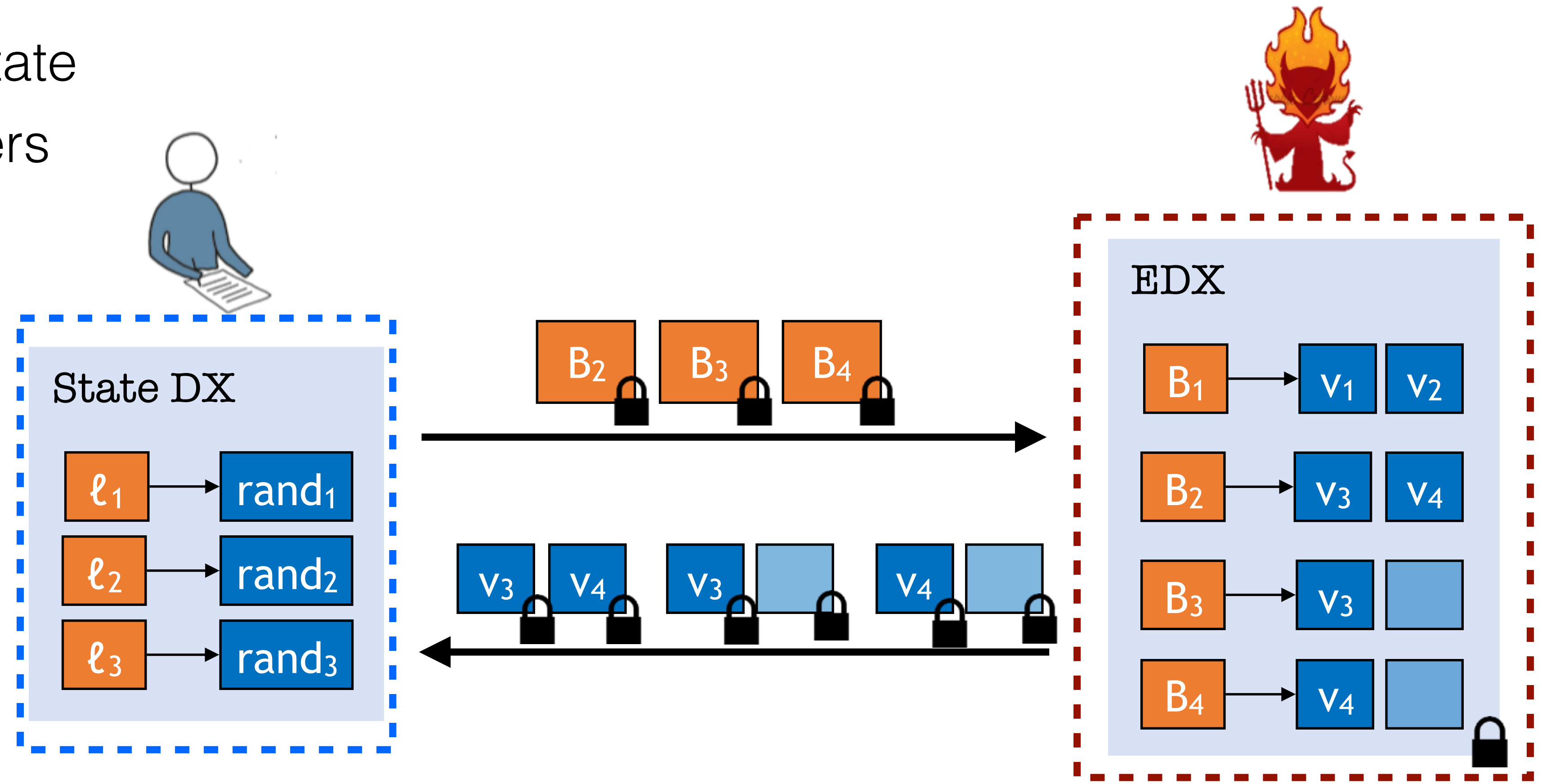
- Store bins in a dictionary DX and encrypt DX



Densest Subgraph Transform

[K.-Moataz19]

- To get ℓ_2,
  - retrieve $rand_2$ from state
  - compute bin identifiers
    - 2:= $F(rand_2, 1)$,
    - 3:= $F(rand_2, 2)$,
    - 4:= $F(rand_3, 3)$
  - retrieve bins



Densest Subgraph Transform
[K.-Moataz19]

Thm: The load of a bin is at most

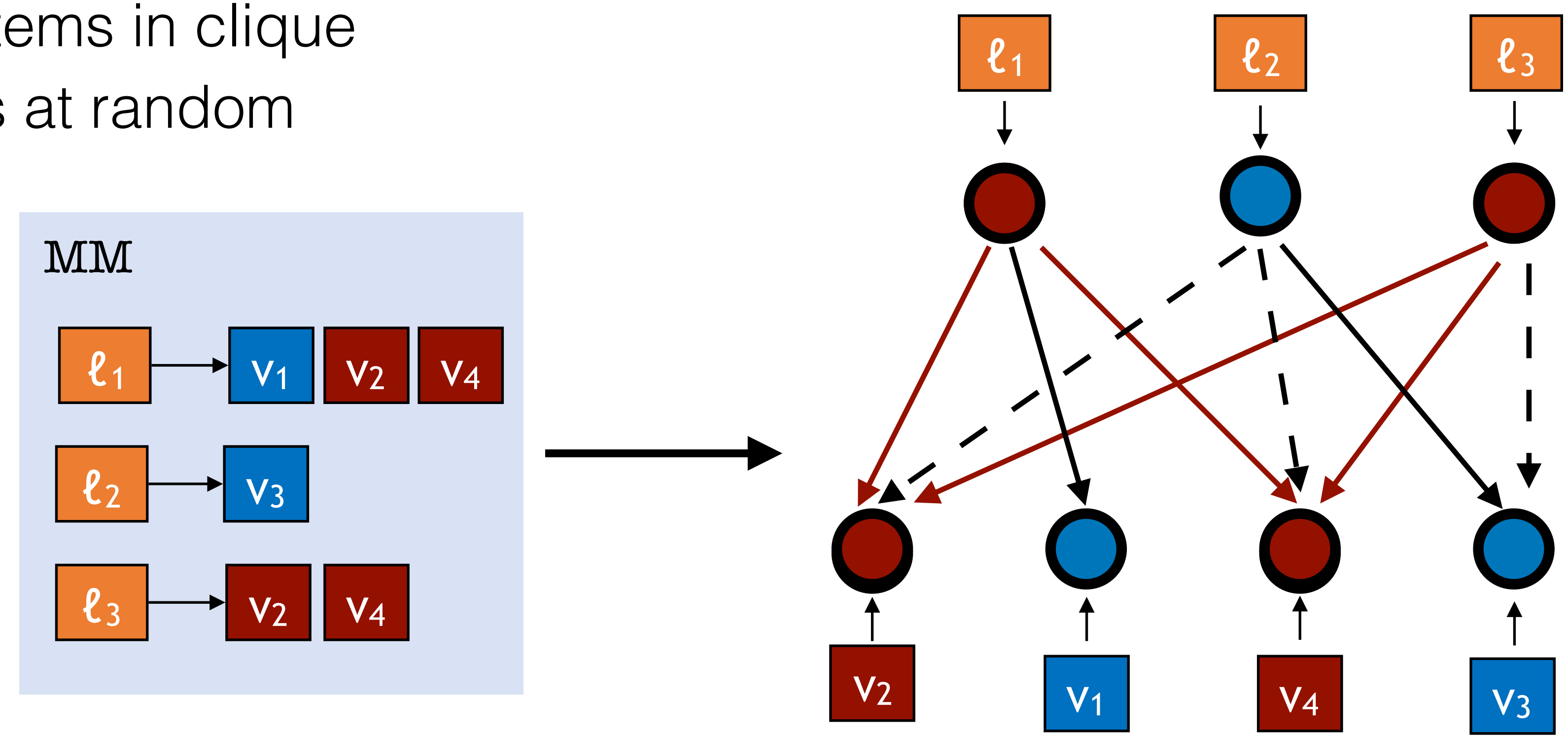
$$\frac{N}{n} + \frac{\ln(1/\varepsilon)}{3}\left(1 + \sqrt{1 + \frac{18N}{n \cdot \ln(1/\varepsilon)}}\right)$$

with probability at least 1 - ε, where $N = \sum_{\ell \in \mathbb{L}_{\mathsf{MM}}} \#\mathsf{MM}[\ell]$

Densest Subgraph Transform

[K.-Moataz19]

- Alternative construction for concentrated MMs
  - **v_2** and **v_4** are duplicated so store them only once
  - Pick bi-partite clique at random
    - store duplicated items in clique
  - Pick remaining edges at random



Densest Subgraph Transform

[K.-Moataz19]

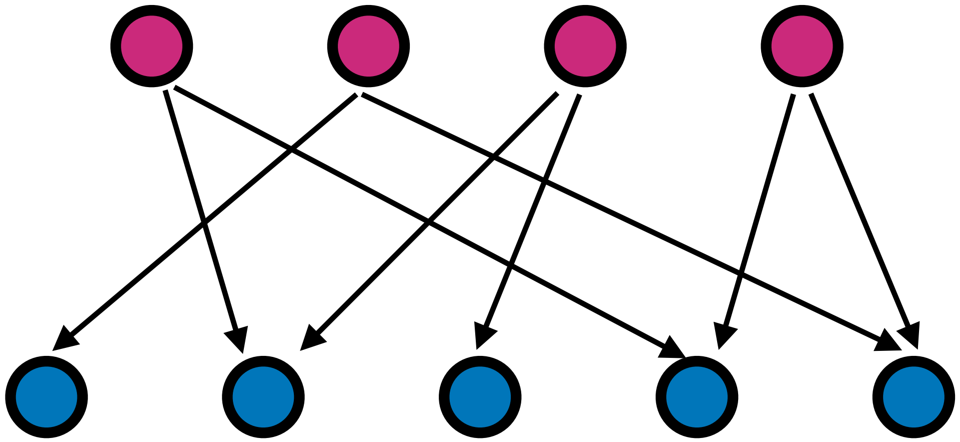
Thm: The load of a bin is at most

$$\frac{N - N_{\mathsf{DS}}}{n} + \frac{\ln(1/\varepsilon)}{3}\left(1 + \sqrt{1 + \frac{18(N - N_{\mathsf{DS}})}{n \cdot \ln(1/\varepsilon)}}\right)$$

with probability at least 1 - ε, where N_{DS} is the size of concentrated part

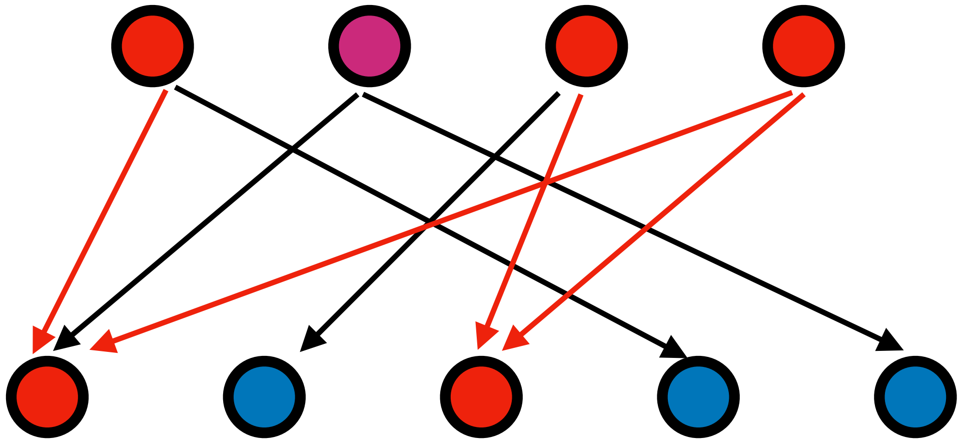
Densest Subgraph Assumption

[Applebaum-Barak-Wigderson10]



Erdös-Rényi graph

$\simeq$



Erdös-Rényi graph with planted dense subgraph

Densest Subgraph Assumption
[Applebaum-Barak-Wigderson10]

- Variant of the planted clique problem
  - central problem in average-case hardness
- Evidence for hardness
  - studied since the mid-70's in CS & statistical physics
  - failure of powerful algorithmic techniques
  - restricted lower bounds
    - Sum-of-squares
    - Statistical query

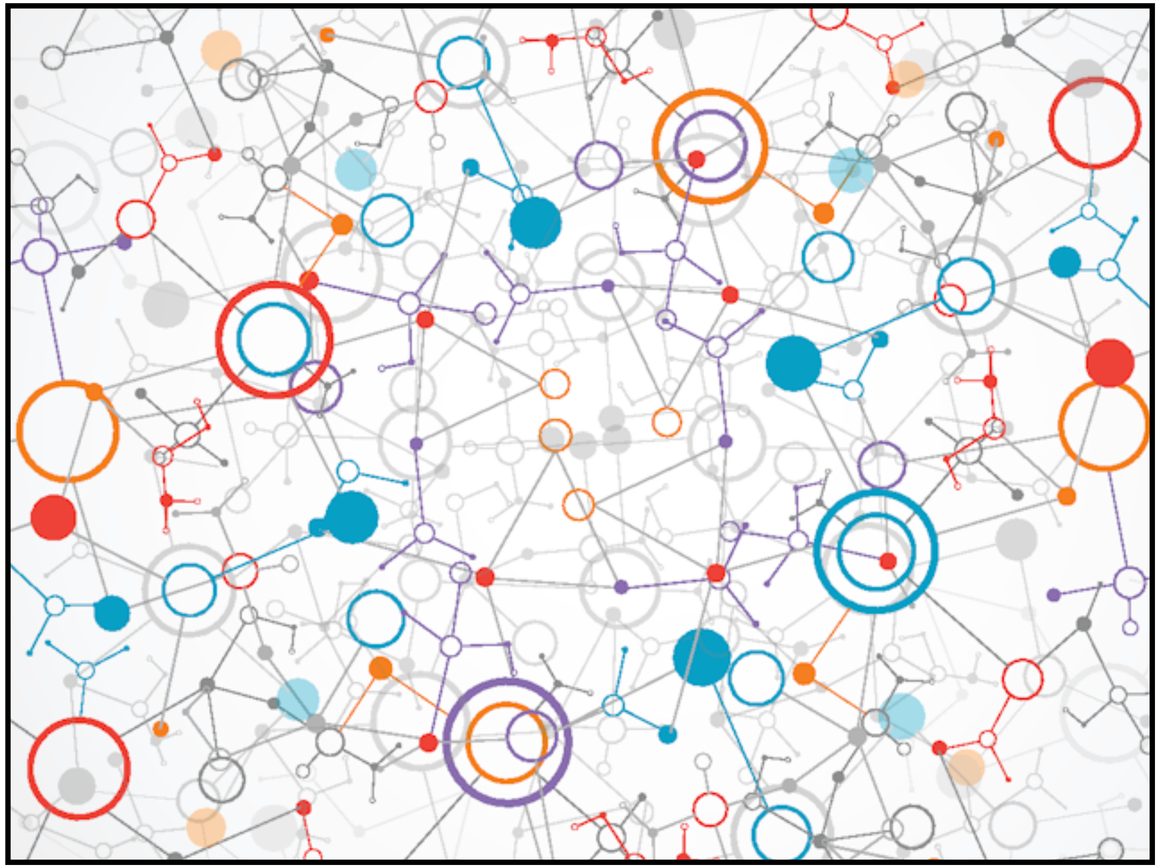
Conclusions

Encrypted Search: 2000-2019

- A large and vibrant area of research
- Many interesting and hard problems
- Many fundamental questions
  - how do we model leakage?
  - how do we quantify leakage?
  - how do we suppress leakage?
  - are the tradeoffs we observe inherent? (i..e, lower bounds)

Encrypted Search: 2000-2019

- Many connections
  - algorithms & data structures
  - database theory & systems
  - statistical learning theory
  - optimization
  - graph theory
  - distributed systems



Encrypted Search: 2000-2019

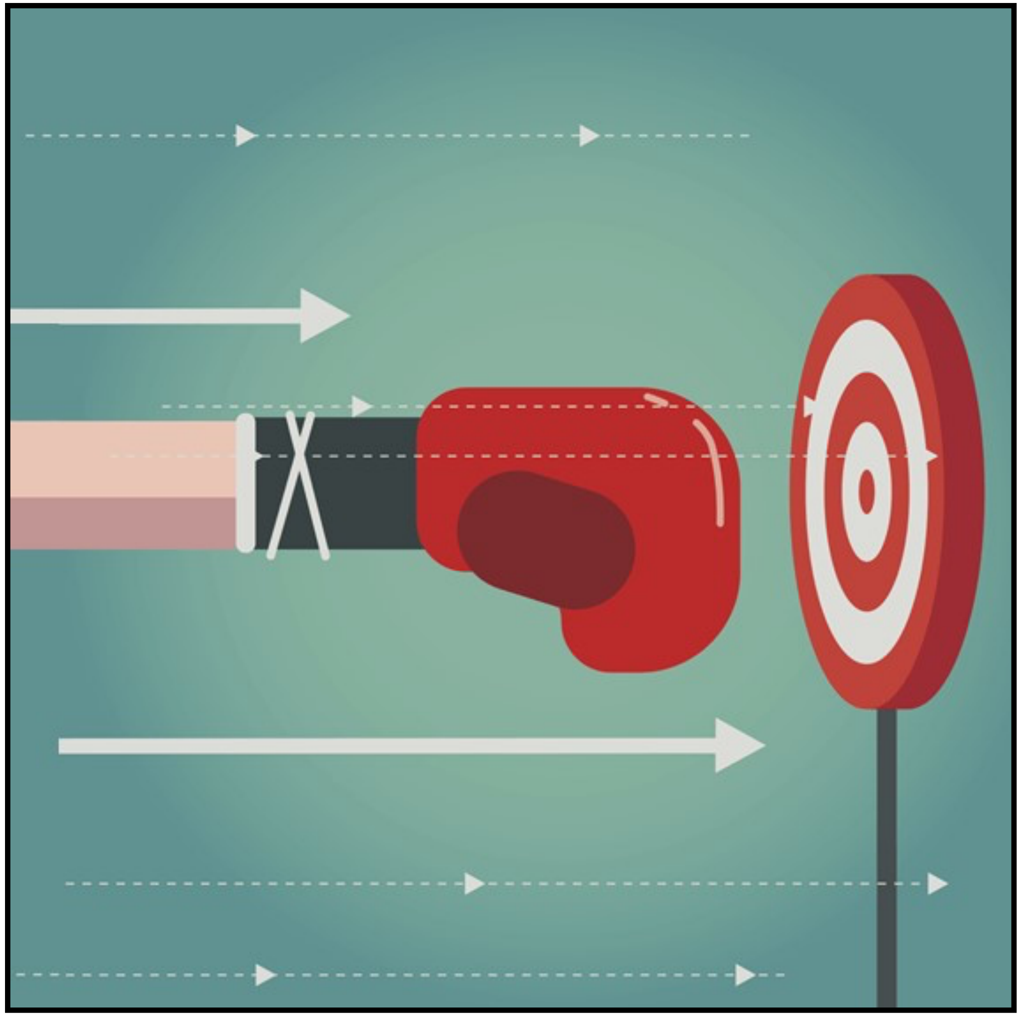
- Many interesting leakage attacks to study
- But many new techniques to bypass leakage attacks
  - padding & clustering techniques [Bost-Fouque17]
  - response-hiding schemes [Blackstone-K.-Moataz19]
  - suppression compilers [K.-Moataz-Ohrimenko18]
  - suppression transforms [K.-Moataz19]
  - worst-case vs. average-case leakage [Agarwal-K.19]
  - distributing data [Agarwal-K.19]

Encrypted Search: 2000-2019

- New tradeoffs to explore
  - leakage vs. correctness [K.-Moataz19]
  - leakage vs. latency [K.-Moataz-Ohrimenko18]

Encrypted Search: 2000-2019

- Real-world impact
  - Microsoft SQL Server
  - MongoDB Field Level Encryption
  - Cisco WebEx
  - Ionic
  - more coming…



Thanks to…

Archita Agarwal

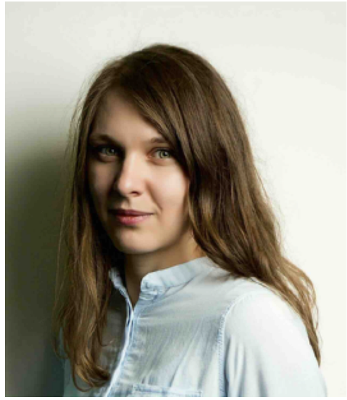
Ghous Amjad

Hajar Alturki

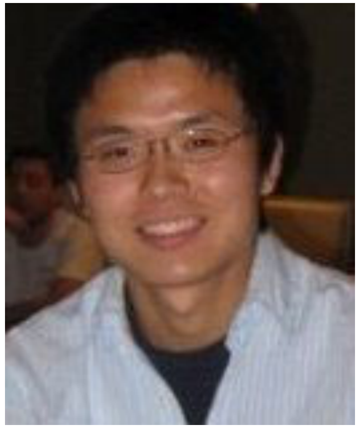
Laura Blackstone

Marilyn George

Tarik Moataz



Olya Ohrimenko



Sam Zhao

The End