H_2O_2RAM

A High-Performance Hierarchical Doubly Oblivious RAM

Leqian Zheng[1], Zheng Zhang[2], Wentao Dong[1], Yao Zhang[2], Ye Wu[2], Cong Wang[1]

[1]City University of Hong Kong, [2]ByteDance Inc.

2025 Aug 15

Access patterns matter in encrypted systems

- Access patterns (which encrypted data are accessed) $\xrightarrow{\text{reveal}}$ sensitive information (about encrypted data)

A toy example

```
if (secret == true) {
    cnt++;
}
else { // do nothing
}
```

Real-world applications

- Searchable encryption
- Private contact discovery for Signal
- Anonymizing Google's Key transparency
- ...

How access patterns pose privacy risks

$$\text{input data } \mathcal{X} \xrightarrow{\Pi_{\mathcal{P}}.\text{exec}(\mathcal{X})} \text{access pattern } \mathcal{AP}_{\mathcal{X}}$$

$$\text{input data } \mathcal{Y} \xrightarrow{\Pi_{\mathcal{P}}.\text{exec}(\mathcal{Y})} \text{access pattern } \mathcal{AP}_{\mathcal{Y}}$$

How access patterns pose privacy risks

$$\begin{tikzcd}[column sep=huge] \text{input data } \mathcal{X} \arrow[r, "\Pi_{\mathcal{P}}.\mathrm{exec}(\mathcal{X})"] & \text{access pattern } \mathcal{AP}_{\mathcal{X}} \arrow[l, dashed, bend right, "\textbf{infer}"'] \end{tikzcd}$$

$$\begin{tikzcd}[column sep=huge] \text{input data } \mathcal{Y} \arrow[r, "\Pi_{\mathcal{P}}.\mathrm{exec}(\mathcal{Y})"] & \text{access pattern } \mathcal{AP}_{\mathcal{Y}} \arrow[l, dashed, bend left, "\textbf{infer}"] \end{tikzcd}$$

General solution: Oblivious RAM

- Oblivious RAM (ORAM):

$$\text{input data } \mathcal{X}/\mathcal{Y} \xrightarrow{\Pi_{\mathcal{P}}} \text{(statistically/computationally) } \textbf{same} \text{ access pattern } \mathcal{AP}$$

$$(|\mathcal{X}| = |\mathcal{Y}|)$$

General solution: Oblivious RAM

- Oblivious RAM (ORAM):

input data $\mathcal{X}/\mathcal{Y}$ $\xrightarrow{\Pi_{\mathcal{P}}}$ (statistically/computationally) **same** access pattern $\mathcal{AP}$

$(|\mathcal{X}| = |\mathcal{Y}|)$

- Technique roadmap: tree-based designs (better practical performance) and hierarchical designs (more of theoretical interests)

General solution: Oblivious RAM

- Oblivious RAM (ORAM):

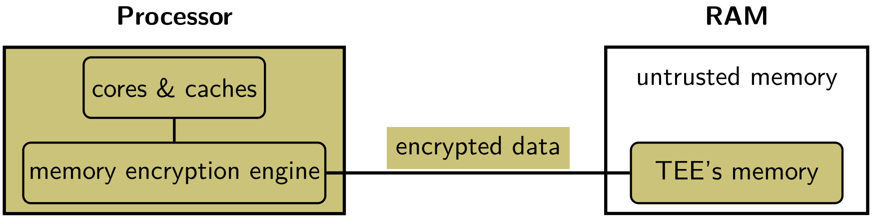
input data $\mathcal{X}/\mathcal{Y}$ $\xrightarrow{\Pi_{\mathcal{P}}}$ (statistically/computationally) **same** access pattern $\mathcal{AP}$

$(|\mathcal{X}| = |\mathcal{Y}|)$

- Technique roadmap: tree-based designs (better practical performance) and hierarchical designs (more of theoretical interests)
- The **de facto standard** for concealing access patterns across diverse scenarios and settings

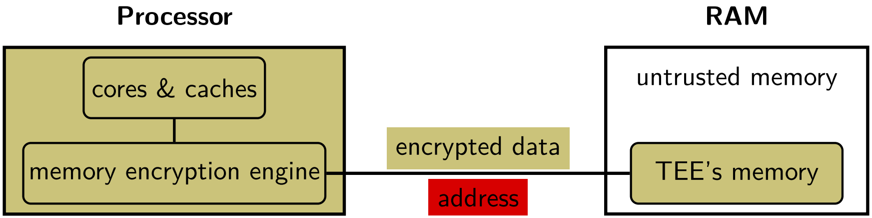
Focused setting: Conceal memory access patterns in TEEs

- Trusted execution environments (TEEs) offer compelling security guarantees with favorable performance and usability 😃



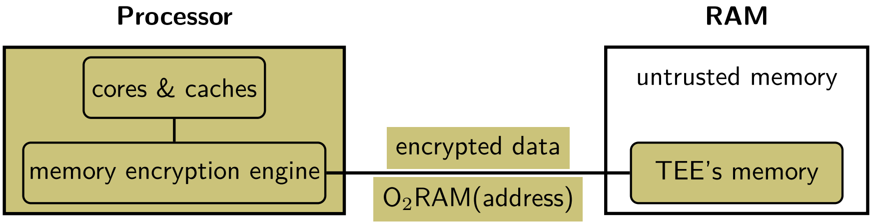
Focused setting: Conceal memory access patterns in TEEs

- Trusted execution environments (TEEs) offer compelling security guarantees with favorable performance and usability 😃
- Memory access patterns remain **exposed by design** in mainstream hardware-based TEEs ☹️



Focused setting: Conceal memory access patterns in TEEs

- Trusted execution environments (TEEs) offer compelling security guarantees with favorable performance and usability 😃
- Memory access patterns remain **exposed by design** in mainstream hardware-based TEEs ☹️



- General solution for the focused setting: **doubly** oblivious RAM (O_2RAM)

Motivation

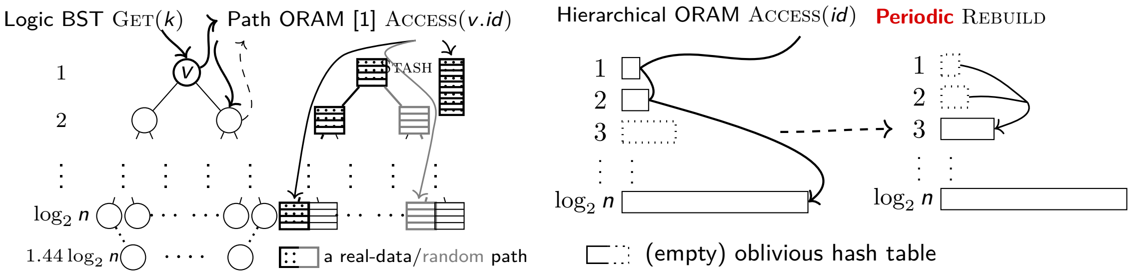
High performance overheads in existing O_2RAM designs:

- Find on a 2^{20}-element map: nanosecs/10^{-9}s (plain) $\longleftrightarrow$ millisecs/10^{-3}s (oblivious)
- Dijkstra on a graph ($|V| \approx 2^{16}$, $|E| \approx 2^{18}$): millisecs/10^{-3}s (plain) $\longleftrightarrow$ ~ 9 hours (oblivious)

Amplified by millions — or more!

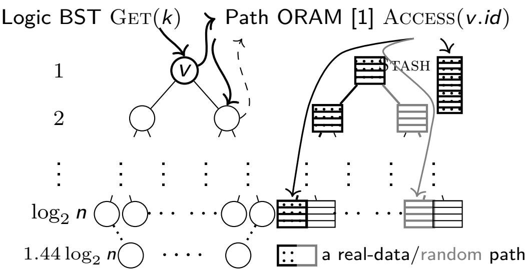
Key insight

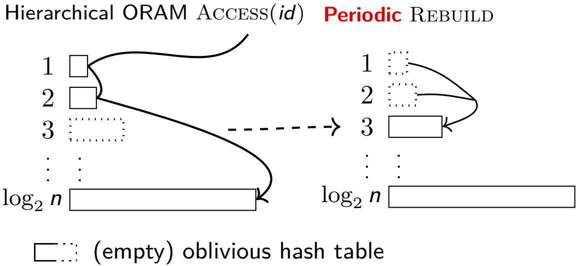
🤔 $\textsc{Access}(id) \xrightarrow{\text{handle}}$ (probably amortized) 1000 data blocks in both approaches?



Key insight

🤔 $\textsc{Access}(id) \xrightarrow{\text{handle}}$ (probably amortized) 1000 data blocks in both approaches?

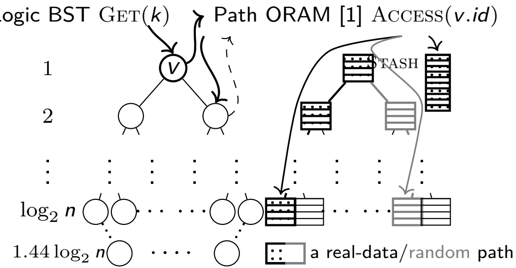


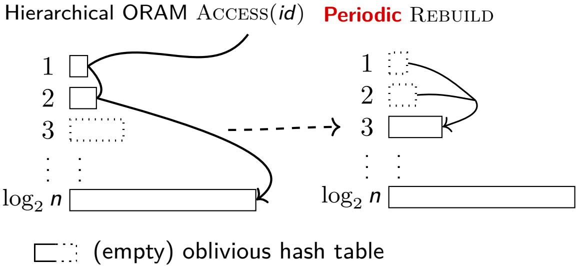


- Better data locality
- Easier to parallelize

Key insight

🧐 Comparable complexity $\Longrightarrow$ HORAM performs better!

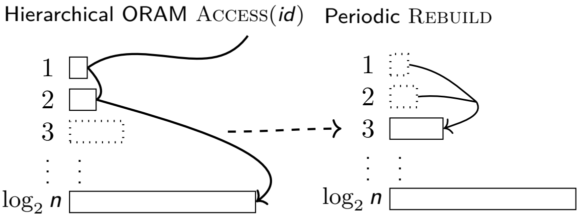




- **Better data locality**
- **Easier to parallelize**

H_2O_2RAM: Technical challenge

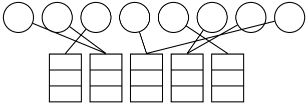
- Design high-performance oblivious hash tables



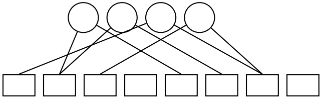
(empty) **oblivious hash table**

H_2O_2RAM: Our approach

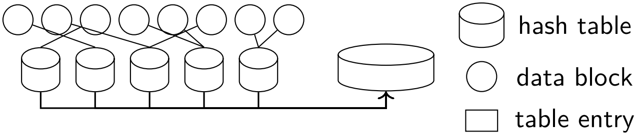
- Design high-performance oblivious hash tables



(a) Bucket hash



(b) Stashless Cuckoo hash



(c) Two-tier hash, where a secret number of blocks from the main hash table will be (obliviously) relocated to a secondary hash table

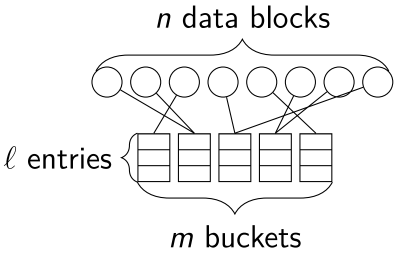
H_2O_2RAM: Core intuitions

- Compute “tight”/appropriate parameters for oblivious hash tables (OHT)

Lifecycle of an OHT (capacity n): 1 **rebuild** $+$ n **lookups**

H_2O_2RAM: Core intuitions

- Compute "tight"/appropriate parameters for oblivious hash tables (OHT)

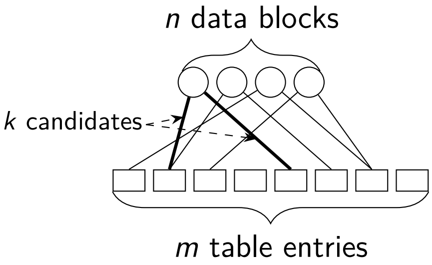


(a) Bucket hash

- each block $\rightarrow$ a bucket via a PRF
- ℓ entries: ensure a negligible probability of bucket overflow
- lookup: linearly scan a bucket (for obliviousness)
- $m\uparrow$: $\ell\downarrow$ lower per-lookup cost, table size $(m \cdot \ell)\uparrow$ higher rebuild cost
- m^*: balance lookup vs. rebuild costs $\rightarrow$ minimal overall cost

H_2O_2RAM: Core intuitions

- Compute "tight"/appropriate parameters for oblivious hash tables (OHT)

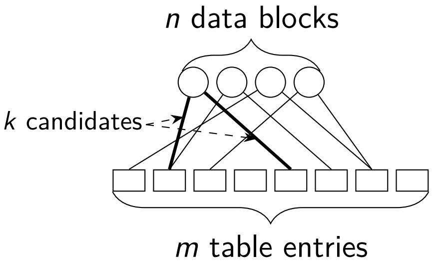


(b) Stashless Cuckoo hash

- each block $\rightarrow$ k candidate entries via k PRFs
- $k = 2$ w/ a stash handling unplaceable blocks
  ($\Omega(\log n)$ for negl overflow prob)

H_2O_2RAM: Core intuitions

- Compute "tight"/appropriate parameters for oblivious hash tables (OHT)

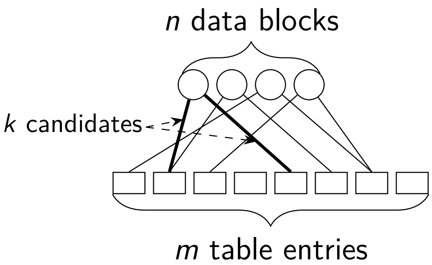


(b) Stashless Cuckoo hash

- each block $\rightarrow$ k candidate entries via k PRFs
- $k =?$ ~~w/o stash~~ + negl overflow prob

H_2O_2RAM: Core intuitions

- Compute "tight"/appropriate parameters for oblivious hash tables (OHT)

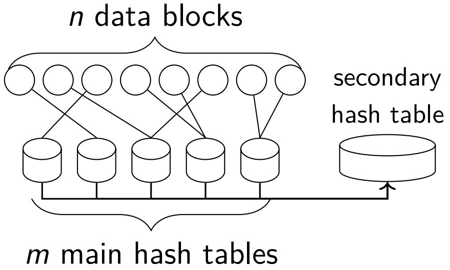


(b) Stashless Cuckoo hash

- each block $\rightarrow$ k candidate entries via k PRFs
- $k \in [4, 6]$ suffices (Crypto 23 [3] + our numeric analysis)
- construction: oblivious bipartite matching (potentially of independent interest)

H_2O_2RAM: Core intuitions

- Compute “tight”/appropriate parameters for oblivious hash tables (OHT)



(c) Two-tier hash

- each block $\rightarrow$ a main HT via a PRF, *nonobliviously*
- ϵ_i (secret) of i^{th} main HT $\rightarrow$ secondary HT, *obliviously*
- to be optimized:
  - #main HTs m
  - sampling parameters for ϵ_i
  - underlying hashing schemes

H_2O_2RAM: Core intuitions

- Compute "tight"/appropriate parameters for oblivious hash tables (OHT)
- Select suitable hash schemes for each H_2O_2RAM level

hash scheme	rebuild time[‡]	lookup time	suitable scenarios
linear scan	$\mathcal{O}(n)$	$\mathcal{O}(n)$	very small n
bucket hash	$\mathcal{O}\left(n \log^2 n\right)$	$\mathcal{O}(\ell)$	small to moderate n
stashless Cuckoo hash	$\mathcal{O}\left(n \log^3 n\right)$	$\mathcal{O}(1)$	$n < t$, secondary table for two-tier hash
two-tier hash	$\mathcal{O}\left(n \log^2 \log n\right)$	$\mathcal{O}(\ell)$	large n

† n: hash table size, ℓ: bucket size, t: the number of lookups performed during its lifetime.
‡ Adopting an $\mathcal{O}(n \log n)$ osort algorithm [2] saves a $\log n$ factor for this column.

H_2O_2RAM: Overall complexity

- Asymptotic comparison of existing O_2RAM schemes, N denotes the capacity

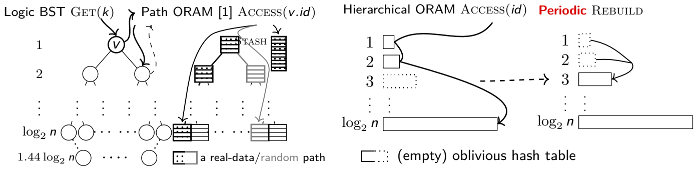
Scheme	Amortized access time
Oblix [4] & GraphOS [5]	$\mathcal{O}\left(\log^3 N\right)$
EnigMap [6]	$\tilde{O}(\log^2 N)$
H_2O_2RAM	$\tilde{O}(\log^2 N)$

H_2O_2RAM: Overall complexity

- Asymptoti … capacity

Key insight

🧐 Comparable complexity $\Longrightarrow$ HORAM performs better!

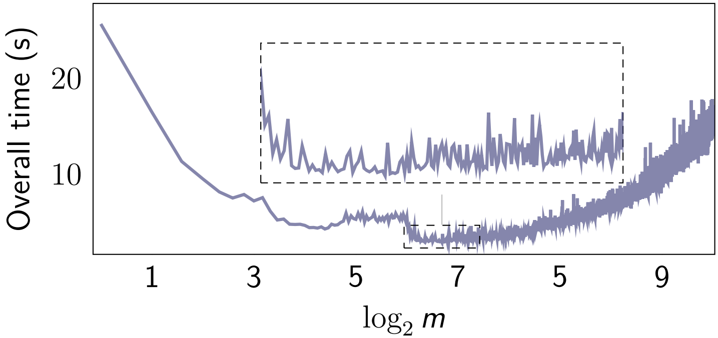


- **Better data locality**
- **Easier to parallelize**

Comparable complexity achieved!

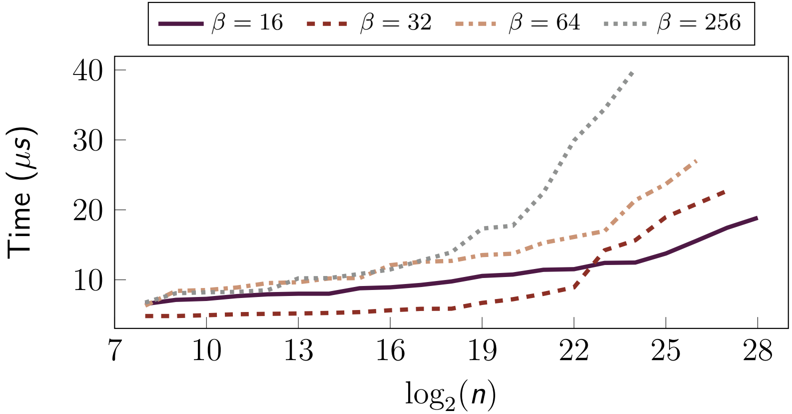
Evaluations: Optimizing OHT parameters

- Overall running time (1 rebuild + n lookups) for a bucket OHT of capacity 8192



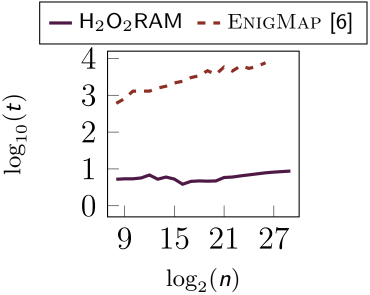
Evaluations: Latency under different settings

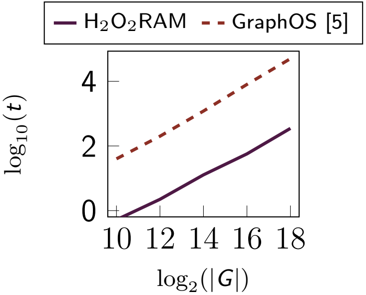
- Amortized access time of H_2O_2RAM with different input data sizes n and data block sizes β



Evaluations: Comparative results

- Get operation time t (in microseconds) of a map, where n denotes its capacity
- Single-source shortest path computation time t (in seconds). $|G| = |E| + |V|$ with $|E| = 4|V|$. The results for GraphOS [5] are taken from the "Gr-V0.13E" line in Figure 6c of their paper.





$\sim 10^3$ speedup!

References

[1] Emil Stefanov et al. "Path ORAM: An extremely simple oblivious RAM protocol". In: **JACM** 65.4 (2018), pp. 1–26.

[2] Sajin Sasy, Aaron Johnson, and Ian Goldberg. "Waks-On/Waks-Off: Fast Oblivious Offline/Online Shuffling and Sorting with Waksman Networks". In: **CCS**. 2023.

[3] Kevin Yeo. "Cuckoo hashing in cryptography: Optimal parameters, robustness and applications". In: **CRYPTO**. 2023.

[4] Pratyush Mishra et al. "Oblix: An efficient oblivious search index". In: **S&P**. 2018.

[5] Javad Ghareh Chamani et al. "GraphOS: Towards Oblivious Graph Processing". In: **VLDB**. 2023.

[6] Afonso Tinoco, Sixiang Gao, and Elaine Shi. "EnigMap: External-Memory Oblivious Map for Secure Enclaves". In: **USENIX Security**. 2023.

Thanks for your attention!

Contact:

leqian.zheng@my.cityu.edu.hk