PanORAMa: Oblivious RAM with Logarithmic Overhead

Sarvar Patel, Giuseppe Persiano, Mariana Raykova, Kevin Yeo
Bob knows what files I am accessing.
Bandwidth is expensive

Retrieve the whole database
Oblivious RAM [GO’96]

Access pattern hiding

polylogarithmic overhead (amortized)
How Efficient Can an ORAM Construction be?
ORAM Lower Bound
ORAM Lower Bound

- Goldreich-Ostrovsky’96
  - Lower bound $O(\log_C N)$ blocks for database of $N$ blocks and client memory of $C$ blocks
ORAM Lower Bound

- Goldreich-Ostrovsky’96
  - Lower bound $O(\log_c N)$ blocks for database of $N$ blocks and client memory of $C$ blocks
  - Caveats:
    - Server only moves and retrieves blocks (does not do any computation)
    - Constructions with statistical security
    - Constructions that work for any block size
    - The client can have oracle access to a private random function
ORAM Lower Bound

- Goldreich-Ostrovsky’96
  - Lower bound $\Omega(\log_c N)$ blocks for database of $N$ blocks and client memory of $C$ blocks
  - Caveats:
    - Server only moves and retrieves blocks (does not do any computation)
    - Constructions with statistical security
    - Constructions that work for any block size
    - The client can have oracle access to a private random function

- Boyle-Naor’16
  - Formalized the “balls and bins” model
  - Evidence for the hardness of extending the lower bound beyond the ball and bins model
    - Reduction from sorting circuits to ORAM
ORAM Lower Bound

- Goldreich-Ostrovsky’96
  - Lower bound $\Omega(\log_c N)$ blocks for database of $N$ blocks and client memory of $C$ blocks
  - Caveats:
    - Server only moves and retrieves blocks (does not do any computation)
    - Constructions with statistical security
    - Constructions that work for any block size
    - The client can have oracle access to a private random function

- Boyle-Naor’16
  - Formalized the “balls and bins” model
  - Evidence for the hardness of extending the lower bound beyond the ball and bins model
    - Reduction from sorting circuits to ORAM

- Larsen-Nielsen’18
  - Lower bound extended to computational online ORAM model (only block uploads/downloads)
PanORAMa:
New Oblivious RAM Construction
- Improved asymptotic communication
  - $O(\log N \cdot \log \log N)$ blocks
- Block size $\Omega(\log N)$
- Can be instantiated in the balls and bins model
- Follows the hierarchical paradigm

PanORAMa:
PanORAMa:

- **New Oblivious RAM Construction**
  - Improved asymptotic communication
    - $O(\log N \cdot \log \log N)$ blocks
  - Block size $\Omega(\log N)$
  - Can be instantiated in the balls and bins model
  - Follows the hierarchical paradigm

- **New Oblivious Hash Table Construction**
  - Obliviousness for non-repeating queries
  - Efficient initialization from random array
  - Amortized query communication complexity
    - $O(\log N + \text{poly}(\log \log \lambda))$
- **New Oblivious RAM Construction**
  - Improved asymptotic communication
    - $O(\log N \cdot \log \log N)$ blocks
  - Block size $\Omega(\log N)$
  - Can be instantiated in the balls and bins model
  - Follows the hierarchical paradigm

- **New Oblivious Hash Table Construction**
  - Obliviousness for non-repeating queries
  - Efficient initialization from random array
  - Amortized query communication complexity
    - $O(\log N + \text{poly}(\log \log \lambda))$

- **New Multi-Array Shuffle Algorithm**
  - Efficient shuffle for input with entropy
  - Shuffle multiple sorted arrays
    - $O(N \log \log \lambda + N \log N \log \lambda)$
    - Not too many very small arrays
Construction Paradigms

- Hierarchical ORAMs
  - Worst case ≠ Average case
  - Computation assumption beyond encryption in most cases

- Tree ORAMs
  - Worst case = Average case
  - Encryption - the only computational assumption
Construction Paradigms

- Hierarchical ORAMs
  - Worst case ≠ Average case
  - Computation assumption beyond encryption in most cases

- Tree ORAMs
  - Worst case = Average case
  - Encryption - the only computational assumption
Construction Paradigms

- **Hierarchical ORAMs**
  - Worst case ≠ Average case
  - Computation assumption beyond encryption in most cases

- **Tree ORAMs**
  - Worst case = Average case
  - Encryption - the only computational assumption

\[ PMAP = \{ \text{item}, \text{path} \} \]
Construction Paradigms

- Hierarchical ORAMs
  - Worst case ≠ Average case
  - Computation assumption beyond encryption in most cases

- Tree ORAMs
  - Worst case = Average case
  - Encryption - the only computational assumption
Timeline and Complexity

**Square Root ORAM**
Hierarchical ORAM: $O(\log^3 N)$

**Cuckoo Hashing**
Security issue

**Fixed Cuckoo Hash approach**
$O(\log^3 N)$

**Binary tree per level**
Only Comp. assump.: Encryption: $O(\log^3 N)$

**Cuckoo Hash + level partition**
Best Complexity: Balls & Bins, Any block size: $O(\log^2 N / \log \log N)$

**Deterministic Eviction Schedule**
$O(\log^2 N / \log \log N)$

**Path ORAM: evict on a path**
$O(\log^2 N)$ blocks of size $\Omega(\log N)$
$O(\log N)$ blocks of size $\Omega(\log^2 N)$

**Circuit ORAM, matches Path ORAM for circuit complexity in MPC**

**Offline ORAM**
$O(\log n \log \log n)$

**Homomorphic encryption Onion-ORAM:**
$O(1)$ blocks of size $\Omega(\log^6 N)$

**Unified framework for all hierarchical ORAM**
The Hierarchical ORAM Paradigm
Hierarchical Construction

Level i has capacity for all items assigned to levels 1 to i at any moment \(2^i\) items.
Hierarchical Construction: Search

Level

1

Linear scan

2

\ldots

i

\ldots

log N - 1

\ldots

log N
Hierarchical Construction: Search

Level
1  Linear scan
2   Look up searched item
i   Look up searched item
log N - 1
log N
Hierarchical Construction: Search

Level

1  Linear scan

2  Look up searched item

i  Look up searched item  Item found ✔

log N - 1

log N  Look up random item
Hierarchical Construction: Search

Level

1

2

i

\log N - 1

\log N

Linear scan

Look up searched item

Look up searched item

Item found

Look up random item

Move Item
Hierarchical Construction

**Deterministic schedule shuffle:**
Shuffle all items residing in level 1 to i and place them in level i, **obliviously!**
Oblivious Hash Table [CGLS’17]

Instantiate each level with oblivious hash table (OHT)
- Access pattern hiding for non-repeating queries

Efficiency costs:
- Query cost
- Oblivious initialization
  - During shuffle
Existing Oblivious Hash Table Constructions
Existing Oblivious Hash Table Constructions

GO’96:
- Use pseudo-random function to match items to level buckets
- Query: retrieve item bucket $O(\log n)$
- Oblivious initializations: several oblivious sorts $O(n \log n)$
Existing Oblivious Hash Table Constructions

**GO’96:**
- Use pseudo-random function to match items to level buckets
- Query: retrieve item bucket $O(\log n)$
- Oblivious initializations: several oblivious sorts $O(n \log n)$

**GM’11:**
- Cuckoo hash table
- Query: $O(1)$
- Oblivious initialization: oblivious sort $O(n \log n)$
Existing Oblivious Hash Table Constructions

KLO’12:
- Level size $n$: $\log n$ Cuckoo hash tables; each shuffle creates a new one
- Query all Cuckoo tables: $O(\log n)$
- Oblivious initialization: $\log n$ oblivious sorts on $n/\log n$ items per $n$ queries: $O(\log n)$
Oblivious Two Tier Hash Table [CGLS'17]
Oblivious Two Tier Hash Table [CGLS'17]

Use PRF1 to assign items into bins: number of bins $B = \frac{n}{\log^\epsilon \lambda}$.
Oblivious Two Tier Hash Table [CGLS'17]

Use PRF1 to assign items into bins : number of bins \( B = \frac{n}{\log^\epsilon \lambda} \)
Oblivious Two Tier Hash Table [CGLS'17]

Initialization:

Oblivious sort!

Overflow buffer of size $288. Be^{-Z/6}$

$Z = \log \epsilon \lambda$

Tier 1

Use PRF1 to assign items into bins: number of bins $B = n/\log \epsilon \lambda$

Tier 2

Insert in Tier 2 table

Use PRF2 to assign items into bins
Oblivious Two Tier Hash Table [CGLS'17]

Retrieve bin PRF1(query)
If not found, Retrieve bin PRF2(query)
else, Retrieve random bin.

Tier 1

Tier 2

Amortized complexity: \(O(\log^2 N / \log \log N)\)
PanORAMa Overview

- Leverage entropy reuse to shuffle more efficiently
PanORAMa Hierarchical Construction

Level
1
2

\[ \text{log } N - 1 \]

\[ \text{i} \]

\[ \vdots \]

\[ \text{log } N - 1 \]

\[ \text{log } N \]
PanORAMa Hierarchical Construction

OHT Extract: extract unqueried items from each level in shuffled order

Level 1
Level 2

\( \log N - 1 \)

\( \log N \)
PanORAMa Hierarchical Construction

OHT Extract: extract unqueried items from each level in shuffled order

Multi-array shuffle: shuffle together all randomly permuted input arrays
PanORAMa Hierarchical Construction

OHT Extract: extract unqueried items from each level in shuffled order

Multi-array shuffle: shuffle together all randomly permuted input arrays

OHT Build: build an OHT from the permuted array

No Oblivious Sorting on a whole large level!
Oblivious Hash Table

- *Oblivious initialization in $o(n \log n)$ leveraging input entropy*
Oblivious Hash Table

- **Definition.** Oblivious Hash Table (OHT):
  - OHT.Init - permutes input
  - OHT.Build - builds OHT from permuted input
  - OHT.Lookup - execute a query
  - OHT.Extract - extracts an array that contains unqueried item in random order

- **Security.**
  - Access hiding: non-repeating query sequences
  - Extract output indistinguishable from random permutation
Oblivious Hash Table

- **Definition.** Oblivious Hash Table (OHT):
  - OHT.Init - permutes input
  - OHT.Build - builds OHT from permuted input
  - OHT.Lookup - execute a query
  - OHT.Extract - extracts an array that contains unqueried item in random order

- **Security.**
  - Access hiding: non-repeating query sequences
  - Extract output indistinguishable from random permutation

- **Oblivious bin** = mini OHT
  - OHT that is instantiated on small input size $O(polylog N)$
  - We can use oblivious sorting without hurting efficiency
Oblivious Bin

- Oblivious Cuckoo Bin

- "Dynamic" Bin
  - Items need to be added continuously in non-amortized manner
  - Smallest ORAM level
  - Size: $O(\log^7 n)$
  - Use existing oblivious ram constructions, e.g. Goodrich, Mitzenmacher [GM'11]

- Cuckoo hash
- Oblivious sort to build and extract
  - Add $n$ dummies in Build
  - Extract $n$ items in Extract
Oblivious Hash Table
Oblivious Hash Table

cutoff: $(1-\varepsilon)\log^c \lambda$

Distribute items into $B = \frac{n}{\log^c \lambda}$ bins using a PRF
Oblivious Hash Table

Distribute items into $B = \frac{n}{\log \lambda}$ bins using a PRF

Oblivious resampling of bucket loads: $\Pr[\text{new load} > \text{cutoff}] < \text{negl}(\lambda)$ & $\Pr[\#\text{items} < \text{new load}] < \text{negl}(\lambda)$
Oblivious Hash Table

Obliviously move items above new loads to overflow buffer.

Distribute items into \( B = \frac{n}{\log c \lambda} \) bins using a PRF.

Oblivious resampling of bucket loads: \( \Pr[\text{new load} > \text{cutoff}] < \negl(\lambda) \) & \( \Pr[\#\text{items} < \text{new load}] < \negl(\lambda) \)

Sample new loads from binomial dist. of \((1-\delta)n\) item in \(B\) bins.
Oblivious Hash Table

Distribute items into \( B = \frac{n}{\log \lambda} \) bins using a PRF.

Obliviously move items above new loads to overflow buffer.

Sample new loads from binomial dist. of \((1-\delta)n\) item in \(B\) bins.

Smallest level: \( \frac{N}{\log N} \)

Oblivious resampling of bucket loads: \( \Pr[\text{new load} > \text{cutoff}] < \text{negl}(\lambda) \) & \( \Pr[\#\text{items} < \text{new load}] < \text{negl}(\lambda) \)
Oblivious Hash Table: Create Oblivious Bins

Items from Cuckoo stash are
- (encrypted) indexed with their source bucket
- merged with items used to instantiate the last level
Oblivious Hash Table: Query

```
<table>
<thead>
<tr>
<th>OBin</th>
<th>OBin</th>
<th>OBin</th>
<th></th>
<th>OBin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OBin</td>
</tr>
</tbody>
</table>
```

```
OBin
```

```
OBin
OBin.Lookup(query)
```
Oblivious Hash Table: Query

If not found, OHT[PRF_1(query)].Lookup(query)
If found, OBin[random].Lookup(rand)

If not found, OHT[PRF_2(query)].Lookup(query)
If found, OBin[random].Lookup(rand)

OBin.Lookup(query)
Oblivious Hash Table: Extract

Move items with origin in the Cuckoo stashes back to their corresponding levels
Oblivious Hash Table: Extract
Oblivious Hash Table: Extract

OHT.Extract: append outputs of OBin.Extract
- Items already randomly distributed across OHT bins
Oblivious Hash Table

Amortized Communication Complexity over N accesses for OHT on N items:

- OCuckooBin: $O(\log N + (\log \log \lambda))$
Oblivious Multi-Array Shuffle

- Random shuffle in $o(n \log n)$ leveraging input entropy: independently sorted input arrays
PanORAMa Hierarchical Construction

OHT Extract: extract unqueried items from each level in shuffled order

Level

1

2

i

log N - 1

log N

Multi-array shuffle: shuffle together all randomly permuted input arrays

OHT Build: build an OHT from the permuted array
Oblivious Multi-Array Shuffle

$A_1, \ldots, A_L$ are randomly permuted
⇒ it suffices to choose a random function

Assign: $[n] \rightarrow [L]$
Oblivious Multi-Array Shuffle

Any subarray in $D$ is assigned approximately proportional fractions from $A_1$, ..., $A_L$ under a random Assign.
Oblivious Multi-Array Shuffle

\[ A_1 \] \[ A_L \]

\[ \text{Obliviously Sample} \]
Oblivious Multi-Array Shuffle

\[ A_1, A_L \]

\[ Bin_{in_1}, Bin_{in_2}, \ldots, Bin_{in_{m-1}}, Bin_{in_{m'}} \]

\[ D, \text{Enc(Assign(b))} \]
Oblivious Multi-Array Shuffle

\[ \text{Bin}^{\text{in}}_1 \rightarrow \text{Bin}^{\text{out}}_1 \rightarrow \text{D} \]

\[ \text{Bin}^{\text{in}}_2 \rightarrow \text{Bin}^{\text{out}}_2 \rightarrow \text{D} \]

\[ \ldots \]

\[ \text{Bin}^{\text{in}}_{m'} \rightarrow \text{Bin}^{\text{out}}_{m'} \rightarrow \text{D} \]

\[ A_1 \rightarrow A_L \]

\[ b, \text{Enc}(\text{Assign}(b)) \]

\[ m' \approx (1 - \varepsilon) m \]
Bin Shuffle

For each $j$ in $[L]$, $\text{Bin}_{in}^i$ contains more elements from $A_j$ than the number of elements from $A_j$ assigned to $\text{Bin}_{out}^i$.
Bin Shuffle

Add dummies with each array index
Bin Shuffle

Add dummies with each array index

Oblivious Sort by input array index
Bin Shuffle

**Bin Shuffle**

- **Bin in**: Input bin
- **Bin out**: Output bin
- **Oblivious Sort by input array index**: Sorting mechanism
- **Moving**: Real items which will be placed in Bin out
- **Unused**: Dummy items which will be discarded
- **Overflow**: Real or dummy items which will be returned as overflow

Add dummies with each array index.
Bin Shuffle

Moving: real items which will be placed in $\text{Bin}^{\text{out}}_i$

Overflow: real or dummy items which will be returned as overflow

Unused: dummy which will be discarded

Adding dummies with each array index
Bin Shuffle

$\text{Bin}^{\text{in}}_i$ $\xrightarrow{\text{Moving}}$ $\text{Bin}^{\text{out}}_i$

$A_1$ $A_2$ $\ldots \ldots \ldots$ $A_L$
Bin Shuffle

Bin\text{in} \rightarrow Bin\text{out}

\begin{align*}
A_1 & \quad A_2 & \quad \cdots & \quad A_L \\
\text{Moving} & & & \\
\end{align*}

Obliviously Sort according to Assign(b)

Pairs: (b, Assign(b))
Bin Shuffle

Match pairs from $\text{Bin}^{\text{out}}_i$ with content from $\text{Bin}^{\text{in}}_i$

Obliviously Sort according to $\text{Assign}(b)$

Pairs: $(b, \text{Assign}(b))$
Bin Shuffle

\[ \text{Bin}_{\text{in}} \]

\[ \text{Bin}_{\text{out}} \]

- \( (b, \text{Assign}(b) = 1) \)
- \( (b', \text{Assign}(b') = 2) \)
- \( (b'', \text{Assign}(b'') = L) \)
- \( (b''', \text{Assign}(b''') = L) \)
Oblivious Multi-Array Shuffle

Let $b_i$ be the $i$-th input bin. Assign $b_i$ values as follows:

- Assign($b_1$) = 1
- Assign($b_2$) = 2
- Assign($b_{m-1}$) = $L$
- Assign($b_m$) = $L$

Sort by the value $b$ to obtain the output bins $b_1', b_2', \ldots, b_m'$. The final output is $b_1''$, $b_2''$, $b_{m-1}''$, and $b_m''$. 

Bin$^{\text{in}}$ - Bin$^{\text{out}}$
Bin Shuffle

Assign to positions $b_i$ in $D$ non-obliviiously
Oblivious Multi-Array Shuffle

$A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_L$

$Bin_{in}^1 \rightarrow Bin_{out}^1 \downarrow \vdots \downarrow \vdots \downarrow Bin_{out}^{m'}$

Bin Shuffle
Oblivious Multi-Array Shuffle

\[ A_1 \rightarrow Bin_{in}^{1} \rightarrow Bin_{out}^{1} \rightarrow A_2 \rightarrow Bin_{in}^{2} \rightarrow Bin_{out}^{2} \rightarrow \ldots \rightarrow海岸 \rightarrow Bin_{in}^{m'} \rightarrow Bin_{out}^{m'} \rightarrow A_L \rightarrow Bin_{out}^{m+1} \rightarrow Bin_{out}^{m} \rightarrow \ldots \]

Recurse

Total Leftover_1 \rightarrow \ldots \rightarrow Total Leftover_L

Bin Shuffle
Ball and Bins Model

- We can instantiate the PanORAMa construction in the model where GO’96 proved $O(\log N)$ lower bound
  - Server does no computation on the data $\Rightarrow$ satisfy “balls and bins” requirement
  - GO’96 allows client to oracle access to private random function $\Rightarrow$ replace PRF

- PanORAMa complexity: $O(\log N \cdot \log \log N)$
Follow-up Work
ORAM with Logarithmic Complexity

- OptORAMa: Optimal Oblivious RAM [AKLNPS18] (eprint 2018/892)
  - Communication complexity: $O(\log N)$
  - Oblivious compaction: $O(N)$
Overview

- **PanORAMa**: new ORAM construction with improved asymptotic complexity
  - $O(\log N \cdot \log \log N)$ for block size $\Omega(\log N)$

- New Efficient Building Blocks
  - Oblivious Hash Table
  - Oblivious Multi-Array Shuffle
Thanks!

Questions?