Perfectly Secure Oblivious Algorithms in the Multi-Server Setting

T-H. Hubert Chan, Jonathan Katz, Kartik Nayak, Antigoni Polychroniadou, Elaine Shi
Defining an Oblivious RAM

Request sequence I: Read(a1), Write(a2, d'), Read(a3), ...

Client snoops on the address bus

Example request sequence I: Read(a1), Write(a2, d'), Read(a3), ...

Server
Defining an Oblivious RAM

Request sequence I

Response

CPU

Client

Sequence ORAM(I)

- Adversary (server) is semi-honest
- No server computation

Security: for I and I’ of the same length,

ORAM(I) ~ ORAM(I’)

bandwidth: #mem locations accessed by ORAM(I) for every access

Server
ORAM(I) \sim ORAM(I')

ORAM(I) \sim ORAM(I') typically

Computationally indistinguishable or Statistically indistinguishable

Statistically indistinguishable:

Adversary cannot distinguish with probability

\[ > \operatorname{negl}(N) \quad \Rightarrow \quad \text{negl}(\lambda) \]

\[ N = \operatorname{poly}(\lambda) \]

If \( N = \operatorname{polylog}(\lambda) \)

\[ \operatorname{negl}(N) \neq \operatorname{negl}(\lambda) \]

Achieving \( \operatorname{negl}(\lambda) \) difference using existing schemes is inefficient;

bandwidth of \( N^c, c < 1 \)
Perfectly-Secure ORAM

ORAM(I) \sim ORAM(I') \quad \text{Identically distributed}

Existing perfectly-secure ORAMs: Bandwidth $O(\log^3 N)$ [DMN’11, CNS’18]
Oblivious RAMs: Bandwidth Trade-offs

Request sequence \( I \)  

Response  

Security: for \( I \) and \( I' \) of the same length, \( \text{view}^{\text{Adv}}(I) \) and \( \text{view}^{\text{Adv}}(I') \) are identically distributed

\[ \text{view}^{\text{Adv}}: \text{denotes what the adversary can observe from the semi-honest corrupt servers} \]

Client

Server \( S_1 \)

Server \( S_2 \)

\ldots

Server \( S_k \)

CPU

[Image]
Oblivious RAMs: Bandwidth Trade-offs

<table>
<thead>
<tr>
<th></th>
<th>Computationally or statistically-secure ORAMs</th>
<th>Perfectly-secure ORAMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-server</td>
<td>$O(\log^2 N/\log \log N)$ [KLO’12]</td>
<td>$O(\log^3 N)$ [DMN’11, CNS’18]</td>
</tr>
<tr>
<td>Multi-server</td>
<td>$O(\log N)$ [LO’13]</td>
<td>$O(\log^2 N)$ [This paper]</td>
</tr>
</tbody>
</table>

1. Multi-server ORAMs were only computationally or statistically secure

2. Are there inherent advantages in the multi-server setting?
# Oblivious RAMs: Bandwidth Trade-offs

<table>
<thead>
<tr>
<th></th>
<th>Computationally or statistically-secure ORAMs</th>
<th>Perfectly-secure ORAMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-server</td>
<td>$O(\log^2 \frac{N}{\log \log N})$ [KLO'12]</td>
<td>$O(\log^3 N)$ [DMN'11, CNS'18]</td>
</tr>
<tr>
<td>Multi-server</td>
<td>$O(\log N)$ [LO'13]</td>
<td>$O(\log^2 N)$ [This paper]</td>
</tr>
</tbody>
</table>

1. Multi-server ORAMs were only computationally or statistically secure.

2. Are there inherent advantages in the multi-server setting?
Our Results

There exists a perfectly-secure 3-server scheme for a single semi-honest corruption to perform

1. Oblivious stable compaction and merging with $O(N)$ bandwidth

Lower bound: Single-server oblivious stable compaction and merging requires $\Omega(N \log N)$ bandwidth in the balls-and-bins model [LSX’18]
Our Results

There exists a perfectly-secure 3-server scheme for a single semi-honest corruption to achieve

1. Oblivious stable compaction and merging with $O(N)$ bandwidth

2. ORAM scheme with $O(\log^2 N)$ bandwidth
Oblivious Sort Incurs $O(N \log N)$ Bandwidth

Typically, shuffle is performed using oblivious sort
Key Idea: Replace Oblivious Sort With Linear Time Operations
Permutation-Storage-Separation Paradigm

Permuted Server

Storage Server

3 8 7 ... 5 2

Assumption: Data encrypted using perfectly-secure encryption scheme
### Permutation-Storage-Separation Paradigm

<table>
<thead>
<tr>
<th>Permuted Server</th>
<th>Storage Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows permutation</td>
<td>Observes accesses</td>
</tr>
<tr>
<td>Fisher-Yates: $O(N)$ bandwidth</td>
<td>$O(1)$ bandwidth (assuming position is known)</td>
</tr>
</tbody>
</table>

Lu-Ostrovsky introduced this paradigm [LO’13]
- Built cuckoo hash tables + used PRFs to access data
- Computationally-secure
Can we perform O(N) bandwidth oblivious sort using this paradigm?
- Not aware of a solution
- Comparison-based (non-oblivious) sorts incur O(N log N)
Our Results

There exists a perfectly-secure 3-server scheme for a single semi-honest corruption to achieve

1. Oblivious stable compaction and merging with $O(N)$ bandwidth

2. ORAM scheme with $O(\log^2 N)$ bandwidth
Oblivious Tight Stable Compaction

Input: $n$ elements, some real, some dummy

Output: $n$ elements, all real elements at the beginning, order of real elements is preserved
Attempt 1: Oblivious Tight Stable Compaction

Server 1

Server 2

Protocol: Read block, if real, write to storage
Pad with dummies

Obliviousness: Each server observes a linear scan
Server 2 observes write time steps
Oblivious Tight Stable Compaction

Server 1: Permute

*Remember head of linked-list
*Maintain a dummy linked-list too

Permute using $\pi$, determine destination

Inverse permute: $\pi^{-1}$

Reverse linear scan to create linked-list

Permute using $\pi$ again
Oblivious Tight Stable Compaction

Server 1: Permute

Server 2: Access

Protocol:
- Traverse real linked list followed by dummy linked list
Oblivious Tight Stable Compaction

Server 1: Permute

Server 2: Access

Security:
Server 1 permutes and performs linear scan. Does not observe accesses.
Server 2 observes accesses, does not know permutation
Oblivious Merge

Input: $S_1$ and $S_2$ have semi-sorted lists with $n_1$ and $n_2$ elements resp.

Server $S_1$

Server $S_2$

Output: Sorted list of $n_1 + n_2$ elements on $S_1$
Our Results

There exists a perfectly-secure 3-server scheme for a single semi-honest corruption to achieve

1. Oblivious stable compaction and merging with $O(N)$ bandwidth

2. ORAM scheme with $O(\log^2 N)$ bandwidth
Hierarchical ORAM [GO’96]

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log N</td>
<td>N reals</td>
</tr>
<tr>
<td>log N - 1</td>
<td>N/2 reals</td>
</tr>
<tr>
<td>log N - 2</td>
<td>N/4 reals</td>
</tr>
<tr>
<td>log N - 3</td>
<td>Level 1</td>
</tr>
</tbody>
</table>

The diagram shows the hierarchical structure of the ORAM with the number of reals at each level.
Hierarchical ORAM [GO’96]

Level $\log N$ - 1

- $N/2$ reals

Level $\log N$

- $N$ reals

[GO’96]: $O(\log N)$ sized buckets, block $b$ stored in $\text{PRF}_k(b)$

N/4 reals

Avoid PRF?
Position-based Hierarchical ORAM [CNS’18]

Store blocks shuffled uniformly at random

Access a block:
- Is the block stored at this level?
- If yes, location?
- else, location of a dummy?

Level $\log N$ - 1

Level $\log N$
Position-based Hierarchical ORAM [CNS’18]

For all levels,
- Is the block stored at this level?
- If yes, location?
- else, location of a dummy?

Level \( \log N \)

- N/2 reals

Level \( \log N - 1 \)

- N/4 reals

Level 1

- N reals
Position-based Hierarchical ORAM [CNS’18]

- Level 1
- Level \(\log N - 1\)
- Level \(\log N\)
Recursive Position-based Hierarchical ORAM [CNS’18]

Position-based ORAM at height-(d-1)

Block b at height-(d-1) stores the level and position of blocks 2b and 2b+1 at height-d

Position-based ORAM at height-d

For all levels, positions of all blocks
Reviewive Position-based Hierarchical ORAM \cite{CNS’18}

Position-based ORAM at depth-(d-1)

Block $b$ at depth-(d-1) stores the level and position of blocks $2b$ and $2b+1$ at depth-$d$.

Caveats:

1. Does not handle dummies
2. Cannot be used in a black-box manner

For all levels,

- Is the block stored at this level?
- If yes, location?
- Else, location of a dummy
Co-ordinated Reshuffle Across Hierarchies

Position-based ORAM at height-(d-1)

Block b at height-(d-1) stores the level and position of blocks 2b and 2b+1 at height-d.

Co-ordinated reshuffle:
When level l at height-d is reshuffled, all levels ≤ l at height < d are reshuffled.

For all levels, positions of all blocks

Position-based ORAM at height-d
Co-ordinated Shuffle in the Multi-Server Setting

Permutation-Storage-Separation paradigm

Linear time oblivious compaction + merging

Linear time co-ordinated shuffle
Conclusion

- Oblivious stable compaction and merging can be performed with $O(N)$ bandwidth using 3 servers

- 3-server ORAM scheme with $O(\log^2 N)$ amortized bandwidth

Thank You!
kartik@cs.duke.edu