Oblivious Computation
Part I - Lower Bounds and Tree Based ORAMs

Gilad Asharov
Bar-Ilan University

Some slides were created by: Elaine Shi, Ilan Komargodski
Models of Computation

**Circuits**

- Emulate easily

\[ T^3 \log T \]

[CR73,PF79]

\((T \gg N)\)

**RAM Model**

- Random Access Machine
- CPU Operation
- Memory access

**Metrics:**

- Size (how many wires, gates)
- Depth (parallelism)
- Time: \(T\)
- Size of the memory: \(N\)
Access Patterns Reveal Information!
func search(val, s, t)
    mid = (s+t)/2
    if val < mem[mid]
        search(val, 0, mid)
    else
        search(val, mid+1, t)

Access Pattern of **binary search** leaks the **rank** of the number being searched.
Access Patterns Reveal Information!

if (secret variable)
    Read mem[x]
else
    Write mem[y]

Access pattern reveals the value of the **secret variable**
Access Patterns Reveal Information!

Kidney Problem

Liver Problem

Heart Problem
A program in the RAM model Access Pattern is “oblivious”: Can be simulated from (T,N)
Example: Sorting

- **Merge sort:** $O(n \log n)$
  - non oblivious

- **Bubble sort:** $O(n^2)$
  - oblivious
A program in the RAM model
Access Pattern is “oblivious”: Can be simulated from (T,N)

Usually, $N' = O(N)$
$T'/T$: overhead of the compilation

Oblivious RAM Compiler

RAM Program with (T,N) $\rightarrow$ Oblivious RAM Program with (T′, N′)

Trivial Compiler

RAM Program with (T,N) $\rightarrow$ Oblivious RAM Program with (TN, N)
Oblivious RAM (ORAM)

An algorithmic technique that provably encrypts access patterns

Goldreich and Ostrovsky (87;90;96)

Permuting and shuffling elements around the memory
ORAM

Read addr
Write addr, data

ORAM

Multiple physical reads/writes

Simulator

Memory

Memory

Security: Physical accesses independent of input logical sequence
Hierarchical ORAM

\[ O(\sqrt{N}) \]
\[ O(\log^3 N) \]

\vspace{1cm}

Hierarchical ORAM

\[ \approx O(\log^2 N) \]

\vspace{1cm}

Tree Based ORAM

\[ O(\log^3 N) \]
\[ \Rightarrow \]
\[ O(\log^2 N) \]

Simple, small constants

Statistical

Hierarchical ORAM

\[ O(\log N) \]

Matching the lower bound!

(Big constant)

Computational

Lower Bound

\[ \Omega(\log N) \]
Oblivious RAM Compiler: State of the Art

Lower bound: $\Omega(\log N)$
[GoldreichOstrovsky'96, LarsenNielsen'18]

Hierarchical
[O90,GO96]
$O(\log N)$

Tree based ORAM
[Shi,Chan,Stefanov11]
$O(\log^2 N)$

Computational security
[OptORAMa,AKLNPS'20]

Statistical security
[PathORAM,CircuitORAM]
Oblivious PRAM compiler:
Introduced by Boyle, Chung and Pass in 2016

Recent work [AKLPS, SODA'22]:
Any PRAM program with $T$ parallel time and $N$ space
$\implies T \log N$ parallel time and $N$ space
Oblivious RAM Compiler: State of the Art

Lower bound: $\Omega(\log N)$

Hierarchical

$O(\log N)$

Tree based ORAM

$O(\log^2 N)$

Computational security

Statistical security

[OptORAMa, AKLNPS’20]

[PathORAM, CircuitORAM]
Lower Bounds

Any ORAM compiler results in $\Omega(\log N)$ overhead
Lower Bounds

Goldreich and Ostrovsky ['96]:
\[ \Omega(\log N) \]
- Balls and Bins model
- Statistical Security
- Offline ORAM

Boyle and Naor ['16]:
An \[ \Omega(\log N) \] lower bound for offline ORAM
not in the balls and bins model
implies an \[ \Omega(N \log N) \] lower bound for sorting circuits

Larsen and Nielsen ['18]:
\[ \Omega(\log N) \]
- Not in Balls and Bins model
- Computational Security
- Online ORAM

Information transfer technique

**Offline ORAM**: the entire logical sequence is known in advance; including all addresses and data
The Lower Bound [LN’18]

- Based on **information transfer** technique of Pătrașcu & Demaine ’06
- Cell probe model [Yao’81] - computation is free, only charge for probes
The Lower Bound [LN’18]

Assign $p_i^j = \text{(Read/Write, addr)}$ to an internal node $\nu$ iff $\nu$ is the lower common ancestor of the two last physical accesses of addr

$P_\nu = \{p_1, \ldots, p_k\}$
Example

Assign $p_i^j = (\text{Read/Write, addr})$ to an internal node $v$ iff $v$ is the lower common ancestor of the two last physical accesses of $\text{addr}$.

$P_v = \{ p_1, \ldots, p_k \}$
Example:

Assign \( p^j_i = (\text{Read/Write, addr}) \) to an internal node \( \nu \) iff \( \nu \) is the lower common ancestor of the two last physical accesses of addr.

Each physical probe is counted at most once.

Total # of probes \( \geq \sum_{\nu \in \text{Tree}} |P_{\nu}| \)

Enough to bound

\[ \sum_{\nu \in \text{Tree}} |P_{\nu}| \geq ?? \]

**Logical Operations:**
- Op1
- Op2
- Op3

**Physical Probes:**
- \( p^1_1 \ldots p^q_1 \)
- \( p^1_2 \ldots p^q_2 \)
- \( p^1_3 \ldots p^q_3 \)

- 5, 10, 20, 1
- 12, 11, 20, 44
- 4, 44, 50, 20
Based on the **physical access pattern** - the adversary can compute the tree

**Security:** For all logical sequences, for all $v$, $|P_v|$ should be similar

For every $v$, we can show a **logical sequence** forcing $|P_v|$ to be large.
Based on the **physical access pattern** - the adversary can compute the tree

**Security:** For all logical sequences, for all $v$, $|P_v|$ should be similar

For every $v$, we can show a **logical sequence** forcing $|P_v|$ to be large

**Assumes online ORAM**

**Logical Operations:**
- $Write(1,r1)$
- $Write(2,r2)$
- $Read(1)$
- $Read(2)$
**Claim:** For every node in depth $d$, $E[|P_v|] \geq \frac{N}{2^d}$

**Proof by encoding / compression argument**

**Logical Operations:**
- Write(1,r1)
- Write(2,r2)
- Read(1)
- Read(2)
Claim: For every node in depth \(d\), \(E[|P_v|] \geq \frac{N}{2^d}\)

\[
E[\text{total #of probes}] \geq \sum_{v \in \text{Tree}} E[|P_v|] = \sum_{v \in \text{Tree}} \frac{N}{2^d} = \sum_{d=0}^{\log N-1} 2^d \cdot \frac{N}{2^d} = N \log N
\]

We considered \textbf{logical sequences} of length \(N\)

\(\Omega(\log N)\) overhead per operation (in expectation)
References

Goldreich and Ostrovsky: Software Protection and Simulation on Oblivious RAMs, JACM 1996

Boyle and Naor: Is There an Oblivious RAM Lower Bound? ITCS 2016

Larsen and Nielsen: Yes! There is an Oblivious RAM Lower Bound, CRYPTO 2018

Weiss and Wichs: Is there an Oblivious RAM Lower Bound for Online Reads? TCC 2018

Pavel Hubacek, Michal Koucky, Karel Kral, Veronika Slivova: Strong Lower Bounds for Online ORAM, TCC 2019

Jacob, Larsen, Nielsen: Lower bounds for oblivious data structures, SODA 2019

Persiano and Yeo: Lower bounds for differentially private RAMs, EUROCRYPT 2019

Larsen, Simkin, Yeo: Lower bounds for multi-server oblivious RAMs, TCC 2020

Komargodski and Lin: A logarithmic lower bound for oblivious RAM (for all parameters), CRYPTO 2021

And more…
Oblivious RAM Compiler: State of the Art

Lower bound: $\Omega(\log N)$

- [GoldreichOstrovsky’96, LarsenNielsen’18]

Hierarchical

- $O(\log N)$
  - Computational security
    - [OptORAMa’20]

Tree based ORAM

- $O(\log^2 N)$
  - Statistical security
    - [PathORAM,CircuitORAM]
Tree Based ORAM

Simple constructions, statistical security, $O(\log^2 N)$ overhead
Strawman: Randomly Permute Blocks in Memory

1  2  3  4  5  6  7  8

4  6  1  5  2  8  3  7
Strawman: Randomly Permute Blocks in Memory
Strawman: Randomly Permute Blocks in Memory

The adversary has no clue what the client is accessing
Strawman: Randomly Permute Blocks in Memory

The adversary has no clue what the client is accessing
Strawman: Randomly Permute Blocks in Memory

Repeated query!!!
Blocks must move around in memory!
Each bucket stores **real** and **dummy** blocks.
Path invariant: every block mapped to a random path
Reading a block is simple!
After being read, block x must relocate!
Pick a new random path and move x there.
Where on the new path can we write block $x$?
Can we write it to the leaf?
Can we write it to the leaf?
Writing to any non-root bucket leaks information
Write it to the root!
Security: every request, visit a random path that has not been revealed
Problem?

Memory

Position map

block x

CPU
Problem: root will overflow

Memory

Position map

block x

CPU
A background eviction process percolates blocks upwards
A background eviction process percolates blocks upwards

Not too slow: prevent overflow
Not too fast: save cost
Every request: pick **2 random buckets per level** to evict
Every request: pick **2 random buckets per level** to evict

Scan, find a real block, write to a child

load > 0
Eviction process does not leak information
Thm: bucket size = log n  \iff  no overflow w.h.p.  \quad [SCSL’11]

Proof: use queuing theory and measure concentration bounds.
Thm: bucket size $= \log n \quad \iff \quad$ no overflow w.h.p. \cite{SCSL11}

Every request incurs $O(\log^2 n)$ cost
Store position map **recursively** in a smaller ORAM
Cost with eviction: $O(\log^3 n)$
Previous construction - \( O(\log^3 N) \) overhead:
- Each path has \( O(\log N) \) nodes
- Each node has a bucket of size \( O(\log N) \)
- Recursion adds another \( O(\log N) \)

Improvement: Path ORAM (\( O(\log^2 N) \) overhead)
- Each node has a bucket of size \( O(1) \)
- Client has local stash of size \( \text{poly} \log N \)
1: $x \leftarrow \text{position}[a]$
2: \text{position}[a] \leftarrow \text{UniformRandom}(0 \ldots 2^L - 1)$
3: for $\ell \in \{0, 1, \ldots, L\}$ do
4: \hspace{1em} $S \leftarrow S \cup \text{ReadBucket}(\mathcal{P}(x, \ell))$
5: end for
6: data $\leftarrow \text{Read block } a \text{ from } S$
7: if $\text{op} = \text{write}$ then
8: \hspace{1em} $S \leftarrow (S - \{(a, \text{data})\}) \cup \{(a, \text{data}^*)\}$
9: end if
10: for $\ell \in \{L, L - 1, \ldots, 0\}$ do
11: \hspace{1em} $S' \leftarrow \{(a', \text{data'}) \in S : \mathcal{P}(x, \ell) = \mathcal{P}(\text{position}[a'], \ell)\}$
12: \hspace{1em} $S' \leftarrow \text{Select min}(|S'|, Z) \text{ blocks from } S'$.
13: \hspace{1em} $S \leftarrow S - S'$
14: \hspace{1em} $\text{WriteBucket}(\mathcal{P}(x, \ell), S')$
15: end for
16: return data
Summary: tree-based ORAMs

- A block is re-mapped to a new random path upon being read.
- The block must be relocated to the new path without revealing the new path.
- Key challenge: design eviction process and prove no overflow.
Tree Based ORAM

Shi, Chan, Stefanov, Li: **Oblivious RAM with $O(\log^3 N)$ Worst-Case Cost**, ASIACRYPT 2011


Chung, Pass: **A Simple ORAM**, 2013

Oblivious RAM Compiler: State of the Art

Lower bound: $\Omega(\log N)$
[GoldreichOstrovsky'96, LarsenNielsen'18]

Hierarchical
$O(\log N)$
[O90,GO96]
Computational security
[OptORAMa,AKLNPS'20]

Tree based ORAM
$O(\log^2 N)$
[Shi,Chan,Stefanov11]
Statistical security
[PathORAM,CircuitORAM]
Thank You!