Techniques for Practical ORAM and ORAM in Hardware

Ling Ren

Joint work with
Chris Fletcher, Albert Kwon, Marten van Dijk and Srini Devadas
Oblivious RAM

- If $|y| = |y'|$, $\text{ORAM}(y)$ and $\text{ORAM}(y')$ indistinguishable

**Applications**
- Encrypted computation using secure processor
- Remote oblivious storage
- Secure RAM computation

This talk
Chris’ talk
Daniel’s talk

Program data
User

Input access pattern
Write a0 d0
Read a0
Read a1
Read a0

Secure Processor

\text{ORAM}\text{Controller}

$\text{ORAM}(y)$

External DRAM

Obfuscated addr & ciphertext:
Read a652 %&X
...
Write a431 #$@
...
Outline

• Introduction to tree-based ORAMs

• Optimizing recursive ORAM

• Ring ORAM

• Hardware ORAM
Path ORAM

- Efficient and simple
- External DRAM structured as a binary tree
Path ORAM

- Position Map: map each block to a random path
- Invariant: if a block is mapped to a path, it must be on that path or in the stash
- Stash: temporarily hold some blocks

<table>
<thead>
<tr>
<th>Block</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>0</td>
</tr>
<tr>
<td>B1</td>
<td>3</td>
</tr>
<tr>
<td>B2</td>
<td>3</td>
</tr>
<tr>
<td>B3</td>
<td>0</td>
</tr>
<tr>
<td>B4</td>
<td>1</td>
</tr>
</tbody>
</table>

ORAM controller

Stash

Position Map

<table>
<thead>
<tr>
<th>Block</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>0</td>
</tr>
<tr>
<td>B1</td>
<td>3</td>
</tr>
<tr>
<td>B2</td>
<td>3</td>
</tr>
<tr>
<td>B3</td>
<td>0</td>
</tr>
<tr>
<td>B4</td>
<td>1</td>
</tr>
</tbody>
</table>

DRAM

root

(B3, 0)

(B0, 0)

dummy

(B2, 3)

dummy

(B1, 3)

path 0 1 2 3
Path ORAM Operation

- **Access Block 1**
  - Read all blocks on path 3
  - Remap B1 to a new random path
  - Write as many blocks as possible back to path 3 (keep the invariant)

<table>
<thead>
<tr>
<th>Block</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>0</td>
</tr>
<tr>
<td>B1</td>
<td>X 1</td>
</tr>
<tr>
<td>B2</td>
<td>3</td>
</tr>
<tr>
<td>B3</td>
<td>0</td>
</tr>
<tr>
<td>B4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Position Map</th>
<th>Stash</th>
<th>dummy</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B0, 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B3, 0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B2, 3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B1, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(B1, 3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DRAM**
- root
  - (B3, 0)
  - (B0, 0)
  - (B2, 3)

**ORAM controller**
Path ORAM Analysis

- Bandwidth $O(ZL) = O(Z\log N)$
- $|Stash| = O(\log N)$ for $Z \geq 4$
- Security: a random path is accessed
- Other tree-based ORAMs differ in eviction

<table>
<thead>
<tr>
<th>Block</th>
<th>Path</th>
<th>Position Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B1</td>
<td>X 1</td>
<td>(B1, 1)</td>
</tr>
<tr>
<td>B2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B4</td>
<td>1</td>
<td>(B4, 1)</td>
</tr>
</tbody>
</table>

ORAM controller

DRAM

root

(B3, 0)

dummy
dummy
dummy

(B2, 3)

path 0 1 2 3
Recursive ORAM

- Position map is too large
- Bandwidth after recursion $O(\log N + \log^3 N/B)$

$H = O(\log N)$ ORAMs in recursion

<table>
<thead>
<tr>
<th>Block</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>92</td>
</tr>
<tr>
<td>1</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N-1</td>
<td>...</td>
</tr>
</tbody>
</table>

Block | Pos
---|---
... | ...
Outline

• Introduction to tree-based ORAMs

• Optimizing recursive ORAM

• Ring ORAM

• Hardware ORAM
Recursive ORAM Access

Request for Block 0

Secure processor

External memory

Data ORAM

$\text{Pos}(0), \text{Pos}(1), \ldots, \text{Pos}(k-1)$

Compression

Locality

$\text{PosMap}\downarrow 2$

<table>
<thead>
<tr>
<th>Block</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>N+1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

$\text{PosMap ORAM}$

<table>
<thead>
<tr>
<th>Block</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Pos(0)</td>
</tr>
<tr>
<td>1</td>
<td>Pos(1)</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>N-1</td>
<td></td>
</tr>
</tbody>
</table>
PLB: Cache PosMap

Request for Block 0

Secure processor

External memory

PLB cache recent PosMap blocks

PosMap↓2

<table>
<thead>
<tr>
<th>Block</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>N+1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Pos(0), Pos(1), ..., Pos(k−1)

Locality

Data

ORAM

Block 0

Pos(0)

Block N

Pos(N)

ORAM

Block N

<table>
<thead>
<tr>
<th>Block</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Pos(0)</td>
</tr>
<tr>
<td>1</td>
<td>Pos(1)</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N-1</td>
<td>...</td>
</tr>
</tbody>
</table>
PLB: Cache PosMap

Request for Block 0
Request for Block 1

Secure processor
External memory

PLB cache recent PosMap blocks

PosMap↓2

<table>
<thead>
<tr>
<th>Block</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>N+1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Locality

Pos(0), Pos(1), ..., Pos(k−1)

PLB Hit/Miss depends on access pattern!
Unified ORAM

Request for Block 0

Secure processor

External memory

PLB cache recent PosMap blocks

PosMap↓2

<table>
<thead>
<tr>
<th>Block</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>N+1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Pos(0), Pos(1), ..., Pos(k-1)

Locality

ORAM for Data & PosMap
Unified ORAM

Request for Block 0

Secure processor

External memory

PLB cache recent PosMap blocks

PosMap↓2

<table>
<thead>
<tr>
<th>Block</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>N+1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

ORAM for Data & PosMap

Pos(0), Pos(1), ..., Pos(k-1)

Locality
Unified ORAM

ORAM for Data & PosMap

Request for Block 0
Request for Block 1

Secure processor
External memory

PLB cache recent PosMap blocks

PosMap↓2

<table>
<thead>
<tr>
<th>Block</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>N+1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

ORAM for Data & PosMap

If \( |\text{ORAM}(y)| = |\text{ORAM}(y')| \) ?? Worth discussion \((1), \cdots, \text{Pos}(k-1)\)

If \( |y| = |y'| \), \( \text{ORAM}(y) \) and \( \text{ORAM}(y') \) indistinguishable
PosMap compression

- **PosMap**: block $\rightarrow$ random leaf ($\log N$ bits)
  - A table of random numbers

- **Block $\rightarrow$ monotonic counter $\rightarrow$ pseudorandom leaf**
  - $|\text{counter}| > \log N$
  - Reduce $|\text{counter}| \rightarrow$ counter overflows

Replacing true randomness with pseudorandomness improves efficiency
PosMap Compression

- Let a group of blocks share a big counter

\[ \text{Pos}(i) = \text{PRF}_{\downarrow K}(GC | IC_{\downarrow i} | i) \]

\[ \text{PRF}_{\downarrow K}(GC | IC_{\downarrow j} | j) \rightarrow \text{PRF}_{\downarrow K}(GC+1 | 0 | j) \]

\[ \frac{k}{2^{\uparrow \beta}} \text{ small } \frac{\alpha}{k} + \beta < \log N \]

\[ \alpha = 64, \ k = 32, \ \beta = 14 \]

\[ \frac{k}{2^{\uparrow \beta}} = 0.2\% \]

\[ \frac{\alpha}{k} + \beta = 16 < 26 = \log N \]
Asymptotic Improvement

- $\beta = \log \log N$, $k = \log N / \log \log N$, $\alpha = \Theta(\log N)^\alpha$ bits
- $k/2 \uparrow \beta = o(1)$, $\alpha/k + \beta = \log \log N < \log N$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Asymptotic bandwidth</th>
<th>$B = \Theta(\log N)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursive Path</td>
<td>$O(\log N + \log \uparrow^3 N/B)$</td>
<td>$O(\log \uparrow^2 N)$</td>
</tr>
<tr>
<td>+ Compression</td>
<td>$O(\log N + \log \uparrow^3 N/B \log \log N)$</td>
<td>$O(\log \uparrow^2 N / \log \log N)$</td>
</tr>
</tbody>
</table>
Simulation Results

![Graph showing simulation results]
Outline

• Introduction to tree-based ORAMs

• Optimizing recursive ORAM

• Ring ORAM

• Hardware ORAM
• Path ORAM: $8\log N$ evict to the [random] path being read
• Load balance between paths
• Allow less frequent evictions (1 per $A$ accesses)
  – $Z=4, A=3$
  – $Z=5, A=5$
  – $Z=7, A=8$

Craig Gentry, Kenny Goldman, Shai Halevi, Charanjit Julta, Mariana Raykova and Daniel Wichs.
Optimizing ORAM and Using it Efficiently for Secure Computation
Permutated buckets

- **Goal:** read only 1 block per bucket
  - Add $Y$ reserved dummy slots, and permute buckets
  - Block of interest or a fresh dummy
  - Re-permute if out of fresh dummies

$Z=4$

$Y > A$
Ring ORAM Performance

- Eviction overhead \((Z + Z + Y) / A\) per bucket

<table>
<thead>
<tr>
<th>Scheme and Parameters</th>
<th>Read</th>
<th>Eviction</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path ORAM (Z = 4)</td>
<td>(4L)</td>
<td>(4L)</td>
<td>(8L)</td>
</tr>
<tr>
<td>Ring ORAM (Z=16, A=22, Y=28)</td>
<td>(L)</td>
<td>(2.8L)</td>
<td>(3.8L)</td>
</tr>
<tr>
<td>Ring ORAM (Z=5, A=5, Y=7)</td>
<td>(L)</td>
<td>(3.8L)</td>
<td>(4.8L)</td>
</tr>
</tbody>
</table>

![Graph showing slowdown comparison between Path ORAM and Ring ORAM](image)

3.8x \(→\) 1.8x
Outline

• Introduction to tree-based ORAMs

• Optimizing recursive ORAM

• Ring ORAM

• Hardware ORAM
Things we take for granted in ORAM algorithms

- Computation is cheap (e.g. ORAM eviction, hashing)
- RAM has uniform latency
Stash Management

- Computation is cheap?

```plaintext
function PUSH_BACK(l, l')
    t_1 ← (l ⊕ l') || 0
    t_2 ← t_1 & ~t_1
    t_3 ← t_2 - 1
    full ← {(Occupied[i] = Z) for i = 0 to L}
    t_4 ← t_3 & ~full
    t_5 ← reverse(t_4)
    t_6 ← t_5 & ~t_5
    t_7 ← reverse(t_6)
    if t_7 ≠ 0 then
        return -1
    return \log_2(t_7)
```

- Bitwise XOR
- Bitwise AND, 2C negation
- 2C subtraction
- Bitwise AND/negation
- Bitwise reverse
- Block is stuck in stash
- Note: \( t_7 \) must be one-hot
• Just use hash tree?
  – A serialization problem. Hashing becomes the bottleneck
PMMAC for Authenticity

• **Construction**
  – Block → monotonic counter
  – MAC(Block address || counter || block data)

• **Problem: privacy under malicious server**
  – Is it fixable? Bounding the leakage?
- An access to a new row is slow
- Very bad for trees
- Solution: subtree layout

DRAM Timing

GIF from http://www.pcmag.com/article2/0,2817,1153175,00.asp
Our Hardware ORAM

• **Techniques we have implemented**
  – PLB + Unified ORAM tree
  – Load balance eviction order (1/A eviction frequency)
  – Efficient stash management
  – PMMAC
  – Subtree layout

• **Techniques we have not implemented**
  – Compressed counter, permuted buckets
FPGA Die Photo

- Area: 6% logic, 14% memory of a mid-range FPGA
- Performance: 250 cycles (200 MHz clock) = 1.25 us
Take-aways

- Recursion accounts for about 50% overhead; PLB (+ Compression) makes it almost free
- Ring ORAM achieves another 2x improvement
- Design challenges in hardware ORAM: computation, hashing, DRAM locality

Thank you! Questions?
Backup
What block size?

- Data block size $B\downarrow d$
- PosMap block size $B\downarrow p = \alpha + k\beta = \Theta(\log N)$

- Will use $B\downarrow p$ (using $B\downarrow d$ is suboptimal)
  - Break a data block into $[B\downarrow d / B\downarrow p]$ sub-blocks of size $B\downarrow p$
  - Treat them as separate blocks in backend, with position being
    - $PRF\downarrow K (GC | IC\downarrow j \mid j \mid t)$ where $t$ is sub-block index
- $(\leftfloor{B\downarrow d / B\downarrow p}\rightfloor + \log N / \log k)\Theta(\log \uparrow 2 N)1 / B\downarrow d = O(\log N + \log \uparrow 3 N / B\downarrow d \log \log N)$
  - $N \rightarrow N[B\downarrow d / B\downarrow p]$
Theoretical Construction

- $\beta = \log \log N$, $k = \log N / \log \log N$, $\alpha = \Theta(\log N)^{\alpha}$ bits
- $k/2^{\beta} = o(1)$, $\alpha/k + \beta = \log \log N < \log N$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Asymptotic bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursive Path</td>
<td>$O(\log N + \log \log N / B)$</td>
</tr>
<tr>
<td>+ Compression</td>
<td>$O(\log N + \log \log N / B \log \log N)$</td>
</tr>
</tbody>
</table>
Outline

Optimize recursive frontend

- Locality
- Compression

Caching (PLB)

- PRF
- Security holes

Unified ORAM
• Ring ORAM also performs well in oblivious storage
  – Online bandwidth: $\ell$ blocks $\rightarrow$ 1 block

• The XOR trick
  – $B, d_1, d_2, d_3, \ldots$
  – $E(B), E(d_1), E(d_2), E(d_3) \ldots$
  – $E(B, r), E(d_1, r_1), E(d_2, r_2), E(d_3, r_3) \ldots$
  – $E(B, r) \oplus E(d_1, r_1) \oplus E(d_2, r_2) \oplus E(d_3, r_3) \oplus \ldots$

  – A property not present in previous tree-based ORAMs
### Computation is cheap?

<table>
<thead>
<tr>
<th>Platform</th>
<th>Core clock</th>
<th>DRAM BW</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA</td>
<td>200 MHz</td>
<td>512 bits/cycle</td>
<td>1 block / cycle</td>
</tr>
<tr>
<td>ASIC</td>
<td>1~2 GHz</td>
<td>~ 64 bits/cycle</td>
<td>1 block / 8 cycles</td>
</tr>
</tbody>
</table>

#### Block size

= 512 bits

#### DRAM bandwidth

100Gbits / second

#### AES/SHA BW

128 bits / cycle