Path Oblivious-RAM for Secure Processors

2013/09/20  Hornet workshop

Ling Ren, Xiangyao Yu, Christopher W. Fletcher, Marten van Dijk, Srinivas Devadas
Outline

• Ascend Overview
• Basic Path ORAM
• Background eviction
• Integrity verification
• Summary
Privacy & Integrity in Cloud

- **Context**: cloud computing

- **Privacy**: user’s data not leaked to anyone

- **Integrity**: computation is done correctly (user gets $P(x)$)
Secure Processors

\[ E_K(x), P \]

\[ E_K(P(x)) \]

User data is decrypted and computed on inside processor

Data can be encrypted but address cannot

+ Integrity (e.g. Aegis)

– Leakage through address/timing/power
Leakage through Addresses

for i = 1 to N
  if (x == 0)
    sum += A[i]
  else
    sum += A[0]

Address sequence: 0x00, 0x01, 0x02 ...
Address sequence: 0x00, 0x00, 0x00 ...

• **Previous work** [HIDE, NDSS12] has shown access pattern leakage in practical applications

• Addresses can be monitored by software
• Existing secure processors (e.g., XOM, Aegis)
  + Can provide integrity
  - Leakage through address/timing/power, or trust the program

• Ascend: terminate leakage over above channels
  - I/O channel: Oblivious RAM
  - Timing channel: Chris’ and Xiangyao’ talks
  - Power Channel:
Oblivious RAM (ORAM)

• Hide access pattern
  – Read vs. write
  – Make all address sequences indistinguishable

• Naïve ORAM
  – Read/write the entire memory on each access
  – Probabilistic encryption → everything always changes
  – $O(N)$ overhead, $N =$ # of data blocks (cache lines) in the memory

Ascend

ORY\M controller

DRAM (encrypted)

scan the entire memory
• **Path ORAM**
  – One of the most efficient ORAMs, simple

• **External DRAM structured as a binary tree**
  – Each node contains $Z$ blocks ($Z = 2$ in the example below)
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>2</td>
</tr>
<tr>
<td>B3</td>
<td>2</td>
</tr>
<tr>
<td>B4</td>
<td>1</td>
</tr>
</tbody>
</table>

ORAM controller

Stash

(B4, 1)

Position Map

DRAM

root

(B3, 2)

(B0, 0)

dummy

path 0

1

2

3

(B2, 2)

dummy

(B1, 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)

\[ O(L) = O(\log N) \]

### ORAM controller

<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>

### 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>

### Stash (dummy)

- (B1, 1)
- (B4, 1)

### DRAM

- **root**: (B3, 0)
- **path 0**: (B0, 0)
- **path 1**: dummy
- **path 2**: dummy
- **path 3**: (B2, 3)
Path ORAM Security

• A random path is read/written on every access
  – Extracted from PosMap, which is always random and fresh due to remapping

• All ciphertexts on the path always change
  – Due to probabilistic encryption
Basic Path ORAM Problems

- **Stash overflow probability?**
  - Negligible (provably or empirically) if \( Z \geq 4 \)
  - Always overflow if \( Z \leq 3 \)
  - Prefer smaller \( Z \) because \( O(ZL) \)

- **Position map too large**
  - Recursive Path ORAM \( \rightarrow \) new problem: integrity

---

**Before:**

<table>
<thead>
<tr>
<th>Block</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Stash:** (B4, 1)

**After:**

<table>
<thead>
<tr>
<th>Block</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Stash:** (B1, 1) (B4, 1)

Background eviction prevents overflow
Outline

• Ascend Overview

• Basic Path ORAM

• Background eviction

• Integrity verification

• Summary
Background eviction

- When the stash is almost full, read/write a \textit{random} path
  - Hope there is a dummy and a block in stash can go there
  - If no dummy, all blocks can at least go back (stash will not increase)

### 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>

### Stash

- (B4, 1)

### ORAM controller

### DRAM

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

- dummy
  - path 0
  - 1
  - 2
  - (B1, 3)
Background eviction

- When the stash is almost full, read/write a **random** path
  - Hope there is a dummy and a block in stash can go there
  - Luckily, path 0 has a dummy
Background eviction

• Security: indistinguishable from normal accesses
  – Read/write a random path
  – No need to remap since no leaf label is exposed

• Impact on performance

![Graph showing impact of background eviction on latency]
Outline

• Ascend Overview

• Basic Path ORAM

• Background eviction

• Integrity verification

• Summary
• Recursive Path ORAM is **insecure** without integrity verification when attackers can modify ORAM
  – Revert PosMap ORAMs to force reuse of old leaf labels

• Another motivation: integrity in Ascend
  – Need to verify input/output and external memory

\[
E_K(x), \ P
\]

\[
Verify_K(s, \ P \ || \ x \ || \ r)
\]
Background – Merkle Signature

- General, can be used for any document, any ORAM
- Efficient $O(L) = O(\log N)$
- Security reduced to collision-resistant hash function

Any Document:

- Chunk 0
- Chunk 1
- Chunk 2
- Chunk 3

Top hash

Hash 4

Hash 0
Hash 1

Hash 5

Hash 2
Hash 3
Merkle Signature for Path ORAM?

• ORAM hides access pattern
  → (pretend to) verify all buckets on a path
  → $O(L^2)$ complexity
    – Path ORAM $O(L)$ complexity

Path ORAM

Top hash
Verify Path ORAM

• Combine Merkle tree and Path ORAM tree
Verify Path ORAM

Authentication Tree

Hash 0

Hash 1

Hash 3

Hash 4

Hash 2

Hash 5

Hash 6

Bucket 0

Bucket 1

Bucket 3

Bucket 4

Bucket 2

Bucket 5

Bucket 6

Path ORAM Tree

$O(L)$
Outline

• Ascend Overview
• Basic Path ORAM
• Background eviction
• Integrity verification
• Summary
Summary

• Background eviction prevents stash overflow, enables smaller $Z$, and improve performance by 20%

• Overhead relative to DRAM
  – Bit movement: ~300x
  – Latency: ~50x (assuming 2 channels)
  – On SPEC: 1x ~ 10x

• Integrity verification adds 17% latency on top of recursive Path ORAM

• We are designing the on-chip Path ORAM controller!