The Ascend Secure Processor

Christopher Fletcher
MIT
Joint work with

• Srini Devadas, Marten van Dijk
• Ling Ren, Albert Kwon, Xiangyao Yu
• Elaine Shi & Emil Stefanov
• David Wentzlaff & Princeton Team (Mike, Tri, Jonathan, Alexey, Yaosheng)
• Omer Khan
Last talk: Intel SGX
Ascend Processor

Memory controller

This talk

Data Integrity
Address Timing
Outline

• Motivation + Oblivious RAM (ORAM) primer

• ORAM in Hardware

• Demo 😊
If (secret variable) {
    ...
    scan memory ...
}

### Binary search

<table>
<thead>
<tr>
<th>Op</th>
<th>Address</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>W</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>R</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>R</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>R</td>
<td>7</td>
<td>17</td>
</tr>
</tbody>
</table>

• SGX broken through page faults  [Xu et al.’15]
• Shared library usage           [Zhuang et al.’04]
• Search queries                 [Islam et al.’12]
Oblivious RAM (ORAM) [Goldreich-Ostrovsky’96]

On-chip

Cache miss

ORAM Controller

Chip pins

Provably removes all access pattern leakage
ORAM security definition

• Access is 3 tuple: \( \text{op} = \text{Read/write}, \, \text{address}, \, \text{data} \)

• Consider access sequences \( A \) and \( A' \)
  \[
  A = [ (\text{op}_1, \text{address}_1, \text{data}_1), \ldots ] \\
  A' = [ (\text{op}_1', \text{address}_1', \text{data}_1'), \ldots ]
  \]

• If \( |A| = |A'| \)
  then \( \text{ORAM}(A) \approx \text{ORAM}(A') \)
Path ORAM [CCS’13]

ORAM Controller (on-chip)

Chip pins

Read/writes

“The ORAM”
Block assigned to *random* path.

Block *lives on* that path.

![Diagram of PosMap and Off-chip block assignment](image)
The diagram illustrates a PosMap with chip pins labeled as A, 2 and B, 3. It shows a binary tree structure with the following nodes:

- Off-chip: B, 3
- Path 1: A, 2 (dummy)
- Path 2: dummy
- Path 3: dummy
- Path 4: dummy

The diagram also illustrates the concept of encrypted and not encrypted data, with empty space indicating dummy encryptions.
Path ORAM Access:

*Read+write the path the block is assigned to.*
Path ORAM Access:
*Read+write the path the block is assigned to.*
Typically, 4 slots per bucket “Z=4”

...for simplicity

Z=1
Too big!

A, 4
B, 3

Off-chip

B, 3

A, 4
Map recursion [Shi et al., 11]

PosMap ORAM

Block

PosMap ORAM 2

On-chip

Map'

Smaller

Small enough

A, 4

B, 3
Map recursion [Shi et al., 11]
Path ORAM summary

Blocks assigned to paths.

Access block: Read+write path.

Adversary sees: random paths.
ORAM in Hardware
Ascend in silicon

• Collaboration with David Wentzlaff’s group @ Princeton
First silicon fully functional @ 500 MHz & .9 V

Design (Verilog) Open Source
Blocks must live on assigned path or in stash.

PosMap

A, 4
B, 2

Stash

A, 4
B, 2

Path

1
2
3
4

Can overflow

Off-chip
Blocks must live on assigned path or in stash.

Bottleneck in prior work [Maas et al. ‘13] Causes $3 \times$ avg. slowdown on SPEC.
def evict(Path_block, Path_evict, occ):
    t_1 = Path_block ⊕ Path_evict
    t_2 = bit_reverse( ( (t_1 ∧ −t_1) − 1) ∧ occ )
    ret = bit_reverse( t_2 ∧ −t_2 )

≈ greatest common prefix
Simple design, no performance bottleneck.
Integrity protection for ORAM

Cache miss

ORAM

Overlay Hash tree

Shuffled
ORAM logic  Test harness  SHA-3

One SHA-3 unit

FPGA Prototype
Cheap Integrity Scheme [ASPLOS’15]

• Per-block MAC

\[ \{ \text{Block data, Hash(Block data, Block addr, counter)} \} \]

• **Good:** Hash 1 block, **NOT** path
• **Bad:** Need to store counters on-chip

Replace entries in map with counters!
Want: Path $P = \text{PosMap}[A]$

Algorithm:

Given $A$: derive $A'$, $A''$, $A'''$

$P' = \text{PRF}(A' \| \text{PosMap}[A'] = \text{Counter})$

$P'' = \text{PRF}(A'' \| \text{ORAMAccess}(A'', P'))$

$P = \text{PRF}(A''' \| \text{ORAMAccess}(A''', P''))$

$\text{Data} = \text{ORAMAccess}(A, P)$

Problem: $|C| > |P|$

More schemes to get $|C| < |P|$
Cheap Integrity Scheme [ASPLOS’15]

Result:

Hashing decreased by 68 X, simple design
ORAM randomizes data layout.

Computer architecture assumes *data locality*.
# Row misses:

\[
\sim \text{tree height} \\
\sim \frac{\text{tree height}}{\text{subtree height}}
\]
Row misses:

60% overhead $\rightarrow$ 13% overhead
Encryption, Stash, Recursion, PLB, Integrity

2 mm

.5 mm

460M transistors

AES rounds

Stash evict()

Hash unit

Recursion, PLB, Integrity

Encryption

Hash unit

ORAM

PLL

Tile 0
Tile 1
Tile 2
Tile 3
Tile 4

Tile 5
Tile 6
Tile 7
Tile 8
Tile 9

Tile 10
Tile 11
Tile 12
Tile 13
Tile 14

Tile 15
Tile 16
Tile 17
Tile 18
Tile 19

Tile 20
Tile 21
Tile 22
Tile 23
Tile 24
Slowdown vs. insecure

2 DRAM channels, In-order core, 2-level cache hierarchy, 1 MByte last-level cache
ORAM = 1208 cycles / tree lookup
Demo
Backup