A HIGH-PERFORMANCE OBLIVIOUS RAM CONTROLLER ON THE CONVEY HC-2EX HETEROGENEOUS COMPUTING PLATFORM

BASED ON “PHANTOM: PRACTICAL OBLIVIOUS COMPUTATION IN A SECURE PROCESSOR” FROM CCS-2013

Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari
Elaine Shi, Krste Asanovic, John Kubiatowicz, Dawn Song

Aspire

Berkeley
University of California
A HIGH-PERFORMANCE OBLIVIOUS RAM CONTROLLER ON THE CONVEY HC-2EX HETEROGENEOUS COMPUTING PLATFORM

Based on “PHANTOM: Practical Oblivious Computation in a Secure Processor” from CCS-2013

Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari
Elaine Shi, Krste Asanovic, John Kubiatowicz, Dawn Song

Cryptographic Construct

High-performance, FPGA-based platform

Secure Processor
Organizations move to the cloud

E.g. government, financial/medical companies

Raises privacy concerns for sensitive data
Attackers with Physical Access

Malicious Employees

Intruders

Government Surveillance
Physical Attack Vectors

E.g. replace **DRAM DIMMs** with NVDIMMs that have non-volatile storage to **record accesses**
Computation on Encrypted Data

- Sealed (tamper-proof), Remote attestation

**Diagram:**
- CPU
- DRAM
- Hard Drive, etc.

- e.g. Secure Processors (AEGIS, XOM, AISE-BMT), IBM Cryptographic Coprocessors, **Intel SGX**
Memory Address Leakage

Leaks e.g. transactions, subjects of surveillance/audit, geolocations, OS fingerprints, *crypto keys*

Sealed (tamper-proof), Remote attestation

CPU

Memory Addresses (plaintext)

DRAM

Hard Drive, etc.

encrypted

encrypted
A real-world example: SQLite

SELECT population FROM all counties in <Texas | California>
We want to prevent this information leakage
In the context of a secure processor
Oblivious RAM (ORAM)

- Problem investigated since 1987
- Originally for memory accesses of a processor, later for e.g. FSs, DBs, ...
- Algorithms required MBs of trusted storage or complex (× Hardware)
Path ORAM (CCS’13, Best Paper)

New algorithm by Stefanov et al.

- Low trusted storage requirement
- Simple enough to implement in hardware on a secure processor
Where’s the problem?

How hard can it be to put Path ORAM into a processor?
1. ORAM Microarchitecture

• Prior work algorithmic, ignores ORAM microarchitectural implementation
• ORAM needs to fully utilize resources
• You need to build it to find the details not apparent from the algorithm
2. Practicality on real system

- Want obliviousness for real systems
- Custom chips (ASICs) very expensive unless widely adopted
- There is a trend towards FPGA-based accelerators (programmable H/W)
PHANTOM: A Practical Oblivious Computing Platform
Featuring an ORAM microarchitecture implemented on an FPGA platform
Overview

1. Overview & Attack Model
2. Path Oblivious RAM
3. The Oblivious Memory System
4. Building PHANTOM
5. Evaluation
PART I
Attack Model and Deployment
PHANTOM Overview

Client → Data Center

Remote Attestation

Custom Application Logic/
Secure CPU

Oblivious Memory System

Untrusted Memory
Our focus: Memory Traffic (data, addresses,...)

Remote Attestation

Custom Application Logic/
Secure CPU

Oblivious Memory System

Attack Model

Orthogonal

Other digital (OS,...)

Analog (radiation, power,...)
PART II
Path Oblivious RAM
Path Oblivious RAM

Oblivious memory is divided into blocks:

```
A B C D E F ...
```

When accessing a block through Path ORAM

- Request block
- Read/write to requested block
- Path ORAM (Confidential state)
- Random appearing DRAM accesses
Path Oblivious RAM

Position Map (secure)

<table>
<thead>
<tr>
<th>Block ID</th>
<th>Leaf ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>101 011</td>
</tr>
<tr>
<td>B</td>
<td>011</td>
</tr>
<tr>
<td>C</td>
<td>000</td>
</tr>
<tr>
<td>D</td>
<td>010</td>
</tr>
<tr>
<td>E</td>
<td>101</td>
</tr>
<tr>
<td>F</td>
<td>010</td>
</tr>
</tbody>
</table>

Stash (secure)
Required Stash Size

• Blocks stay behind in the stash

• How large does the stash have to be to never overflow?

• Bound known up to constant factors: determined constants empirically
PART III
The Oblivious Memory System
The Oblivious Memory System
High-throughput Memory

Challenge:

- Design
  - Basic 128bit: 34816 Cycles
  - 8x Memory BW: 4352 Cycles

Note: 1,000 cycles = 6.6us @ 150 MHz
(ORAM Size 1GB, 17 level tree, 4KB blocks)

AES counter mode
High-throughput Memory

Challenge: Keep up with memory

Note: 1,000 cycles = 6.6us @ 150 MHz
(ORAM Size 1GB, 17 level tree, 4KB blocks)

<table>
<thead>
<tr>
<th>Design</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic 128bit</td>
<td>34816</td>
</tr>
<tr>
<td>8x Memory BW</td>
<td>4352</td>
</tr>
</tbody>
</table>
For each node in the path, select an entry from the stash to write to it (or put a dummy).
Time to pick a block

- In our case, we have 32 cycles to pick the next block (otherwise we will stall the memory system).
- Examining all blocks takes C cycles for each block.
Picking from the full stash

**Challenge:**
Keep up with memory

**Note:** 1,000 cycles = 6.6μs @ 150 MHz
(ORAM Size 1GB, 17 level tree, 4KB blocks)

<table>
<thead>
<tr>
<th>Design</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic 128bit</td>
<td>34816</td>
</tr>
<tr>
<td>8x Memory BW</td>
<td>4352</td>
</tr>
<tr>
<td></td>
<td>10880</td>
</tr>
</tbody>
</table>

C cycles to select next block to write back, C = 128
Adding a sorting step

Note: 1,000 cycles = 6.6us @ 150 MHz
(ORAM Size 1GB, 17 level tree, 4KB blocks)

Challenge:
Keep up with memory
Heap-based Sorting

Challenge:

Keep up with memory

Design | Cycles
---|---
Basic 128bit | 34816
8x Memory BW | 4352
C log C Sorting | 5248
Fully overlap | 4352

Note: 1,000 cycles = 6.6us @ 150 MHz
(ORAM Size 1GB, 17 level tree, 4KB blocks)
Timing Channels

Operation is data-driven; risk to leak information from timing

1. Operation always take the maximum amount of time (avoiding large overheads) or are overlapped
2. Decouple DRAM timing variations

Challenge: Side Channels
DRAM Buffer

Absorb timing variations at periphery

Trust Boundary

Timing isolation

DRAM

DRAM Buffer

Memory Request Generation Logic

All other PHANTOM state (Stash, Sorter, AES Units)

Address of path to access

✔ Challenge: Side Channels
The Whole Picture

More details can be found in the paper
PART IV
Building PHANTOM
PHANTOM Prototype

Implemented on Convey HC-2ex platform
Integrated with RISC-V CPU

Developed by UC Berkeley’s Architecture Group
PHANTOM Secure Processor

- Integrated a **RISC-V CPU** with ORAM
- Loads and runs real-world programs, including (in-memory) **SQLite**
- Not optimized for FPGA yet, very small cache sizes (4KB/4KB/8KB)
Implementation on the HC-2ex

- Use Convey development kit, bundles Convey and user logic into personality
- Implement Verilog module, interfaces with MCs, management unit, etc.
- Personality loaded by Convey runtime
Convey Personality Workflow

Using this to build **two-way communication channel**
Interaction with RISC-V CPU

RISC-V CPU runs independently but talks to host.

- **Management Processor Glue**
  - Write values to shared registers
  - Blocking caep

- **Front-end Server**
  - Load programs
  - Execute sys-calls for the RISC-V CPU
ORAM Microarchitecture

- Fully implemented, except remote attestation and AES units
- ORAM controller tested/verified for millions of random ORAM accesses
- ORAM Block Size of 4KB (for now)
Implementation Challenges

• Many challenges and unknown details
• Min-heap, BRAM multiplexing, block headers, stash management, block caching, timing domains, inter-FPGA communication, block buffering,...
Min-heap Implementation

Need to write and look at **two children** at every step, running at 150 Mhz

Pre-fetch four grandchildren to avoid long combinational path (read and write to BRAM in the same cycle)

Split into multiple BRAMs to avoid limitation to 2 ports
Synthesized FPGA Design

Virtex-6 LX760 FPGA
PART V
Evaluation
ORAM Access times

Average of 1M ORAM accesses each (4KB)

Row 1: ORAM size in levels (64MB-4GB) - Row 2: # cached levels (k)

Time per ORAM access (us)

4719 cycles

Total access time

4352 cycles

0.812us

Time for read phase
Application Performance

<table>
<thead>
<tr>
<th>Application</th>
<th>Execution Time normalized to no ORAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlite-query3</td>
<td></td>
</tr>
<tr>
<td>sqlite-query2</td>
<td></td>
</tr>
<tr>
<td>sqlite-query1</td>
<td></td>
</tr>
<tr>
<td>sqlite-warmup</td>
<td></td>
</tr>
</tbody>
</table>

- **Simulation (Caches: 16KB/32KB/1MB)**
- **FPGA Prototype (Caches: 4KB/4KB/8KB)**
- **No ORAM**

20%-5.5x Overhead (1MB LLC)  1GB ORAM
Future Work

- Prototype is a starting point
- Integrate additional Path ORAM optimizations, HW/SW co-design
- Compiler/OS support to avoid ORAM accesses and reduce size of ORAMs
Conclusion

- Investigated ORAM microarchitecture to exploit high memory bandwidth

Phantom: Make oblivious computation practical on existing hardware
Thank you! Any Questions?

Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari
Elaine Shi, Krste Asanovic, John Kubiatowicz, Dawn Song

{maas, ericlove, emil}@eecs.berkeley.edu, tiwari@austin.utexas.edu,
elaine@cs.umd.edu, {krste, kubitron, dawnsong}@eecs.berkeley.edu