Deconstructing New Cache Designs for Thwarting Software Cache-based Side Channel Attacks

Jingfei Kong*  
Onur Acicimez  
Jean-Pierre Seifert  
Huiyang Zhou  

University of Central Florida  
Samsung Electronics  
Samsung Electronics  
University of Central Florida
Outline

• Background (more details are in related papers)
• Security evaluation on previously proposed secure cache designs
• Potential solutions and conclusion
What are software cache-based side channel attacks?

• Side channel attacks
  – Exploit any observable information generated as a byproduct of the cryptosystem implementation
  – Infer the secret information

• Cache implementation of modern general-purpose microprocessors
  – A hardware component sits between the CPU pipeline and main memory
  – Take advantage of spatial and temporal locality to minimize communication latency

• Software-managed observation
  – No need for physical possession of the attacked system
Cache design of modern general-purpose microprocessors

Requested address
Tag bits | Cache set index bits | Offset bits

Cache set array
Tag | Cache Block
Tag | Cache Block

CPU
"on-board"
registers
data cache
Instruction cache
Cache miss
unified data+inst.
L2 cache

physical memory
(e.g., SIMMs)

disk
holds virtual memory

OS page fault handler

Page faults
The case of Advanced Encryption Standard (AES)

\[
\begin{align*}
    x_{0}^{i+1}, x_{1}^{i+1}, x_{2}^{i+1}, x_{3}^{i+1} \\
    x_{4}^{i+1}, x_{5}^{i+1}, x_{6}^{i+1}, x_{7}^{i+1} \\
    x_{8}^{i+1}, x_{9}^{i+1}, x_{10}^{i+1}, x_{11}^{i+1} \\
    x_{12}^{i+1}, x_{13}^{i+1}, x_{14}^{i+1}, x_{15}^{i+1}
\end{align*}
\]

\[
\begin{align*}
    T_{0} [x_{0}^{i}] \oplus T_{1} [x_{5}^{i}] \oplus T_{2} [x_{10}^{i}] \oplus T_{3} [x_{15}^{i}] \oplus \{k_{0}^{i}, k_{1}^{i}, k_{2}^{i}, k_{3}^{i}\} \\
    T_{0} [x_{4}^{i}] \oplus T_{1} [x_{9}^{i}] \oplus T_{2} [x_{14}^{i}] \oplus T_{3} [x_{3}^{i}] \oplus \{k_{4}^{i}, k_{5}^{i}, k_{6}^{i}, k_{7}^{i}\} \\
    T_{0} [x_{8}^{i}] \oplus T_{1} [x_{13}^{i}] \oplus T_{2} [x_{2}^{i}] \oplus T_{3} [x_{7}^{i}] \oplus \{k_{8}^{i}, k_{9}^{i}, k_{10}^{i}, k_{11}^{i}\} \\
    T_{0} [x_{12}^{i}] \oplus T_{1} [x_{1}^{i}] \oplus T_{2} [x_{6}^{i}] \oplus T_{3} [x_{11}^{i}] \oplus \{k_{12}^{i}, k_{13}^{i}, k_{14}^{i}, k_{15}^{i}\}
\end{align*}
\]
**Access-driven attacks**

- The attacker does not have direct access to the victim process’ memory address space
- The attacker controls one or multiple process(es) which share the cache with the victim process

![Cache Diagram](image)
Time-driven attacks

- The attacker sends the encryption request to the victim process and upon response records the execution time
Why hardware support against software cache-based attacks?

- Generic
- Secure against both attacks
- Performance efficient
Partition Locked Cache (PLcache) & Random Permutation Cache (RPcache)

PLcache

<table>
<thead>
<tr>
<th>L</th>
<th>ID</th>
<th>Original Cache Block</th>
</tr>
</thead>
</table>

• Control interface
  - ld.lock/ld.unlock and st.lock/st.unlock
  - lock_mem_region(addr, length)
  - unlock_mem_region(addr, length)

Previous work from Wang et.al at International Symposium on Computer Architecture (ISCA) 2007

University of Central Florida
Simulation Environment

<table>
<thead>
<tr>
<th>Superscalar Core</th>
<th>7-stage pipeline: Fetch/Dispatch/Issue/RegisterRead/EXE/WriteBack/Retire</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fetch/Dispatch/Issue/ MEM issue/Retire Bandwidth: 4</td>
</tr>
<tr>
<td></td>
<td>Fully-symmetric Function Units: 4</td>
</tr>
<tr>
<td></td>
<td>Reorder Buffer size: 64</td>
</tr>
<tr>
<td></td>
<td>Issue Queue Size: 32</td>
</tr>
<tr>
<td></td>
<td>Load Store Queue Size: 32</td>
</tr>
<tr>
<td>Execution Latencies</td>
<td>Address Generation: 1 cycle</td>
</tr>
<tr>
<td></td>
<td>Memory Access: 2 cycles</td>
</tr>
<tr>
<td></td>
<td>(hit in data cache)</td>
</tr>
<tr>
<td>Instruction Cache</td>
<td>Integer ALU ops: 1 cycle</td>
</tr>
<tr>
<td></td>
<td>Complex ops: MIPS R10000 latencies</td>
</tr>
<tr>
<td>L1 Data Cache</td>
<td>32KB 2-way, Block size: 64B</td>
</tr>
<tr>
<td></td>
<td>10-cycle miss penalty</td>
</tr>
<tr>
<td>L2 Unified Cache</td>
<td>2MB 16-way, Block size: 64B</td>
</tr>
<tr>
<td></td>
<td>300-cycle miss penalty</td>
</tr>
</tbody>
</table>
Security Evaluation of PLcache

- Initial loading exposure of PLcache

- Potential vulnerability to denial of service attacks
Security Evaluation of RPcache

- Vulnerable to latest cache-collision timing attacks

![Graph showing timing deviation and number of samples](image)
Possible solutions and future work

• Secure PLcache
  – Using locking mechanism to do the preloading first before cryptographic operations take place
  – Allow the locked cache lines to be used by other processes when the owner process is inactive

• Secure RPcache
  – Preloading only supported from OS is not enough
  – The best possible solution is to detect the event of cache misses in hardware and respond to the event in software
Summary

• Latest software cache-based side channel attacks are emerging threats
• Current secure cache designs, although providing generic security protection and incurring small performance overhead, are still missing some important security issues
Thank you!

Questions?