Attack of the Knights: Side-Channel Attack on Non-Uniform Cache Architecture

Farabi Mahmud, Sungkeun Kim, Harpreet Singh Chawla, EJ Kim, Chia-Che Tsai, Abdullah Muzahid
Texas A&M University, College Station, TX
Non-Uniform Cache Architecture is Everywhere!

Most of the servers and datacenters use multicore processors

<table>
<thead>
<tr>
<th>AWS Instance Type</th>
<th>M6i/M6id</th>
<th>M5zn</th>
<th>M5n</th>
<th>M5 (Burstable)</th>
<th>T3 (Burstable)</th>
<th>T2 (Burstable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Processor</td>
<td>3rd Gen Intel® Xeon® Scalable</td>
<td>2nd Gen Intel® Xeon® Scalable Processors</td>
<td>2nd Gen Intel® Xeon® Scalable Processors</td>
<td>Intel® Xeon® Platinum 8175M Processors</td>
<td>Intel® Xeon® Scalable Processors</td>
<td>Intel® Xeon® Processors</td>
</tr>
<tr>
<td>AWS Instance Type</td>
<td>DL1</td>
<td>VT1</td>
<td>P4</td>
<td>G4</td>
<td>P3</td>
<td></td>
</tr>
<tr>
<td>Intel® Processor</td>
<td>2nd Gen Intel® Xeon®</td>
<td>2nd Gen Intel® Xeon®</td>
<td>2nd Gen Intel® Xeon®</td>
<td>Intel® Xeon®</td>
<td>Intel® Xeon®</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AWS Instance Type</th>
<th>R6i/R6id</th>
<th>X2idn/X2iedn</th>
<th>X2iezn</th>
<th>R5b</th>
<th>R5n</th>
<th>R5/R5d</th>
<th>X1e/X1</th>
<th>Z1d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Processor</td>
<td>3rd Gen Intel® Xeon®</td>
<td>3rd Gen Intel® Xeon® Scalable</td>
<td>2nd Gen Intel® Xeon®</td>
<td>2nd Gen Intel® Xeon®</td>
<td>2nd Gen Intel® Xeon®</td>
<td>Intel® Xeon® E7 8880 v3</td>
<td>Intel® Xeon® Platinum</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AWS Instance Type</th>
<th>C6i/C6id</th>
<th>C5</th>
<th>C5n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Processor</td>
<td>3rd Gen Intel® Xeon® Scalable Processors</td>
<td>2nd Gen Intel® Xeon® Scalable Processors</td>
<td>Intel® Xeon® Platinum 8124M Processors</td>
</tr>
</tbody>
</table>
Non-Uniform Cache Architecture is Everywhere!

Most of the servers and datacenters use multicore processors.

Multiple cores are connected via different types of communication networks.

Exploiting Secret dependent communication can be fun (and $$$)
NUCA Architecture

Every Core has -
• Private Cache
• Shared LLC (L2 Cache)
NUCA Architecture

Every Core has -
• Private Cache
• Shared LLC (L2 Cache)
NUCA Architecture

Every Core has -
- Private Cache
- Shared LLC (L2 Cache)

4 x 4 configuration
- Connected via 2D mesh network
- Data can be available at one or multiple L2 Cache locations
NUCA Architecture

Every Core has -
• Private Cache
• Shared LLC (L2 Cache)

4 x 4 configuration
• Connected via 2D mesh network
• Data can be available at one or multiple L2 Cache locations

LLC Hit Timing is dependent on LLC Location -
• Nearer L2 Cache accesses are faster
• Farther L2 Cache accesses are slower
Every Core has -
• Private Cache
• Shared LLC (L2 Cache)

4 x 4 configuration
• Connected via 2D mesh network
• Data can be available at one or multiple L2 Cache locations

LLC Hit Timing is dependent on LLC Location -
• Nearer L2 Cache accesses are faster
• Farther L2 Cache accesses are slower
NUCA Architecture

Nearer L2 Cache accesses are faster

Farther L2 Cache accesses are slower
Cache Side-Channel Attacks on NUCA Architecture

DON’T MESH AROUND [USENIX’22]

MESHUP [S&P’22]

LORD OF THE RING(S) [USENIX’21]

ADVERSARIAL PREFETCH [S&P’21]
Existing Cache Side-Channel Attacks

Prime + Probe
[Crypto’06]
Existing Cache Side-Channel Attacks

Prime + Probe
[Crypto’06]
Existing Cache Side-Channel Attacks

Prime + Probe
[Crypto’06]
Existing Cache Side-Channel Attacks

Prime + Probe
[Crypto’06]
Existing Cache Side-Channel Attacks

Prime + Probe [Crypto’06]

Don’t Mesh Around [USENIX’22]
Existing Cache Side-Channel Attacks

Prime + Probe [Crypto’06]

Victim

Attacker

Probe

Don’t Mesh Around [USENIX‘22]

Victim

Attacker

L2 Cache 0

L2 Cache 1

L2 Cache 2

L2 Cache 3

L2 Cache 4

L2 Cache 5

L2 Cache 6

L2 Cache 7

L2 Cache 8

L2 Cache 9

L2 Cache 10

L2 Cache 11

L2 Cache 12

L2 Cache 13

L2 Cache 14

L2 Cache 15

Attack of the Knights: Farabi Mahmud et al.
Existing Cache Side-Channel Attacks

Prime + Probe
[Crypto’06]

Don’t Mesh Around
[USENIX’22]

Attack of the Knights: Farabi Mahmud et al.
Target Architecture – Intel Xeon Phi
Target Architecture – Intel Xeon Phi

CHA = Caching and Homing Agent
- Part of the Tag Directory
- Tracks Locations
Target Architecture – Intel Xeon Phi

CHA = Caching and Homing Agent
• Part of the Tag Directory
• Tracks Locations

Core 4 requests Line A
• Request Goes to CHA A
• CHA Forwards Request to LLC hosting Line A
• LLC hosting Line A sends Response to Core 4
Target Architecture – Intel Xeon Phi

CHA = Caching and Homing Agent
- Part of the Tag Directory
- Tracks Locations

Core 4 requests Line A
- Request Goes to CHA A
- CHA Forwards Request to LLC hosting Line A
- LLC hosting Line A sends Response to Core 4

Core 0 requests Line A
- Request Goes to CHA A
- CHA Forwards Request to LLC hosting Line A
- LLC Hosting Line A sends Response to Core 0
Target Architecture – Intel Xeon Phi

CHA = Caching and Homing Agent
- Part of the Tag Directory
- Tracks Locations

Core 4 LLC hit on Cacheline A is Faster than Core 0 LLC hit on Cacheline A

Core 4 requests Line A
- Request Goes to CHA A
- CHA Forwards Request to LLC hosting Line A
- LLC hosting Line A sends Response to Core 4

Core 0 requests Line A
- Request Goes to CHA A
- CHA Forwards Request to LLC hosting Line A
- LLC Hosting Line A sends Response to Core 0
LLC Hit Timing Depends on Physical Location
Attack on Intel Xeon Phi: Setup

**Victim**
- AES Decrypt
- Contains Secret Key
- Accesses Near/Far Tile based on Secret

**Attacker**
- Observe timing of Victim
- Does not have access to Secret
- Can run multiple iterations
Vulnerable Access Patterns in AES

AES has many decryption tables (Td tables) for improving performance

By default, no-asm AES use these tables now

Commit

aes: make the no-asm constant time code path not the default
After OMC and OTC discussions, the 95% performance loss resulting from the constant time code was deemed excessive for something outside of our security policy.

The option to use the constant time code exists as it was in OpenSSL 1.1.1.

Reviewed-by: ""
(Merged from #17600)

openssl-3.0 (#15786) + openssl-3.1
Vulnerable Access Patterns in AES

AES has many decryption tables (Td tables) for improving performance

By default, no-asn AES use these tables now

Last round of decryption use part of secret key rk[0] and has secret dependent memory accesses

Output of last round AES decrypt is plaintext

/* The last round */

\[
\begin{align*}
    s0 &= ((u32)Td4[(t0 >> 24)] << 24) ^ \\
        & \quad ((u32)Td4[(t3 >> 16) & 0xff] << 16) ^ \\
        & \quad ((u32)Td4[(t2 >> 8) & 0xff] << 8) ^ \\
        & \quad ((u32)Td4[(t1 & 0xff]) \) \) ^ \\
        & \quad rk[0]; \\
    \end{align*}
\]

PUTU32(out, s0);

\[
\begin{align*}
    s1 &= ((u32)Td4[(t1 >> 24)] << 24) ^ \\
        & \quad ((u32)Td4[(t0 >> 16) & 0xff] << 16) ^ \\
        & \quad ((u32)Td4[(t3 >> 8) & 0xff] << 8) ^ \\
        & \quad ((u32)Td4[(t2 & 0xff]) \) \) \) ^ \\
        & \quad rk[1]; \\
    \end{align*}
\]

PUTU32(out + 4, s1);
Attack Steps: Generate Keys
Attack Steps: Measure Known Data
Attack Steps: Train AdaBoost
Attack Steps: Allow Victim Access
Attack Steps: Measure Victim Access Time
Attack Steps: Classify Victim Access Time
Attack Steps: Recover Key
**Decryption function contains many memory accesses**

- Many Td table accesses are made
- End-to-end timing of Decrypt contains a lot of noise

**Fine Grained Timing utilizes access to shared buffer**

- Allows more precise measurement of LLC Hit Latency
- Only monitor accesses to Td4 table
- Access the unprotected buffer (out buffer)

```c
/* The last round */
s0 =
  (((u32)Td4[(t0 >> 24)] << 24) ^
   (((u32)Td4[(t3 >> 16) & 0xff] << 16) ^
    (((u32)Td4[(t2 >> 8) & 0xff] << 8) ^
      (((u32)Td4[(t1) & 0xff]) ^
        rk[0];
  PUTU32(out, s0);

s1 =
  (((u32)Td4[(t1 >> 24)] << 24) ^
   (((u32)Td4[(t0 >> 16) & 0xff] << 16) ^
    (((u32)Td4[(t3 >> 8) & 0xff] << 8) ^
      (((u32)Td4[(t2) & 0xff]) ^
        rk[1];
  PUTU32(out + 4, s1);
```
/* The last round */

\[
\begin{align*}
\text{s0} &= ((u32)\text{Td4}[t0 >> 24] & \text{<< 24} & \text{^} \\
& & ((u32)\text{Td4}[t3 >> 16] & \text{& 0xff} & \text{<< 16} & \text{^} \\
& & ((u32)\text{Td4}[t2 >> 8] & \text{& 0xff} & \text{<< 8} & \text{^} \\
& & ((u32)\text{Td4}[t1] & \text{& 0xff}) & \text{^} \\
& & \text{rk[0];} \\
\text{PUTU32(out, s0);} \\

\text{s1} &= ((u32)\text{Td4}[t1 >> 24] & \text{<< 24} & \text{^} \\
& & ((u32)\text{Td4}[t0 >> 16] & \text{& 0xff} & \text{<< 16} & \text{^} \\
& & ((u32)\text{Td4}[t3 >> 8] & \text{& 0xff} & \text{<< 8} & \text{^} \\
& & ((u32)\text{Td4}[t2] & \text{& 0xff}) & \text{^} \\
& & \text{rk[1];} \\
\text{PUTU32(out + 4, s1);} 
\end{align*}
\]
Experiment Setup

Configuration Parameters

- Intel Xeon Phi 7290 CPU
- Cluster set to All-to-all configuration
- MCDRAM set as part of the memory

Side Channel Attack

- Number of different plaintexts \([2, 2^{20}]\)
- Number of trials for same plaintext \([1, 100]\)

Covert Channel Attack

- Payload Size \([2^0, 2^{17}]\)
Side-Channel Results: Key Extraction Accuracy

We can extract 4 bytes of any random key with 100% accuracy by using only \( \approx 4000 \) trials.
Side-Channel Attack Result: ML Model Accuracy

100% accuracy for >40 samples for each plaintext using AdaBoost
Covert Channel Bandwidth & Error Rates

Max 0.02% error rate with 205 KBPS bandwidth
Generalizability: Beyond Xeon Phi

Intel Xeon SP Scalable and Intel Core processors have mesh network

Similar latency distribution found in Intel 10700k with 16 cores from Comet Lake processor family

Similar vulnerabilities may exist in other mesh network processors
Conclusions

- Implemented Covert Channel Intel Xeon Phi 7290
  - 205KBPS data bandwidth & 0.2% Error Rate
- Implemented Side-channel in Intel Xeon Phi 7290
  - 4000 trials to get 4 bytes of AES key with 100% accuracy
- Other processors with mesh network might be vulnerable
- Other cryptographic algorithms with similar T-table might be vulnerable

Artifacts Available: https://github.com/farabimahmud/aok_ae

Farabi Mahmud
Texas A&M University
farabi@tamu.edu
www.farabimahmud.com
Extra Slides
Allocate Large array to LLC

Let Victim Make LLC Hit

Measure Latency of LLC Hit

Classify LLC Hit to HIGH vs LOW

Identify HIGH vs LOW accesses
<table>
<thead>
<tr>
<th>Using Cache replacement</th>
<th>Attacker run on a separate core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bring data to attacker’s core before the Victim is allowed to execute</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Using PREFETCHW Instruction</th>
<th>Attacker run on a separate core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalidate L1D cache of the victim core with PREFETCHW from remote core</td>
<td></td>
</tr>
</tbody>
</table>
Two Key Requirements

Vulnerable T-table implementation that access different memory location based on secret key

Fine-grained timing channel that allows running timer thread based on a shared variable with AES engine
Out of Order CPUs can issue multiple load into pipeline

Latency of multiple loads can overlap

Highest latency load will dominate overall latencies

/* The last round */
s0 =
((u32)Td4[(t0 >> 24)] << 24)
^ ((u32)Td4[(t3 >> 16) & 0xff] << 16)
^ ((u32)Td4[(t2 >> 8) & 0xff] << 8)
^ ((u32)Td4[(t1 & 0xff])
  rk[0];
PUTU32(out, s0);
s1 =
((u32)Td4[(t1 >> 24)] << 24)
^ ((u32)Td4[(t0 >> 16) & 0xff] << 16)
^ ((u32)Td4[(t3 >> 8) & 0xff] << 8)
^ ((u32)Td4[(t2 & 0xff])
  rk[1];
PUTU32(out + 4, s1);
Problem

• Attacker can only monitor the timing of Decrypt function
• Decrypt contains multiple rounds of Td4 table usages

Solution:

• Use multiple trials and AdaBoost algorithm to decide
1. Attacker Generate N ciphertext and key pairs (Ca and Ka)
2. Attacker thread use AES Decryption to get plaintext
3. Timer thread measures latency during each decryption
4. Attacker classify labels with LOW and HIGH based on latency
5. Attacker Train AdaBoost Model with these labels
Classifying Victim’s Accesses

4. Victim use the AES engine to decrypt ciphertext $C_v$ with its own secret key $K_v$

5. Timer monitors the victim accesses and measure latency

6. Latency is predicted to be HIGH/LOW using AdaBoost model

7. If the latency is classified as LOW, plaintext can be XORed with Td4 values associated with LOW label

Repeat Step 4-7 multiple times and take majority voting
Intel Xeon SP Scalable and Intel Core processors have mesh network

Similar latency distribution found in Intel 10700k with 16 cores from Comet Lake processor family

Similar vulnerabilities may exist in other mesh network processors
Similar T-table implementation in many cryptographic software

Recent AES version has disabled patch which would prevent this attack

Camellia & ARIA also have similar structure

```c
static void sl1(ARIA_u128 *o, const ARIA_u128 *x, const ARIA_u128 *y)
{
    unsigned int i;
    for (i = 0; i < ARIA_BLOCK_SIZE; i += 4) {
        o->c[i] = sb1[x->c[i]] ^ y->c[i];
        o->c[i + 1] = sb2[x->c[i + 1]] ^ y->c[i + 1];
        o->c[i + 2] = sb3[x->c[i + 2]] ^ y->c[i + 2];
        o->c[i + 3] = sb4[x->c[i + 3]] ^ y->c[i + 3];
    }
}
```

Attack of the Knights: Farabi Mahmud et al.
Decrypt function Contains many memory accesses

- Many of these accesses are made to Td table
- End-to-end timing of Decrypt contains noise

Fine-grained Timing utilizes access to shared buffer

- Allows more precise measurement of LLC Hit Latency
- Only monitor accesses to Td4 table
SHMEM
Facilitates fine-grain data sharing

Memory Protection Keys
Restricts access to specific memory regions
Allows sharing of other regions

Multithreading
Different threads can be sharing some variables

SHMEM
Allows threads or processes to exchange specific values
Do not need to share the memory space entirely
Is this Generalizable?

Software Targets

Hardware Platforms

Configurations

• MCDRAM
• Cluster
Intel Xeon SP Scalable and Intel Core processors have mesh network

Similar latency distribution found in Intel 10700k with 16 cores from Comet Lake processor family

Similar vulnerabilities may exist in other mesh network processors
MCDRAM configuration will impact LLC Hit Latency

- Cache Mode
- Flat Mode
- Hybrid Mode

We have used Flat Mode
Three available modes

• All-to-all
• Quadrant/Hemisphere Mode
• Sub NUMA Cluster (SNC-2/SNC-4)

We have used All-to-all cluster mode
Overlapping Loads

Problem

• Multiple loads overlap within the region of interest
• Measured latency is affected by overlapped loads

Solution

• Take multiple samples
• Use AdaBoost algorithm to classify samples
Side-Channel Result: ML Model

Attack of the Knights: Farabi Mahmud et al.
We can extract 4 bytes of any random key with 100% accuracy by using only $\approx 4000$ trials.
Covert Channel Results

Attack of the Knights: Farabi Mahmud et al.
Distance-based NUCA Cache Side-Channel Attack

Implemented in Gem5 Simulator
- 95% Accuracy even with Rodinia background application

Implemented Covert Channel Intel Xeon Phi 7290
- 205KBPS data bandwidth
- 0.2% Error Rate

Implemented Side-channel in Intel Xeon Phi 7290
- 4000 trials to get 4 bytes of AES key with 100% accuracy

Other processors with mesh network might be vulnerable

Other cryptographic algorithms with similar T-table might be vulnerable
ATTACK EXAMPLE ON GEM5 SIMULATOR
<table>
<thead>
<tr>
<th>Architecture</th>
<th>8x8 Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Distributed Directory</td>
</tr>
<tr>
<td></td>
<td>2D Mesh Network</td>
</tr>
<tr>
<td>Each tile contains</td>
<td>A Core</td>
</tr>
<tr>
<td></td>
<td>Private L1I Cache</td>
</tr>
<tr>
<td></td>
<td>Private L1D Cache</td>
</tr>
<tr>
<td></td>
<td>Shared LLC (L2) Bank</td>
</tr>
<tr>
<td>L1D Cache</td>
<td>2-way associative</td>
</tr>
<tr>
<td></td>
<td>4kB</td>
</tr>
<tr>
<td></td>
<td>LRU Replacement</td>
</tr>
<tr>
<td>LLC</td>
<td>8-way associative</td>
</tr>
<tr>
<td></td>
<td>2MB</td>
</tr>
<tr>
<td></td>
<td>Distributed across 64 Tiles</td>
</tr>
</tbody>
</table>

- **Core**
  - L1 Cache
  - LLC Slice
- **Core**
  - L1 Cache
  - LLC Slice
Step 1. Identify vulnerable access pattern in Victim function

Step 2. Prepare for L1 Miss but LLC Hit

Step 3. Allow Victim to access entries that would be LLC hit

Step 4. Measure the latency and classify accordingly

Attack of the Knights: Farabi Mahmud et al.
Step 1a. Reverse Engineer LLC Slice Selection Function

<table>
<thead>
<tr>
<th>Offset</th>
<th>Tag</th>
<th>LLC Slice ID</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Step 1b. Determine Addresses belonging to Different LLC Slice

<table>
<thead>
<tr>
<th>Array Index</th>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>LLC Slice</th>
</tr>
</thead>
<tbody>
<tr>
<td>117 * 64</td>
<td>0x4C7FC0</td>
<td>0xC6FC0</td>
<td>63</td>
</tr>
<tr>
<td>118 * 64</td>
<td>0x4C8000</td>
<td>0xC7000</td>
<td>0</td>
</tr>
<tr>
<td>L1$</td>
<td>Set</td>
<td>Way 0</td>
<td>Way 1</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>0</td>
<td>0-0-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LLC</th>
<th>Set</th>
<th>Way 0</th>
<th>Way 1</th>
<th>Way 2</th>
<th>Way 3</th>
<th>Way 4</th>
<th>Way 5</th>
<th>Way 6</th>
<th>Way 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0-0-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Slice-L2-L1
ADDR 0-0-0
LLC Miss

Prepare for L1 Miss, L2 Hit
Way 0
Way 1
Set 0 - 0 - 0
Way 2
Way 3
Way 4
Way 5
Way 6
Way 7

Attack of the Knights: Farabi Mahmud et al.
Prepare for L1 Miss, L2 Hit

Way 0
Way 1
Set
0
0-0-0
1-0-0
1

slice-L2-L1

ADDR
1-0-0

LLC Miss

<table>
<thead>
<tr>
<th>L1$</th>
<th>Set</th>
<th>Way 0</th>
<th>Way 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0-0-0</td>
<td>1-0-0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LLC</th>
<th>Set</th>
<th>Way 0</th>
<th>Way 1</th>
<th>Way 2</th>
<th>Way 3</th>
<th>Way 4</th>
<th>Way 5</th>
<th>Way 6</th>
<th>Way 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0-0-0</td>
<td>1-0-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Attack of the Knights: Farabi Mahmud et al.
Prepare for L1 Miss, L2 Hit

Way 0
Way 1
Set 2 - 0 - 0
1 - 0 - 0
Way 0
Way 1
Set 0 - 0 - 0
1 - 0 - 0
Way 2
Way 3
Way 4
Way 5
Way 6
Way 7
Slice-L2-L1
ADDR 2-0-0
LLC Miss

<table>
<thead>
<tr>
<th>L1$</th>
<th>Set</th>
<th>Way 0</th>
<th>Way 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0-0-0</td>
<td>1-0-0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LLC</th>
<th>Set</th>
<th>Way 0</th>
<th>Way 1</th>
<th>Way 2</th>
<th>Way 3</th>
<th>Way 4</th>
<th>Way 5</th>
<th>Way 6</th>
<th>Way 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0-0-0</td>
<td>1-0-0</td>
<td>2-0-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### L1$\$ Cache Table

<table>
<thead>
<tr>
<th>Set</th>
<th>Way 0</th>
<th>Way 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2-0-0</td>
<td>3-0-0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### LLC Cache Table

<table>
<thead>
<tr>
<th>Set</th>
<th>Way 0</th>
<th>Way 1</th>
<th>Way 2</th>
<th>Way 3</th>
<th>Way 4</th>
<th>Way 5</th>
<th>Way 6</th>
<th>Way 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0-0-0</td>
<td>1-0-0</td>
<td>2-0-0</td>
<td>3-0-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Slice-L2-L1**

**ADDR**

1-0-0

**LLC Hit**
• Only allow Victim to have one memory access

• Memory Location dependent on Secret Bit

```c
void victim(unsigned int mask) {
    uint8_t s = secret & mask;
    if (s == 0) s ^= arr[117 * 64]; // LLC bank 63
    else s ^= arr[118 * 64]; // LLC bank 0
}
```
• Use RDTSCP to measure latency
• Based on the threshold, we can classify whether its bit 0 or 1

```c
// Time the victim function
t1 = __rdtscp(&junk);
victim(mask);
t2 = __rdtscp(&junk) - t1;
// If the bit is 0, the latency > 100
printf("BIT[%d]: %d\n", i, t2 > THRESHOLD? 0:1);
```
Attack in Gem5 Simulator: Results

Green Region Secret Bit 0
White Region Secret Bit 1

> 95 % accuracy

Red Cross Bit Predicted 1
Blue Cross Bit Predicted 0