

# ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture

USENIX Security 2022

**Pietro Borrello**

Sapienza University of Rome

**Moritz Lipp**

Amazon Web Services

**Andreas Kogler**

Graz University of Technology

**Daniel Gruss**

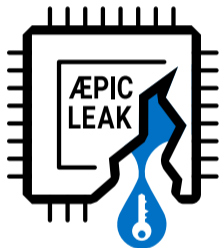
Graz University of Technology

**Martin Schwarzl**

Graz University of Technology

**Michael Schwarz**

CISPA Helmholtz Center for Information Security



1. **ÆPIC Leak**
2. **Understand** what we leak
3. **Control** what we leak
4. **Exploit** ÆPIC Leak
5. **Mitigations**

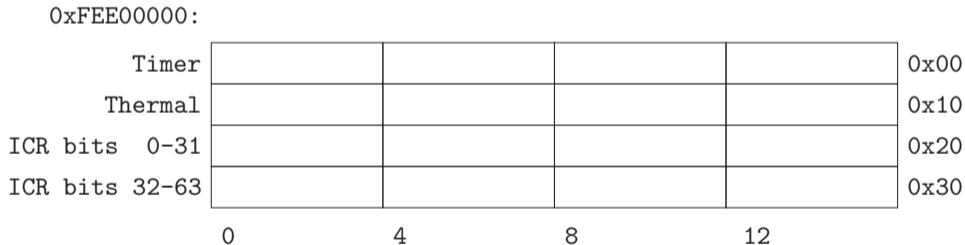
**What is ÆPIC Leak?**

- **Memory-mapped APIC registers**

Timer					0x00
Thermal					0x10
ICR bits 0-31					0x20
ICR bits 32-63					0x30

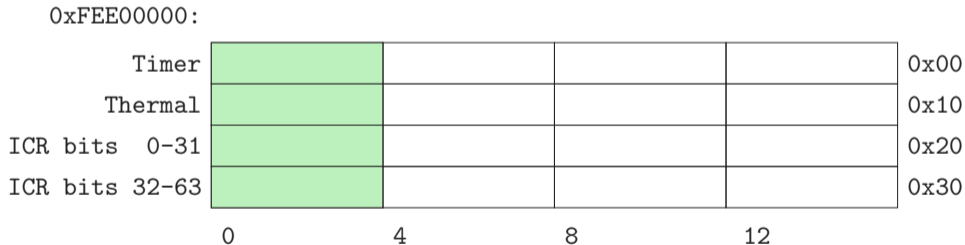
0                      4                      8                      12

- **Memory-mapped** APIC registers
  - Controlled by MSR IA32\_APIC\_BASE (default 0xFEE00000)

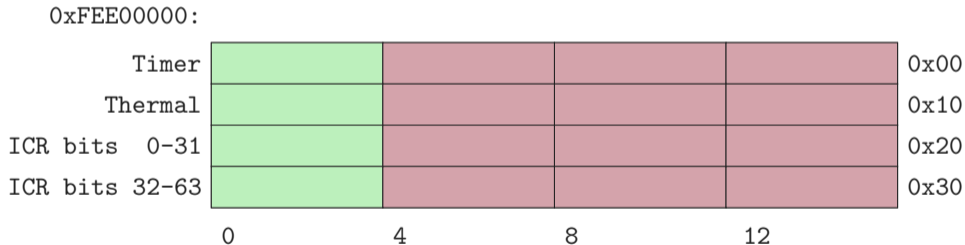


# Advanced Programmable Interrupt Controller (APIC) MMIO

- **Memory-mapped APIC registers**
  - Controlled by MSR IA32\_APIC\_BASE (default 0xFEE00000)
  - Mapped as **32bit** values, **aligned to 16 bytes**



- **Memory-mapped APIC registers**
  - Controlled by MSR IA32\_APIC\_BASE (default 0xFEE00000)
  - Mapped as **32bit** values, **aligned to 16 bytes**
  - **Should not** be accessed at bytes 4 through 15.



*Any access that touches bytes 4 through 15 of an APIC register may cause undefined behavior and must not be executed. This undefined behavior could include hangs, incorrect results, or unexpected exceptions.*



*Any access that touches bytes 4 through 15 of an APIC register may cause undefined behavior and must not be executed. This undefined behavior could include hangs, incorrect results, or unexpected exceptions.*

## Let's try this!

```
u8 *apic_base = map_phys_addr(0xFEE00000);  
dump(&apic_base[0]);  
dump(&apic_base[4]);  
dump(&apic_base[8]);  
dump(&apic_base[12]);  
/* ... */
```

output:

```
u8 *apic_base = map_phys_addr(0xFEE00000);  
dump(&apic_base[0]); // no leak  
dump(&apic_base[4]);  
dump(&apic_base[8]);  
dump(&apic_base[12]);  
/* ... */
```

output:

FEE00000: 00 00 00 00

....

```
u8 *apic_base = map_phys_addr(0xFEE00000);  
dump(&apic_base[0]); // no leak  
dump(&apic_base[4]); // LEAK!  
dump(&apic_base[8]);  
dump(&apic_base[12]);  
/* ... */
```

output:

```
FEE00000: 00 00 00 00 57 41 52 4E
```

```
....WARN
```

```
u8 *apic_base = map_phys_addr(0xFEE00000);  
dump(&apic_base[0]); // no leak  
dump(&apic_base[4]); // LEAK!  
dump(&apic_base[8]); // LEAK!  
dump(&apic_base[12]);  
/* ... */
```

output:

```
FEE00000: 00 00 00 00 57 41 52 4E 5F 49 4E 54          ....WARN_INT
```

```
u8 *apic_base = map_phys_addr(0xFEE00000);  
dump(&apic_base[0]); // no leak  
dump(&apic_base[4]); // LEAK!  
dump(&apic_base[8]); // LEAK!  
dump(&apic_base[12]); // LEAK!  
/* ... */
```

output:

```
FEE00000: 00 00 00 00 57 41 52 4E 5F 49 4E 54 45 52 52 55 ....WARN_INTERRU
```

```
u8 *apic_base = map_phys_addr(0xFEE00000);  
dump(&apic_base[0]); // no leak  
dump(&apic_base[4]); // LEAK!  
dump(&apic_base[8]); // LEAK!  
dump(&apic_base[12]); // LEAK!  
/* ... */
```

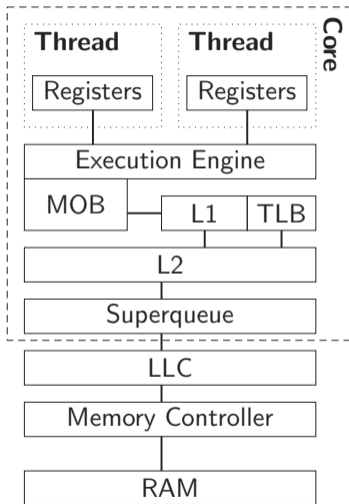
output:

```
FEE00000: 00 00 00 00 57 41 52 4E 5F 49 4E 54 45 52 52 55 ....WARN_INTERRU  
FEE00010: 00 00 00 00 4F 55 52 43 45 5F 50 45 4E 44 49 4E ....OURCE_PENDIN  
FEE00020: 00 00 00 00 46 49 5F 57 41 52 4E 5F 49 4E 54 45 ....FI_WARN_INTE  
FEE00030: 00 00 00 00 54 5F 53 4F 55 52 43 45 5F 51 55 49 ....T_SOURCE_QUI
```

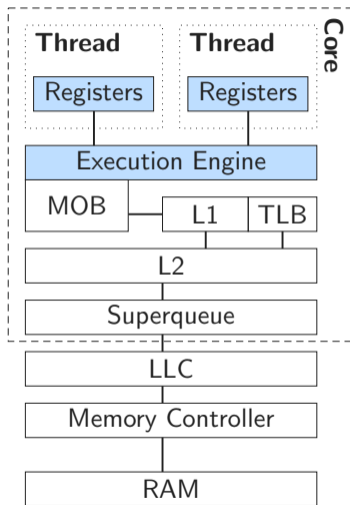
**Where do we leak from?**



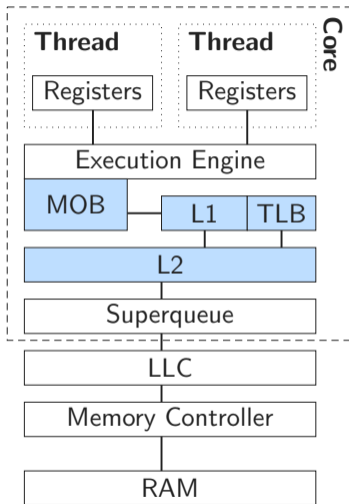
# Ruling out microarchitectural elements



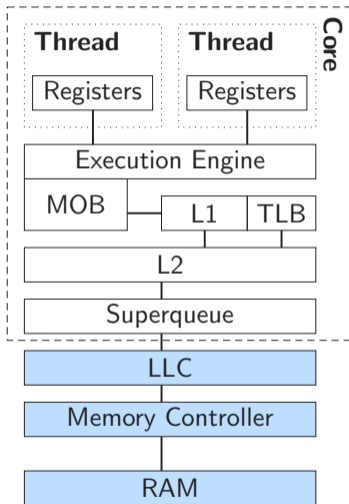
# Ruling out microarchitectural elements



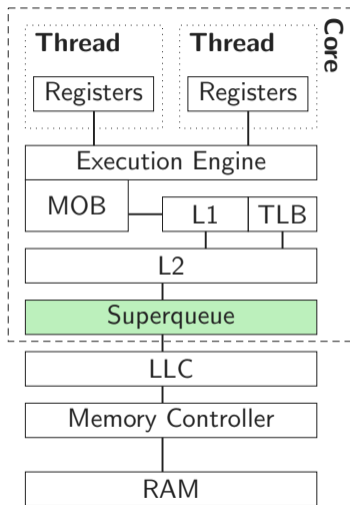
# Ruling out microarchitectural elements



# Ruling out microarchitectural elements



# Ruling out microarchitectural elements



- **Decoupling buffer** between L2 and LLC
- **Data** passed between L2 and LLC

- We can leak only **undefined** APIC offsets: *i.e.*, **3/4** of a cache line

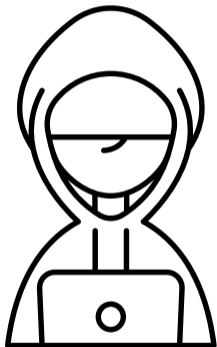
0	4	8	12	
				0x00
				0x10
				0x20
				0x30
				0x40
				0x50
				0x60
				0x70

Leaked Addresses

- We can leak only **undefined** APIC offsets: *i.e.*, **3/4** of a cache line
- We only observe **even** cache lines

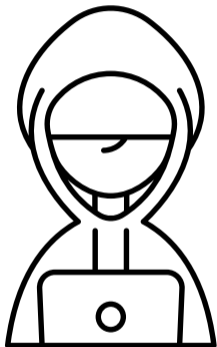
0	4	8	12	
				0x00
				0x10
				0x20
				0x30
				0x40
				0x50
				0x60
				0x70

Leaked Addresses

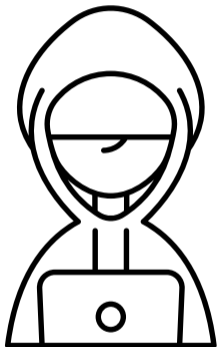


- We leak data from the **Superqueue (SQ)**

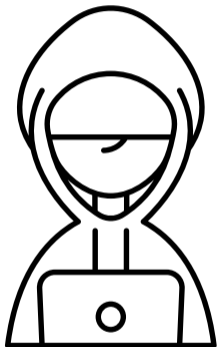




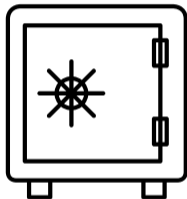
- We leak data from the **Superqueue (SQ)**
- Like an **uninitialized** memory read, but in the CPU



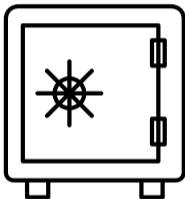
- We leak data from the **Superqueue (SQ)**
- Like an **uninitialized** memory read, but in the CPU
- We need **access** to APIC MMIO region



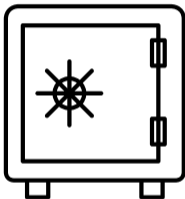
- We leak data from the **Superqueue (SQ)**
  - Like an **uninitialized** memory read, but in the CPU
  - We need **access** to APIC MMIO region
- Let's leak data from **SGX enclaves!**



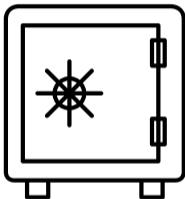
- SGX: isolates environments against **priviledged** attackers



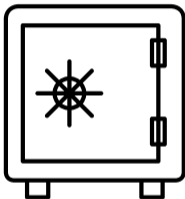
- SGX: isolates environments against **priviledged** attackers
- Transparently **encrypts** pages in the Enclave Page Cache (EPC)



- SGX: isolates environments against **priviledged** attackers
- Transparently **encrypts** pages in the Enclave Page Cache (EPC)
- Pages can be **moved** between EPC and RAM

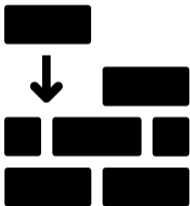


- SGX: isolates environments against **priviledged** attackers
- Transparently **encrypts** pages in the Enclave Page Cache (EPC)
- Pages can be **moved** between EPC and RAM
- Use State Save Area (SSA) for context switchces

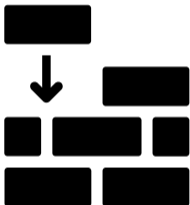


- SGX: isolates environments against **priviledged** attackers
- Transparently **encrypts** pages in the Enclave Page Cache (EPC)
- Pages can be **moved** between EPC and RAM
- Use State Save Area (SSA) for context switches
  - Stores enclave state during switches
  - **Inlcuding** register values

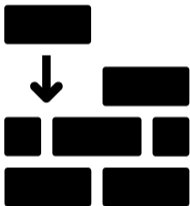




- We can already **sample** data from SGX enclaves!
- But, how to leak interesting data?



- We can already **sample** data from SGX enclaves!
- But, how to leak interesting data?
  - Can we **force** data into the SQ?



- We can already **sample** data from SGX enclaves!
- But, how to leak interesting data?
  - Can we **force** data into the SQ?
  - Can we **keep** data in the SQ?

# Enclave Shaking



- Abuse the **EWB** and **ELDU** instructions for page swapping

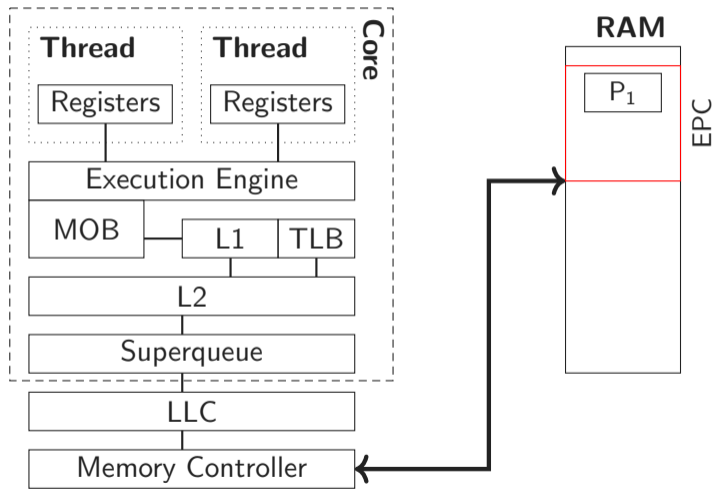


- Abuse the **EWB** and **ELDU** instructions for page swapping
- **EWB** instruction:
  - Encrypts and stores an enclave page to RAM



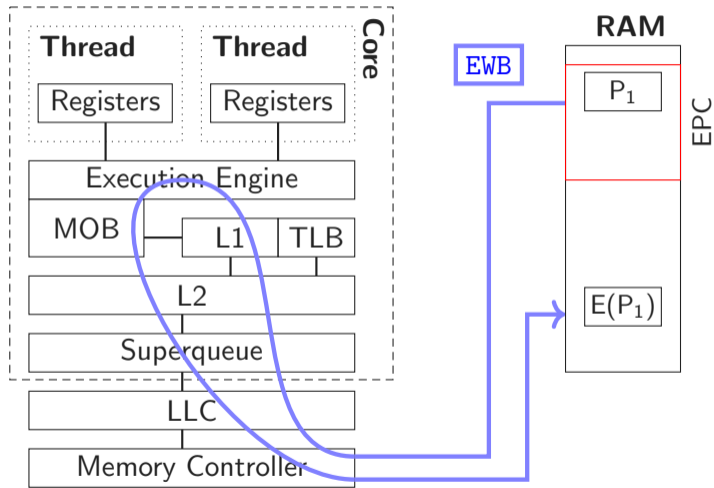
- Abuse the **EWB** and **ELDU** instructions for page swapping
- **EWB** instruction:
  - Encrypts and stores an enclave page to RAM
- **ELDU** instruction:
  - Decrypts and loads an enclave page from RAM

# Force Data into the SQ: Enclave Shaking

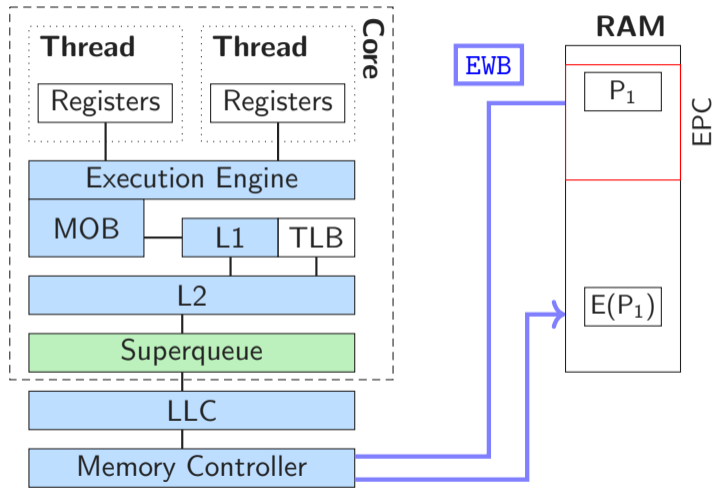




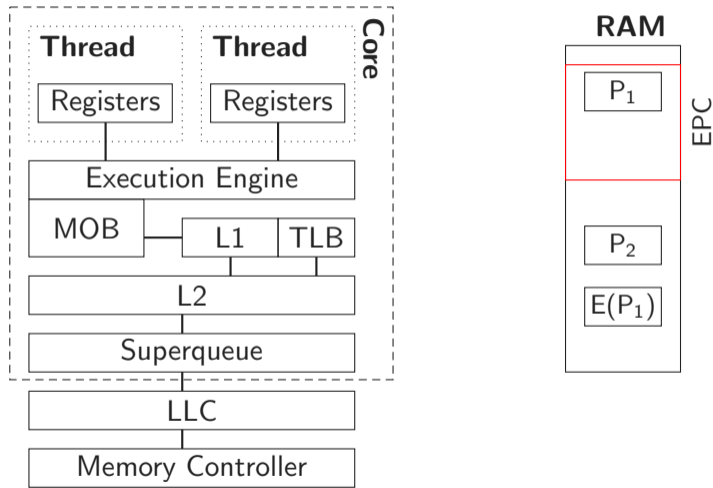
# Force Data into the SQ: Enclave Shaking



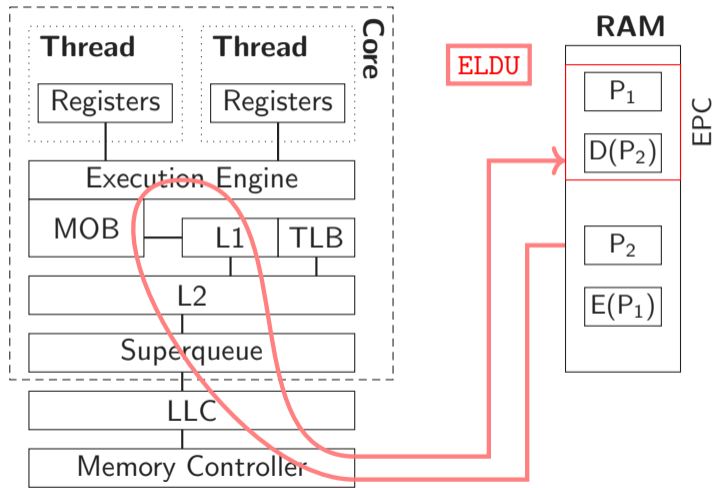
# Force Data into the SQ: Enclave Shaking



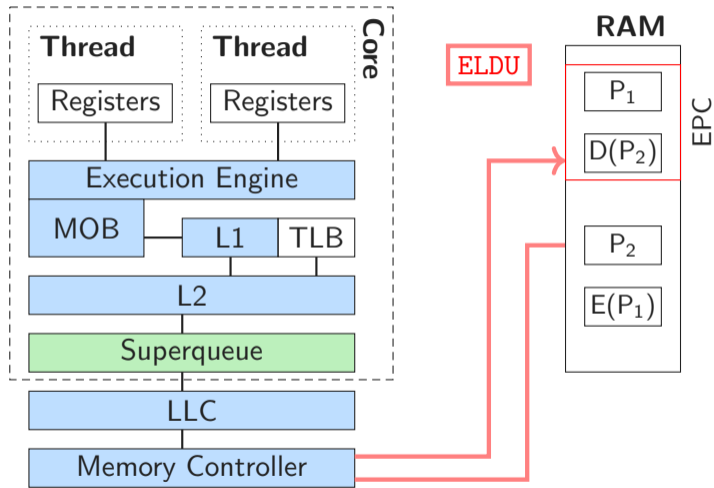
# Force Data into the SQ: Enclave Shaking



# Force Data into the SQ: Enclave Shaking



# Force Data into the SQ: Enclave Shaking



# Cache Line Freezing



We **do not need** hyperthreading, but we can use it!



We **do not need** hyperthreading, but we can use it!

- The SQ is **shared** between hyperthreads





We **do not need** hyperthreading, but we can use it!

- The SQ is **shared** between hyperthreads
- An hyperthread **affects** the SQ's content



We **do not need** hyperthreading, but we can use it!

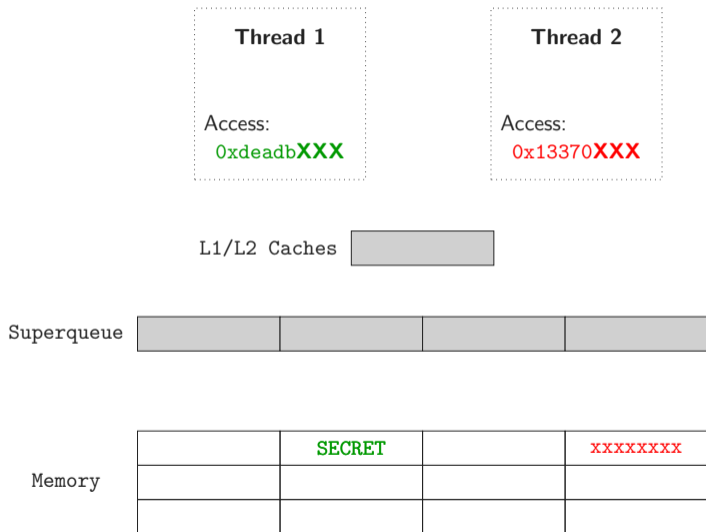
- The SQ is **shared** between hyperthreads
- An hyperthread **affects** the SQ's content
- Theory: **Zero blocks are not transfered over the SQ**



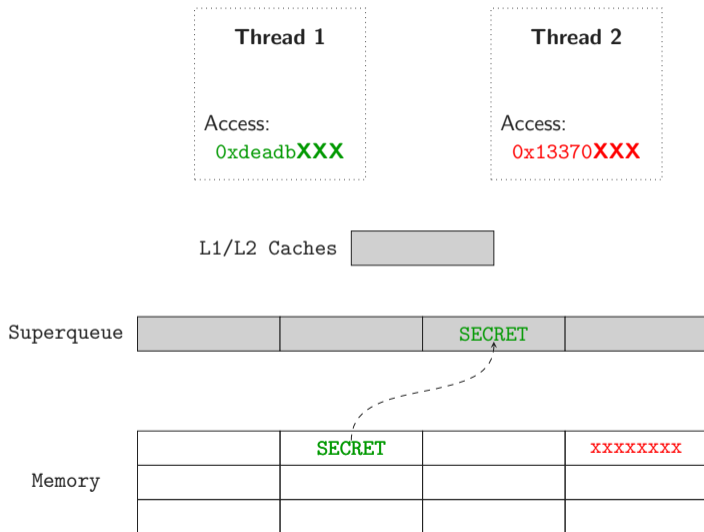
We **do not need** hyperthreading, but we can use it!

- The SQ is **shared** between hyperthreads
- An hyperthread **affects** the SQ's content
- Theory: **Zero blocks are not transfered over the SQ**
- But how?

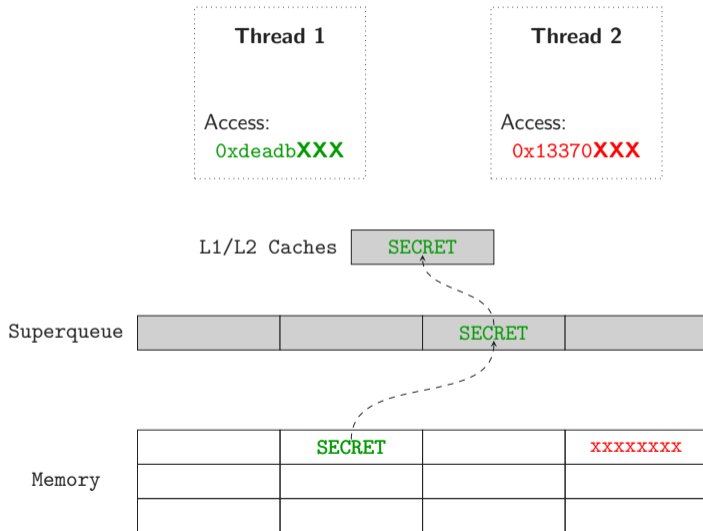
# Keep Data in the SQ: Cache Line Freezing



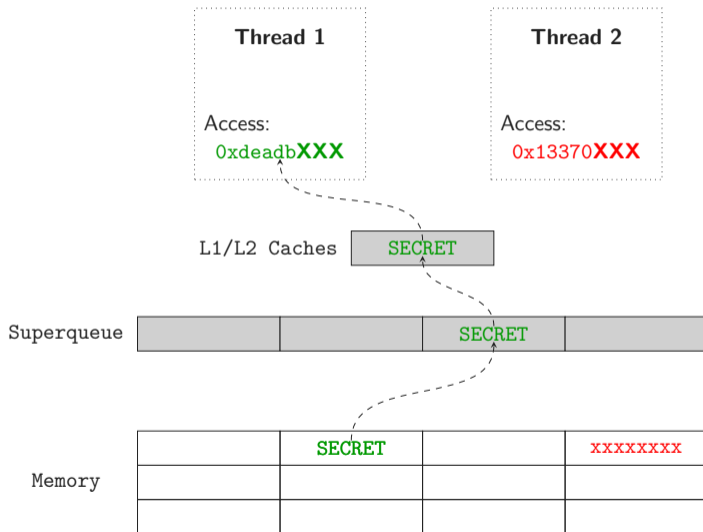
# Keep Data in the SQ: Cache Line Freezing



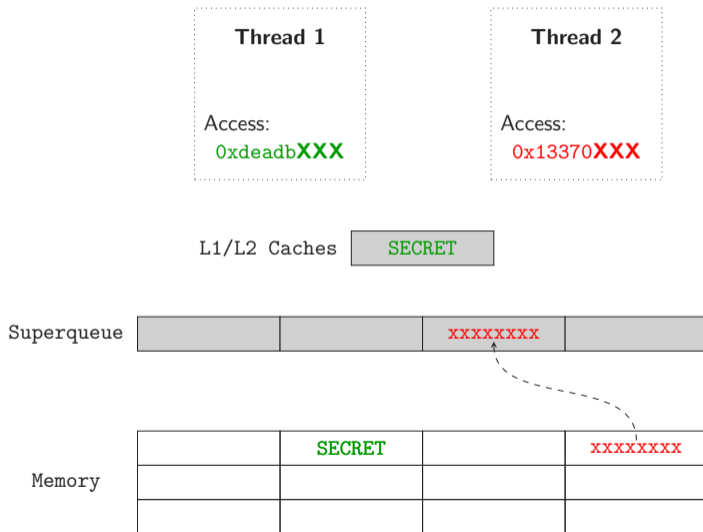
# Keep Data in the SQ: Cache Line Freezing



# Keep Data in the SQ: Cache Line Freezing

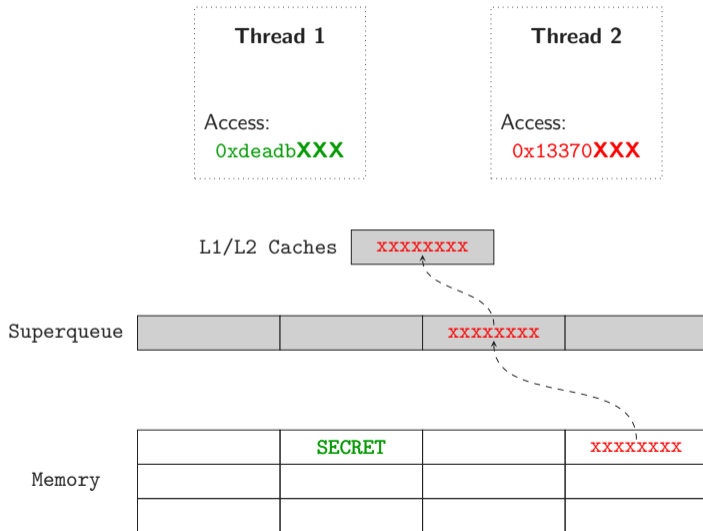


# Keep Data in the SQ: Cache Line Freezing

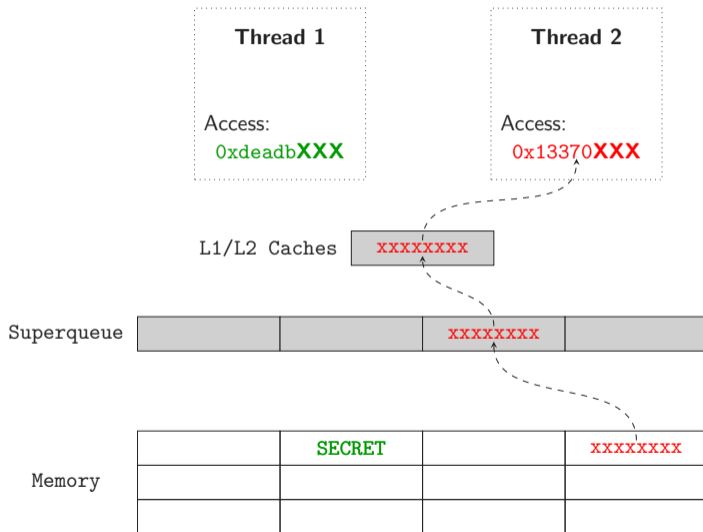




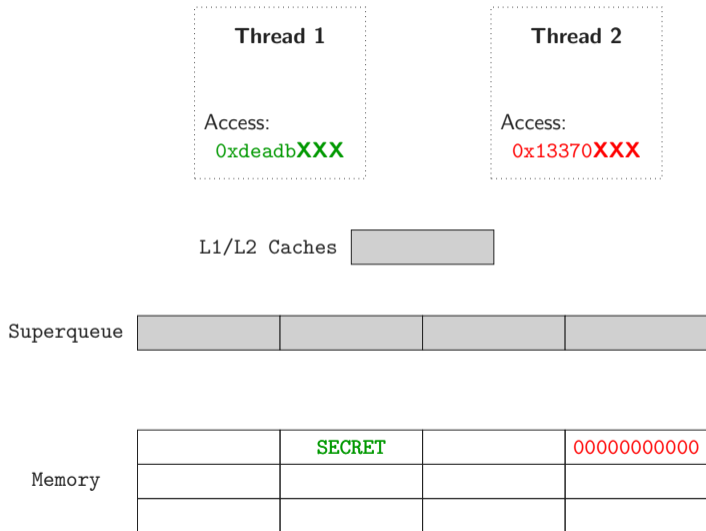
# Keep Data in the SQ: Cache Line Freezing



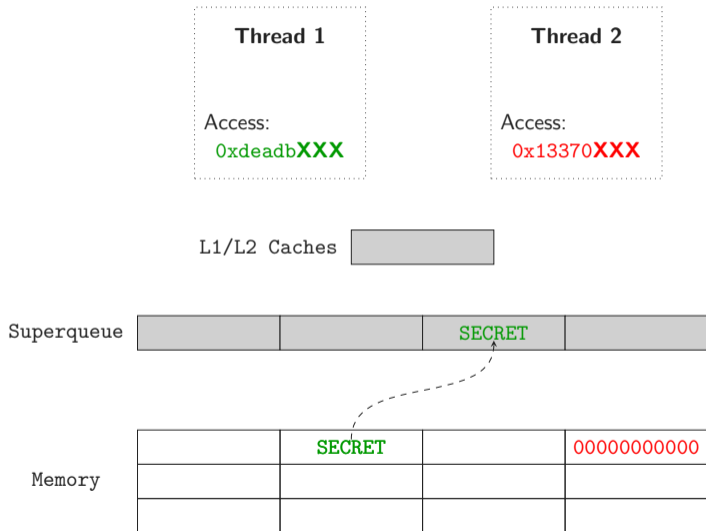
# Keep Data in the SQ: Cache Line Freezing



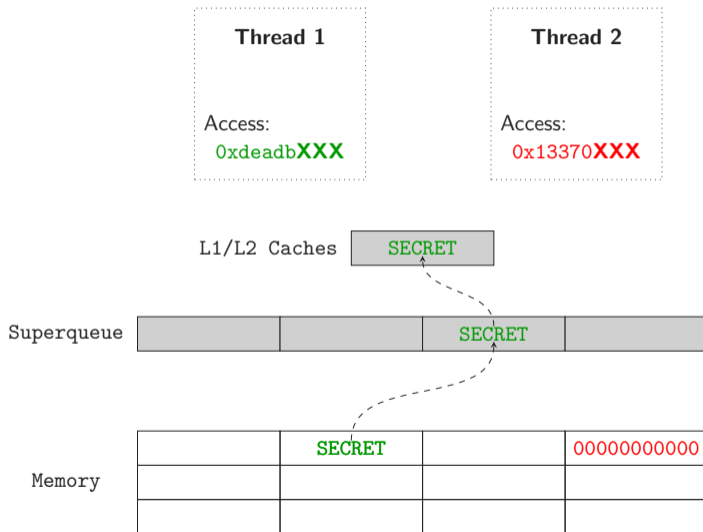
# Keep Data in the SQ: Cache Line Freezing



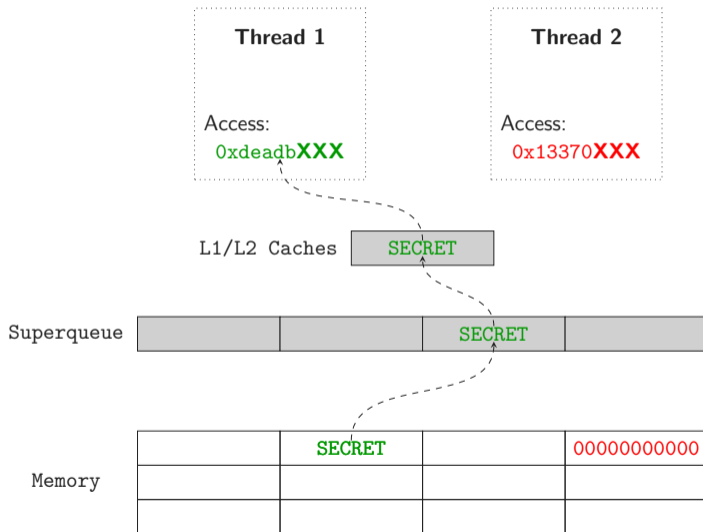
# Keep Data in the SQ: Cache Line Freezing



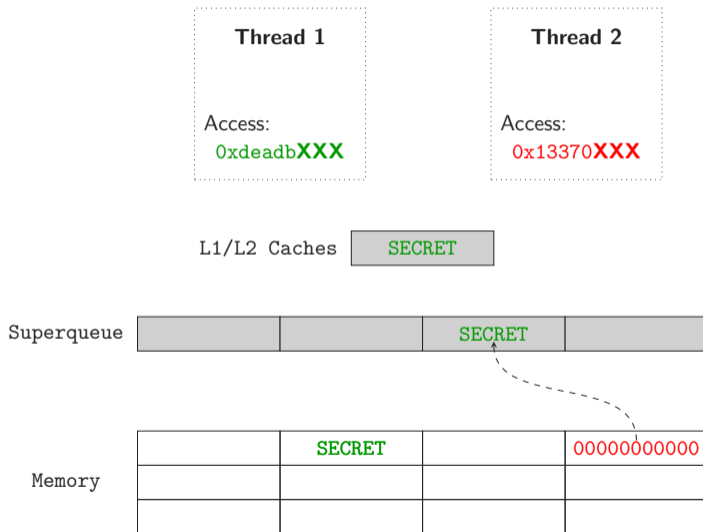
# Keep Data in the SQ: Cache Line Freezing



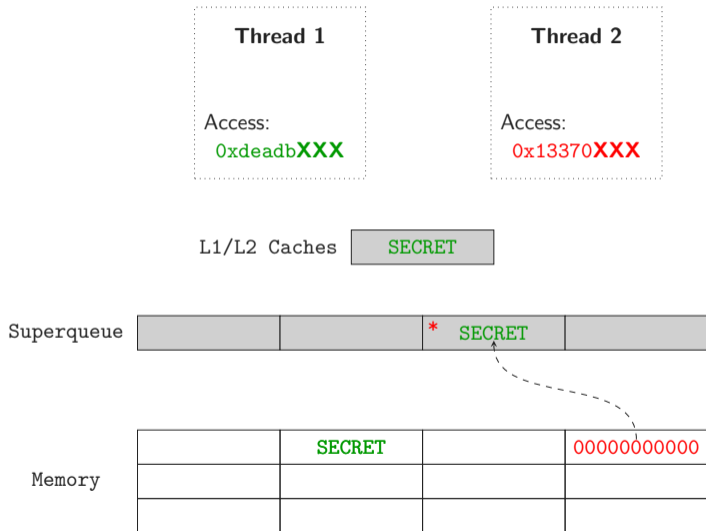
# Keep Data in the SQ: Cache Line Freezing



# Keep Data in the SQ: Cache Line Freezing

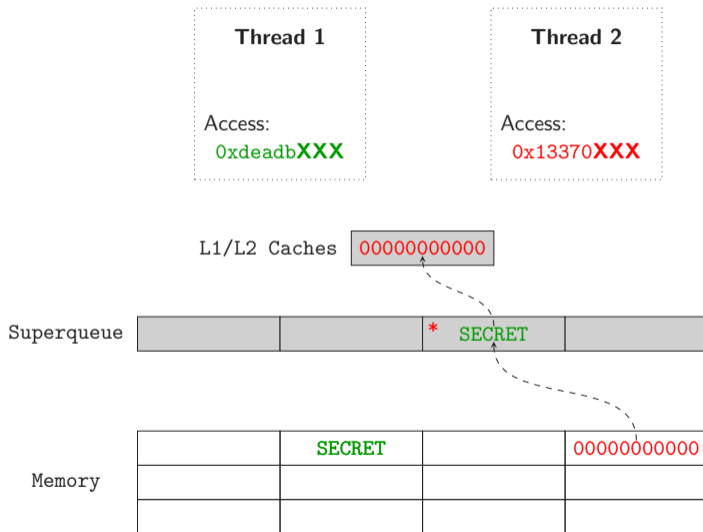


# Keep Data in the SQ: Cache Line Freezing

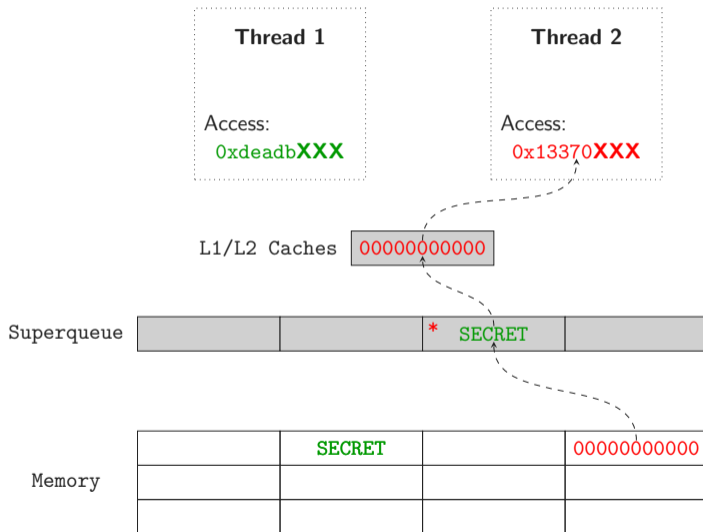




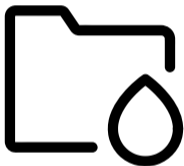
# Keep Data in the SQ: Cache Line Freezing



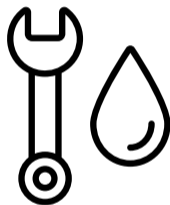
# Keep Data in the SQ: Cache Line Freezing



**ÆPIC Leak**



1. **Start** the enclave
2. **Stop** when the data is **loaded**
3. **Move** the page out (EWB) *and* perform Cache Line Freezing
4. **Leak** via APIC MMIO
5. **Move** the page in (ELDU)
6. **Goto** 3 until enough confidence



1. **Start** the enclave
2. **Stop** at the target instruction
3. **Move** SSA page out (EWB) *and* perform Cache Line Freezing
4. **Leak** via APIC MMIO
5. **Move** SSA page in (ELDU)
6. **Goto** 3 until enough confidence

---

Class	Leakable Registers
General Purpose	<u>rdi</u> r8 <u>r9</u> r10 <u>r11</u> r12 <u>r13</u> r14
SIMD	xmm0-1 xmm6-9

---

```
l~  
$ ./runner enclave.signed.so  
[enclave] enclave init!  
[runner ] waiting for user input ot termiante!  
□
```

```
l~  
$ ./dumper `pidof runner` 0 dsr /dev/null
```

```
$ sudo ./stepper aes.enclave 0 config
[idt.c] locking IRQ handler pages 0x55c2b1e6f000/0x55c2b1e80000
__key0: offset=0x 20bc0 reg: xmm0 line= 2 start= 8 end=12 if=[ 13, 14)+1 pf=[ 0, 2)+1
[attacker] ssa @ 0x7f0d94d9ef48
[attacker] base @ 0x7f0d94c00000
[attacker] size 512 pages
[attacker] irq vector 0x400ec
[victim] starting on core 1

[0x20bc0] __key0[ 1]+ 13 = 00000000|15ca71be|2b73aef0|857d7781|

[victim] finished!
[main] finished with 29275 aep callbacks!
$ cat aes.cpp | grep secret_key -A 5
static Ipp8u secret_key[KEY_SIZE] = {
    0x60, 0x3d, 0xeb, 0x10,
    0x15, 0xca, 0x71, 0xbe,
    0x2b, 0x73, 0xae, 0xf0,
    0x85, 0x7d, 0x77, 0x81
};
$ █
```



- Recommend to **disable** APIC MMIO

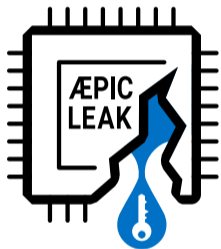




- Recommend to **disable** APIC MMIO
- Microcode update to **flush SQ** on SGX transitions



- Recommend to **disable** APIC MMIO
- Microcode update to **flush SQ** on SGX transitions
- Disable hyperthreading when using SGX



- ÆPIC Leak: the **first** architectural CPU vulnerability that leaks data from cache hierarchy
- Does not require hyperthreading
- 10<sup>th</sup>, 11<sup>th</sup> and 12<sup>th</sup> gen Intel CPUs affected

[aepicleak.com](https://aepicleak.com)

## **ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture**

Pietro Borrello<sup>1</sup>, Andreas Kogler<sup>2</sup>, Martin Schwarzl<sup>2</sup>, Moritz Lipp<sup>3</sup>, Daniel Gruss<sup>2</sup>, Michael Schwarz<sup>4</sup>

<sup>1</sup> Sapienza University of Rome, <sup>2</sup> Graz University of Technology,

<sup>3</sup> Amazon Web Services, <sup>4</sup> CISPA Helmholtz Center for Information Security