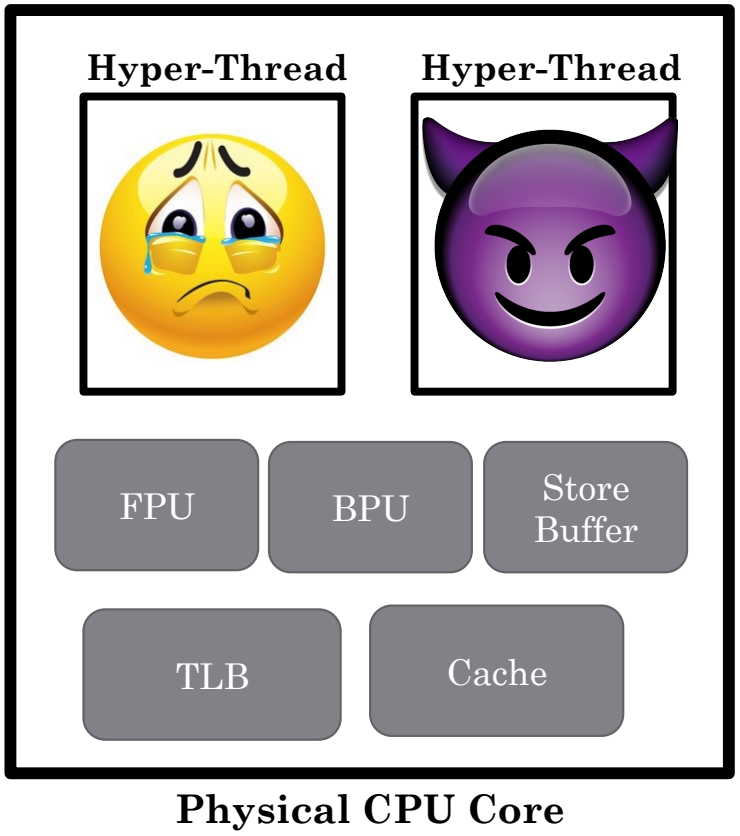
Software Solutions to Micro-architectural Side Channels

Yinqian Zhang
Assistant Professor
Computer Science & Engineering
The Ohio State University

Introduction

- Research interests
  - Computer system security in general
  - (Micro-architectural) side-channel attacks and defenses
- Recent publications on micro-architectural side channels
  - Cloud computing (S&P'11, CCS'12, CCS'13, CCS'14, Security'15, Security'16, RAID'16, CCS'16a, AsiaCCS'17b, TDSC 2018)
  - Intel SGX (AsiaCCS'17a, CCS'17a, CCS'17b, S&P'18a, SgxPectre)
  - Smartphones (CCS'16b), side-channel measurement (ACSAC'18)
- Recent publications on system-level side channels
  - Mobile systems (CCS'15, NDSS'18a), searchable encryption (INFOCOM'18), vulnerability detection (S&P'18b)
- Service on the program committees of security conferences
  - IEEE S&P (2016, 2017, 2018), ACM CCS (2015, 2016, 2017, 2018), Usenix Security (2017), NDSS (2017, 2018, 2019)

Hyper-Threading Side Channels



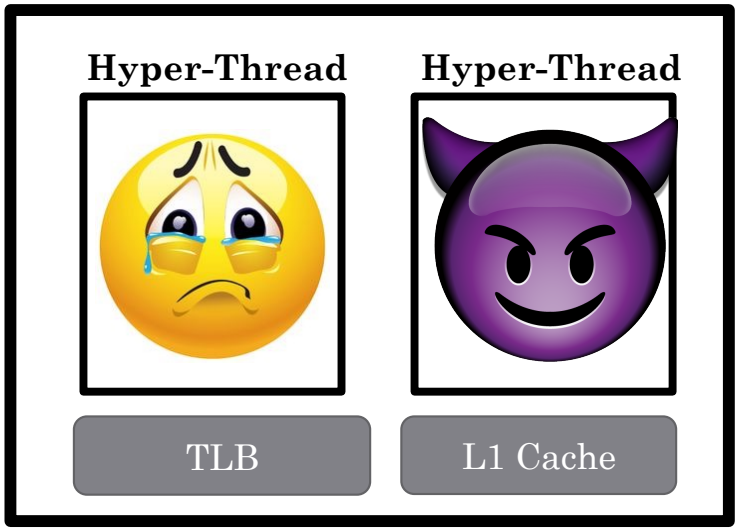
Hyper-Threading creates side channels

- Floating point units
- Branch prediction units
- Store buffers
- Translation lookaside buffers
- Caches (L1I, L1D, L2)

Cache missing for fun and profit, Colin Percival 2005

Cache Attacks and Countermeasures: the Case of AES, Dag Arne Osvik , Adi Shamir, and Eran Tromer, 2005

Hyper-Threading Side Channels



Hyper-Threading enhances existing side channels

- Page monitoring through page tables (Wang et al. 2017)

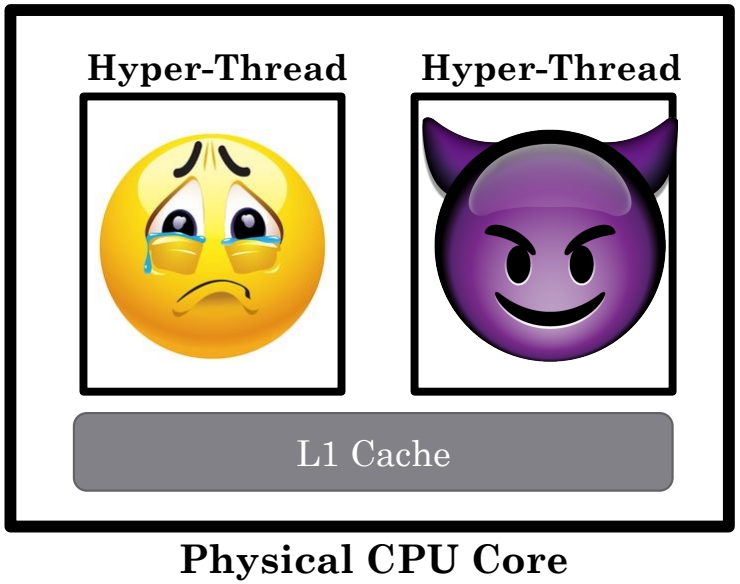
Hyper-Threading thwarted defenses

- Hyper-Threading invalidates some defense techniques that leverage Intel's TSX (Gruss et al. 2017)

Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX, Wenhao Wang, Guoxing Chen, Xiaorui Pan, *Yinqian Zhang*, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, Carl A. Gunter, A*CM Conference on Computer and Communications Security*, Dallas, Texas, USA, Oct. 2017

Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory, Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, Manuel Costa, USENIX Security Symposium, 2017.

Hyper-Threading Side Channels



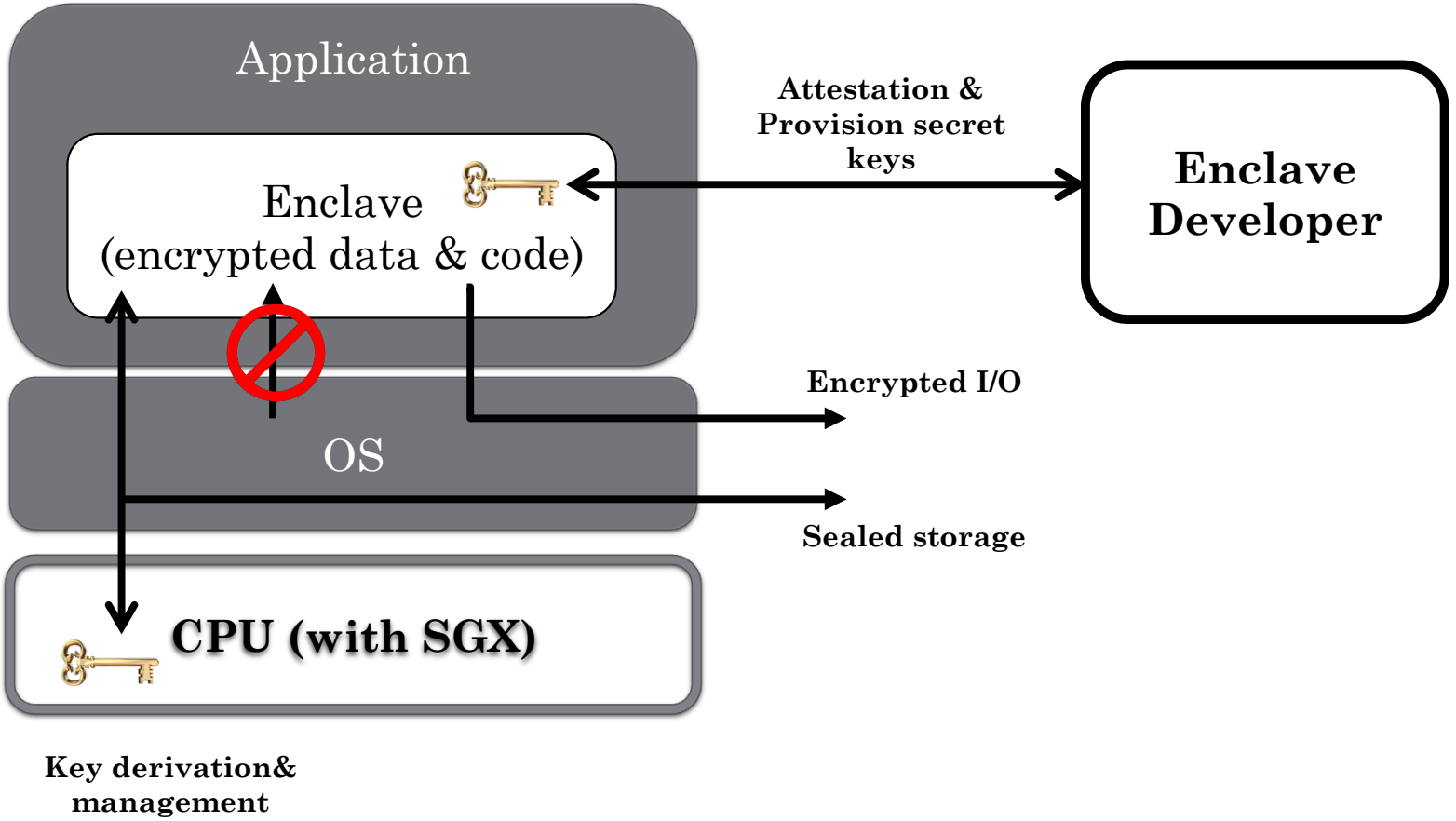
Hyper-Threading facilities speculative-execution based side channels

- Meltdown attacks (Lipp et al. 2018)
- Foreshadow attacks / L1 Terminal Fault (Bulck et al. 2018)

Meltdown: Reading Kernel Memory from User Space, Moritz Lipp and Michael Schwarz and Daniel Gruss and Thomas Prescher and Werner Haas and Anders Fogh and Jann Horn and Stefan Mangard and Paul Kocher and Daniel Genkin and Yuval Yarom and Mike Hamburg, USENIX Security Symposium, 2018.

Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution, Van Bulck, Jo and Minkin, Marina and Weisse, Ofir and Genkin, Daniel and Kasikci, Baris and Piessens, Frank and Silberstein, Mark and Wenisch, Thomas F. and Yarom, Yuval and Strackx, Raoul, USENIX Security Symposium, 2018.

Intel Software Guard Extension (SGX)



Hyper-Threading side channels on SGX

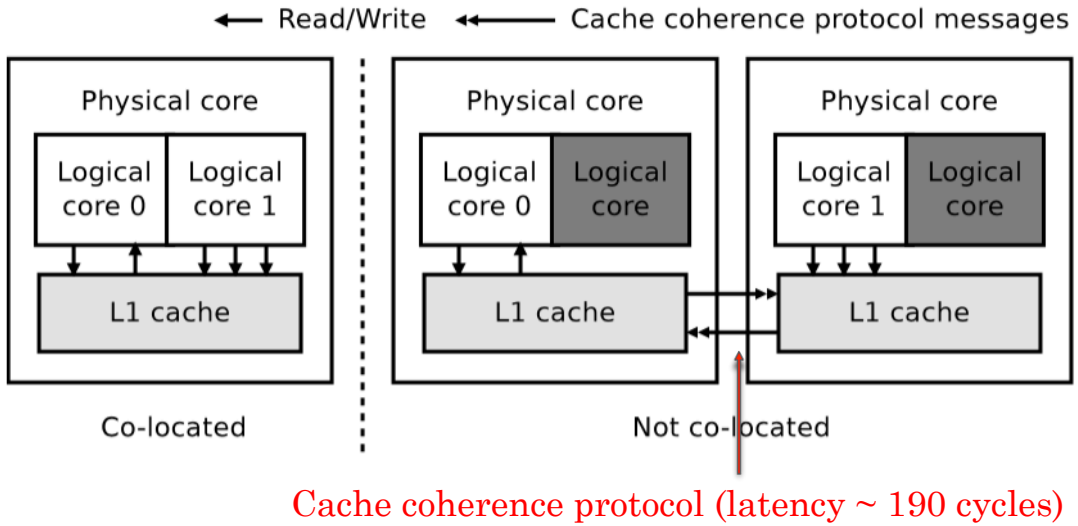
- Simply disabling Hyper-Threading is not a solution
  - No effective way to verify

- **HyperRace (Chen et al. 2018)**
  - Create an auxiliary thread, called shadow thread, to occupy the sibling Hyper-Thread
  - Request the untrusted OS to schedule the protected enclave thread and the shadow enclave thread on the same physical core
  - Verify these two threads are co-located on the same physical core

Racing in Hyperspace: Closing Hyper-Threading Side Channels on SGX with Contrived Data Races, Guoxing Chen, Wenhao Wang, Tianyu Chen, Sanchuan Chen, Yinqian Zhang, XiaoFeng Wang, Ten-Hwang Lai, Dongdai Lin, *IEEE Symposium on Security and Privacy*, 2018

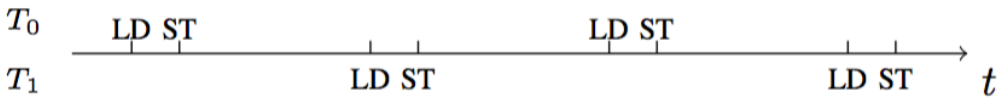
Hyper-Thread co-location tests

- Two threads operate on the same shared variable ν inside the enclave:
  - Thread T_0
    - **While (1)**
      Writes 0 to ν
      Waits for 10 cycles
      Reads ν
  - Thread T_1
    - **While (1)**
      Writes 1 to ν



- Thread T_0 reads ν=1 with very high probability when co-located
- Thread T_0 reads ν=0 with very high probability when not co-located

A refined design



Thread T_0

```
1  <initialization>:
2     mov $colocation_count, %rdx
3     xor %rcx, %rcx
4     ; co-location test counter
5  <synchronization>:
6     ··· ; acquire lock 0
7   .sync0:
8     mov %rdx, (sync_addr1)
9     cmp %rdx, (sync_addr0)
10    je .sync1
11    jmp .sync0
12  .sync1:
13    mfence
14    mov $0, (sync_addr0)
15 <initialize a round>:
16    mov $begin0, %rsi
17    mov $1, %rbx
18    mfence
19    mov $addr_v, %r8
20 <co-location test>:
21  .L0:
22 <load>:
23    mov (%r8), %rax
24 <store>:
25    mov %rsi, (%r8)
26 <update counter>:
27    mov $0, %r10
28    mov $0, %r11
29    cmp $end0, %rax
30    ; a data race happens?
31    cmovl %rbx, %r10
32    sub %rax, %r9
33    cmp $1, %r9
34    ; continuous number?
35    cmova %r11, %r10
36    add %r10, %rcx
37    shl $b_count, %rbx
38    ; bit length of $count
39    mov %rax, %r9
40    ; record the last number
41 <padding instructions 0>:
42    nop
43    nop
44     ⋮
45    nop
46    mov (%r8), %rax
47    mov (%r8), %rax
48     ⋮
49    mov (%r8), %rax
50    dec %rsi
51    cmp $end0, %rsi
52    jne .L0
53    ; finish 1 co-location test
54 <all rounds finished?>:
55    ··· ; release lock 1
56    dec %rdx
57    cmp $0, %rdx
58    jne .sync0
```

Thread T_1

```
1  <initialization>:
2     mov $colocation_count, %rdx
3     xor %rcx, %rcx
4     ; co-location test counter
5  <synchronization>:
6     ··· ; release lock 0
7   .sync2:
8     mov %rdx, (sync_addr0)
9     cmp %rdx, (sync_addr1)
10    je .sync3
11    jmp .sync2
12  .sync3:
13    mfence
14    mov $0, (sync_addr1)
15 <initialize a round>:
16    mov $begin1, %rsi
17    mov $1, %rbx
18    mfence
19    mov $addr_v, %r8
20 <co-location test>:
21  .L2:
22 <load>:
23    mov (%r8), %rax
24 <update counter>:
25    mov $0, %r10
26    mov $0, %r11
27    cmp $end0, %rax
28    ; a data race happens?
29    cmovg %rbx, %r10
30    sub %rax, %r9
31    cmp $1, %r9
32    ; continuous number?
33    cmova %r11, %r10
34    add %r10, %rcx
35    shl $b_count, %rbx
36    ; bit length of $count
37     mov %rax, %r9
38    ; record the last number
39 <store>:
40    mov %rsi, (%r8)
41 <padding instructions 1>:
42    mov (%r8), %rax
43    lfence
44    mov (%r8), %rax
45    lfence
46    mov (%r8), %rax
47    lfence
48    mov (%r8), %rax
49    lfence
50    mov (%r8), %rax
51    lfence
52    dec %rsi
53    cmp $end1, %rsi
54    jne .L2
55    ; finish 1 co-location test
56 <all rounds finished?>:
57    ··· ; acquire lock 1
58    dec %rdx
59    cmp $0, %rdx
60    jne .sync2
```

More details behind the scene

- Co-location test via statistical hypothesis testing:
  - Consider n data race unit tests: X_j, $j = 1, 2, \ldots, n$; probability of passing each test $P(X_j = 1) = p$.
  - Null hypothesis and alternative hypothesis

    H_0: $\hat{p} \geq p$; the two threads are co-located.

    H_1: $\hat{p} < p$; the two threads are not co-located.

- Threat model
  - Altering the execution speed of the enclave program by (1) causing cache contention, (2) altering CPU frequency, and (3) disabling caching.

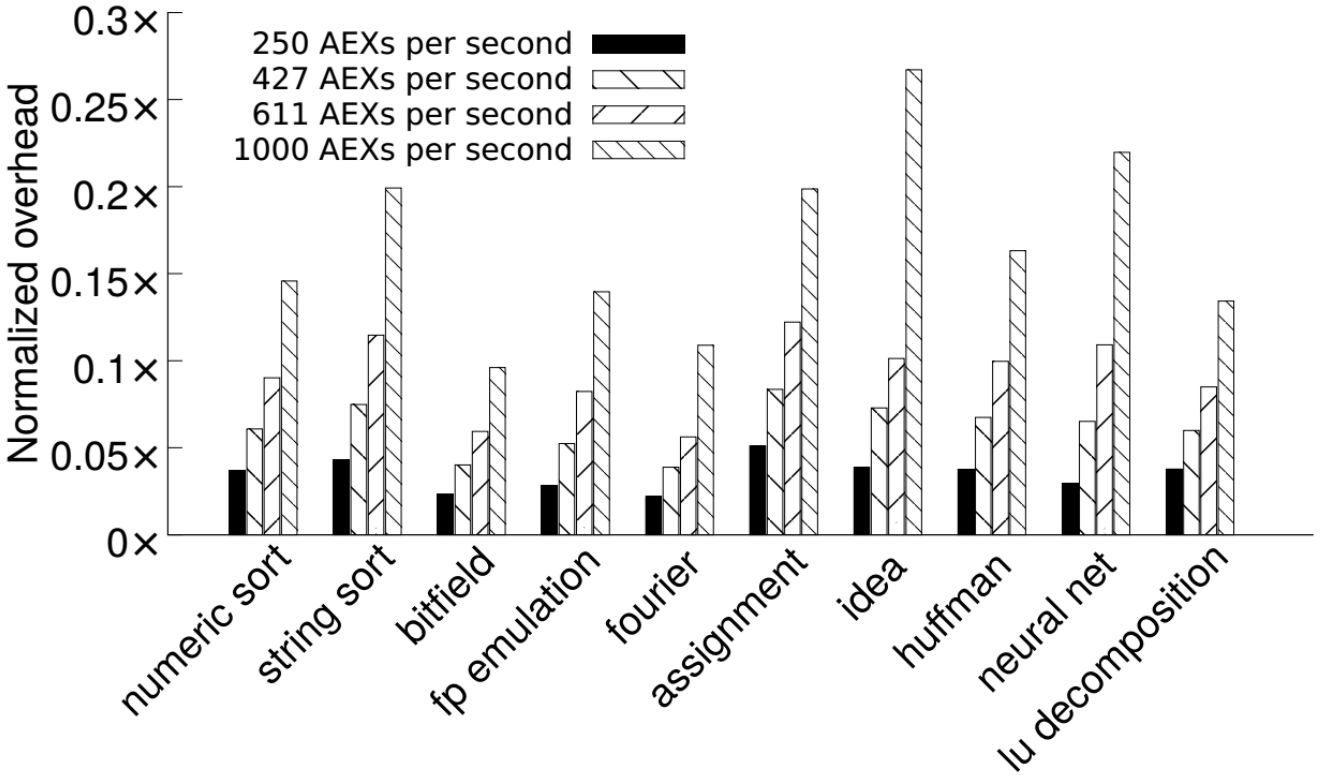
- Security analysis
  - Two threads not co-located: the co-location test will fail with very high probability
  - Two threads co-located: the co-location test will fail with very low probability

Implementation of HyperRace

- HyperRace is a compiler-assisted tool
  - A tool based on the LLVM framework
  - Instrument enclave program automatically to insert attack detection

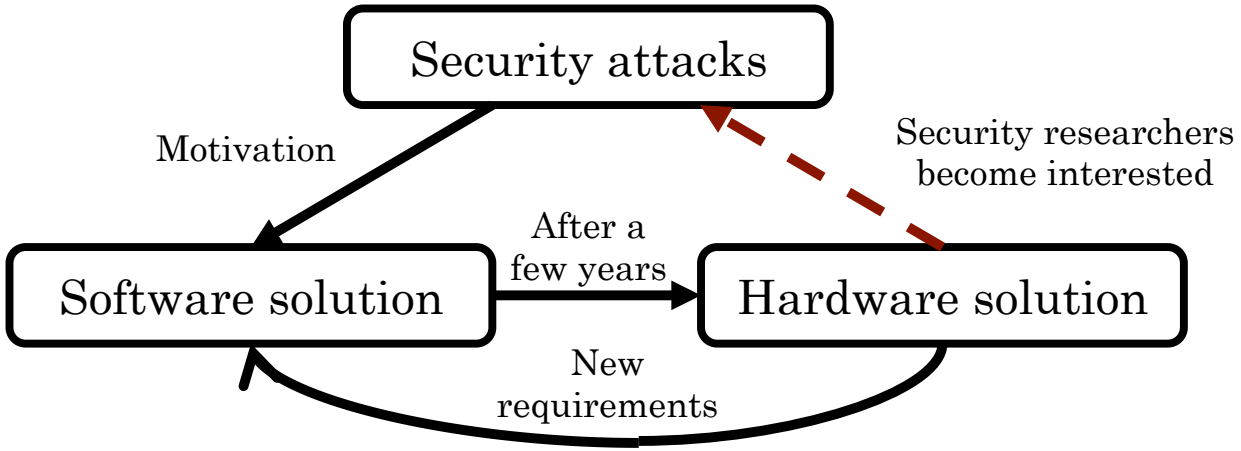
- HyperRace working logic:
  - Create a shadow thread for each enclave thread
  - Co-location test at EENTER of the enclave thread
  - Check AEX at every basic block (of a control flow graph)
    - Reasoning: no core migration without AEX
    - May insert more checks in the same basic block if needs higher security
  - Perform co-location tests when AEXs detected

Performance overhead of co-location tests



Hardware Solutions

- Disable Hyper-Threading
  - Inform enclave developers through remote attestation
- Enable Hyper-Threading
  - Temporarily disable sibling Hyper-Thread upon EENTER, or
  - Enclave code checks co-location through MSRs, and
  - Enclave code selectively disable sibling Hyper-Thread by updating MSRs



Hardware-Software Co-design

- Collaboration of researchers in **architecture**, **system**, and **security**
  - Micro-architectural side channels are rooted in the computer (micro-)architecture
  - Improper assumption made by system designer for program isolation
  - Insufficient understanding of the attack models, methods, techniques
- **Key research questions & challenges:**
  - Trade-off: cost effectiveness, performance, security
    - Metrics of measurement
    - Separation of responsibilities
  - Side channels are "side" channels by definition
    - Motivate security researchers to involve in the design stage
    - Leakage-free design ≠ leakage-free implementation