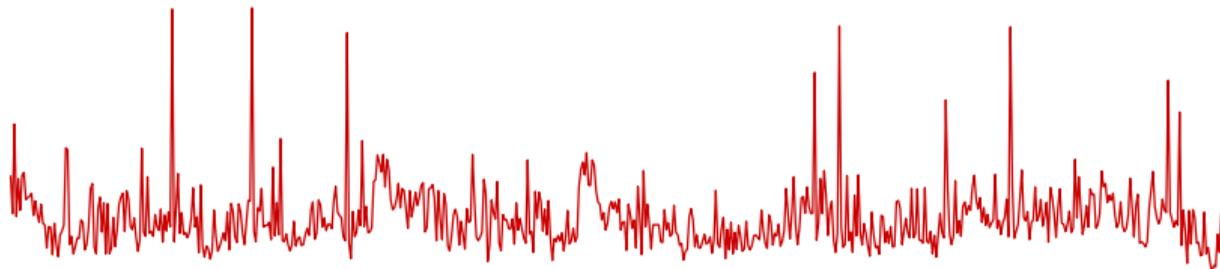


Microarchitectural Side-Channel Attacks for Privileged Adversaries

Jo Van Bulck

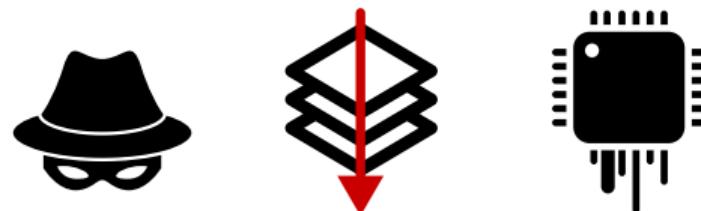
🏡 imec-DistriNet, KU Leuven 📩 jo.vanbulck@cs.kuleuven.be 🐦 jovanbulck

COSIC hardware security course, October 21, 2019



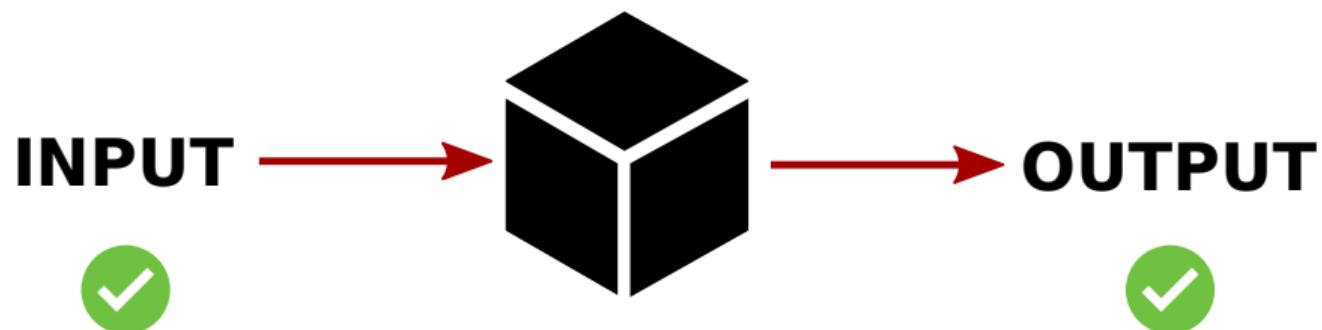
Today's goals and perspective

- Limitations of **trusted execution** environments (Sancus, Intel SGX)
 - Side-channel attacks from *untrusted operating system* to enclave
- **Software** viewpoint on hardware optimizations
 - Security cross-cuts hardware-software *abstraction layers*(!)



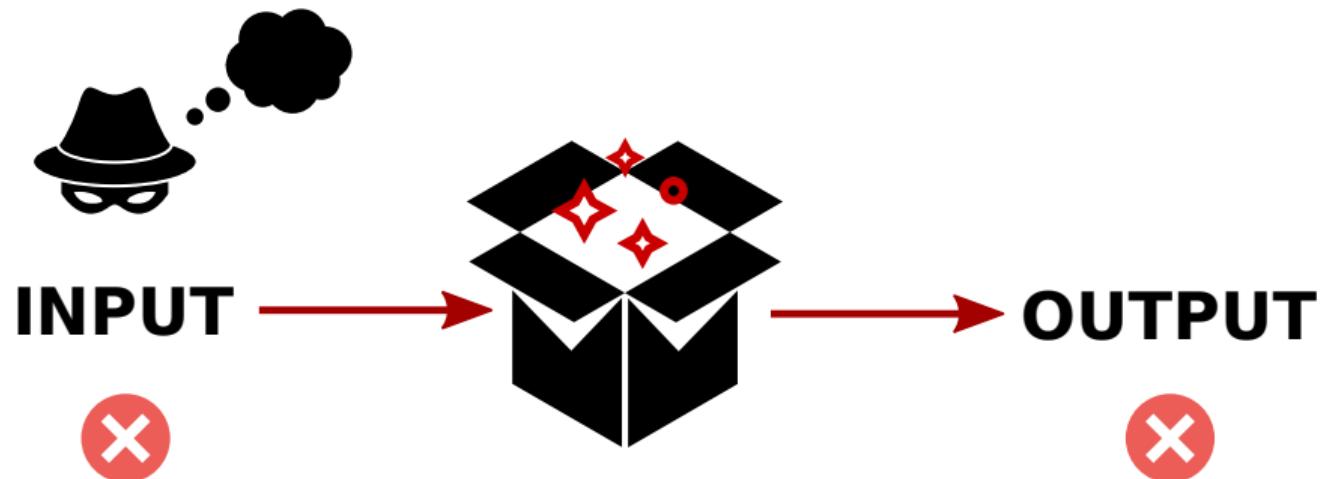
A primer on software security

Secure program: convert all input to *expected output*



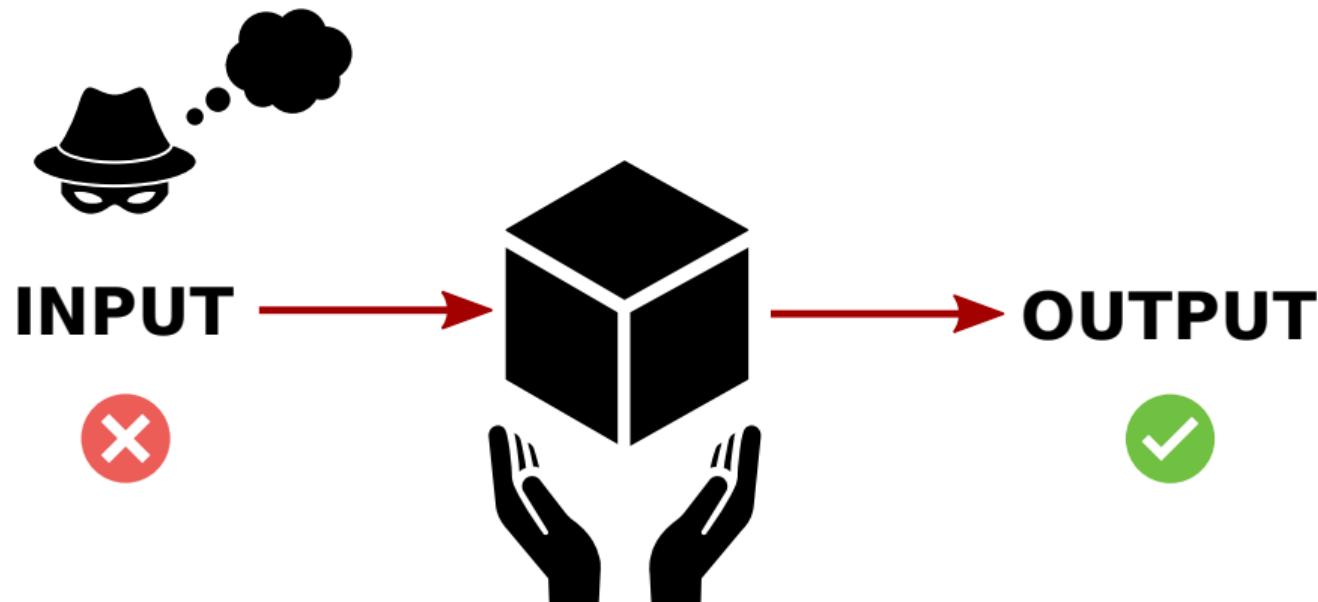
A primer on software security (previous lecture)

Buffer overflow vulnerabilities: trigger *unexpected behavior*



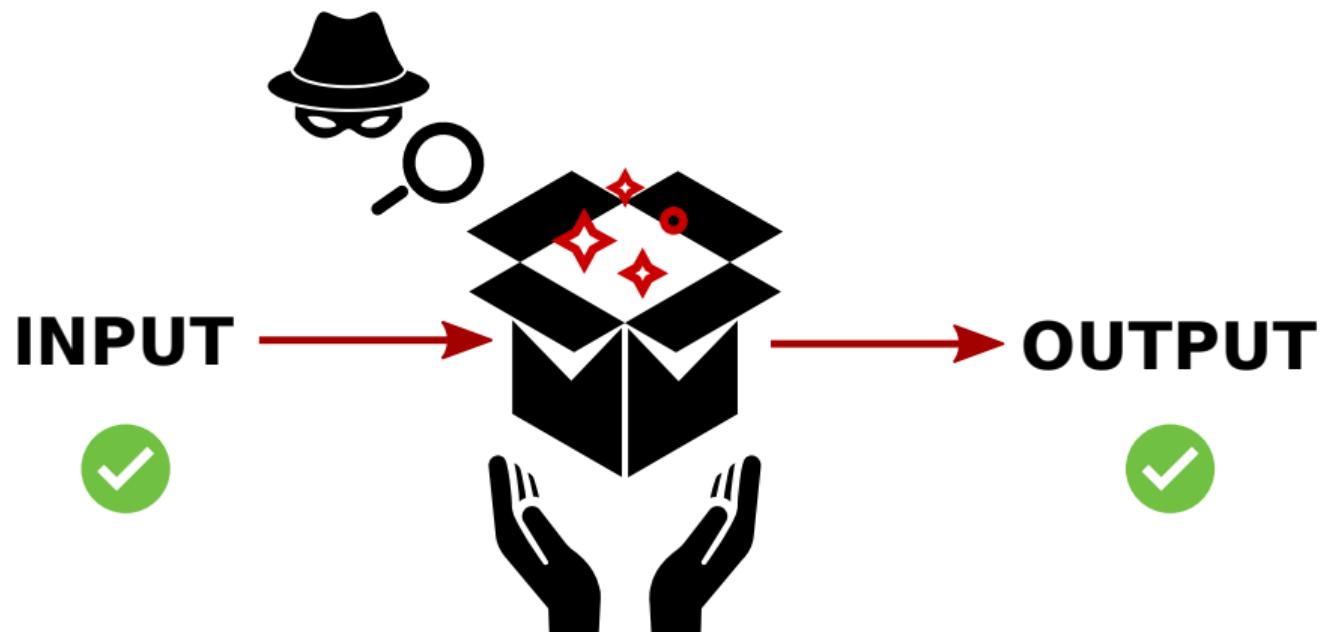
A primer on software security (previous lecture)

Safe languages & formal verification: preserve expected behavior



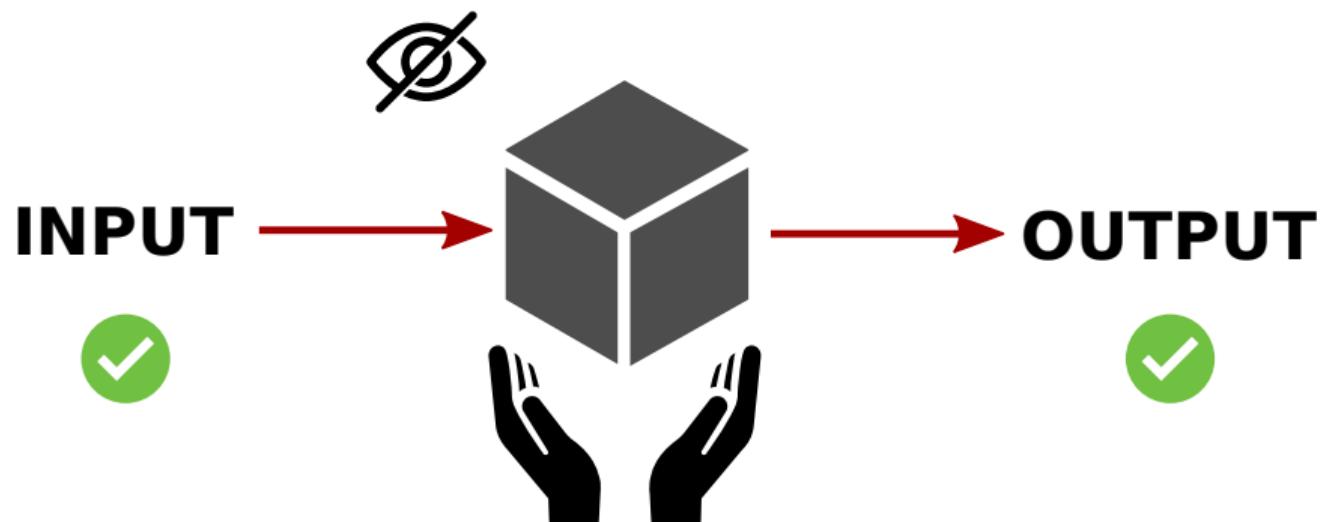
A primer on software security (this lecture)

Side-channels: observe *side-effects* of the computation



A primer on software security (this lecture)

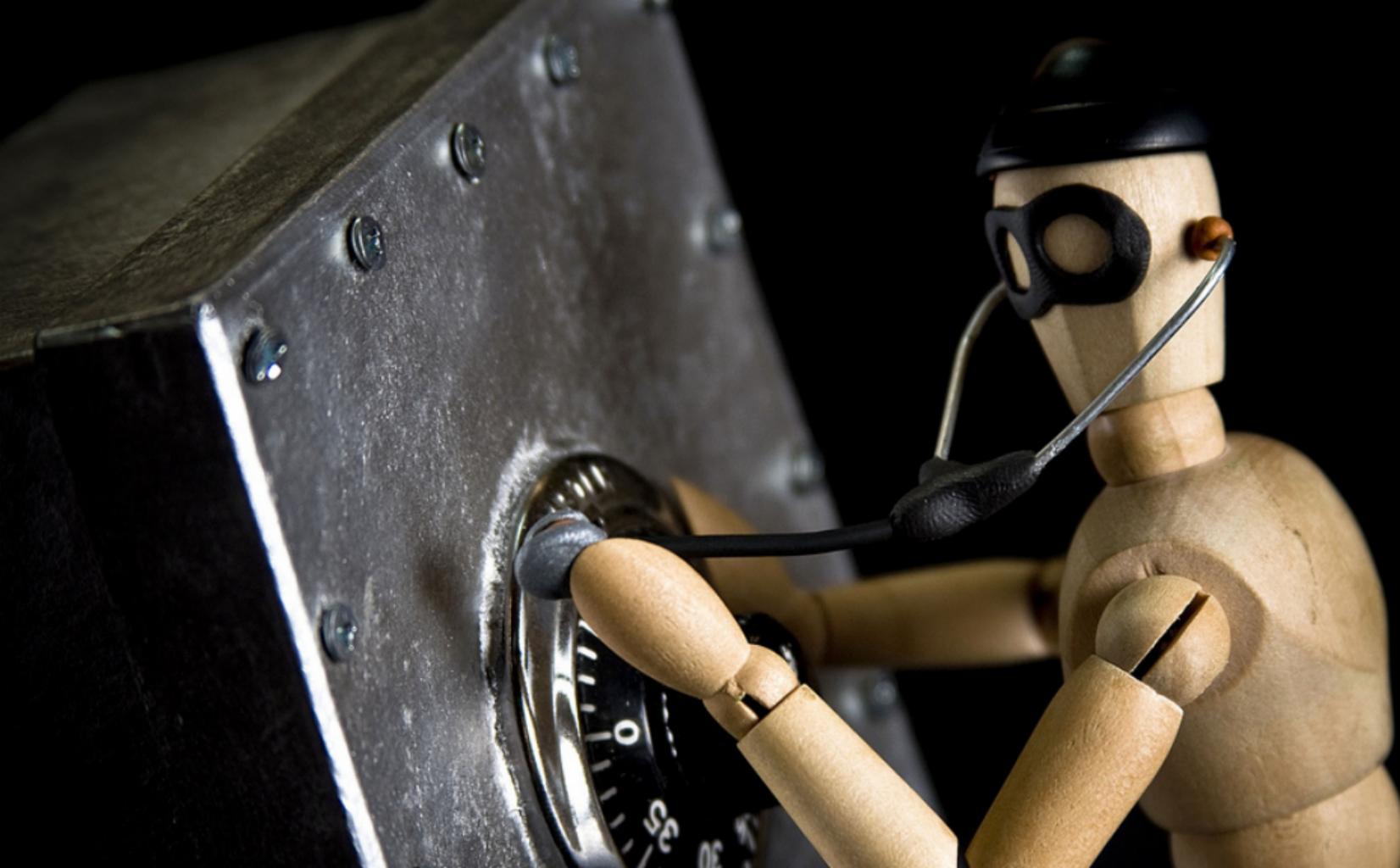
Constant-time code: eliminate *secret-dependent* side-effects





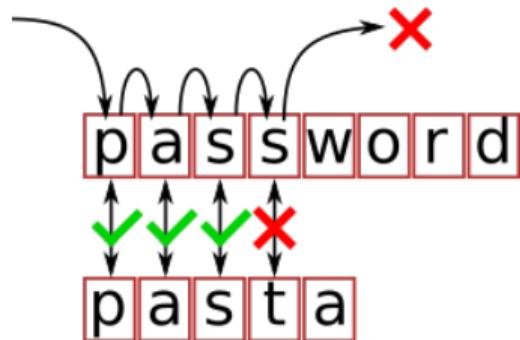
VAULT DOOR

WEIGHT: 22 1/2 Tons
THICKNESS: 22 Inches
STEEL: 11 Layers of Special
Cutting and Drill Resistant
LOCKS: 4 Hamilton Watch
Movements for Time Locks



A vulnerable example program and its constant-time equivalent

```
1 void check_pwd(char *input)
2 {
3     for (int i=0; i < PWD_LEN; i++)
4         if (input[i] != pwd[i])
5             return 0;
6
7     return 1;
8 }
```



Overall execution time reveals correctness of individual password bytes!

→ reduce brute-force attack from an exponential to a linear effort...

A vulnerable example program and its constant-time equivalent

```
1 void check_pwd(char *input)
2 {
3     for (int i=0; i < PWD_LEN; i++)
4         if (input[i] != pwd[i])
5             return 0;
6
7     return 1;
8 }
```

```
1 void check_pwd(char *input)
2 {
3     int rv = 0x0;
4     for (int i=0; i < PWD_LEN; i++)
5         rv |= input[i] ^ pwd[i];
6
7     return (rv == 0);
8 }
```

Rewrite program such that execution time does not depend on secrets

→ manual, error-prone solution; side-channels are likely here to stay...

Vulnerable patterns: Secret-dependent code/data memory accesses

```
1 void secret_vote(char candidate)
2 {
3     if (candidate == 'a')
4         vote_candidate_a();
5     else
6         vote_candidate_b();
7 }
```

```
1 int secret_lookup(int s)
2 {
3     if (s > 0 && s < ARRAY_LEN)
4         return array[s];
5     return -1;
6 }
7 }
```

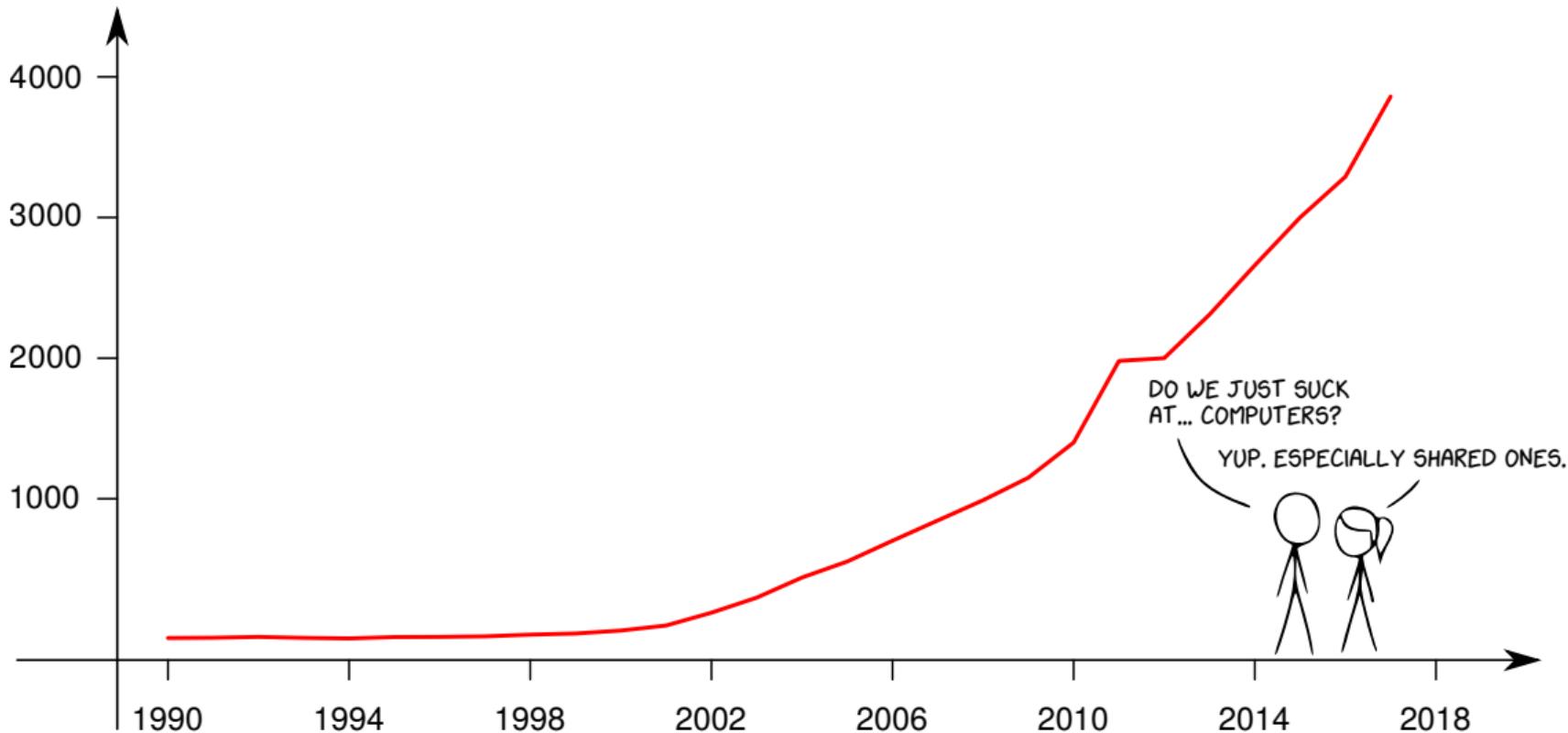
Vulnerable patterns: Secret-dependent code/data memory accesses

```
1 void secret_vote(char candidate)
2 {
3     if (candidate == 'a')
4         vote_candidate_a();
5     else
6         vote_candidate_b();
7 }
```

```
1 int secret_lookup(int s)
2 {
3     if (s > 0 && s < ARRAY_LEN)
4         return array[s];
5     return -1;
6 }
7 }
```

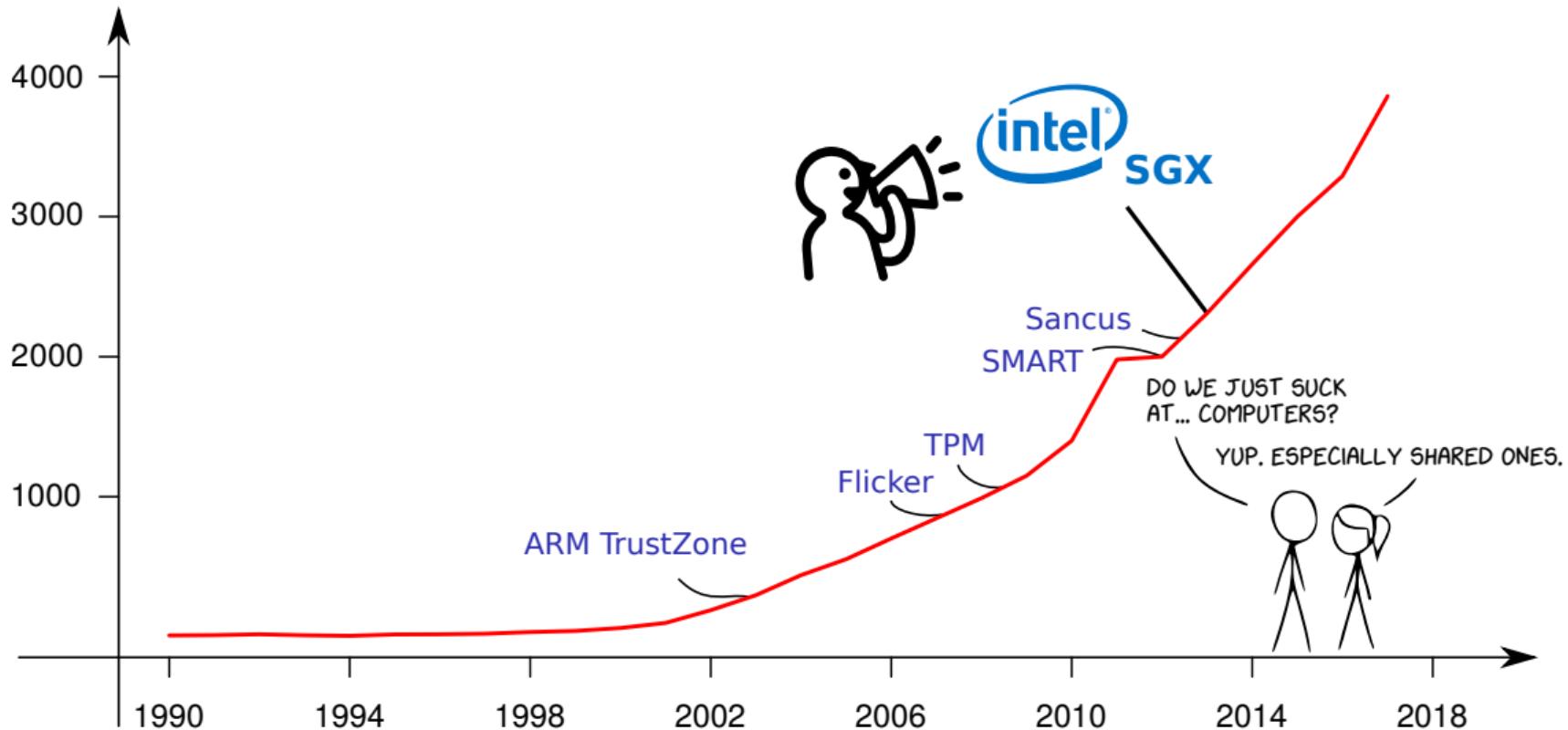
What are the ways for adversaries to create an “oracle” for all victim code+data memory access sequences?

Evolution of “side-channel attack” occurrences in Google Scholar



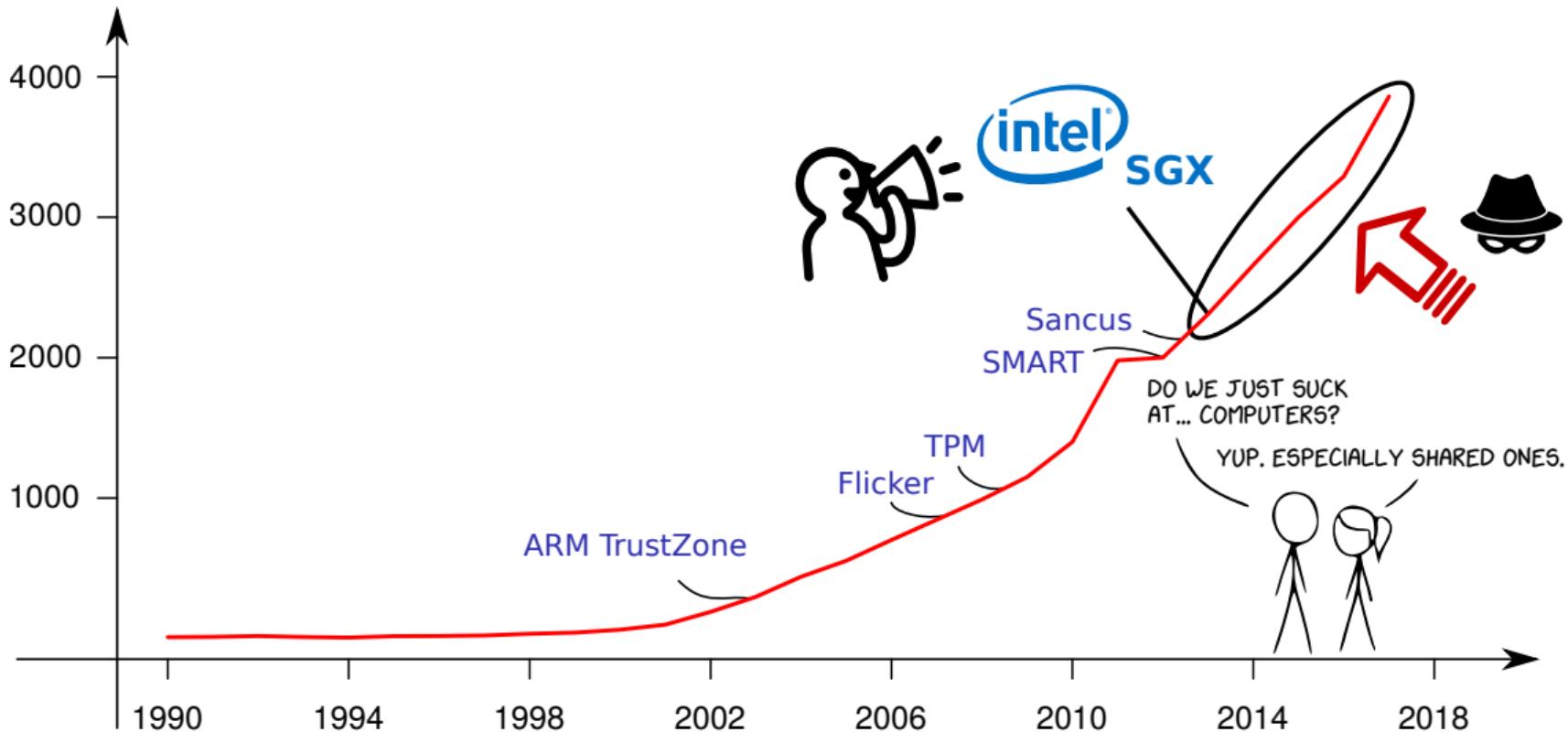
Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

Side-channel attacks and trusted computing

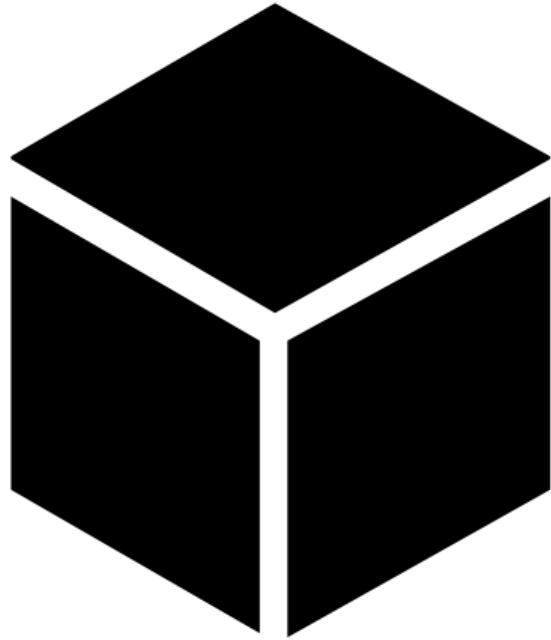


Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

Side-channel attacks and trusted computing (focus of today)

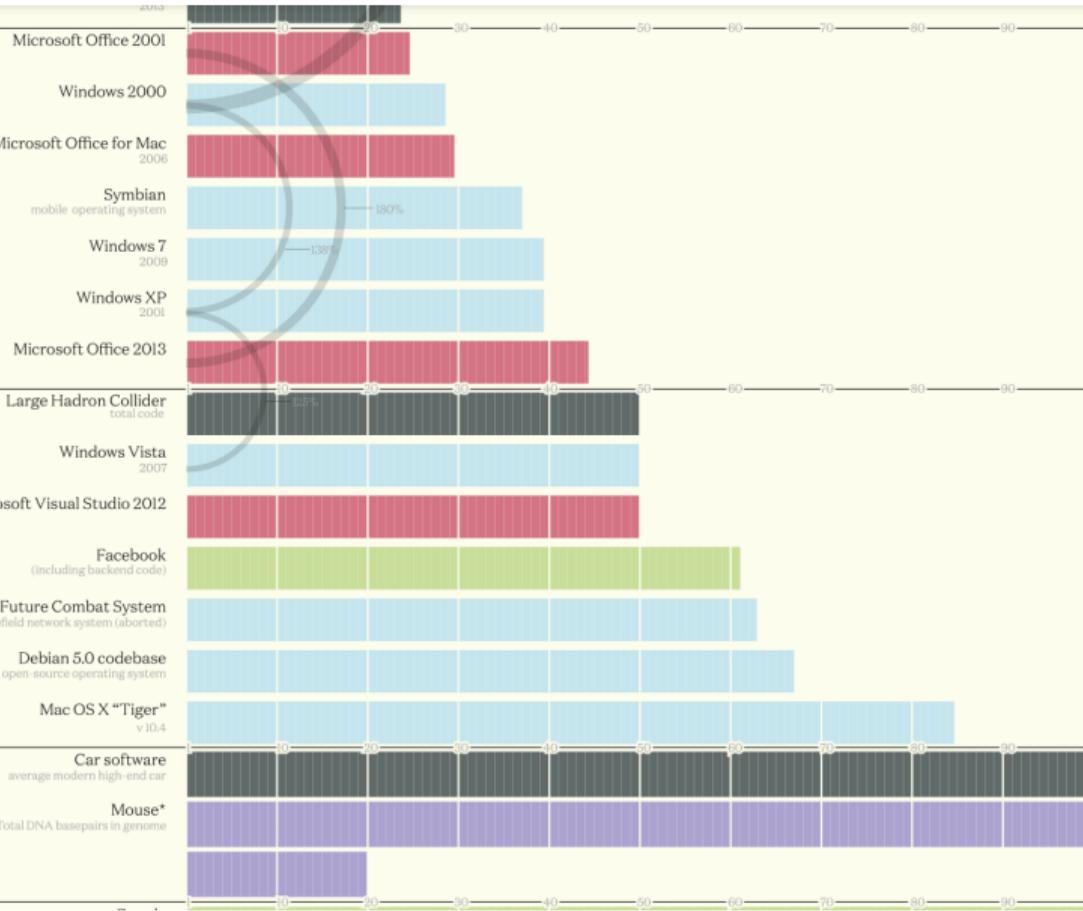


Based on github.com/Pold87/academic-keyword-occurrence and xkcd.com/1938/

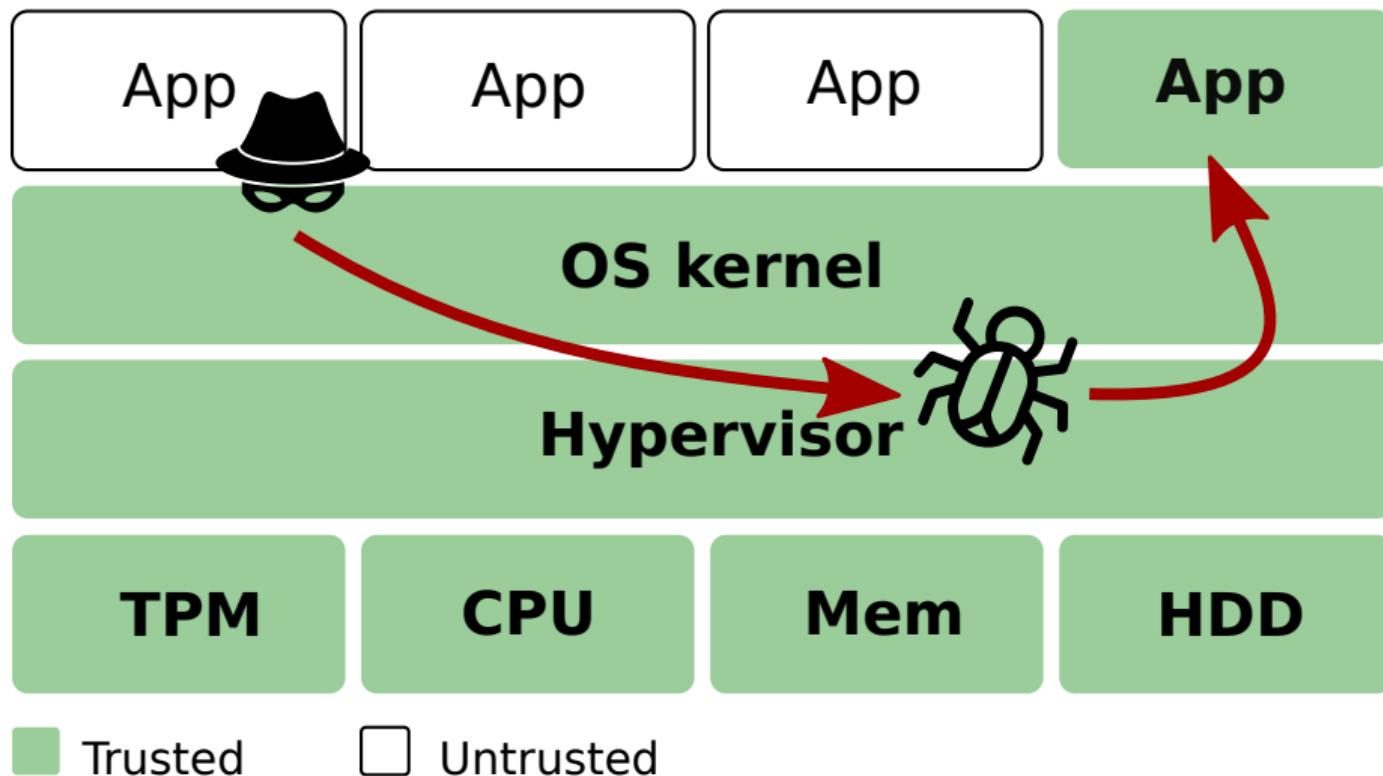


What's inside the black box?

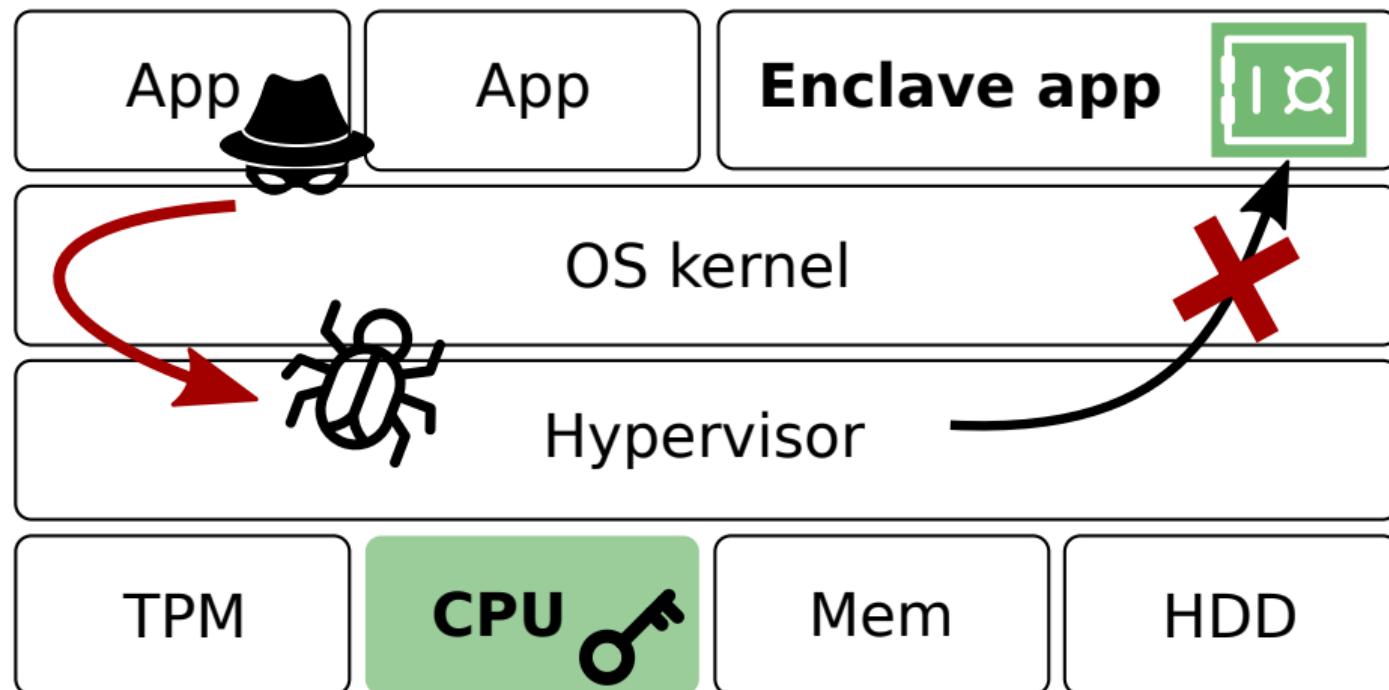
25



Enclaved execution: Reducing attack surface

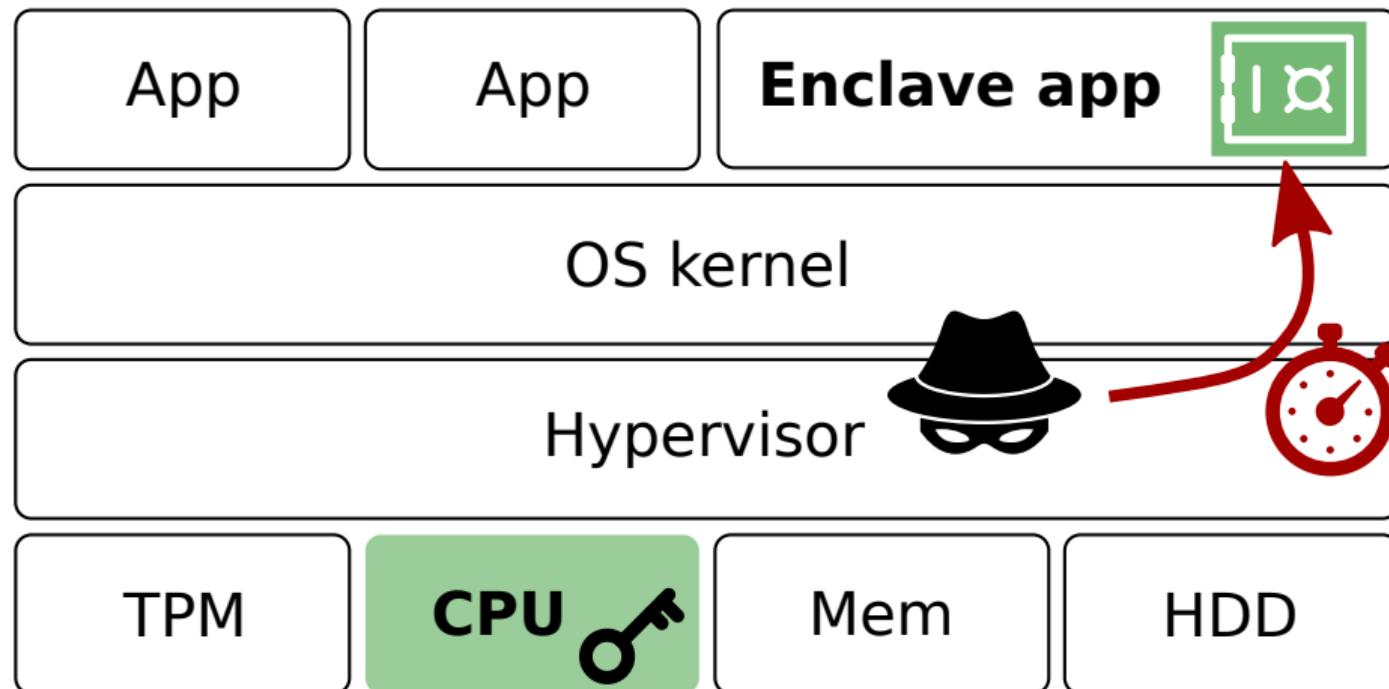


Enclaved execution: Reducing attack surface



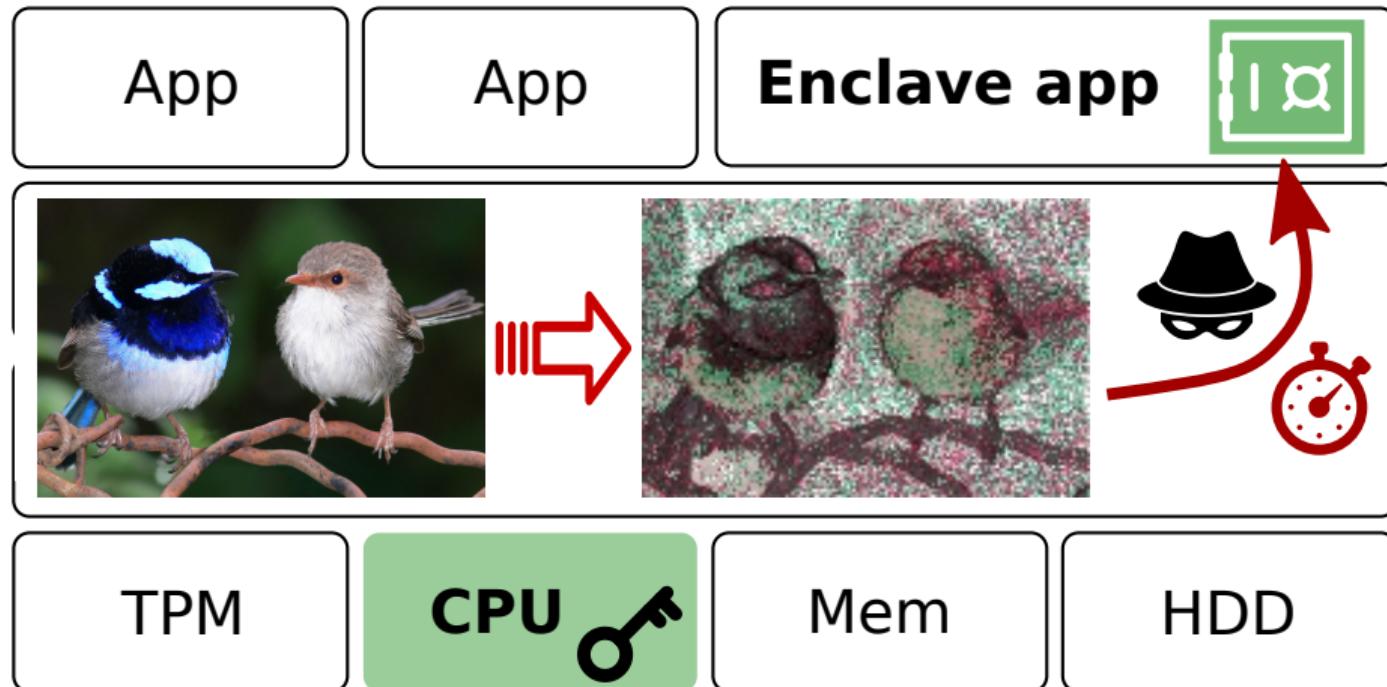
Intel SGX promise: hardware-level **isolation and attestation**

Enclaved execution: Privileged side-channel attacks



Game-changer: Untrusted OS → new class of powerful **side-channels**

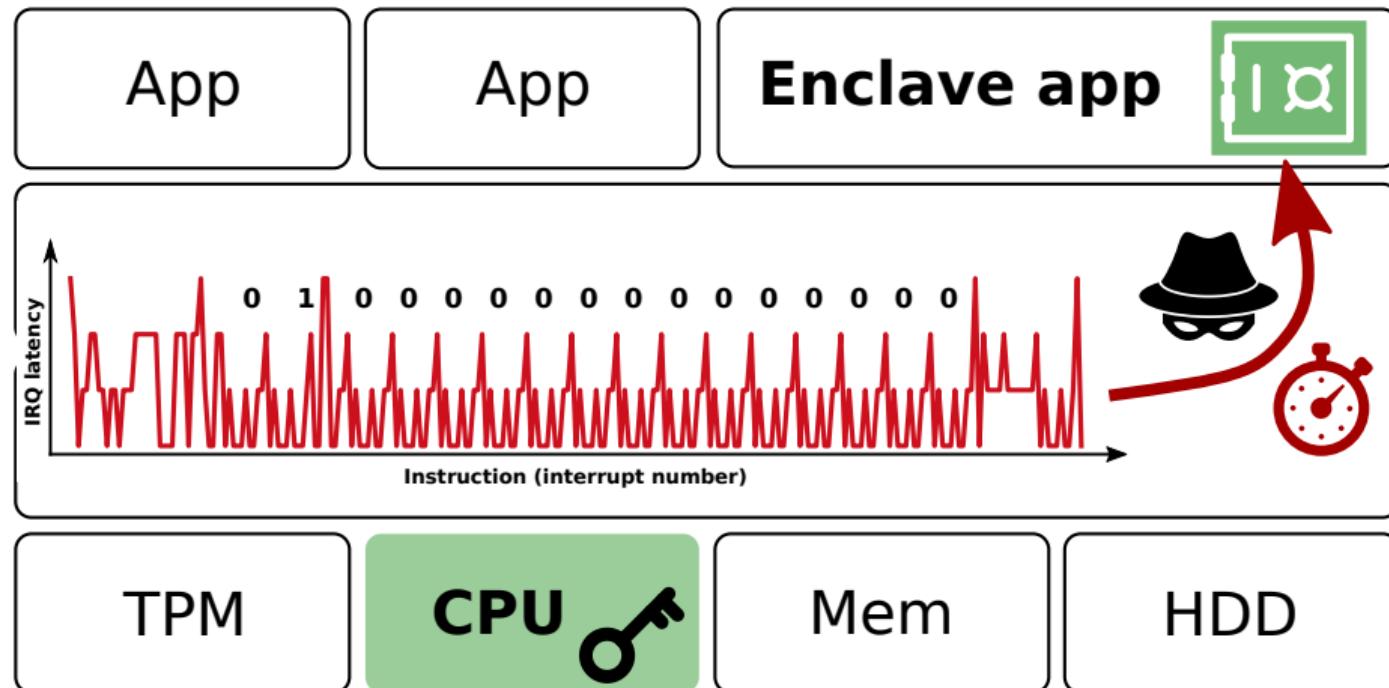
Enclaved execution: Privileged side-channel attacks



Game-changer: Untrusted OS → new class of powerful **side-channels**

Xu et al. "Controlled-channel attacks: Deterministic side-channels for untrusted operating systems", IEEE S&P 2015 [XCP15]

Enclaved execution: Privileged side-channel attacks



Game-changer: Untrusted OS → new class of powerful **side-channels**



KEEP CALM
IT IS
OUT OF SCOPE

A note on side-channel attacks (Intel)

Protection from Side-Channel Attacks

Intel® SGX does not provide explicit protection from side-channel attacks. It is the enclave developer's responsibility to address side-channel attack concerns.

In general, enclave operations that require an OCall, such as thread synchronization, I/O, etc., are exposed to the untrusted domain. If using an OCall would allow an attacker to gain insight into enclave secrets, then there would be a security concern. This scenario would be classified as a side-channel attack, and it would be up to the ISV to design the enclave in a way that prevents the leaking of side-channel information.

An attacker with access to the platform can see what pages are being executed or accessed. This side-channel vulnerability can be mitigated by aligning specific code and data blocks to exist entirely within a single page.

More important, the application enclave should use an appropriate crypto implementation that is side channel attack resistant inside the enclave if side-channel attacks are a concern.



Today's agenda: Understanding privileged side-channel leakage

- Critical remarks on **TEE isolation**:
 - Which **side-channels** exist? Which enclave **applications** are vulnerable? (Not only crypto!)
 - How to (not) **defend** against them, and at what cost?
- Focus on **privileged attack surfaces**: page tables, interrupts (Game-changer!)

Out-of-scope:

- “Traditional” leakage sources: caches, branch predictors, etc. (cf. next lecture?)
- Speculative execution attacks (Spectre, Meltdown, Foreshadow, etc.)

Today's agenda: Understanding privileged side-channel leakage

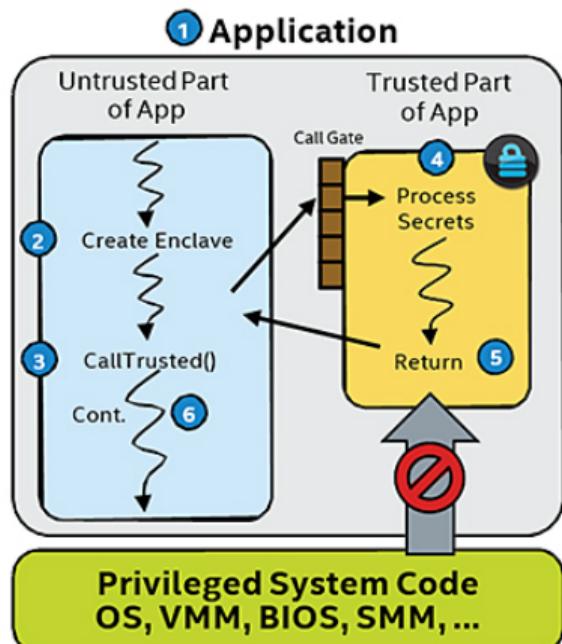
- Critical remarks on **TEE isolation**:
 - Which **side-channels** exist? Which enclave **applications** are vulnerable? (Not only crypto!)
 - How to (not) **defend** against them, and at what cost?
- Focus on **privileged attack surfaces**: page tables, interrupts (Game-changer!)

Out-of-scope:

- “Traditional” leakage sources: caches, branch predictors, etc. (cf. next lecture?)
- Speculative execution attacks (Spectre, Meltdown, Foreshadow, etc.)

 **Key question:** Infer secrets from functionally correct enclave programs through untrusted OS?
→ overview several *attack avenues*... with an explicit focus on *Intel SGX TEEs*

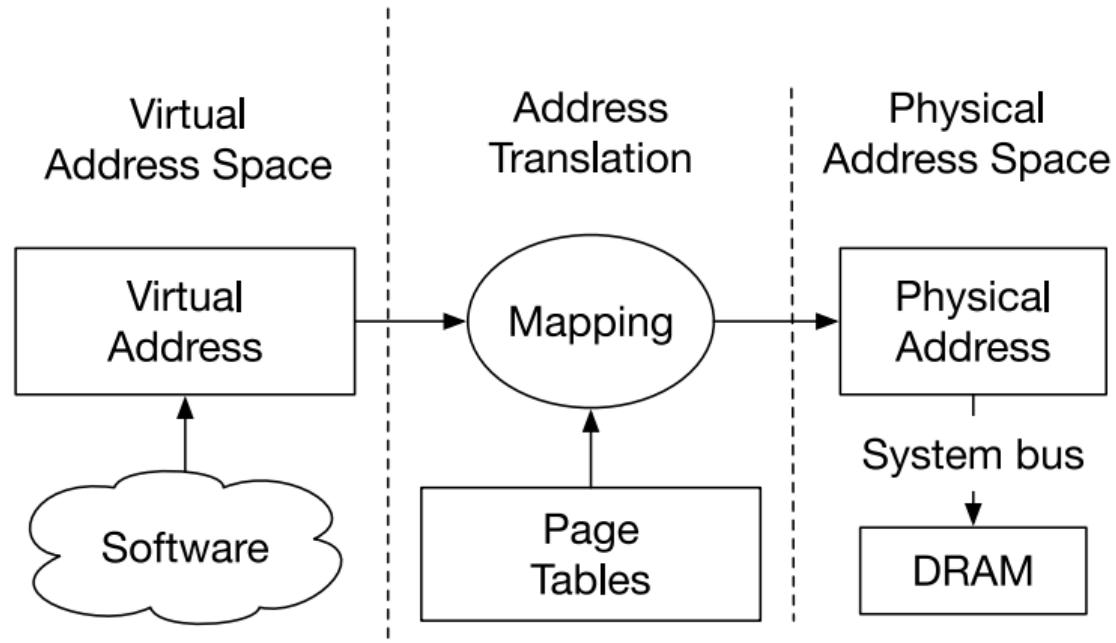
Intel SGX: A helicopter view



- Enclaves live in **user-space guest application**
- Inaccessible by all outside software (including OS)
- **Virtual memory** extensions enforce isolation
- Memory **encrypted** when outside processor package
- x86 ISA **instruction extensions:**
 - `eenter/eexit, eresume/aex`: switch in/out enclave
 - `egetkey`: hardware-level key derivation, attestation

<https://software.intel.com/en-us/sgx/details>

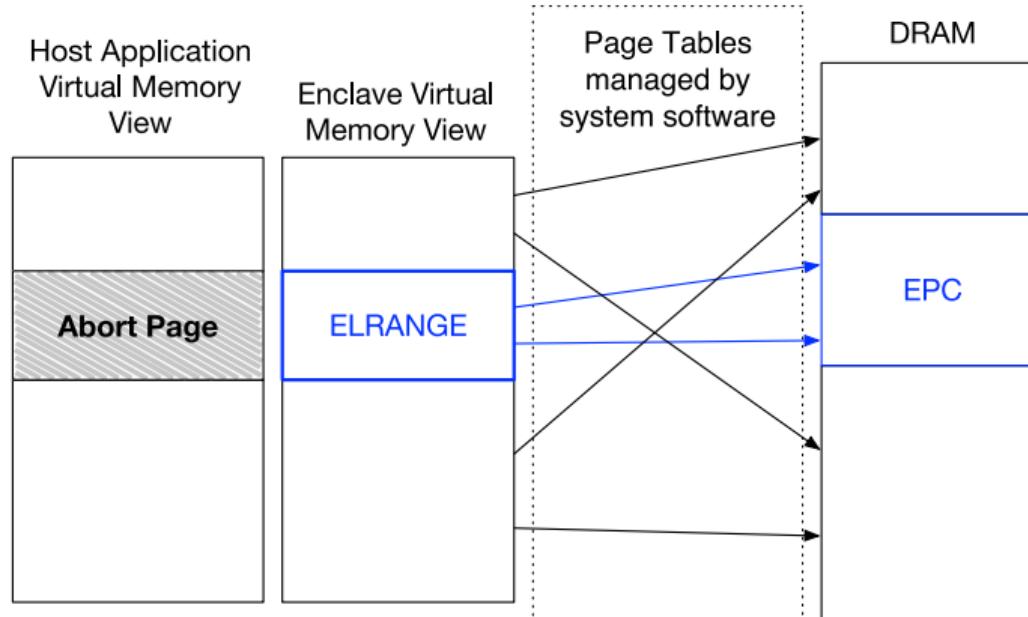
The virtual memory abstraction



Costan et al. "Intel SGX explained", IACR 2016 [CD16]

Intel SGX: How enclave accesses are enforced

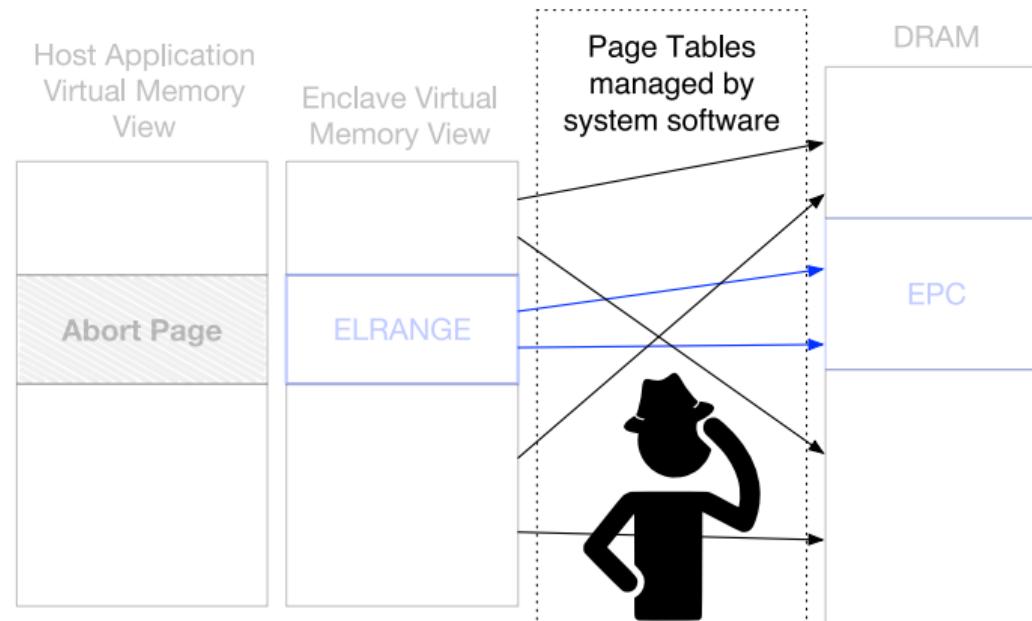
Intel SGX enclaves live in **virtual address space** of untrusted host application



Costan et al. "Intel SGX explained", IACR 2016 [CD16]

Intel SGX: How enclave accesses are enforced

Challenge: Untrusted OS controls **virtual-to-physical mapping** → address-remapping attacks!



Costan et al. "Intel SGX explained", IACR 2016 [CD16]

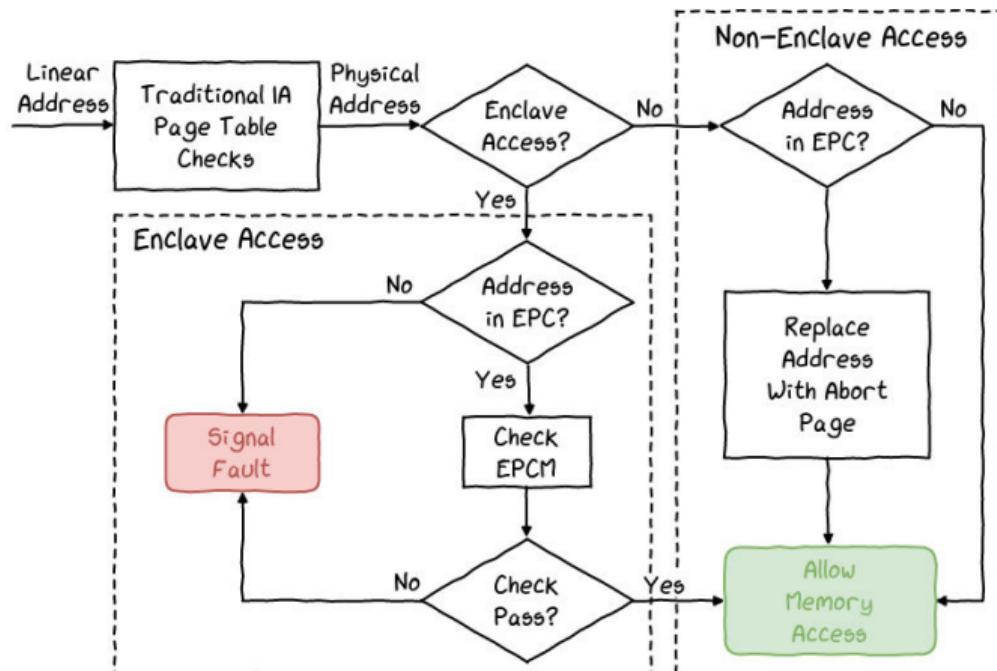
Intel SGX: How enclave accesses are enforced

Solution: Additional checks to **verify** untrusted address translation outcome



Intel SGX: How enclave accesses are enforced

Solution: Additional checks to **verify** untrusted address translation outcome



Attack idea #1



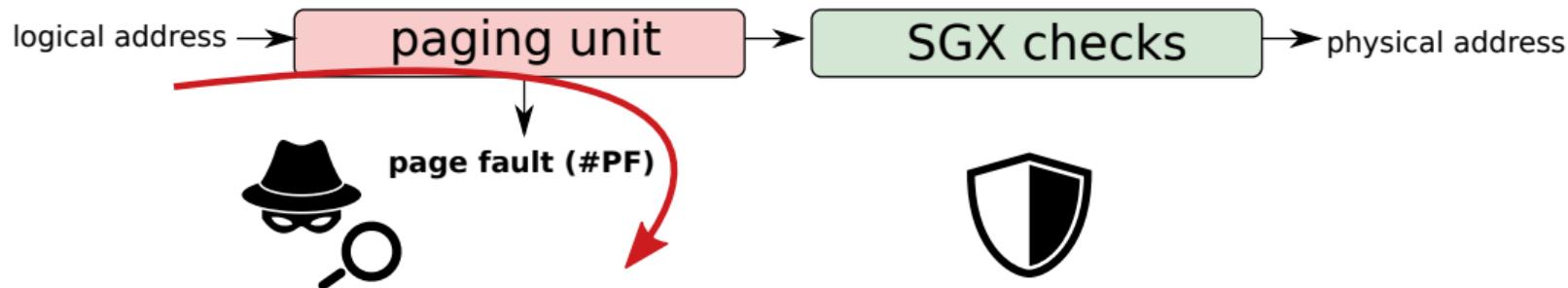
Can we abuse untrusted address translation as a side-channel?

Page faults as a side-channel



SGX machinery protects against direct address remapping attacks

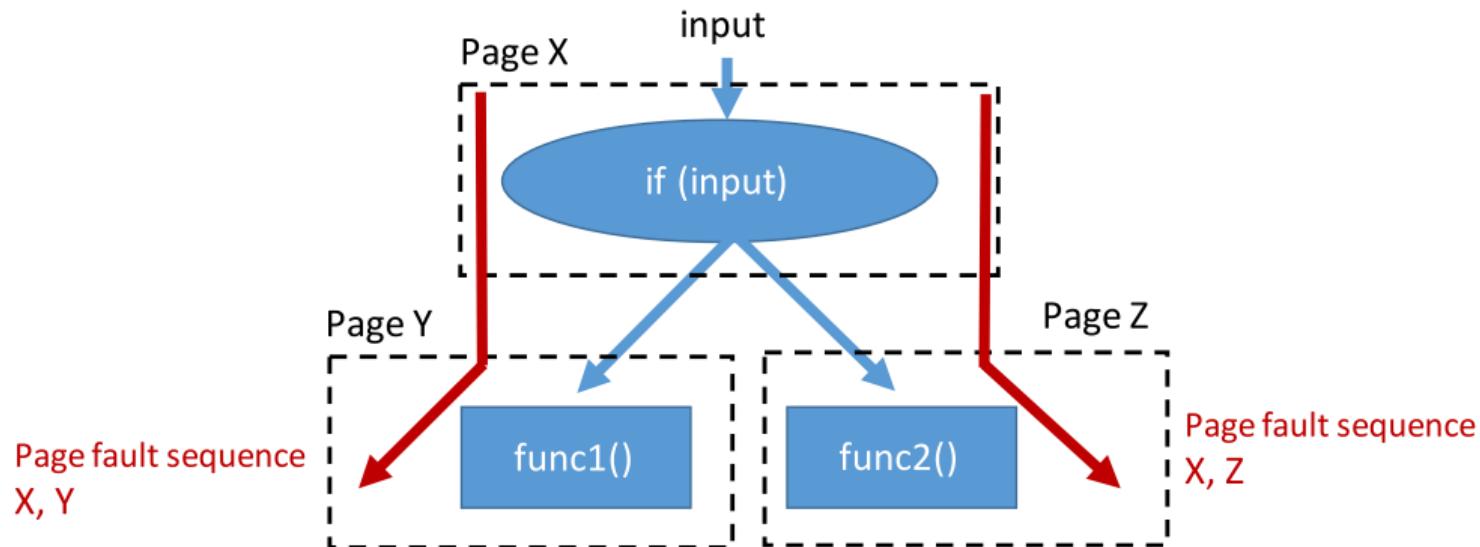
Page faults as a side-channel



... but untrusted address translation may **fault** during enclaved execution (!)

→ page fault deterministically reveals that the enclave tried to access a certain 4 KiB memory page...

Page faults as a side-channel



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

⇒ Page fault traces leak **private control data/flow**

#PF attacks: An end-to-end example

```
void inc_secret( void )
{
    if (secret)
        *a += 1;
    else
        *b += 1;
}
```

Page Table

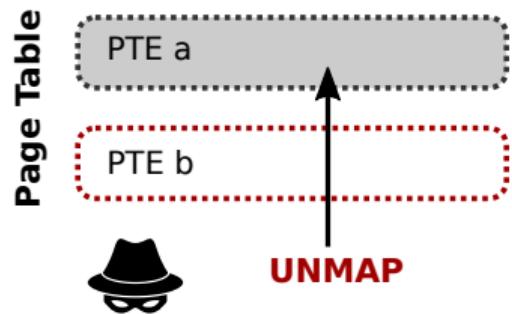
PTE a

PTE b

#PF attacks: An end-to-end example

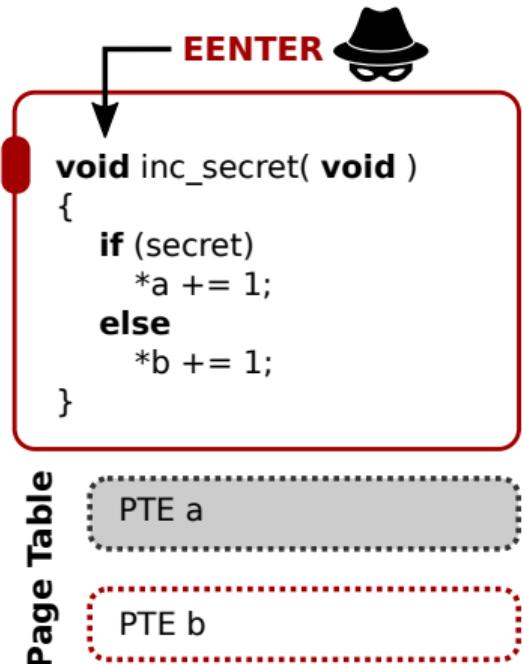
- ① Revoke access rights on *unprotected* enclave page table entry

```
void inc_secret( void )  
{  
    if (secret)  
        *a += 1;  
    else  
        *b += 1;  
}
```



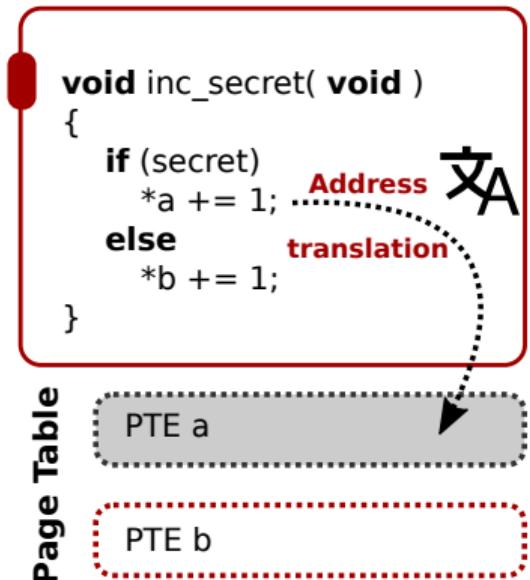
#PF attacks: An end-to-end example

- ① Revoke access rights on *unprotected* enclave page table entry
- ② Enter victim enclave



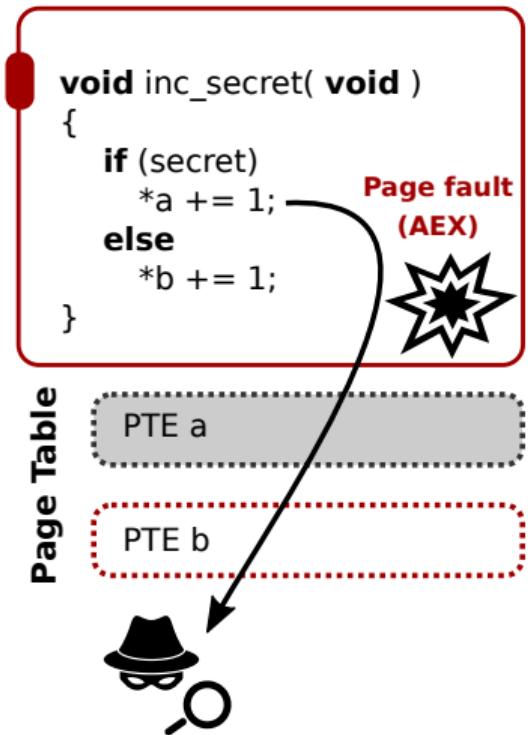
#PF attacks: An end-to-end example

- ① Revoke access rights on *unprotected enclave page table entry*
- ② Enter victim enclave
- ③ Secret-dependent data memory access
 - ~ Processor performs virt-to-phys address translation!
 - ~ By reading page table entry setup by *untrusted OS*



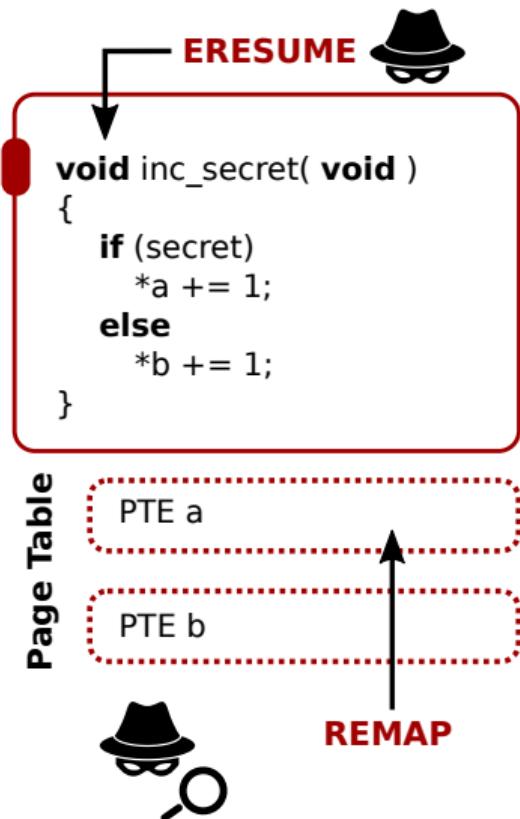
#PF attacks: An end-to-end example

- ① Revoke access rights on *unprotected enclave page table entry*
- ② Enter victim enclave
- ③ Secret-dependent data memory access
 - ~ Processor performs virt-to-phys address translation!
 - ~ By reading page table entry setup by *untrusted OS*
- ④ Virtual address not present → raise page fault
 - ~ Processor exits enclave and vectors to untrusted OS
 - ~ Noise-free side-channel signal that the enclave wants to access page A(!)

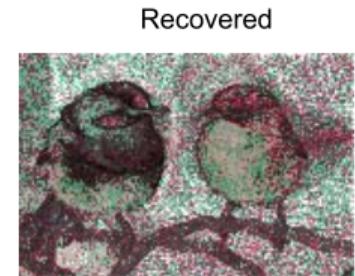


#PF attacks: An end-to-end example

- ① Revoke access rights on *unprotected enclave page table entry*
- ② Enter victim enclave
- ③ Secret-dependent data memory access
 - ~ Processor performs virt-to-phys address translation!
 - ~ By reading page table entry setup by *untrusted OS*
- ④ Virtual address not present → raise page fault
 - ~ Processor exits enclave and vectors to untrusted OS
 - ~ Noise-free side-channel signal that the enclave wants to access page A(!)
- ⑤ Restore access rights and resume victim enclave



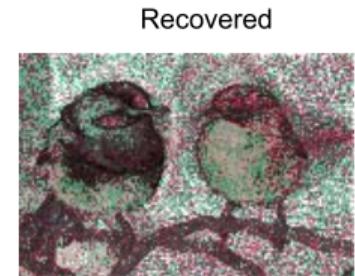
Page table-based attacks in practice



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

⇒ **Low-noise, single-run** exploitation of legacy applications

Page table-based attacks in practice



Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015 [XCP15]

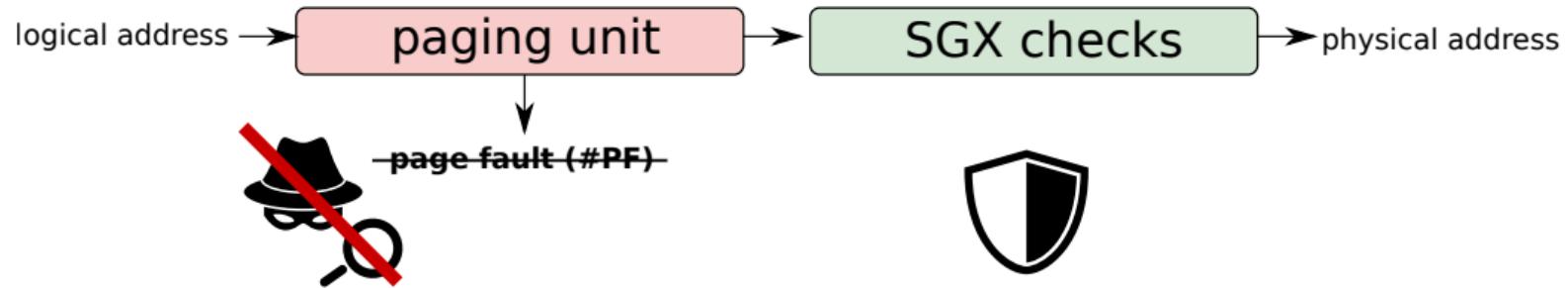
... but at a relative coarse-grained **4 KiB granularity**

Attack idea #2



What about other side-effects of address translation?

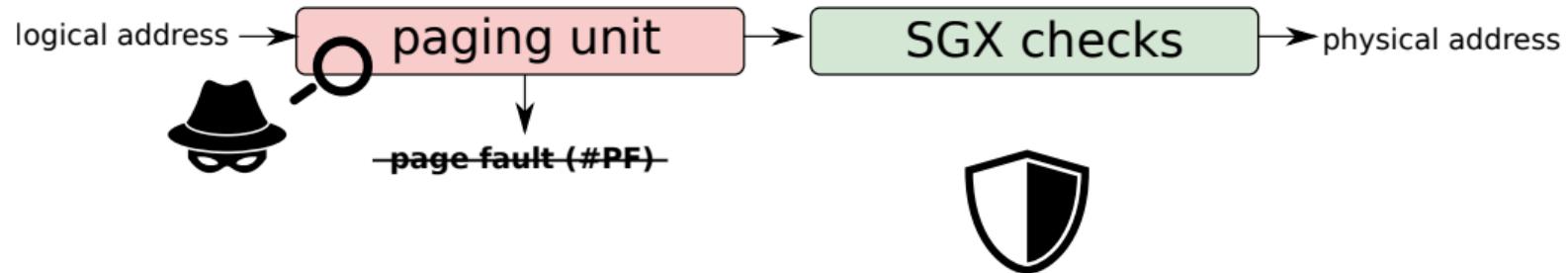
Naive solutions: Hiding enclave page faults



Shih et al. "T-SGX: Eradicating controlled-channel attacks against enclave programs", NDSS 2017 [SLKP17]

Shinde et al. "Preventing page faults from telling your secrets", AsiaCCS 2016 [SCNS16]

Naive solutions: Hiding enclave page faults



... But stealthy attacker can still learn page accesses without triggering faults!

Documented side-effects of address translation

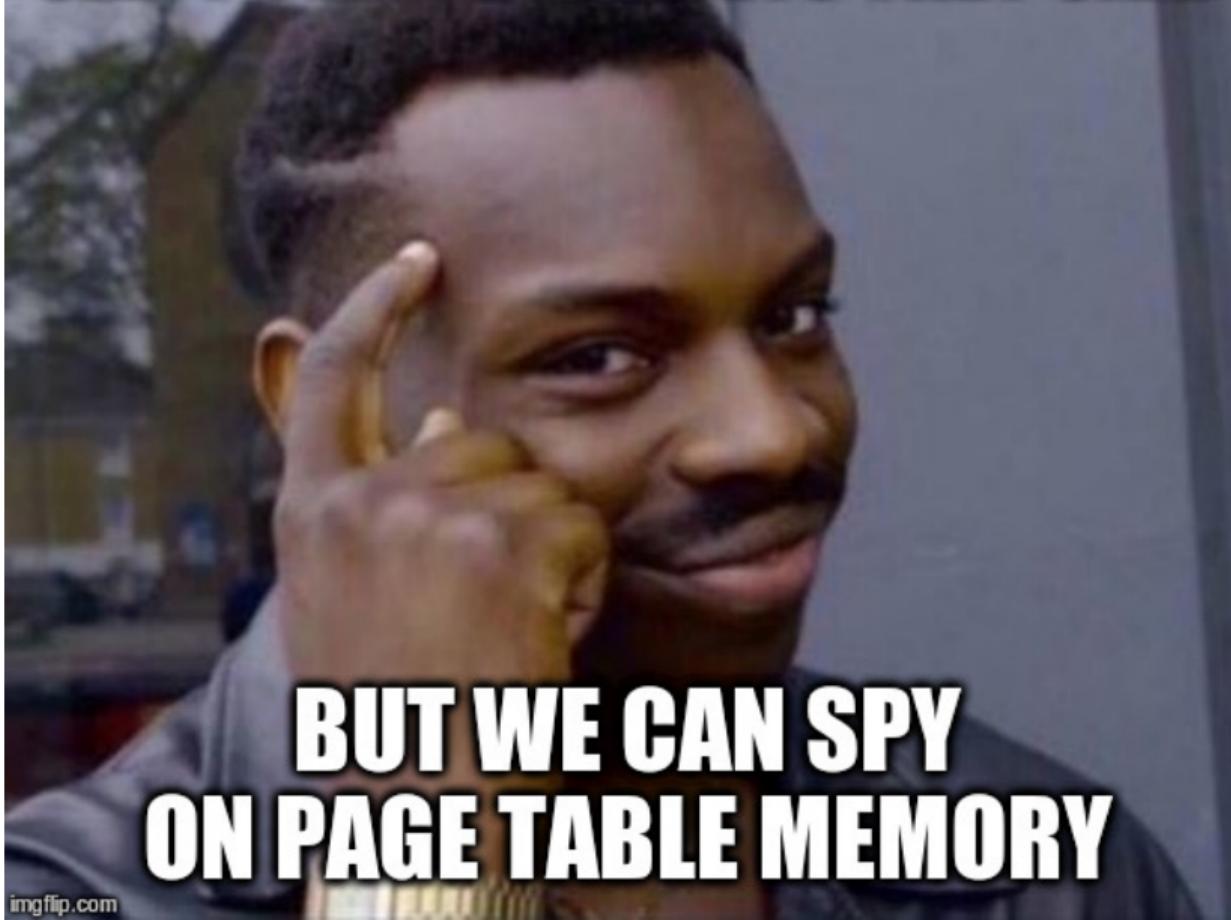
4.8 ACCESSED AND DIRTY FLAGS

For any paging-structure entry that is used during linear-address translation, bit 5 is the **accessed** flag.² For paging-structure entries that map a page (as opposed to referencing another paging structure), bit 6 is the **dirty** flag. These flags are provided for use by memory-management software to manage the transfer of pages and paging structures into and out of physical memory.

Whenever the processor uses a paging-structure entry as part of linear-address translation, it sets the accessed flag in that entry (if it is not already set).

Whenever there is a write to a linear address, the processor sets the dirty flag (if it is not already set) in the paging-structure entry that identifies the final physical address for the linear address (either a PTE or a paging-structure entry in which the PS flag is 1).

CAN'T SEE PAGE FAULTS THEY SAID



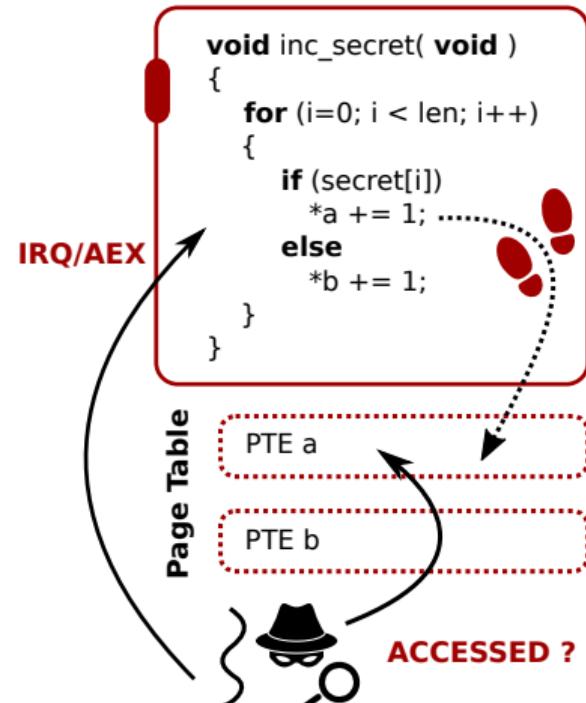
BUT WE CAN SPY
ON PAGE TABLE MEMORY

Telling your secrets without page faults

① Attack vector: PTE status flags:

- A(ccessed) bit
- D(irty) bit

~ Also updated in enclave mode!



Telling your secrets without page faults

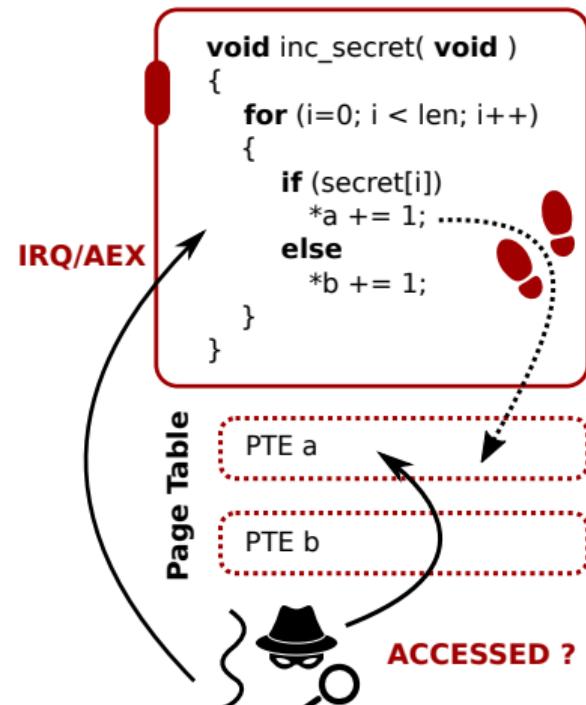
① Attack vector: PTE status flags:

- A(ccessed) bit
- D(irty) bit

~ Also updated in enclave mode!

② Attack vector: Unprotected page table memory:

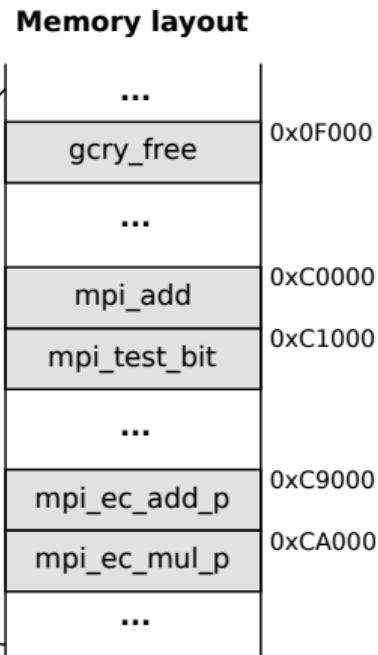
- Cached as regular data
 - Accessed during address translation
- ~ Flush+Reload cache timing attack!
(cf. next lecture)



Attacking Libgcrypt EdDSA (simplified)

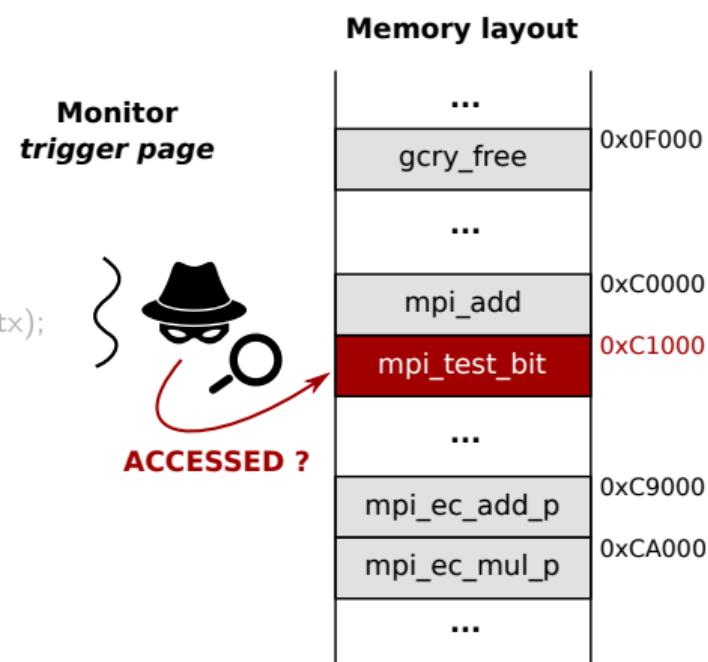
```
1 if (mpi_is_secure (scalar)) {
2     /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4     point_init (&tmppt);
5
6     for (j=nbits-1; j >= 0; j--) {
7         _gcry_mpi_ec_dup_point (result, result, ctx);
8         _gcry_mpi_ec_add_points (&tmppt, result, point, ctx);
9         point_swap_cond (result, &tmppt, mpi_test_bit (scalar, j), ctx);
10    }
11    point_free (&tmppt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17    }
18 }
```

22 Code pages per iteration



Attacking Libgcrypt EdDSA (simplified)

```
1 if (mpi_is_secure (scalar)) {
2     /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4     point_init (&tmppt);
5
6     for (j=nbits-1; j >= 0; j--) {
7         _gcry_mpi_ec_dup_point (result, result, ctx);
8         _gcry_mpi_ec_add_points (&tmppt, result, point, ctx);
9         point_swap_cond (result, &tmppt, mpi_test_bit (scalar, j), ctx);
10    }
11    point_free (&tmppt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17    }
18 }
```



Attacking Libgcrypt EdDSA (simplified)

```
1 if (mpi_is_secure (scalar)) {
2     /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4     point_init (&tmppt);
5
6     for (j=nbits-1; j >= 0; j--) {
7         _gcry_mpi_ec_dup_point (result, result, ctx);
8         _gcry_mpi_ec_add_points (&tmppt, result, point, ctx);
9         point_swap_cond (result, &tmppt, mpi_test_bit (scalar, j), ctx);
10    }
11    point_free (&tmppt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17    }
18 }
```

INTERRUPT

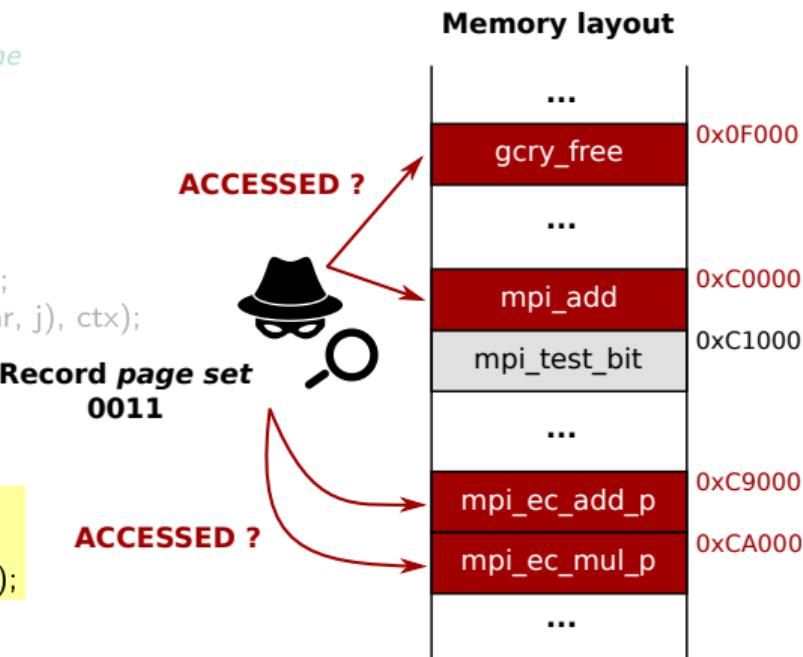


Memory layout

...	0x0F000
gcry_free	0xC0000
...	0xC1000
mpi_add	0xC1000
mpi_test_bit	0xC9000
...	0xCA000
mpi_ec_add_p	0xCA000
mpi_ec_mul_p	...

Attacking Libgcrypt EdDSA (simplified)

```
1 if (mpi_is_secure (scalar)) {
2     /* If SCALAR is in secure memory we assume that it is the
3        secret key we use constant time operation. */
4     point_init (&tmppt);
5
6     for (j=nbits-1; j >= 0; j--) {
7         _gcry_mpi_ec_dup_point (result, result, ctx);
8         _gcry_mpi_ec_add_points (&tmppt, result, point, ctx);
9         point_swap_cond (result, &tmppt, mpi_test_bit (scalar, j), ctx);
10    }
11    point_free (&tmppt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17    }
18 }
```



Attacking Libgcrypt EdDSA (simplified)

```
1 if (mpi_is_secure (scalar)) {
2     /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4     point_init (&tmppt);
5
6     for (j=nbits-1; j >= 0; j--) {
7         _gcry_mpi_ec_dup_point (result, result, ctx);
8         _gcry_mpi_ec_add_points (&tmppt, result, point, ctx);
9         point_swap_cond (result, &tmppt, mpi_test_bit (scalar, j), ctx);
10    }
11    point_free (&tmppt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17    }
18 }
```



RESUME

Full 512-bit key recovery, single run

Memory layout

...	0x0F000
gcry_free	0xC0000
...	0xC1000
mpi_add	0xC1000
mpi_test_bit	0xC9000
...	0xCA000
mpi_ec_add_p	0xCA000
mpi_ec_mul_p	...

Attack idea #3



Can we further improve the temporal resolution?

Intel's note on side-channel attacks (revisited)

Protection from Side-Channel Attacks

Intel® SGX does not provide explicit protection from side-channel attacks. It is the enclave developer's responsibility to address side-channel attack concerns.

In general, enclave operations that require an OCall, such as thread synchronization, I/O, etc., are exposed to the untrusted domain. If using an OCall would allow an attacker to gain insight into enclave secrets, then there would be a security concern. This scenario would be classified as a side-channel attack, and it would be up to the ISV to design the enclave in a way that prevents the leaking of side-channel information.

An attacker with access to the platform can see what pages are being executed or accessed. This side-channel vulnerability can be mitigated by aligning specific code and data blocks to exist entirely within a single page.

More important, the application enclave should use an appropriate crypto implementation that is side channel attack resistant inside the enclave if side-channel attacks are a concern.

<https://software.intel.com/en-us/node/703016>

Temporal resolution limitations for the page fault oracle

Counting `strlen` loop iterations

Note: page fault-driven attacks cannot make progress for single code + data page

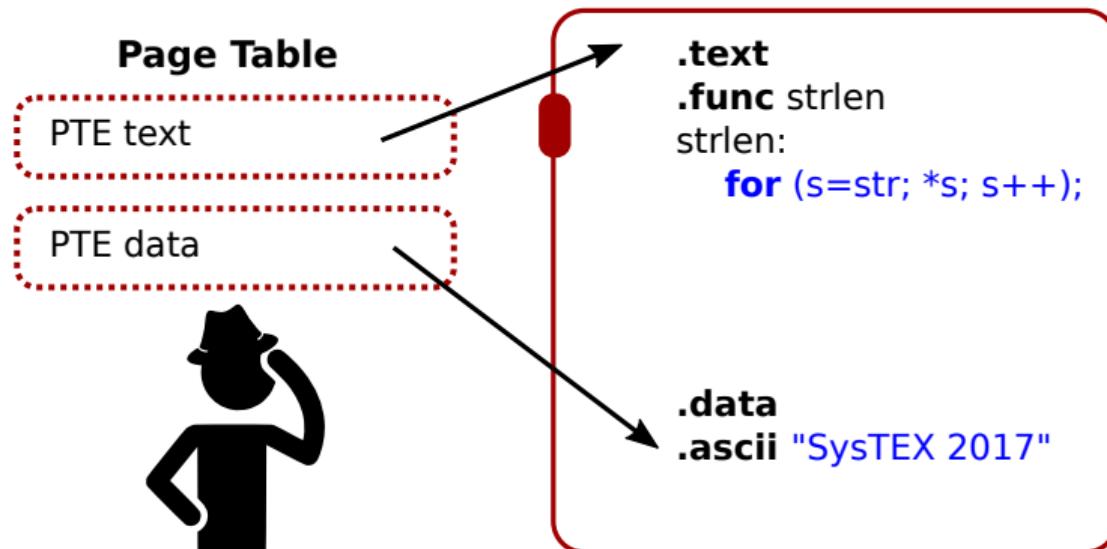
```
1 size_t strlen (char *str)
2 {
3     char *s;
4
5     for (s = str; *s; ++s);
6     return (s - str);
7 }
```

⇒ tight loop: 4 instructions, single memory operand, single code + data page

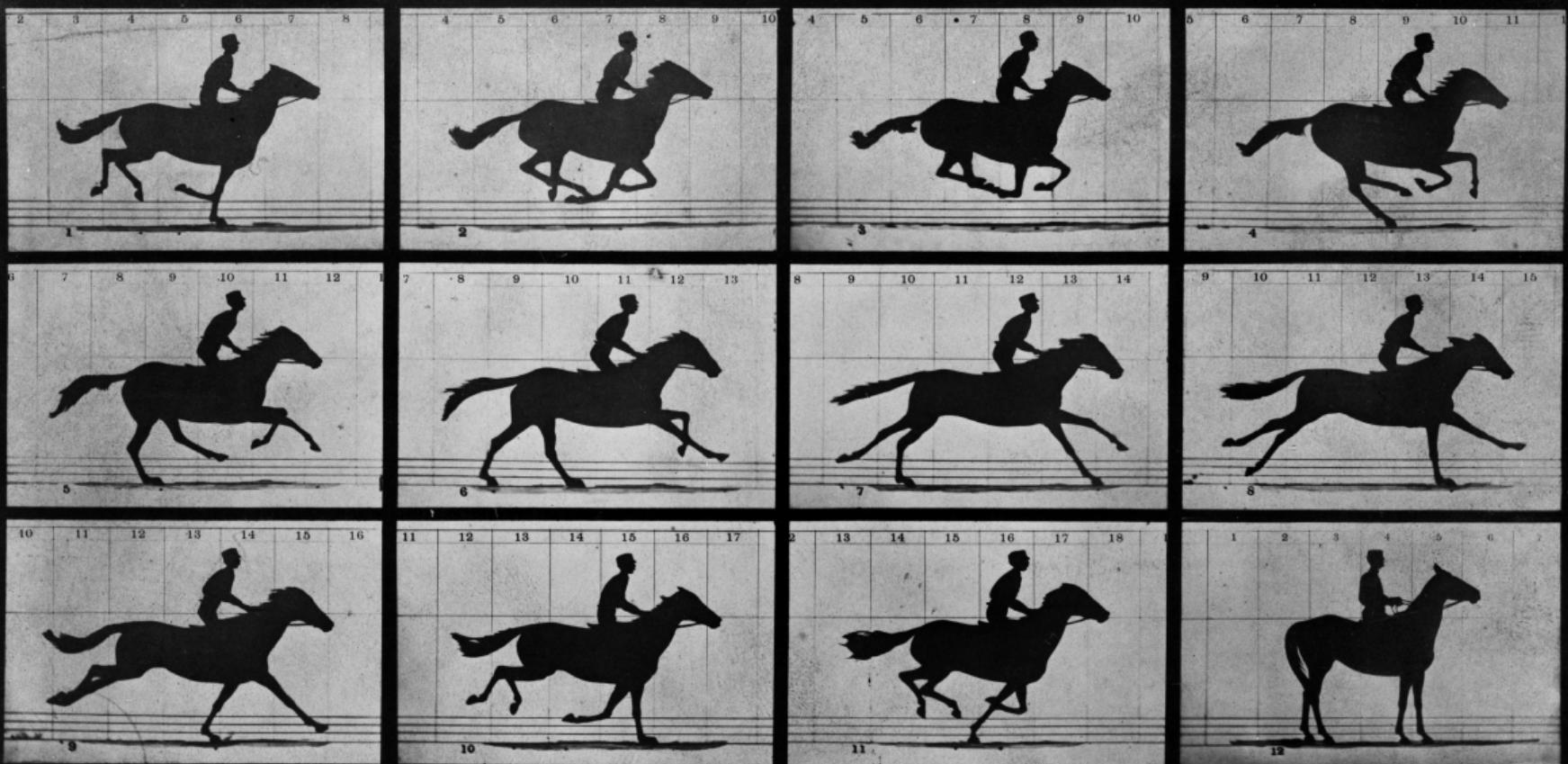
Temporal resolution limitations for the page fault oracle

Counting `strlen` loop iterations

⇒ **progress** requires both pages present ↔ page fault **oracle** requires non-present pages







Copyright, 1878, by MUYBRIDGE.

MORSE'S Gallery, 417 Montgomery St., San Francisco.

THE HORSE IN MOTION.

Illustrated by

LEWIS CARROLL

CHARLES L. DODGSON

Building a precise single-stepping primitive

⚠ **SGX-Step goal:** executing enclaves one instruction at a time

Challenge: we need a very precise timer interrupt:

- (:(x86 hardware *debug features* disabled in enclave mode
- (:) ... but we have *root access!*

Building a precise single-stepping primitive

⚠ **SGX-Step goal:** executing enclaves one instruction at a time

Challenge: we need a very precise timer interrupt:

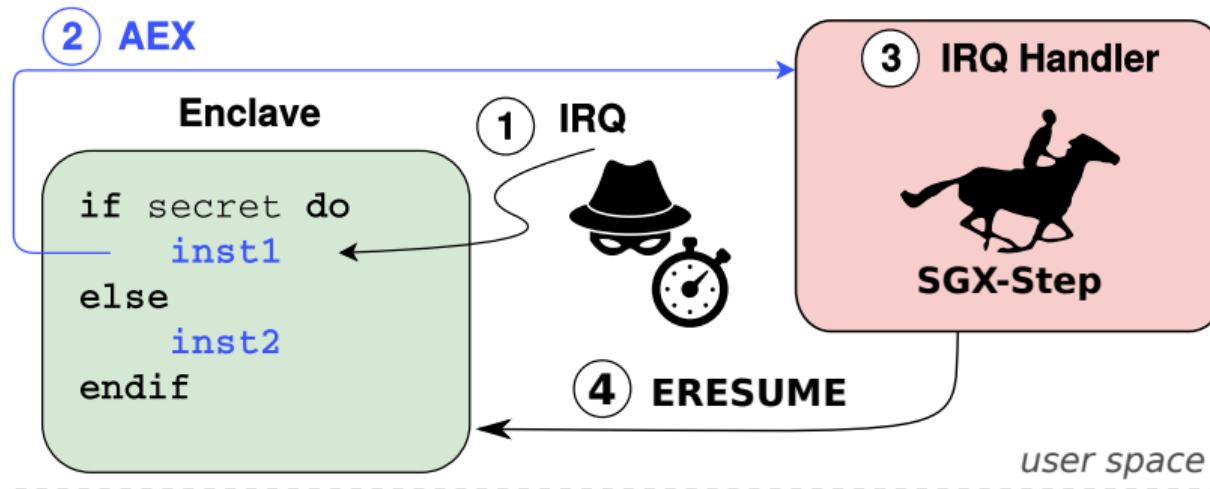
- (:(x86 hardware *debug features* disabled in enclave mode
- (:) ... but we have *root access!*

⇒ Setup user-space virtual **memory mappings** for x86 APIC

```
jo@sgx-laptop:~$ cat /proc/iomem | grep "Local APIC"
fee00000-fee00fff : Local APIC
jo@sgx-laptop:~$ sudo devmem2 0xFEE00030 h
/dev/mem opened.
Memory mapped at address 0x7f37dc187000.
Value at address 0xFEE00030 (0x7f37dc187030): 0x15
jo@sgx-laptop:~$ █
```

SGX-Step: Executing enclaves one instruction at a time

SGX-Step: user space APIC timer + interrupt handling 😊

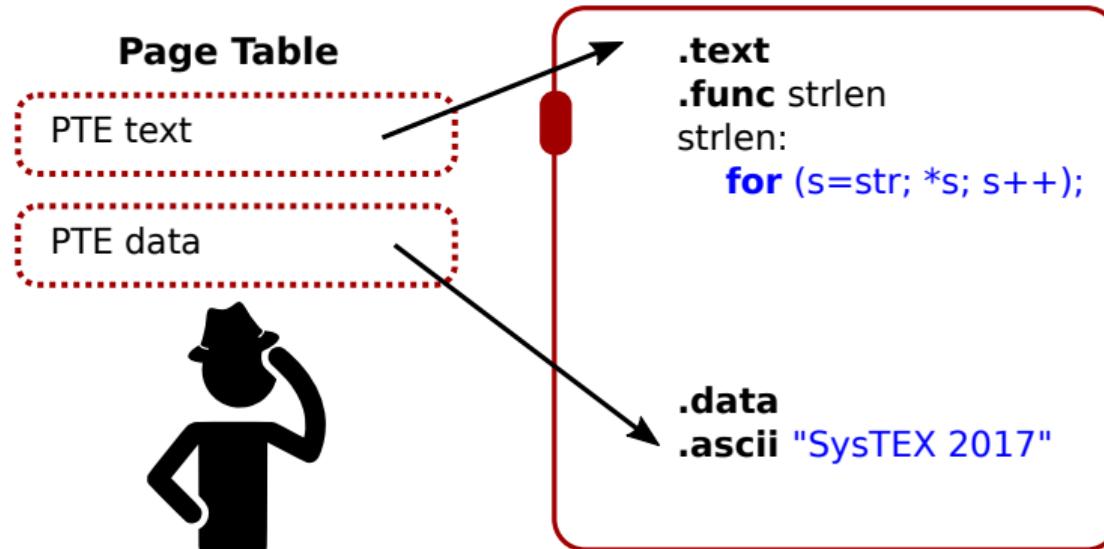


Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017 [VBPS17]

High-resolution attack example: Counting `strlen` loop iterations

Page fault adversary

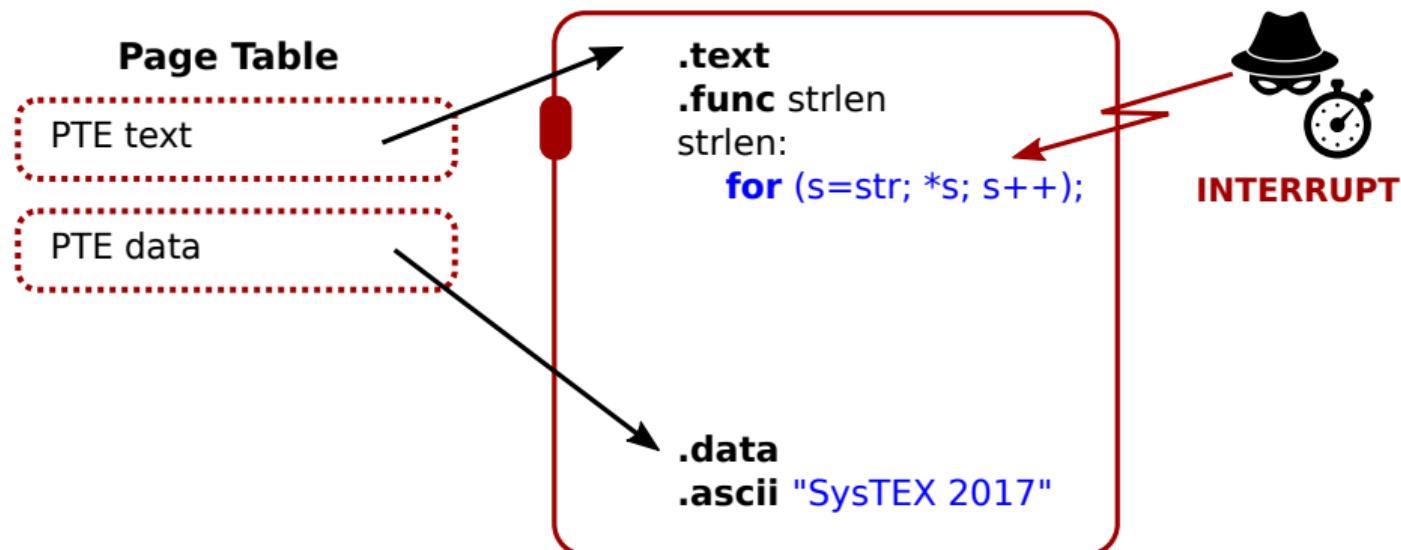
Progress ⇒ both code + data pages present 😐



High-resolution attack example: Counting `strlen` loop iterations

Single-stepping adversary

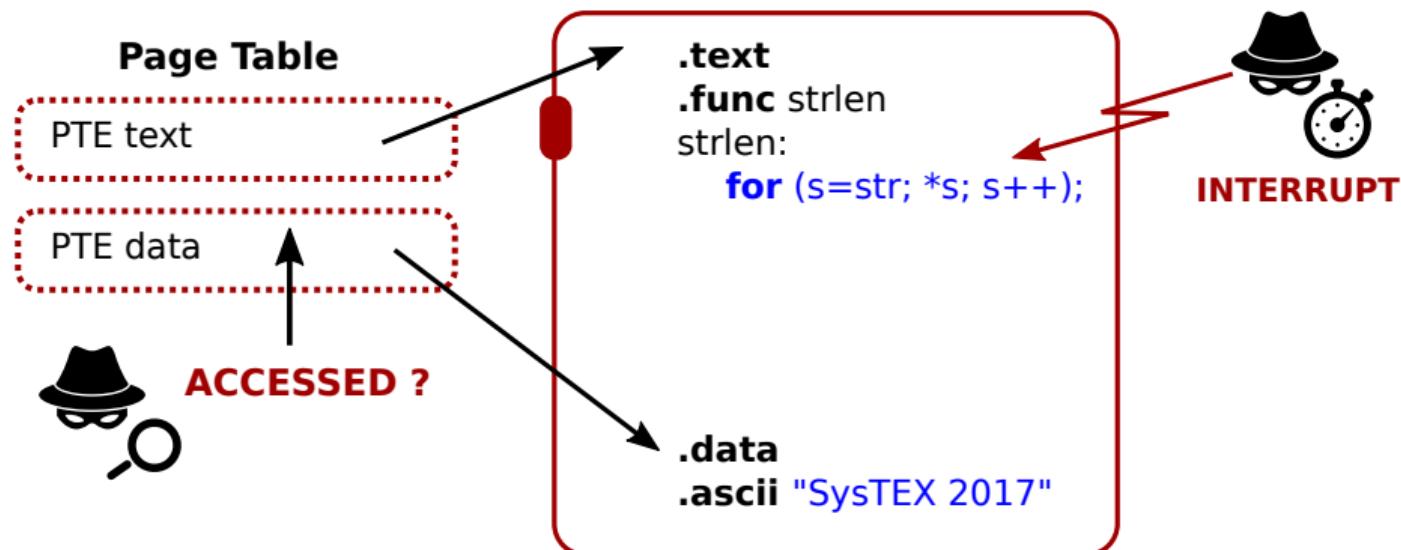
Execute one step → **interrupt** → probe accessed bit ☺ → resume



High-resolution attack example: Counting `strlen` loop iterations

Single-stepping adversary

Execute one step → **interrupt** → probe accessed bit ☺ → resume





2 Background

On February 16th, 2018, a team of security researchers at Catholic University of Leuven (KU Leuven) disclosed to Intel Corporation an issue with Edger8r Tool within the Intel® Software Guard Extensions (Intel® SGX) Software Developer's Kit (SDK). This issue could cause the Edger8r tool to generate source code that could, when used as intended within an SGX enclave, expose the enclave to a side-channel attack. The attack would then have the potential to disclose confidential data within the enclave.

https://software.intel.com/sites/default/files/managed/e1/ec/180309_SGX_SDK_Developer_Guidance_Edger8r.pdf

CVE-2018-3626: `strlen()` side-channel attacks in practice

```
static sgx_status_t SGX_CDECL sgx_ecall_pointer_string(void* pms)
{
    CHECK_REF_POINTER(pms, sizeof(ms_ecall_pointer_string_t));
    ms_ecall_pointer_string_t* ms =
        SGX_CAST(ms_ecall_pointer_string_t*, pms);
    sgx_status_t status = SGX_SUCCESS;
    char* _tmp_str = ms->ms_str;
    size_t _len_str = _tmp_str ? strlen(_tmp_str) + 1 : 0;
    char* _in_str = NULL;

    CHECK_UNIQUE_POINTER(_tmp_str, _len_str);
```

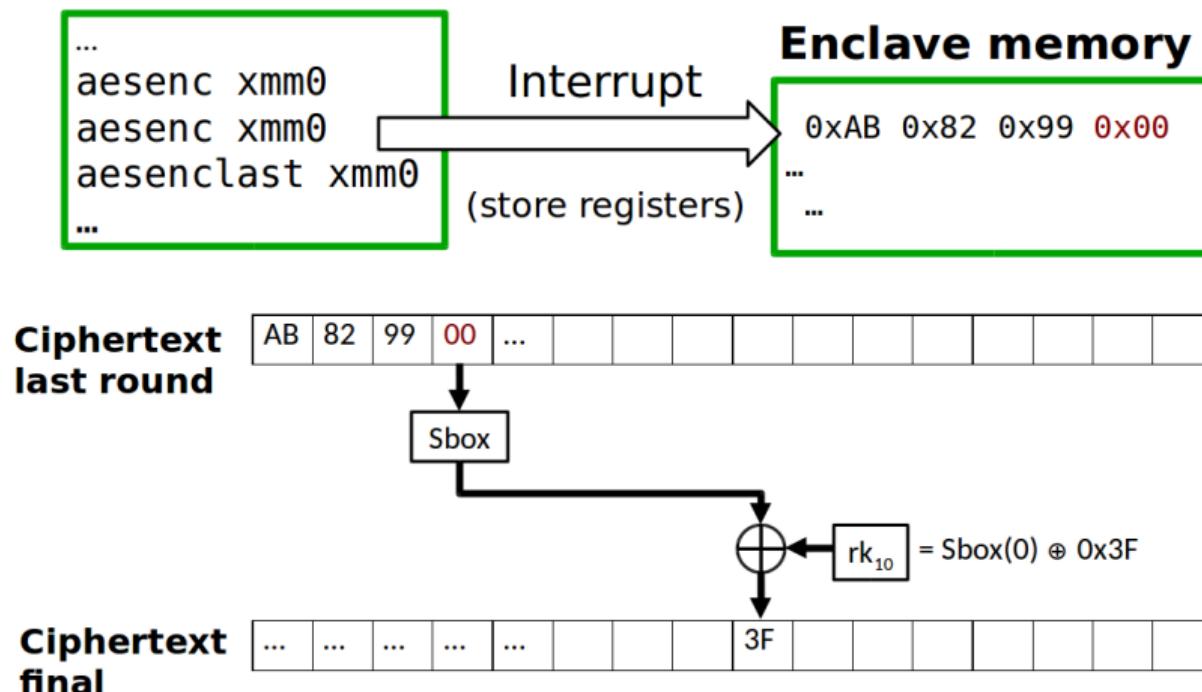
 **Side-channel oracle:** Execute `strlen()` on attacker-provided pointer!

- First execute `strlen()`, only then validate *untrusted argument pointer*...
- ⇒ Side-channel leakage reveals positions of 0x00 bytes in enclave memory

ALL YOUR ZERO BYTES

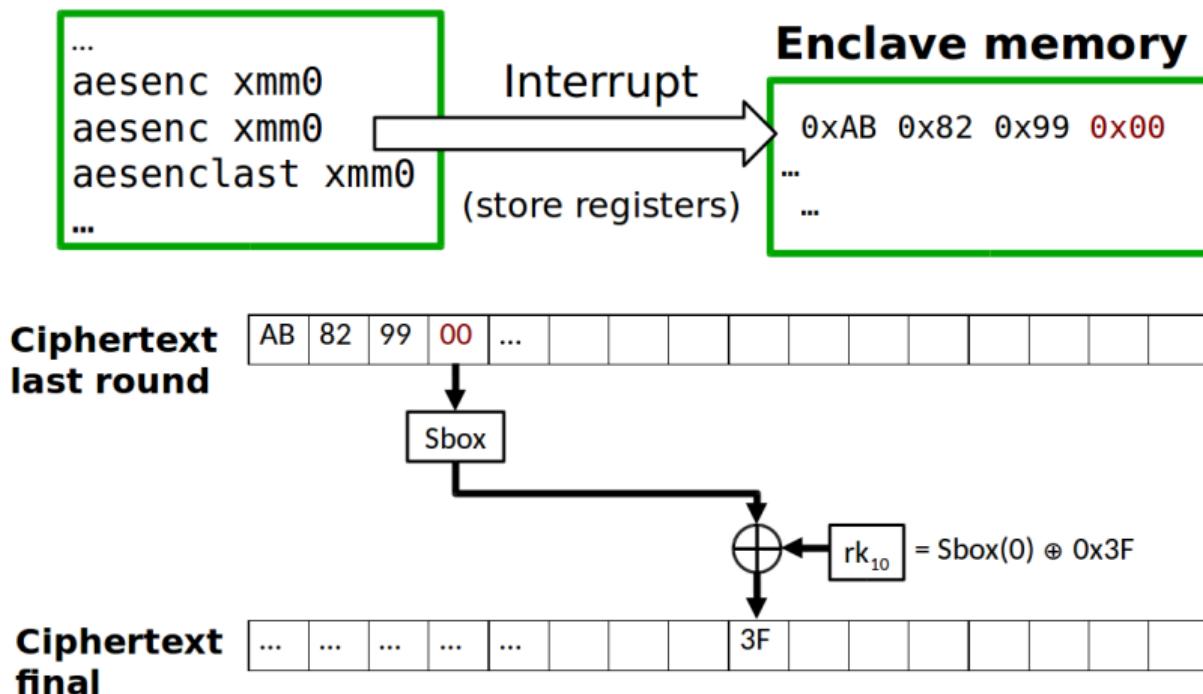
ARE BELONG TO US

Breaking AES-NI with strlen() null byte oracle



Van Bulck et al., "A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes". CCS 2019 (to appear) [VBOM⁺19]

Breaking AES-NI with `strlen()` null byte oracle



Van Bulck et al. "A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes", CCS 2019 (to appear) [VBOM⁺19]

Reconstructing the full AES-NI round key

Algorithm 1 `strlen()` oracle AES key recovery where $S(\cdot)$ denotes the AES SBox and $SR(p)$ the position of byte p after AES ShiftRows.

```
while not full key  $K$  recovered do
     $(P, C, L) \leftarrow$  random plaintext, associated ciphertext, strlen oracle
    if  $L < 16$  then
         $K[SR(L)] \leftarrow C[SR(L)] \oplus S(0)$ 
    end if
end while
```

Attack idea #4



What about simplified processors without virtual memory?

Sancus: Open-source trusted computing for the IoT (cf. lecture 2)

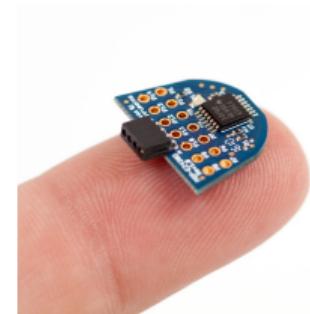
Embedded **enclaved execution**:

- ISA extensions for **isolation** & **attestation**
- Save + clear CPU state on **enclave interrupt** (~SGX)



Extremely **low-end processor** (openMSP430):

- **Area:** ≤ 2 kLUTs
 - **Deterministic** execution: *no pipeline/cache/MMU/...*
 - CPU “as simple as it gets”
- No known microarchitectural **side-channels** (!)



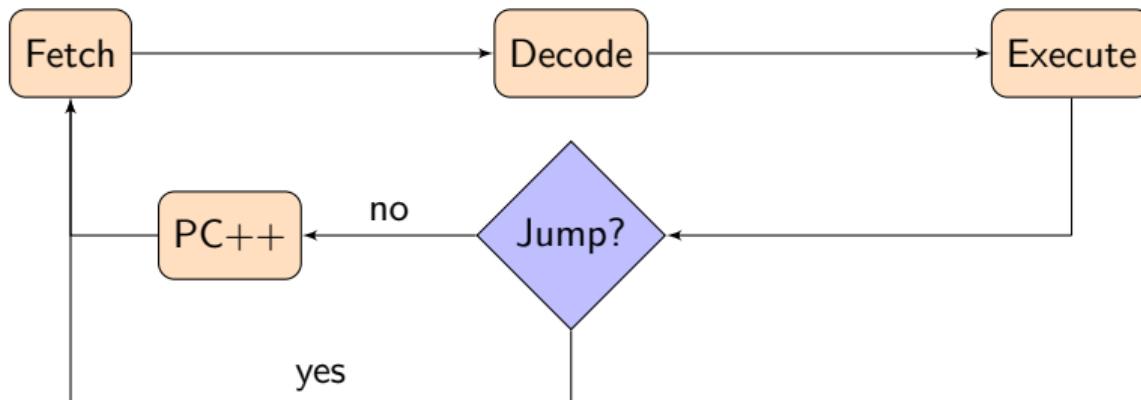
Noorman et al. "Sancus 2.0: A Low-Cost Security Architecture for IoT devices", ACM TOPS 2017 [NVBM⁺17]

De Clercq et al. "Secure interrupts on low-end microcontrollers", IEEE ASAP 2014 [dCPSV14]

⌚ <https://github.com/sancus-pma> and <https://distrinet.cs.kuleuven.be/software/sancus/>

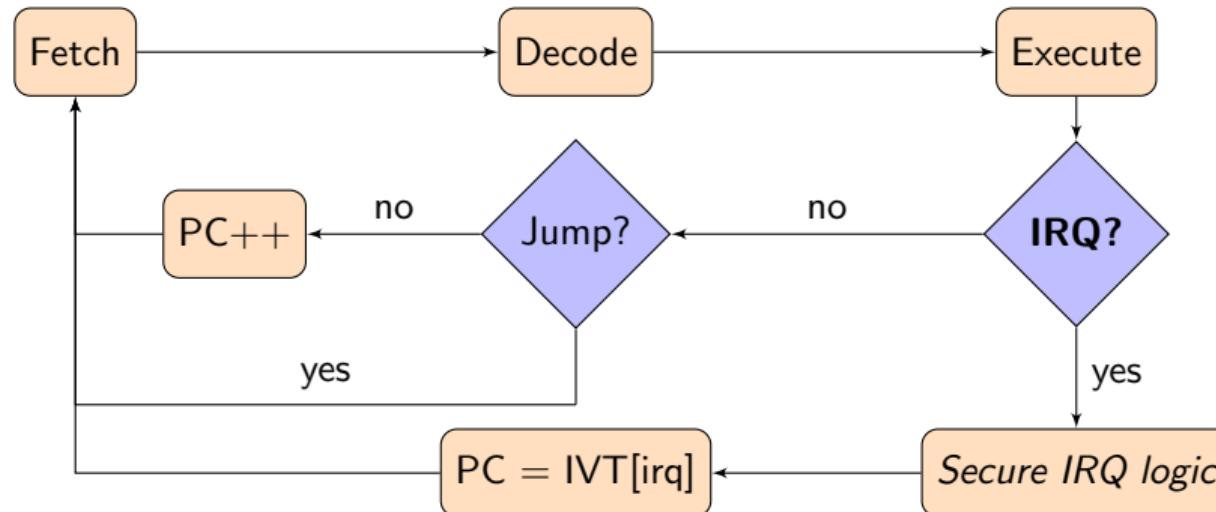
Back to basics: Fetch-decode-execute

Elementary CPU behavior: stored program computer



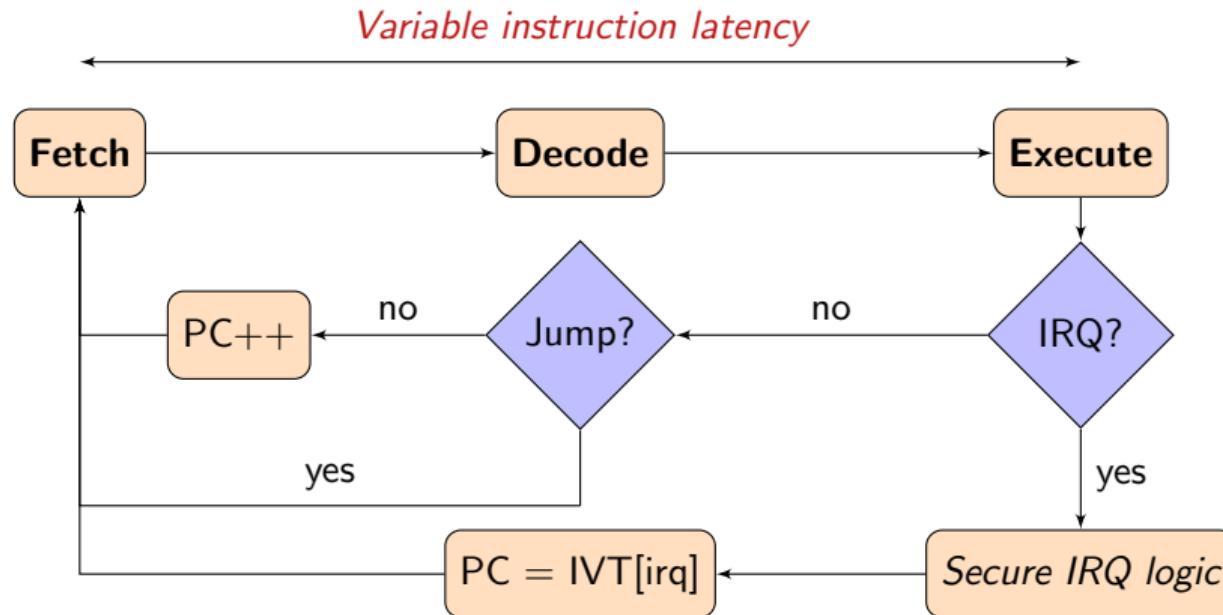
Back to basics: Fetch-decode-execute

Interrupts: asynchronous real-world events, handled on instruction retirement

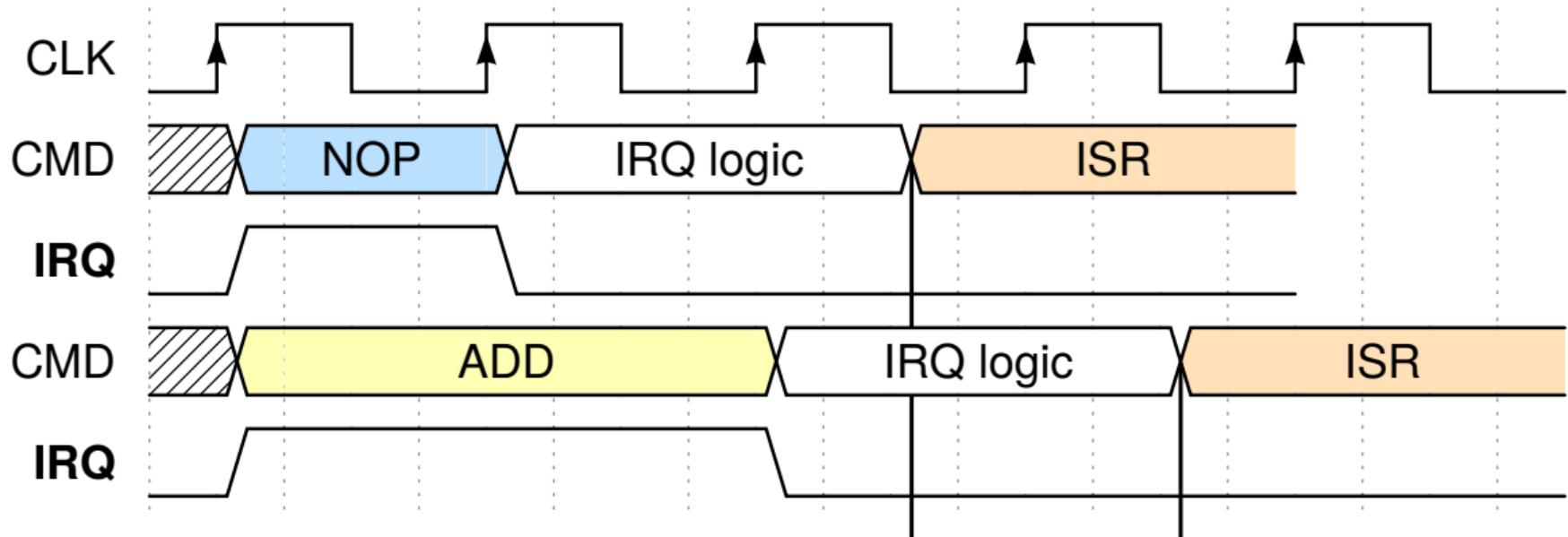


Back to basics: Fetch-decode-execute

⚠ **Timing leak:** IRQ response time depends on currently executing instruction(!)



Wait a cycle: Interrupt latency as a side-channel



```
if (secret) { ADD @R5+, R6; } // 2 cycles  
else { NOP; NOP; } // 2*1 cycle
```



WHAT COULD POSSIBLY



GO WRONG?

Attacking a Sancus application with interrupt latency

Secure keypad: enclave has exclusive access to memory-mapped I/O device



Attacking a Sancus application with interrupt latency

Driver enclave: 16-bit vector indicates which keys are down



PIN code enclave

0100000000000000

→ *traverse bits*

Attacking a Sancus application with interrupt latency

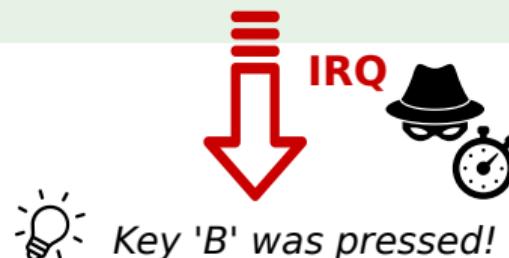
Attacker: Interrupt *conditional control flow* to infer secret PIN



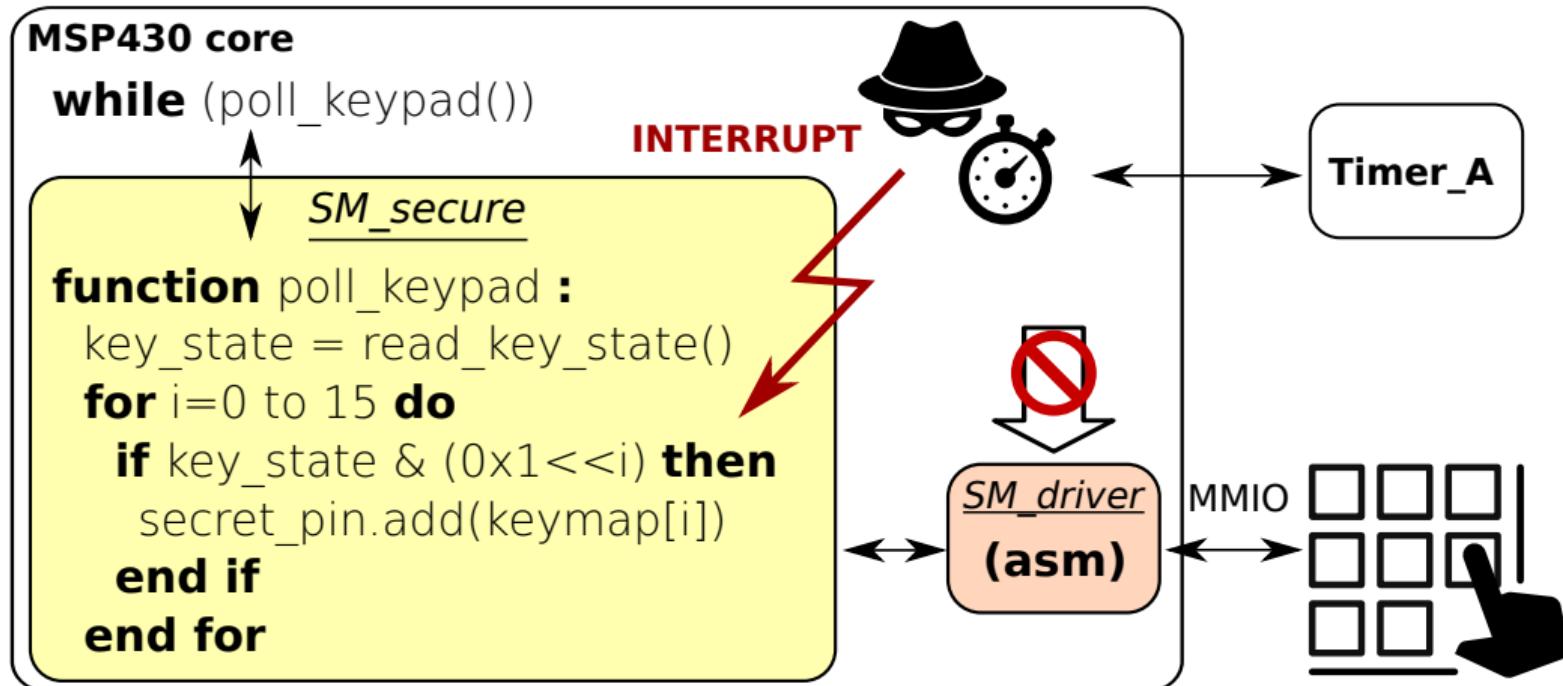
PIN code enclave

0100000000000000

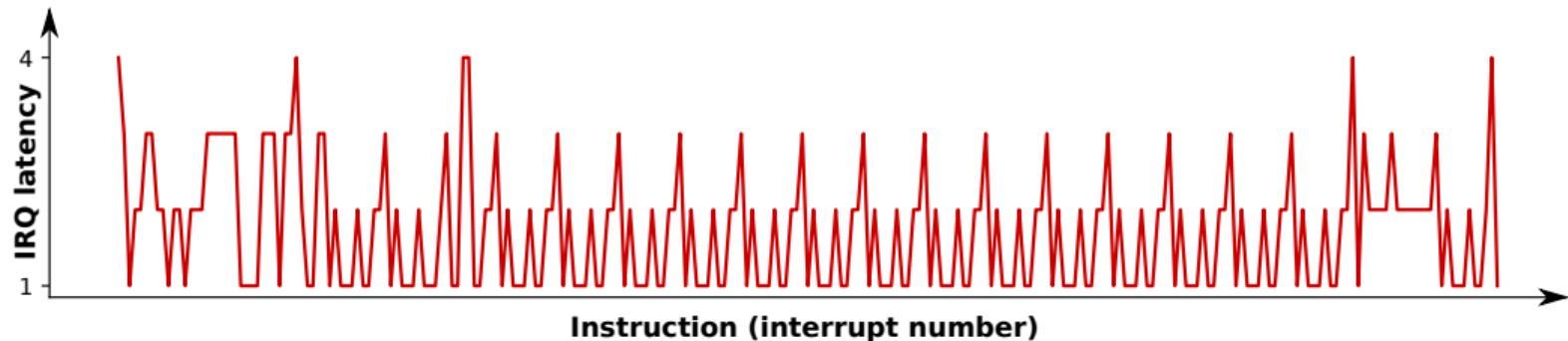
→ *traverse bits*



Attacking a Sancus application with interrupt latency

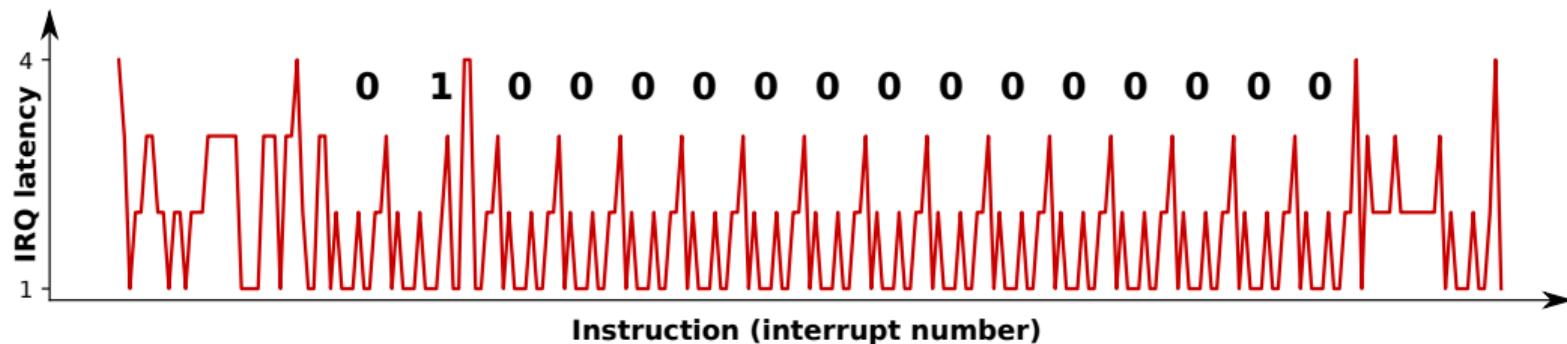


Sancus IRQ timing attack: Inferring key strokes



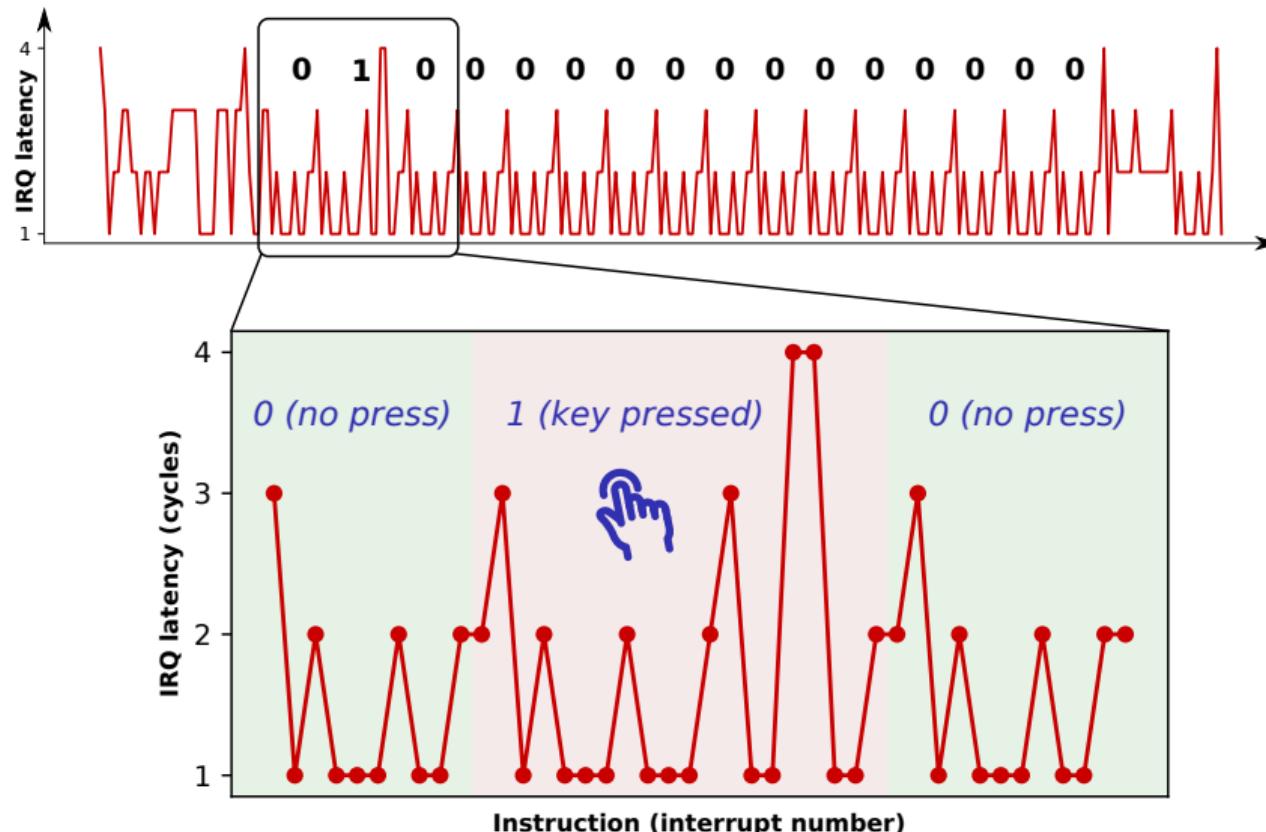
Enclave x-ray: Start-to-end trace enclaved execution

Sancus IRQ timing attack: Inferring key strokes



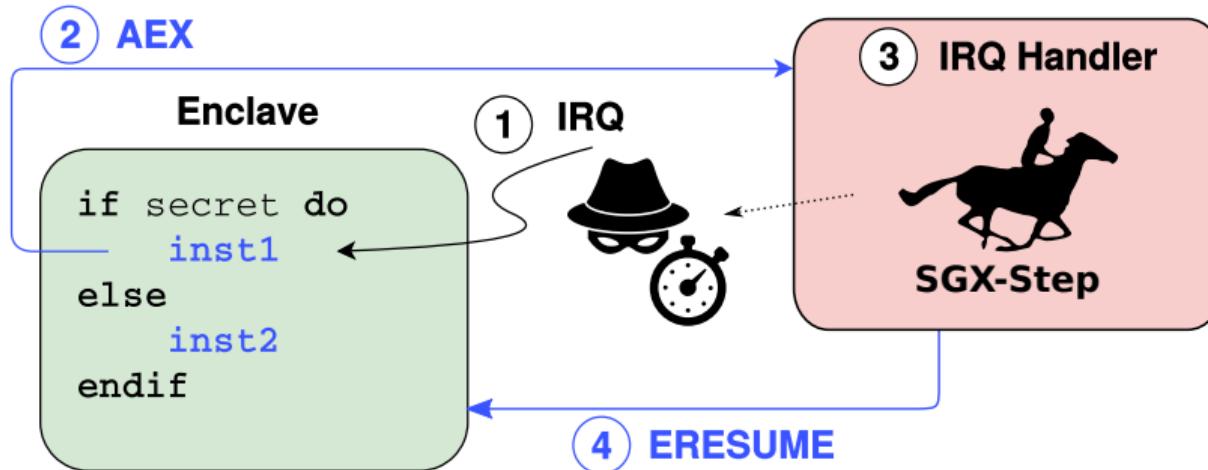
Enclave x-ray: Keymap bit traversal (ground truth)

Sancus IRQ timing attack: Inferring key strokes



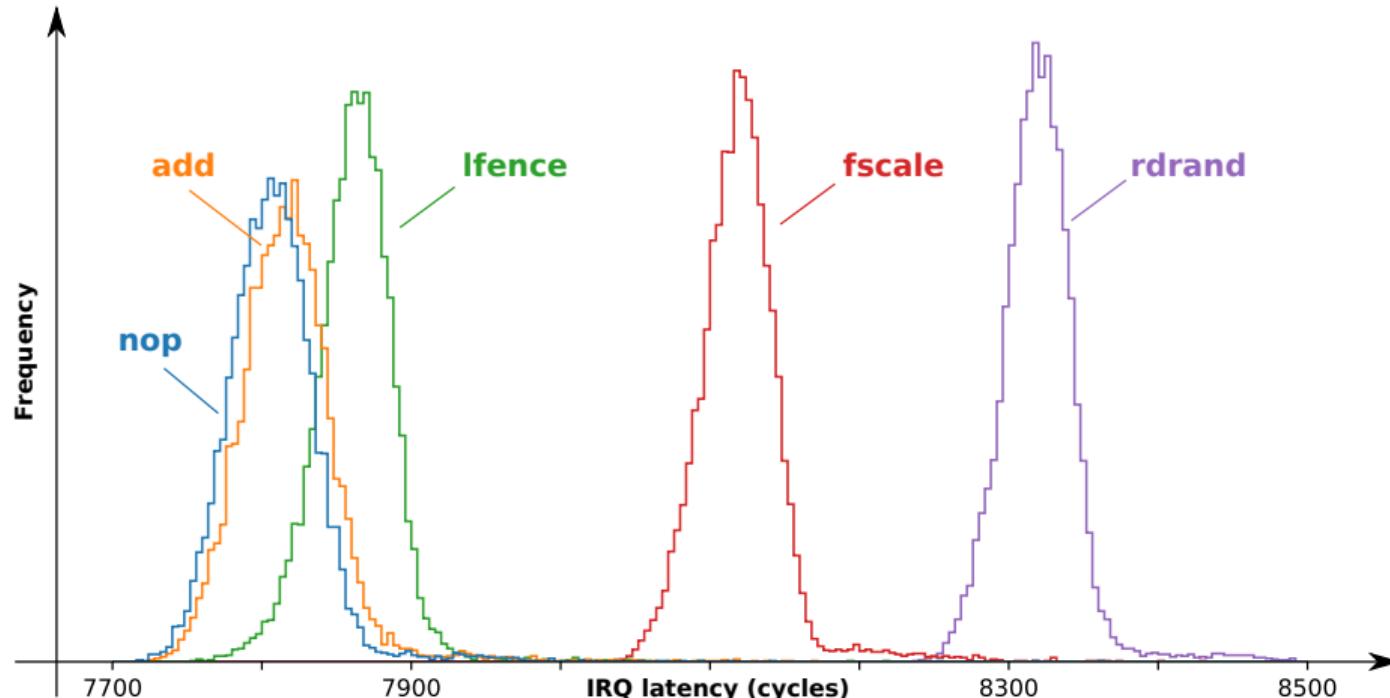
Does this also work for Intel SGX enclaves?

Yes(!): precise x86 APIC timer interrupts can be abused to reconstruct execution timings for *individual* enclave instructions → same attack vector as on Sancus...



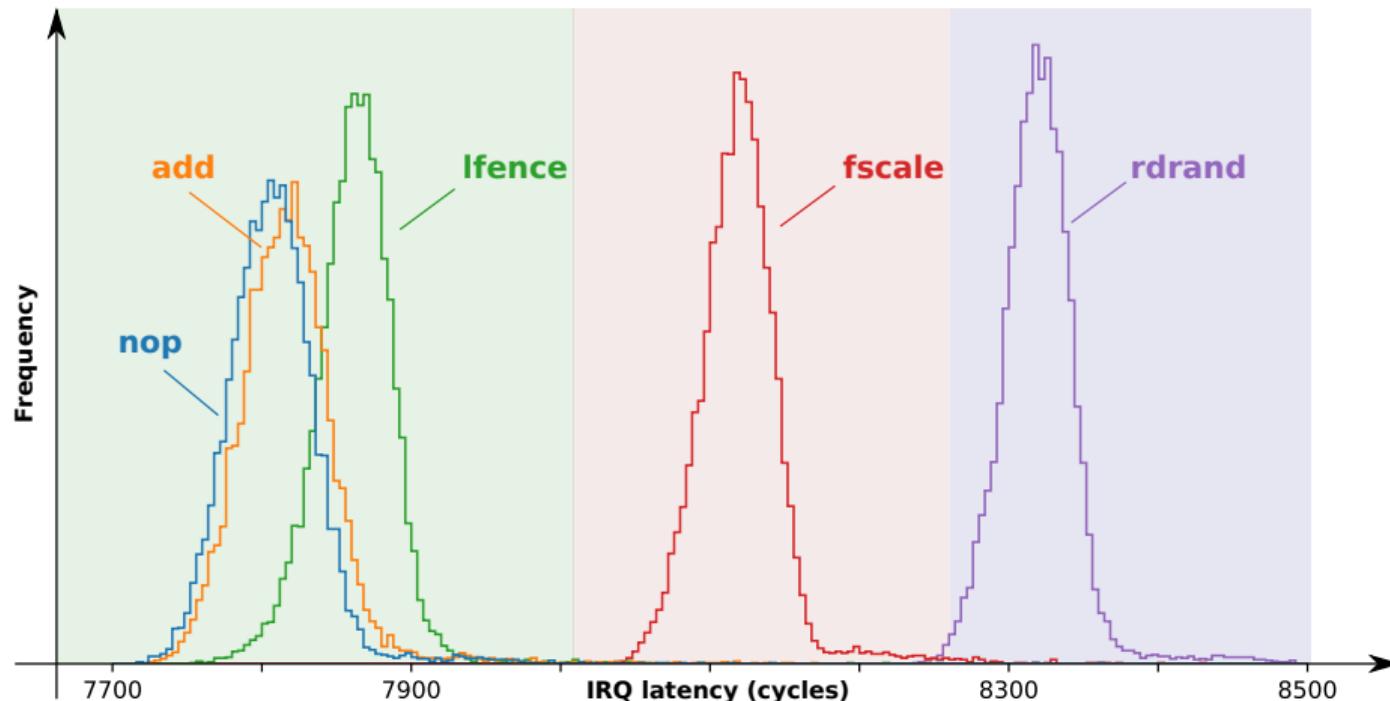
Microbenchmarks: Measuring Intel x86 instruction latencies

Latency distribution: 10,000 samples from benchmark enclave



Microbenchmarks: Measuring Intel x86 instruction latencies

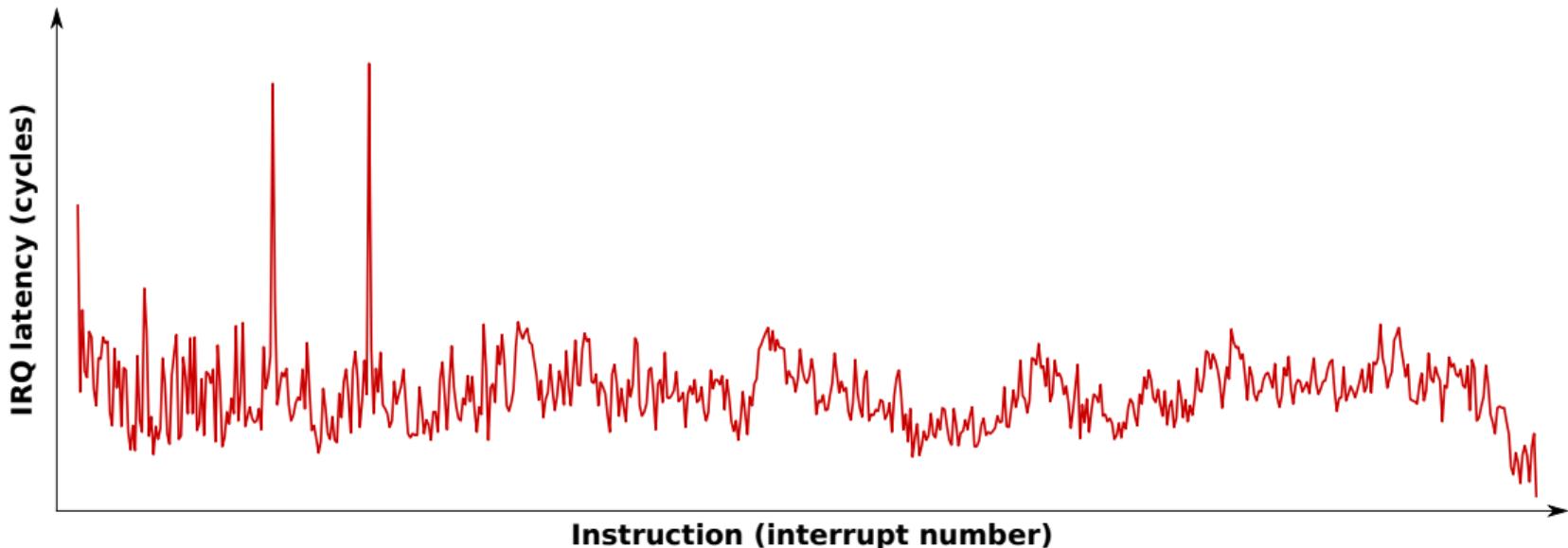
Timing leak: reconstruct *instruction latency class*



Single-stepping Intel SGX enclaves in practice



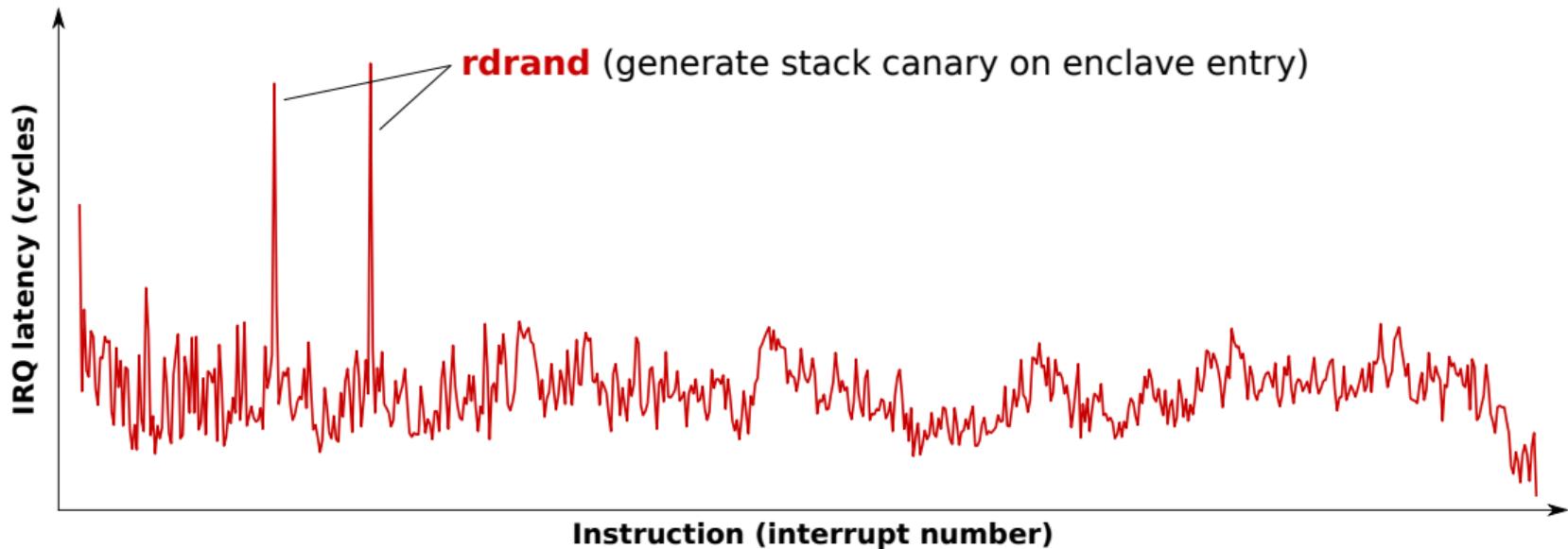
Enclave x-ray: Start-to-end trace enclaved execution



Single-stepping Intel SGX enclaves in practice



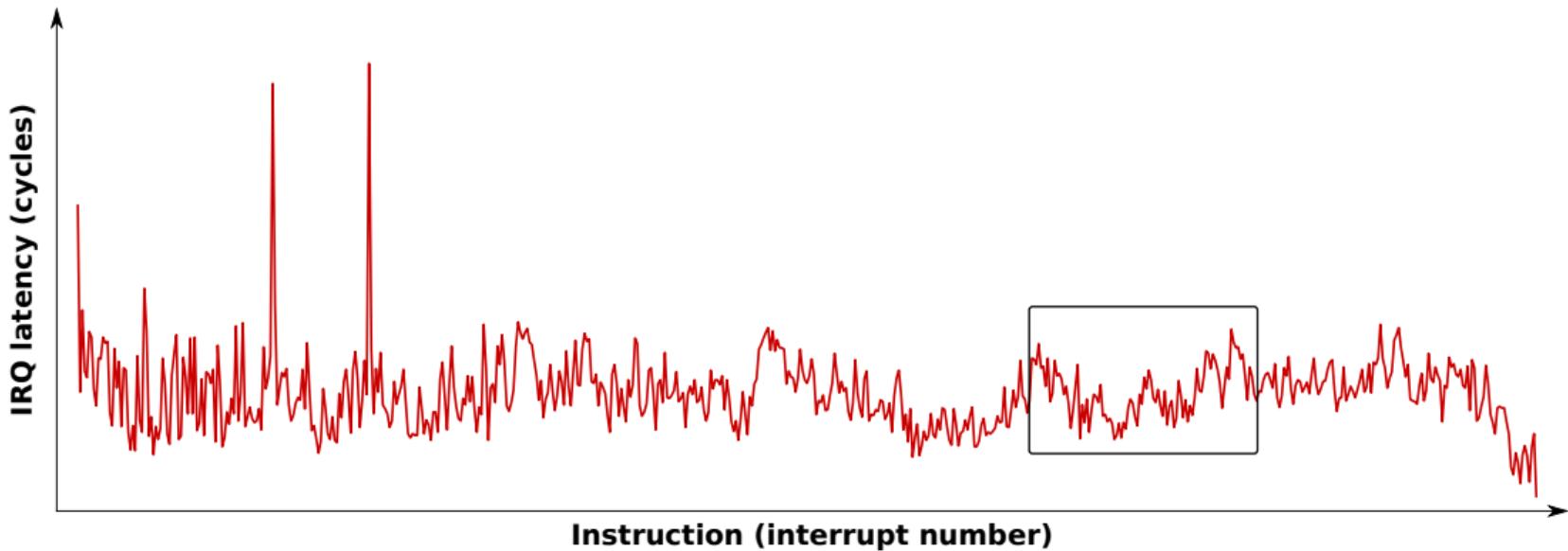
Enclave x-ray: Spotting high-latency instructions



Single-stepping Intel SGX enclaves in practice

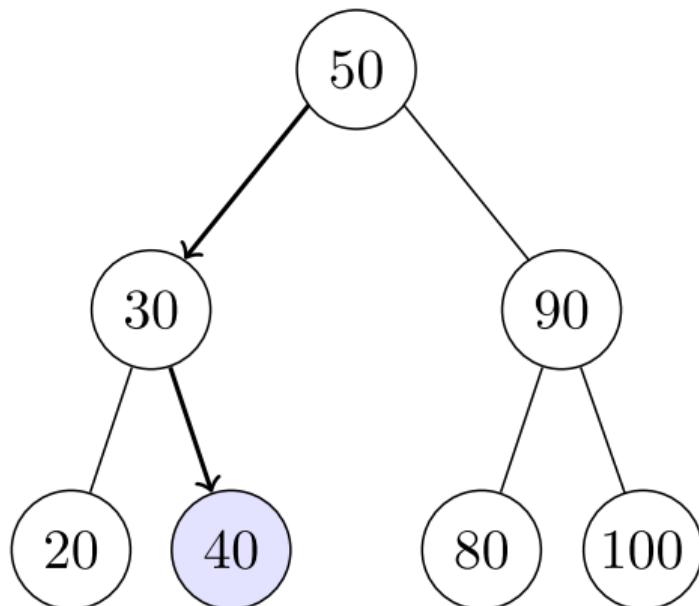


Enclave x-ray: Zooming in on bsearch function



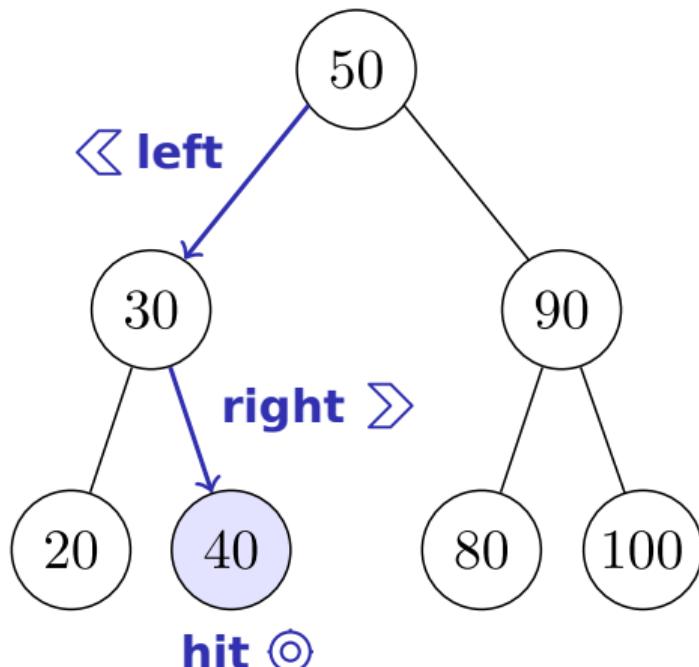
De-anonymizing enclave lookups with interrupt latency

Binary search: Find 40 in {20, 30, 40, 50, 80, 90, 100}



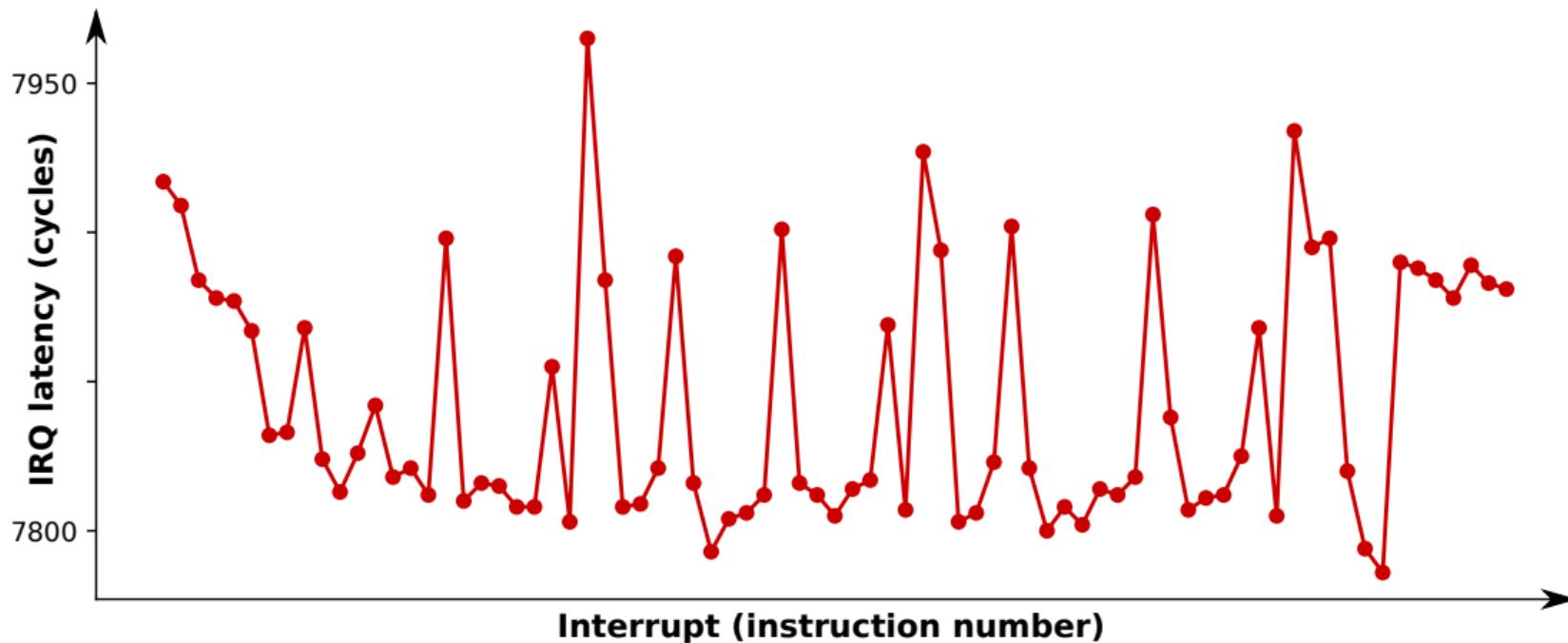
De-anonymizing enclave lookups with interrupt latency

Adversary: Infer secret lookup in known array



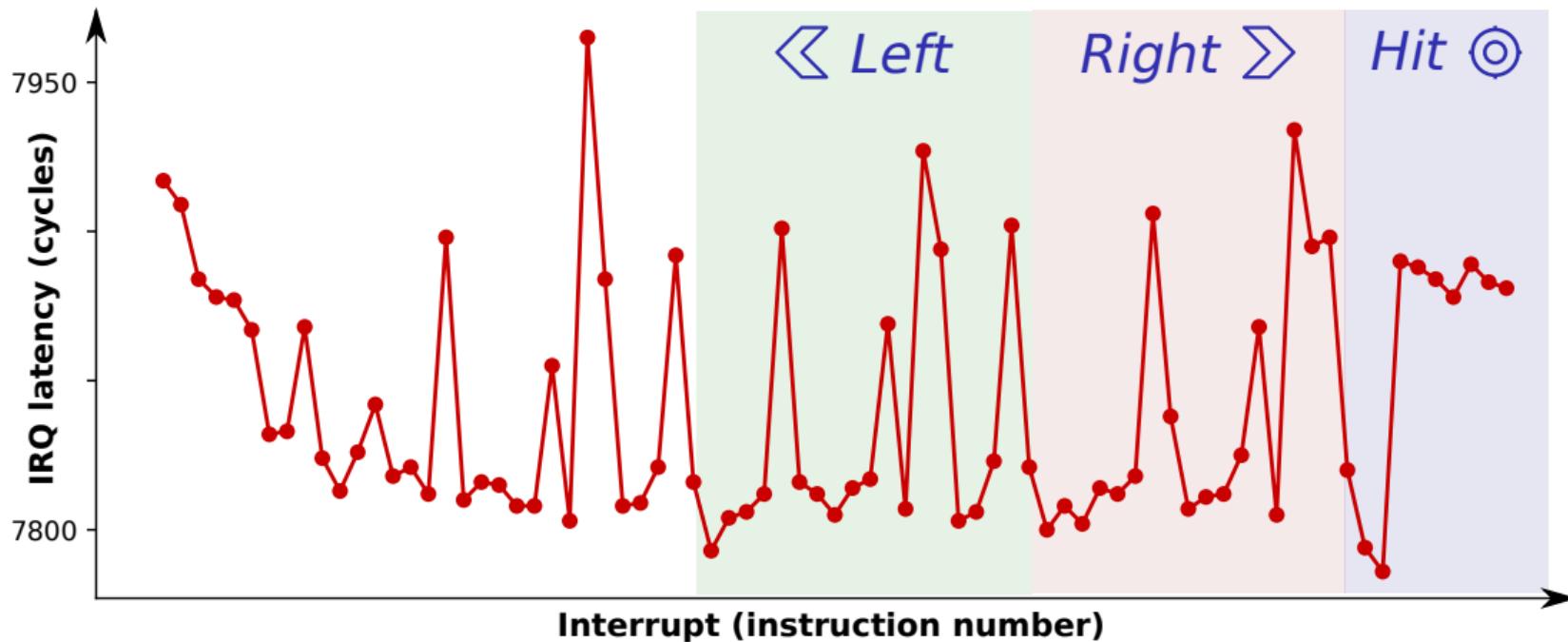
De-anonymizing enclave lookups with interrupt latency

Goal: Infer lookup → reconstruct bsearch control flow



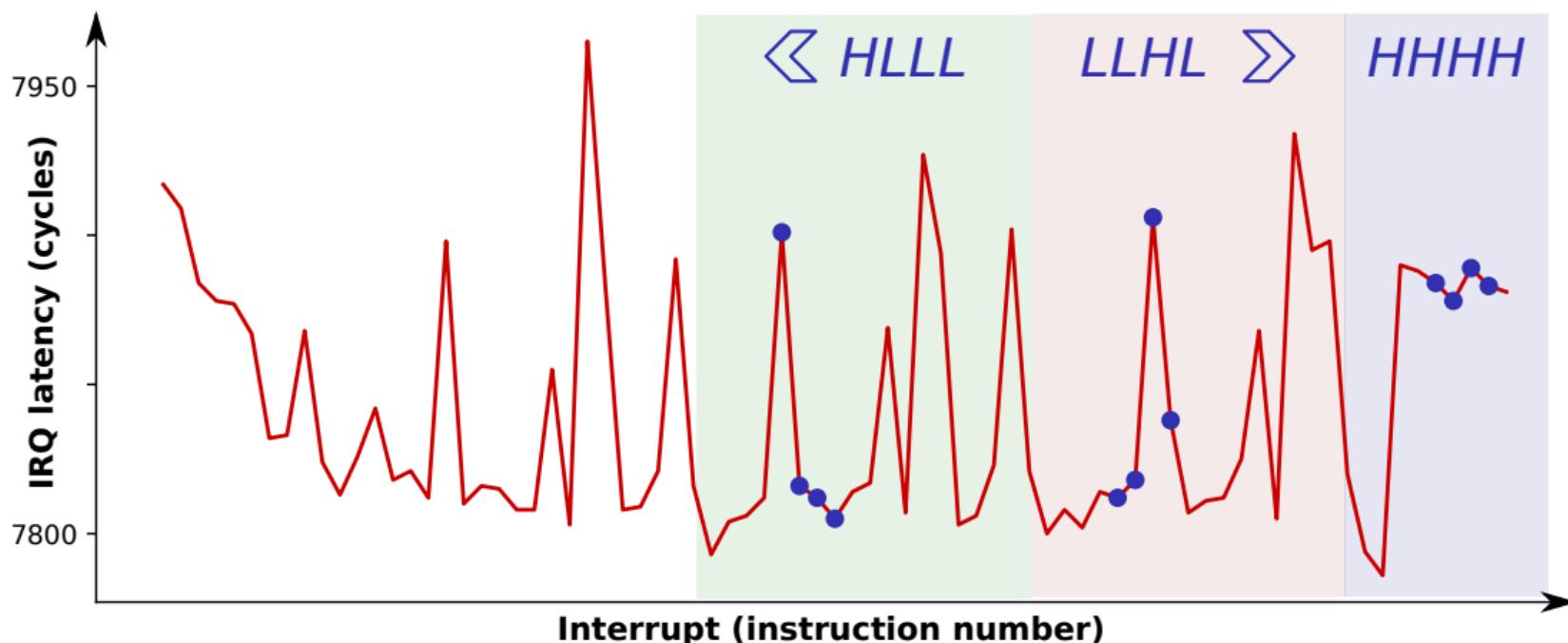
De-anonymizing enclave lookups with interrupt latency

Goal: Infer lookup → reconstruct bsearch control flow



De-anonymizing enclave lookups with interrupt latency

⇒ Sample **instruction latencies** in secret-dependent path



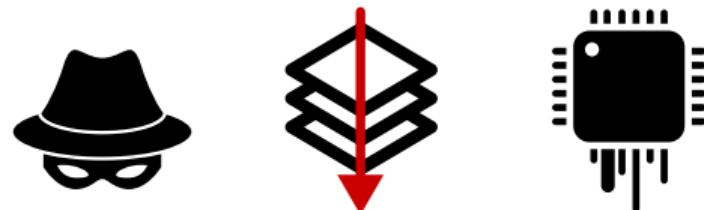


SHARING IS NOT CARING

SHARING IS LOSING YOUR STUFF TO OTHERS

Conclusions and take-away

- Security cross-cuts **hardware-software boundaries**
- Trusted execution environments are not perfect(!)
- No silver-bullet **defenses**: write constant-time code



References I

-  V. Costan and S. Devadas.
Intel SGX explained.
Cryptology ePrint Archive, Report 2016/086, 2016.
-  R. de Clercq, F. Piessens, D. Schellekens, and I. Verbauwhede.
Secure interrupts on low-end microcontrollers.
In *25th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2014)*, ASAP'14, pp. 1–6. IEEE, 2014.
-  J. Noorman, J. Van Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. Freiling.
Sancus 2.0: A low-cost security architecture for IoT devices.
ACM Transactions on Privacy and Security (TOPS), 2017.
-  S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena.
Preventing page faults from telling your secrets.
In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIA CCS)*, pp. 317–328. ACM, 2016.
-  M.-W. Shih, S. Lee, T. Kim, and M. Peinado.
T-SGX: Eradicating controlled-channel attacks against enclave programs.
In *Proceedings of the 2017 Network and Distributed System Security Symposium (NDSS 2017)*, February 2017.
-  J. Van Bulck, J. T. Mühlberg, and F. Piessens.
VulCAN: Efficient component authentication and software isolation for automotive control networks.
In *Proceedings of the 33th Annual Computer Security Applications Conference (ACSAC'17)*. ACM, 2017.
-  J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. Garcia, and F. Piessens.
A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes.
In *Proceedings of the 26th ACM Conference on Computer and Communications Security (CCS'19)*. ACM, November 2019.

References II



J. Van Bulck, F. Piessens, and R. Strackx.

SGX-Step: A practical attack framework for precise enclave execution control.

In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, SysTEX'17, pp. 4:1–4:6. ACM, 2017.



J. Van Bulck, F. Piessens, and R. Strackx.

Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic.

In *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS'18)*. ACM, October 2018.



J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx.

Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution.

In *Proceedings of the 26th USENIX Security Symposium*. USENIX Association, August 2017.



Y. Xu, W. Cui, and M. Peinado.

Controlled-channel attacks: Deterministic side channels for untrusted operating systems.

In *36th IEEE Symposium on Security and Privacy*. IEEE, May 2015.