KU LEUVEN

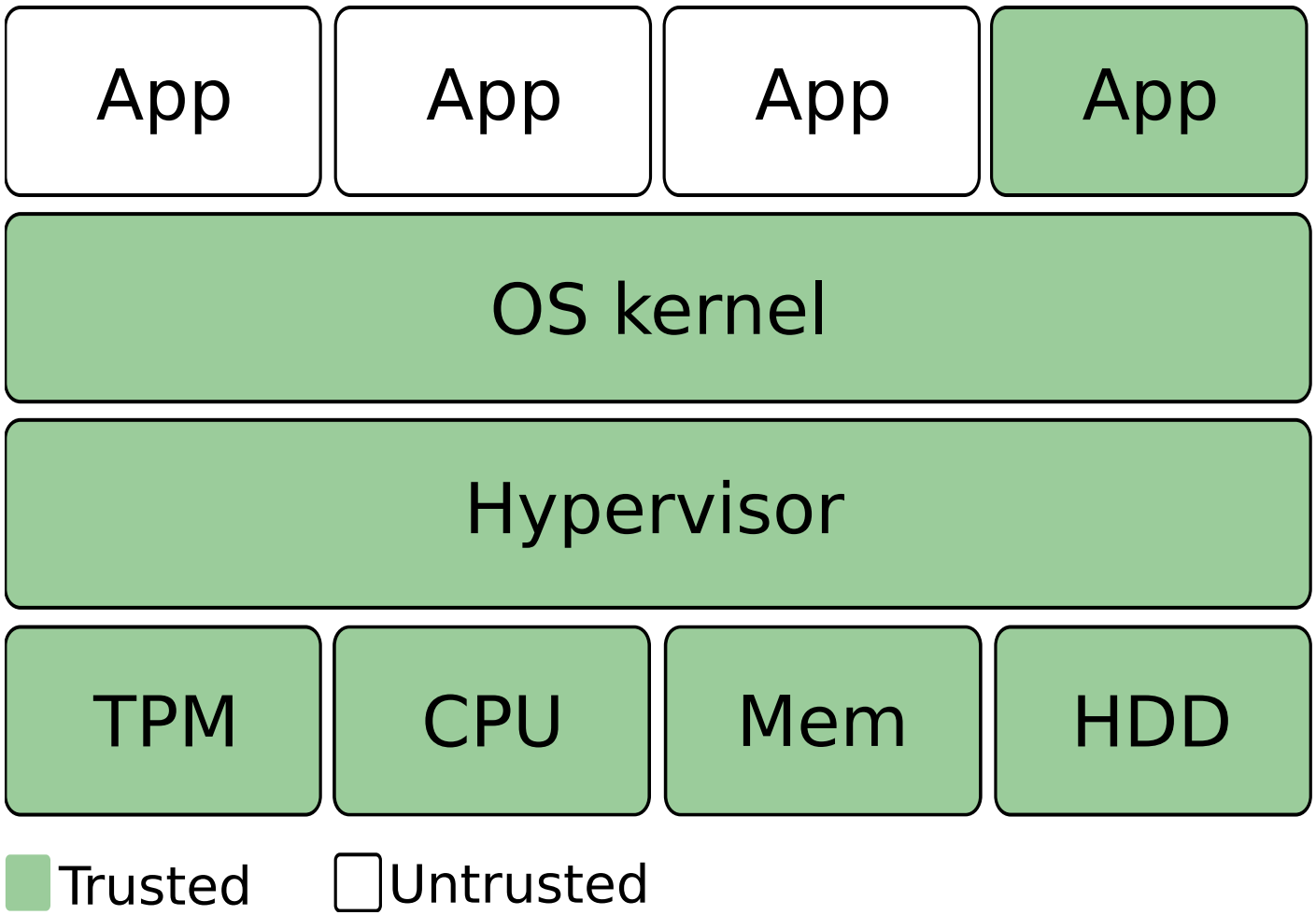
A Security Analysis of Interrupts in Embedded Enclaved Execution

Sven Cuyt

Distrinet, KU Leuven

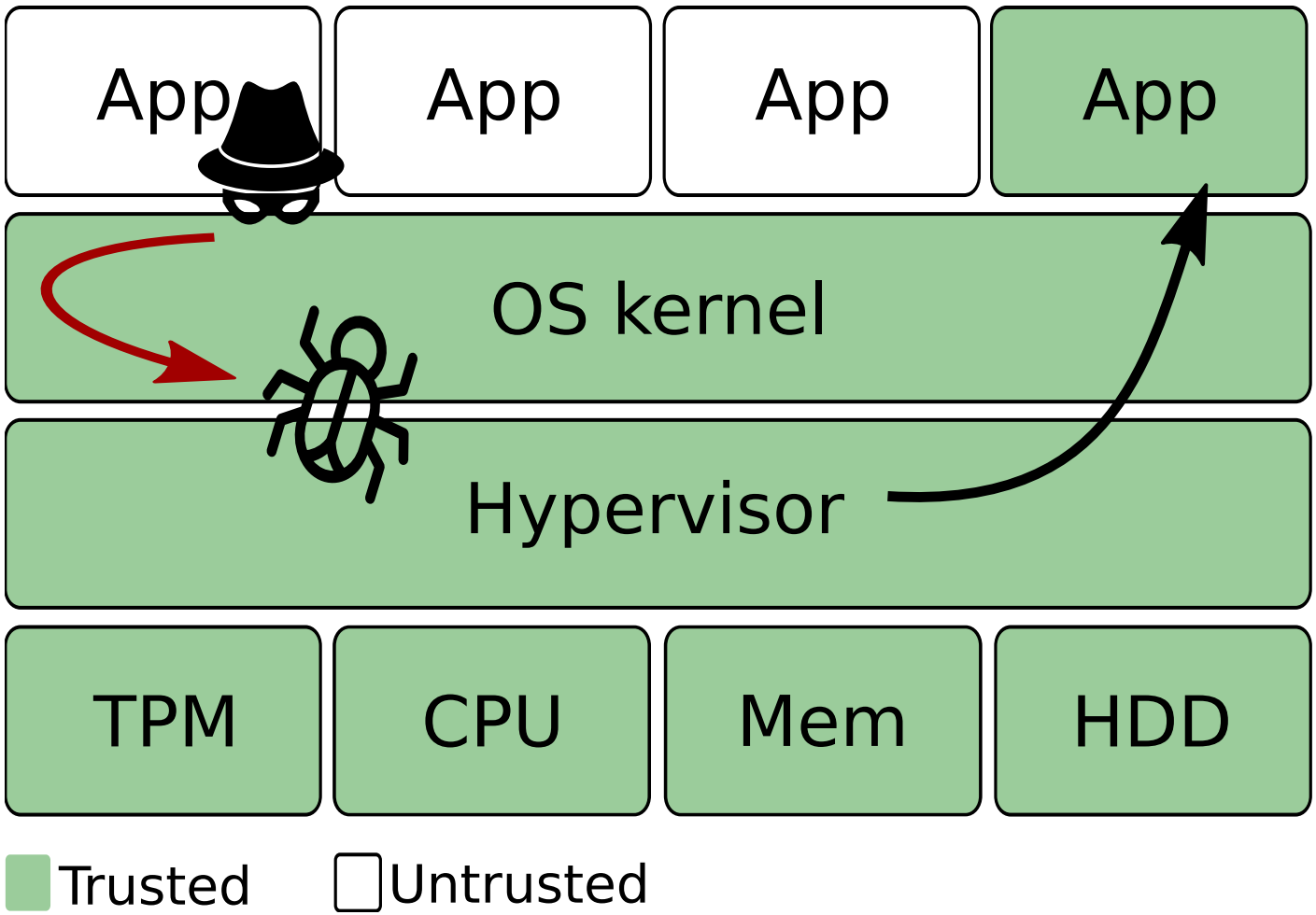
June 28, 2019

Conventional Software Isolation



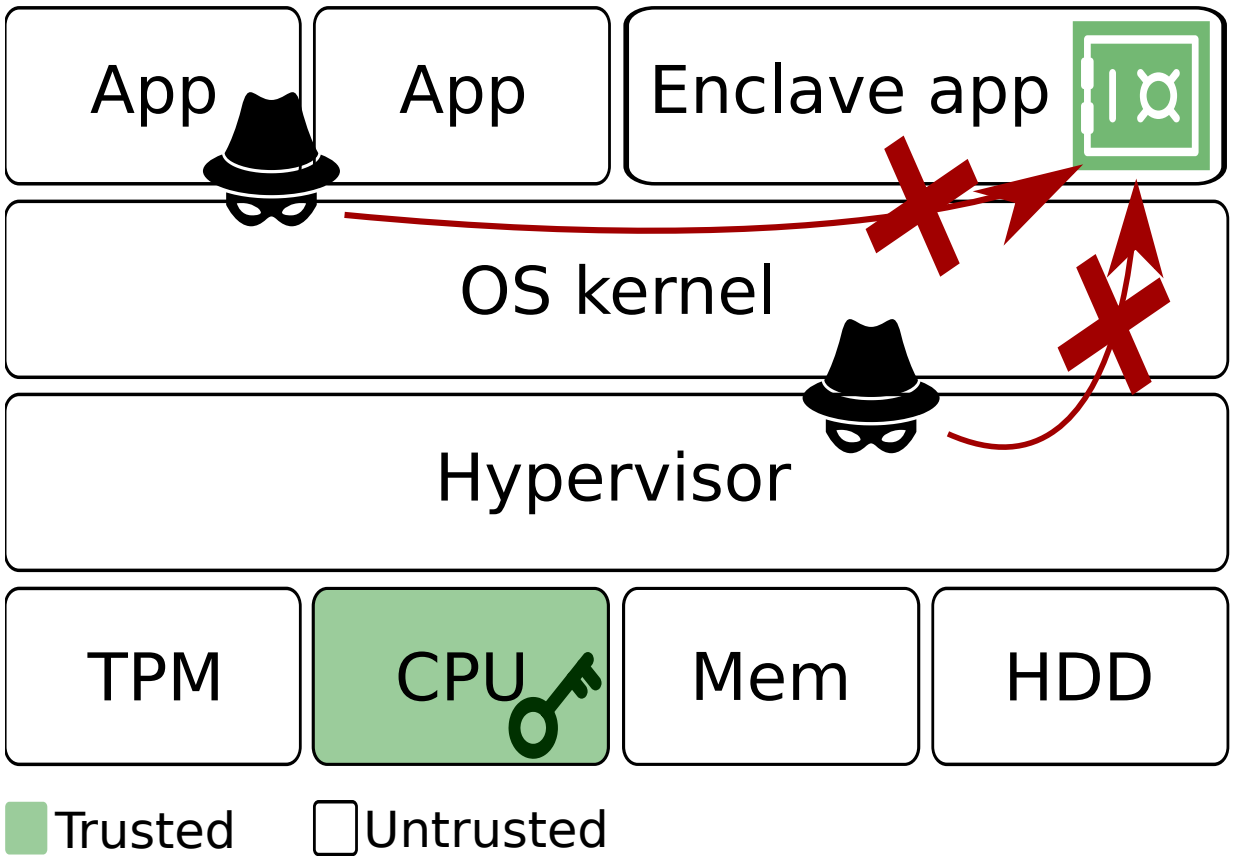
based on: https://distrinet.cs.kuleuven.be/software/sancus/publications/ccs18-slides.pdf

TCB: Security Risk



based on: https://distrinet.cs.kuleuven.be/software/sancus/publications/ccs18-slides.pdf

Enclaved Execution



based on: https://distrinet.cs.kuleuven.be/software/sancus/publications/ccs18-slides.pdf

Key feature: **Isolation** and **Attestation**

Sancus

Sancus

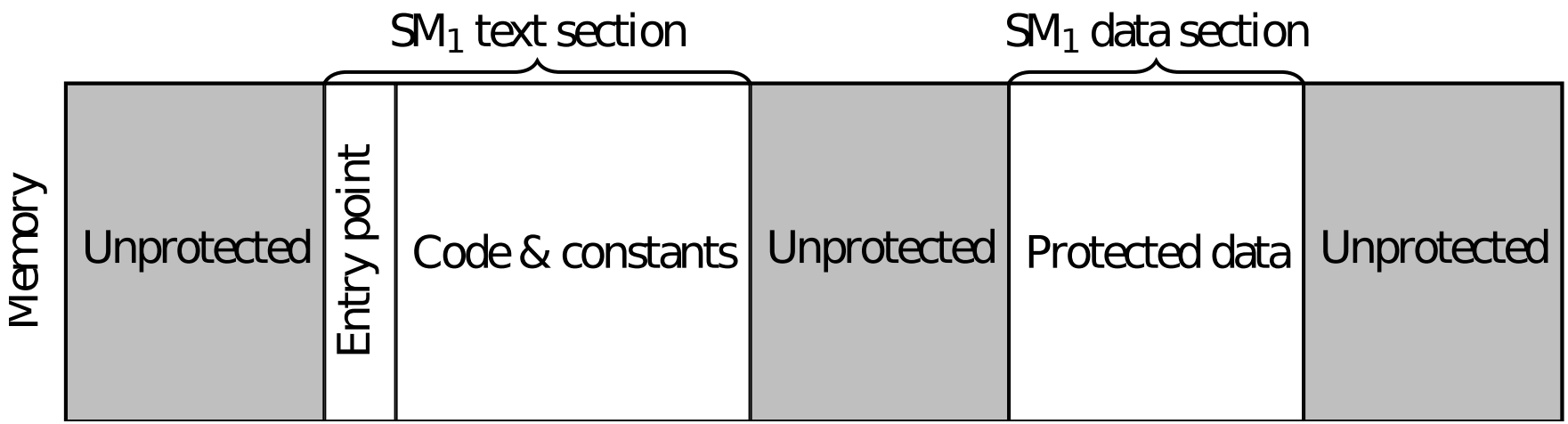
- Embedded PMA

Sancus

- Embedded PMA
- Hardware TCB only

Sancus

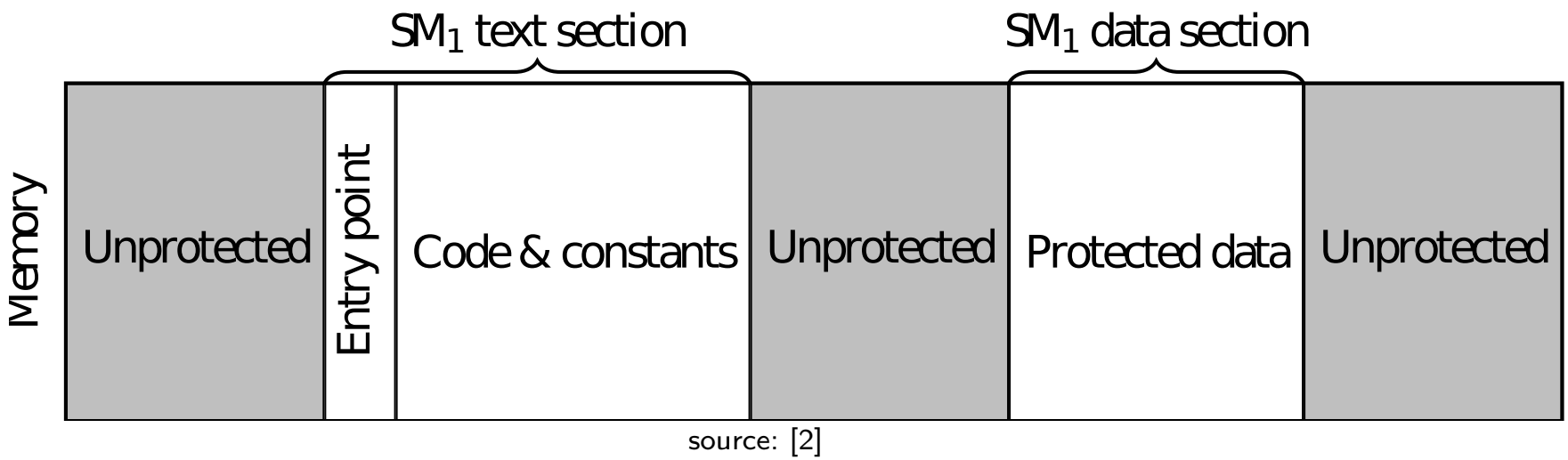
- Embedded PMA
- Hardware TCB only
- Enclave layout



source: [2]

Sancus

- Embedded PMA
- Hardware TCB only
- Enclave layout



source: [2]

- Fully abstract compilation [1]

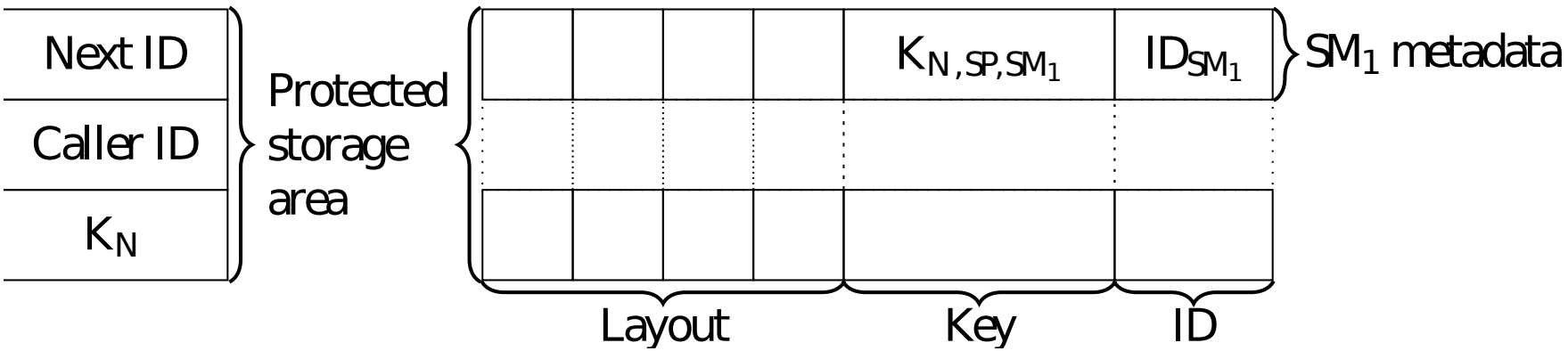
Sancus: Isolation

Program counter based access control policy

From\To	Entry	Text	Data	Unprotected
Entry	r-x	r-x	rw-	rwx
Text	r-x	r-x	rw-	rwx
Other	--x	---	---	rwx

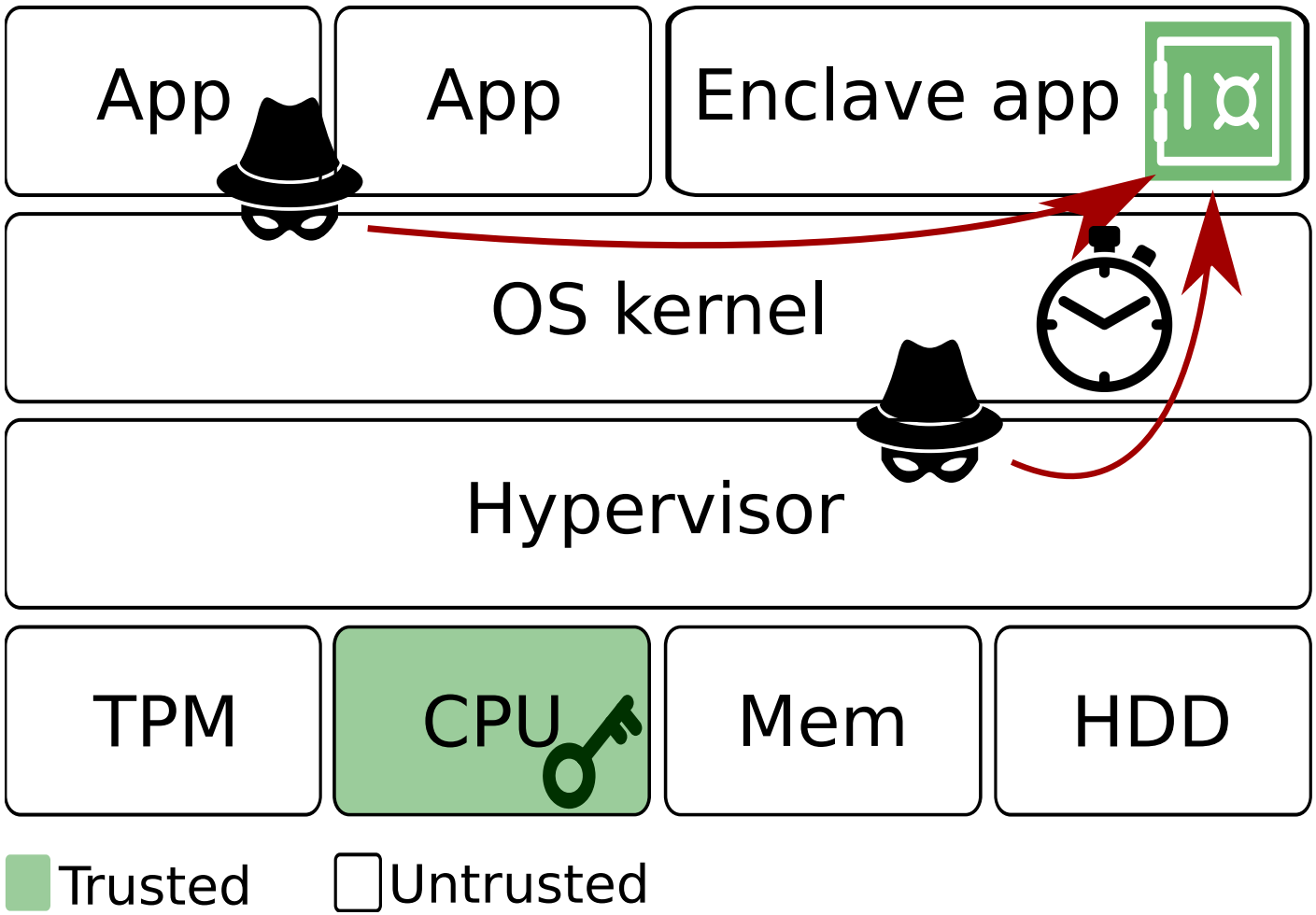
source: [2]

Sancus: Attestation



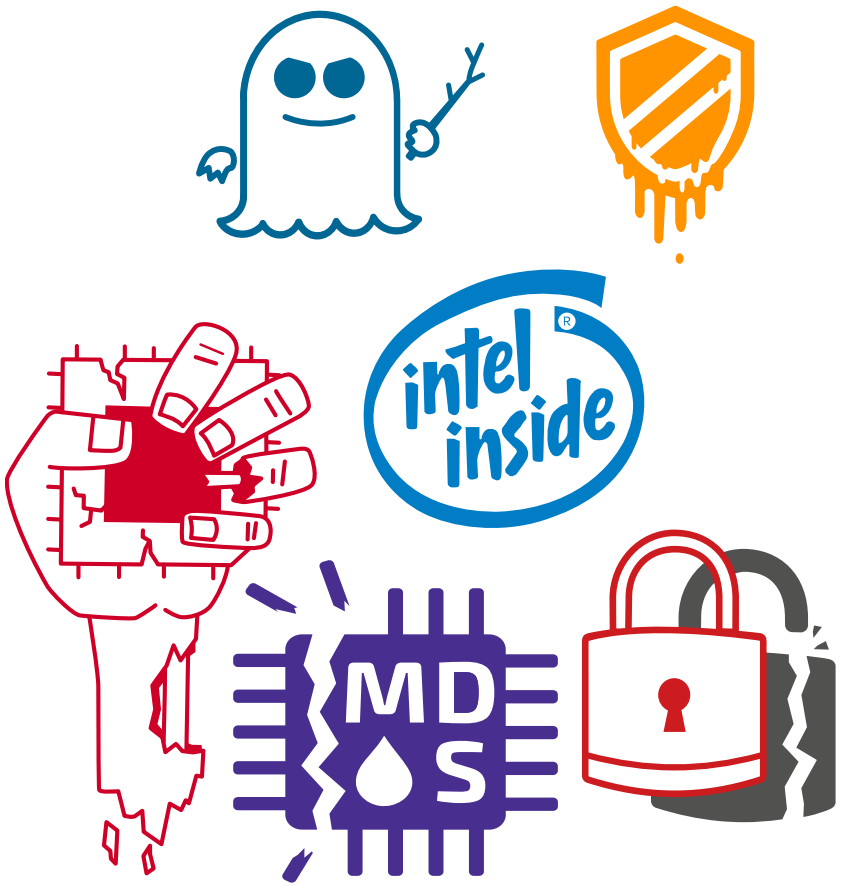
source: [2]

Side-Channel Attacks



based on: https://distrinet.cs.kuleuven.be/software/sancus/publications/ccs18-slides.pdf

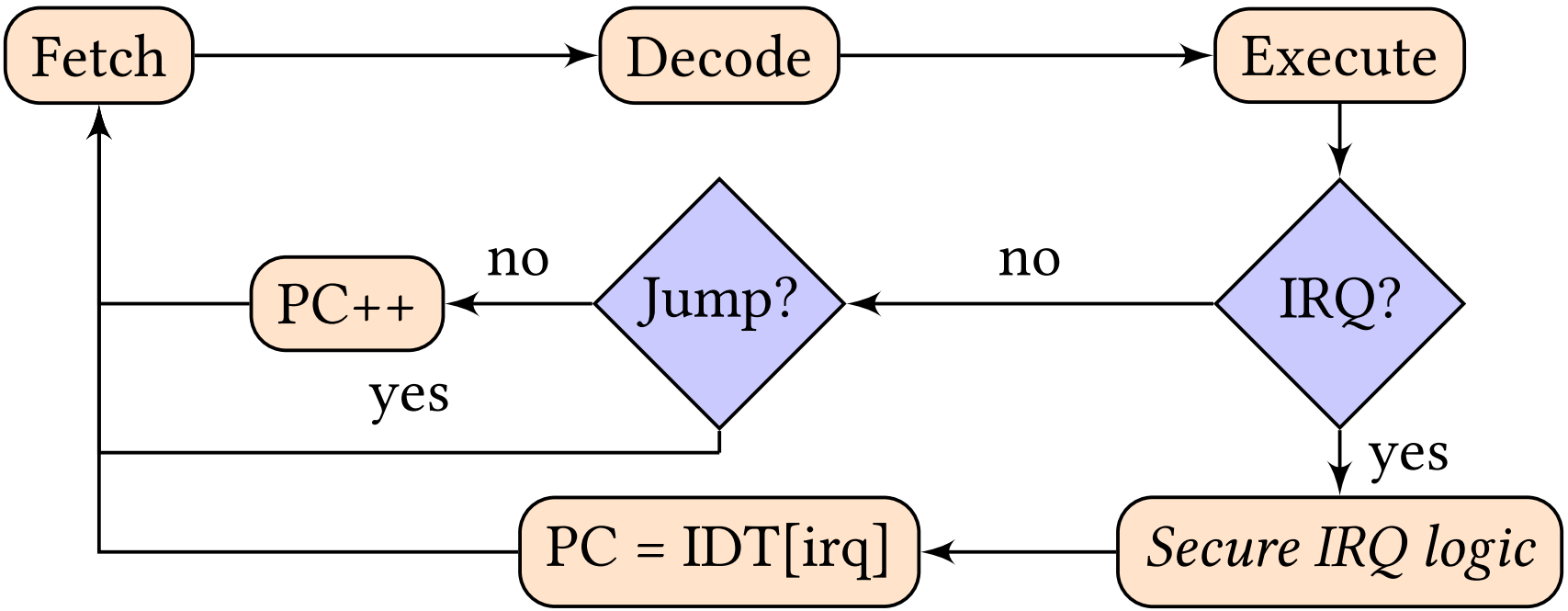
Side-Channels inside Intel



Abusing microarchitectural optimizations

$\rightarrow$ Not present in embedded

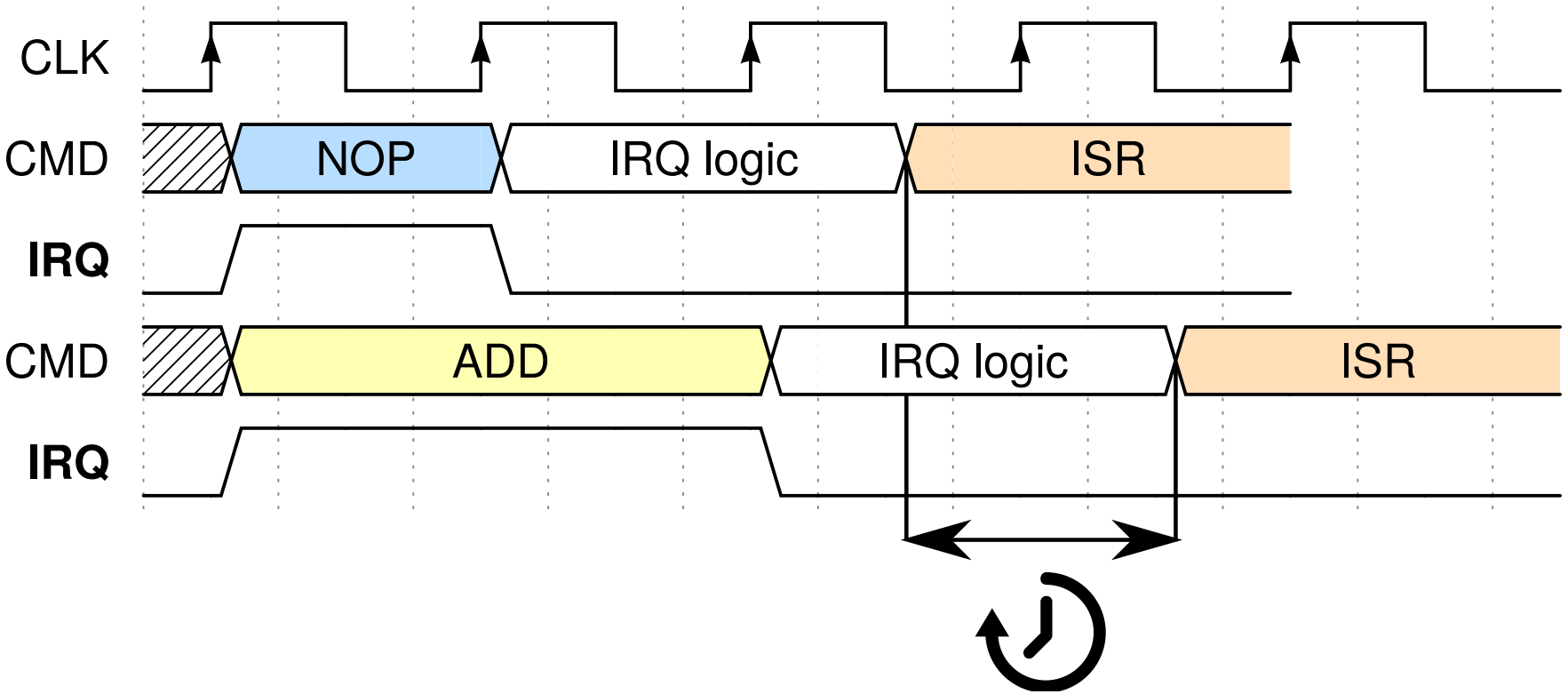
Nemesis [5]: Fetch-Decode-Execute Cycle



source: [5]

Interrupts only served on instruction retirement

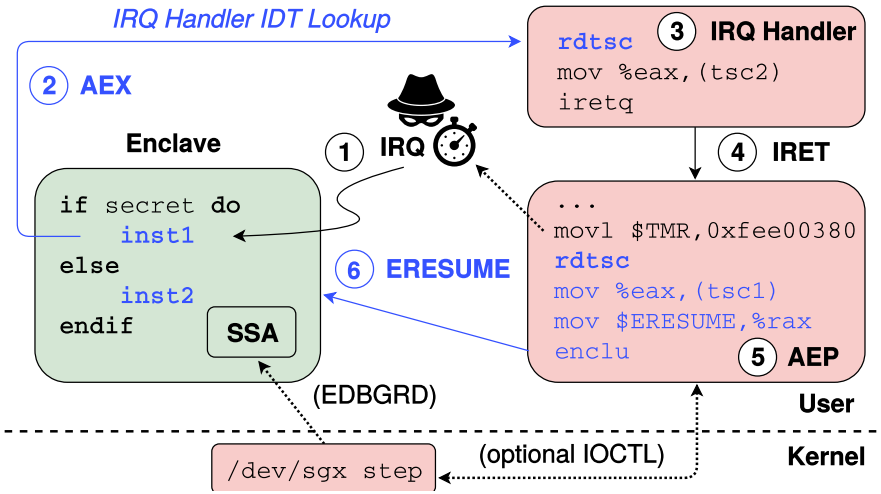
Nemesis [5]: Interrupt Latency



source: https://distrinet.cs.kuleuven.be/software/sancus/publications/ccs18-slides.pdf

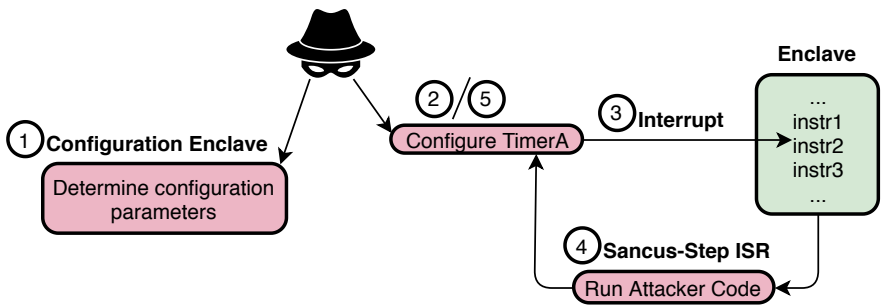
Attacker Frameworks

SGX-Step: Increase granularity and ease of use



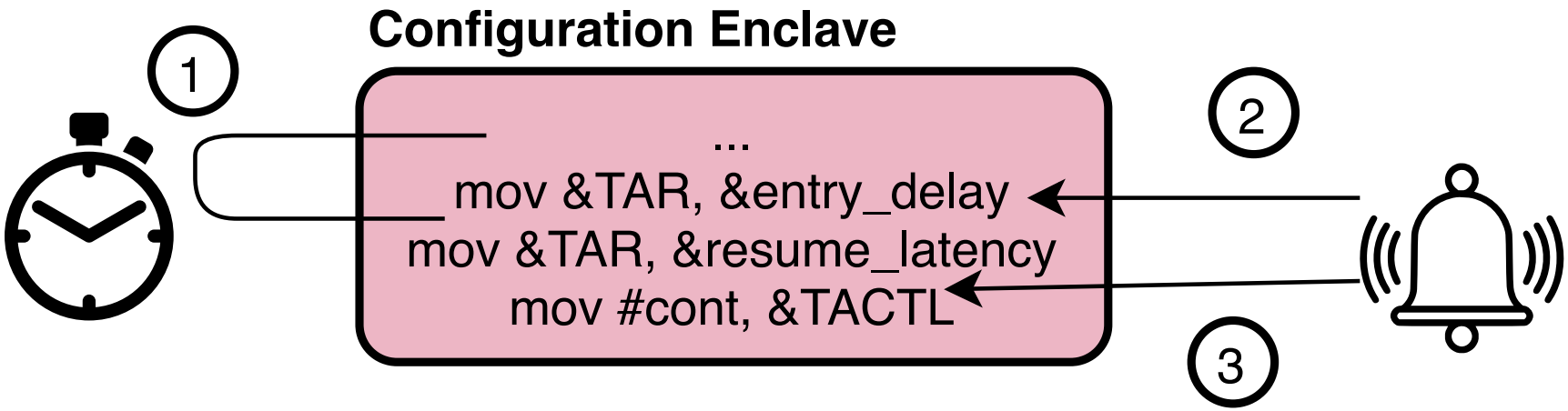
source: [5]

Sancus-Step: Ease of use



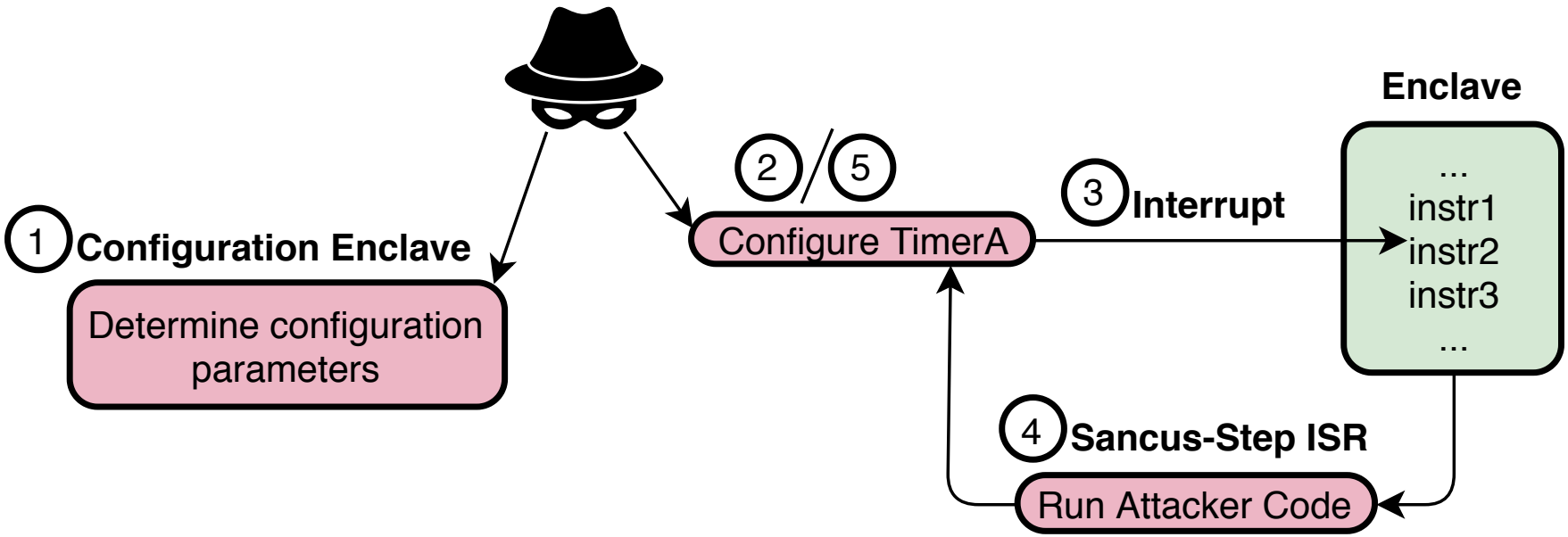
KU LEUVEN

Sancus-Step Configuration



KU LEUVEN

Sancus-Step Overview



KU LEUVEN

Verification TOCTOU

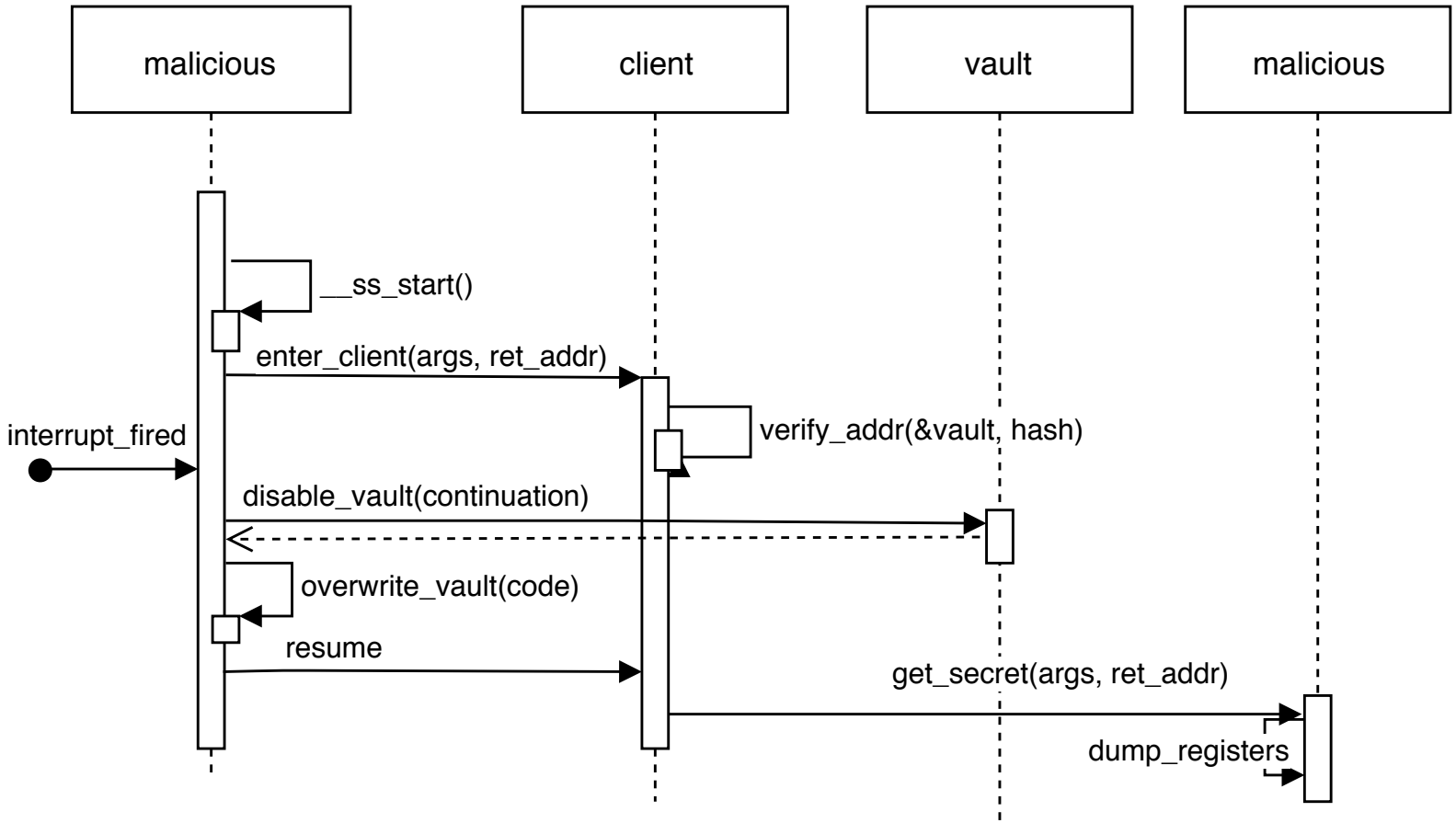
- Making enclaves interruptible $\rightarrow$ well known issues [2, 3, 4]

Verification TOCTOU

- Making enclaves interruptible $\rightarrow$ well known issues [2, 3, 4]
- ... In theory, not in practice

Verification TOCTOU

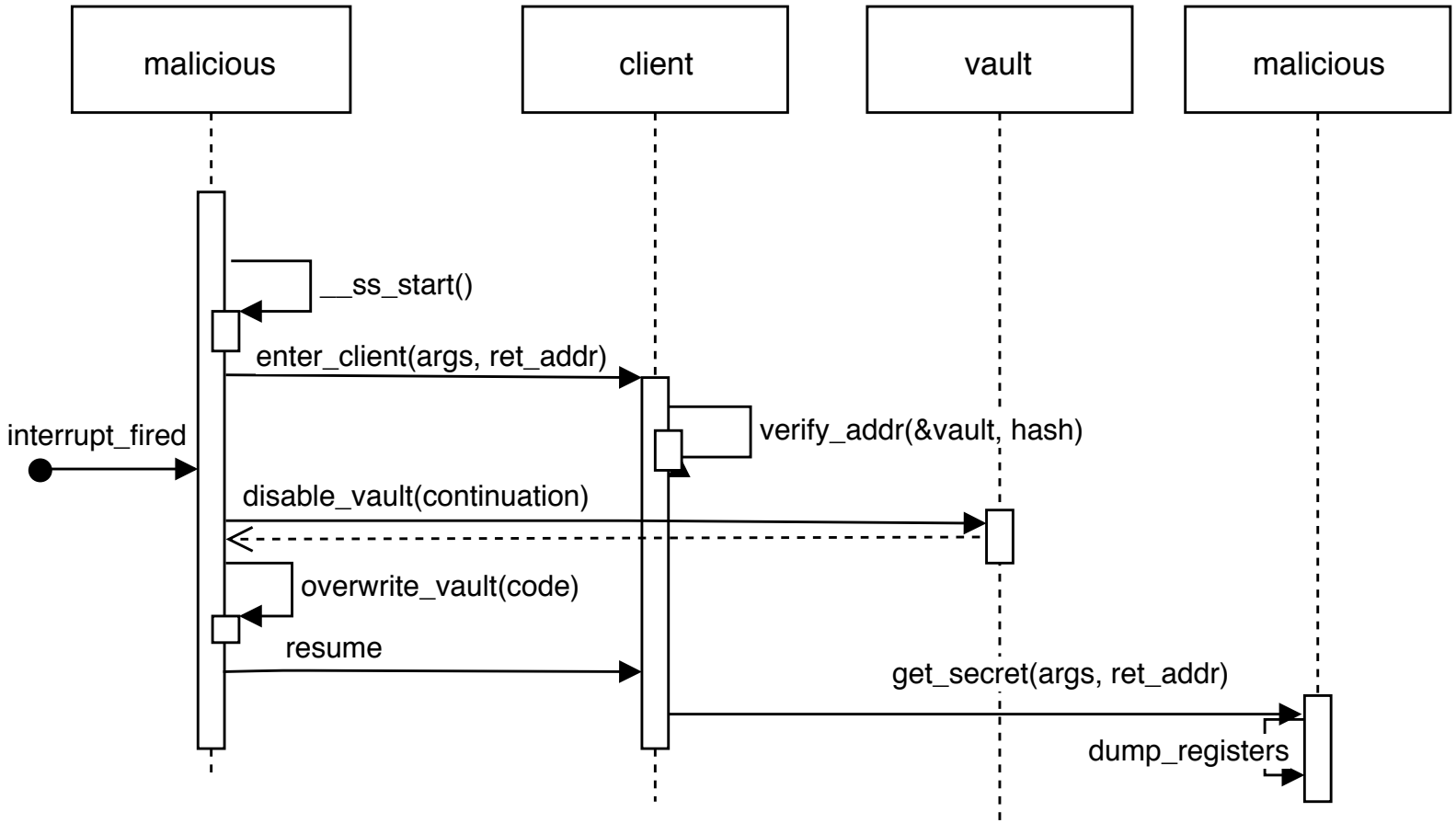
- Making enclaves interruptible → well known issues [2, 3, 4]
- ... In theory, not in practice



KU LEUVEN

Verification TOCTOU

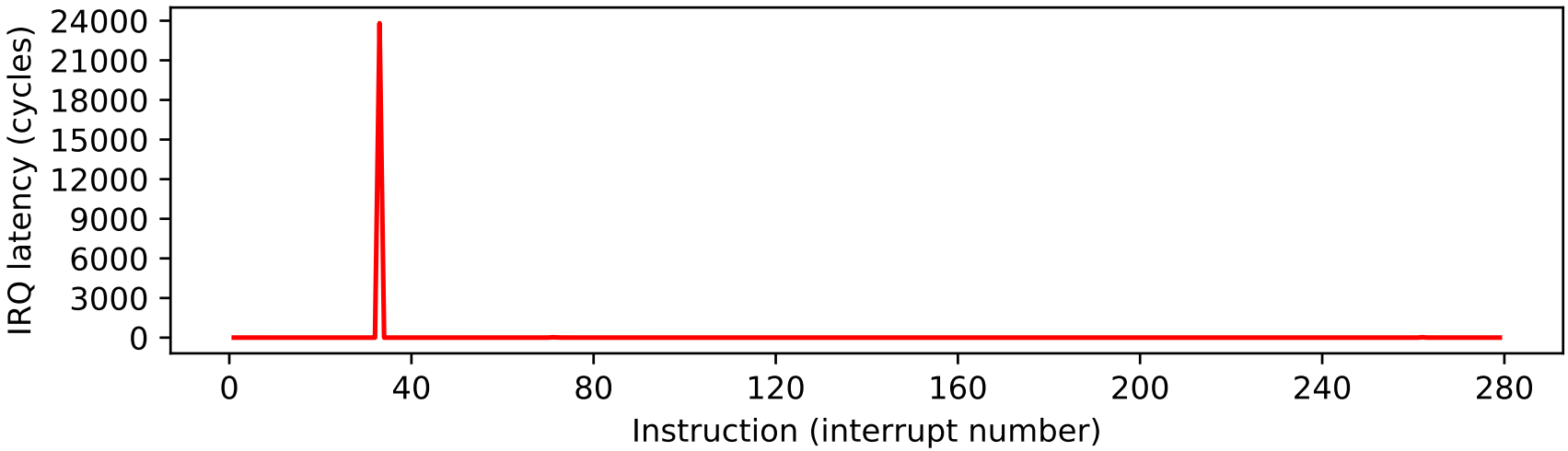
- Making enclaves interruptible → well known issues [2, 3, 4]
- ... In theory, not in practice



- Our ad-hoc implementation reveals vulnerabilities

KU LEUVEN

Verifcation TOCTOU: Automatic Exploit



KU LEUVEN

Side-Channel Analysis in Compiler Generated Code

Side-Channel Analysis in Compiler Generated Code

- Stubs:
  - Multiplication stub
  - Other arithmetic stubs
  - Entry and Exit stub

Side-Channel Analysis in Compiler Generated Code

- Stubs:
  - Multiplication stub
  - Other arithmetic stubs
  - Entry and Exit stub
- Both start-to-end and Nemesis

Reversing the Multiplication Stub

```
__sm_mulhi3:
    mov     r15, r13
    clr     r15
1:  tst     r14
    jz      3f
    clrc
    rrc     r13
    jnc     2f
    add     r14, r15
2:  rla     r14
    tst     r13
    jnz     1b
3:  ret
```

```
unsigned int mul(unsigned  int a,
                 unsigned int b)
{
    unsigned int rv = 0;
    while (a!=0)
    {
        if (b & 1)
            rv += a;
        b >>= 1;
        a <<= 1;
        if (b == 0)
            break;
    }
    return rv;
}
```

Intuition of the Multiplication Stub

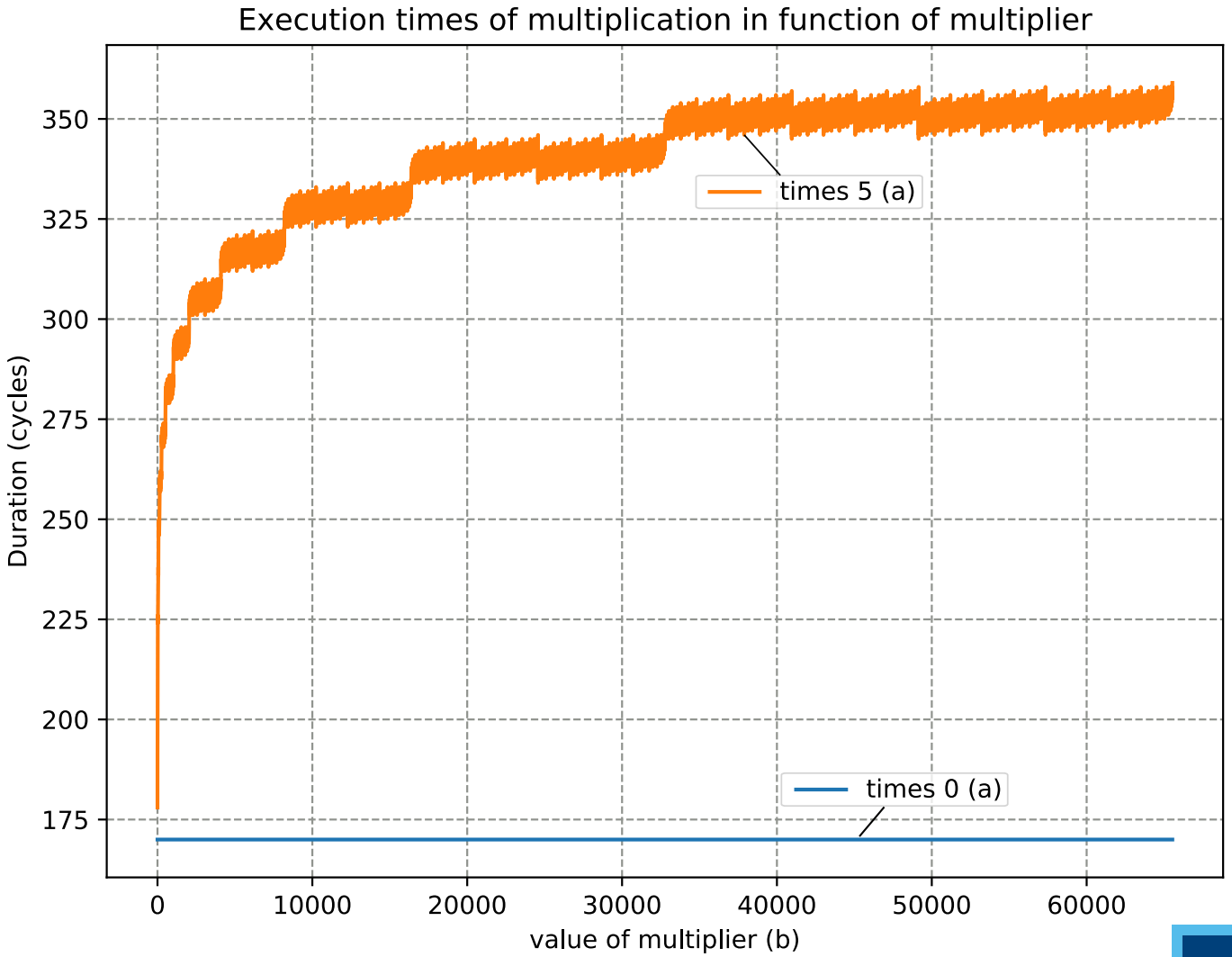
```
unsigned int mul(unsigned  int a,
                 unsigned int b)
{
    unsigned int rv = 0;
    while (a!=0)
    {
        if (b & 1)
            rv += a;
        b >>= 1;
        a <<= 1;
        if (b == 0)
            break;
    }
    return rv;
}
```

```
   10(a)
x 101(b)
------
   10
  00
+10
------
 1010
```

Case Study: SM_mul

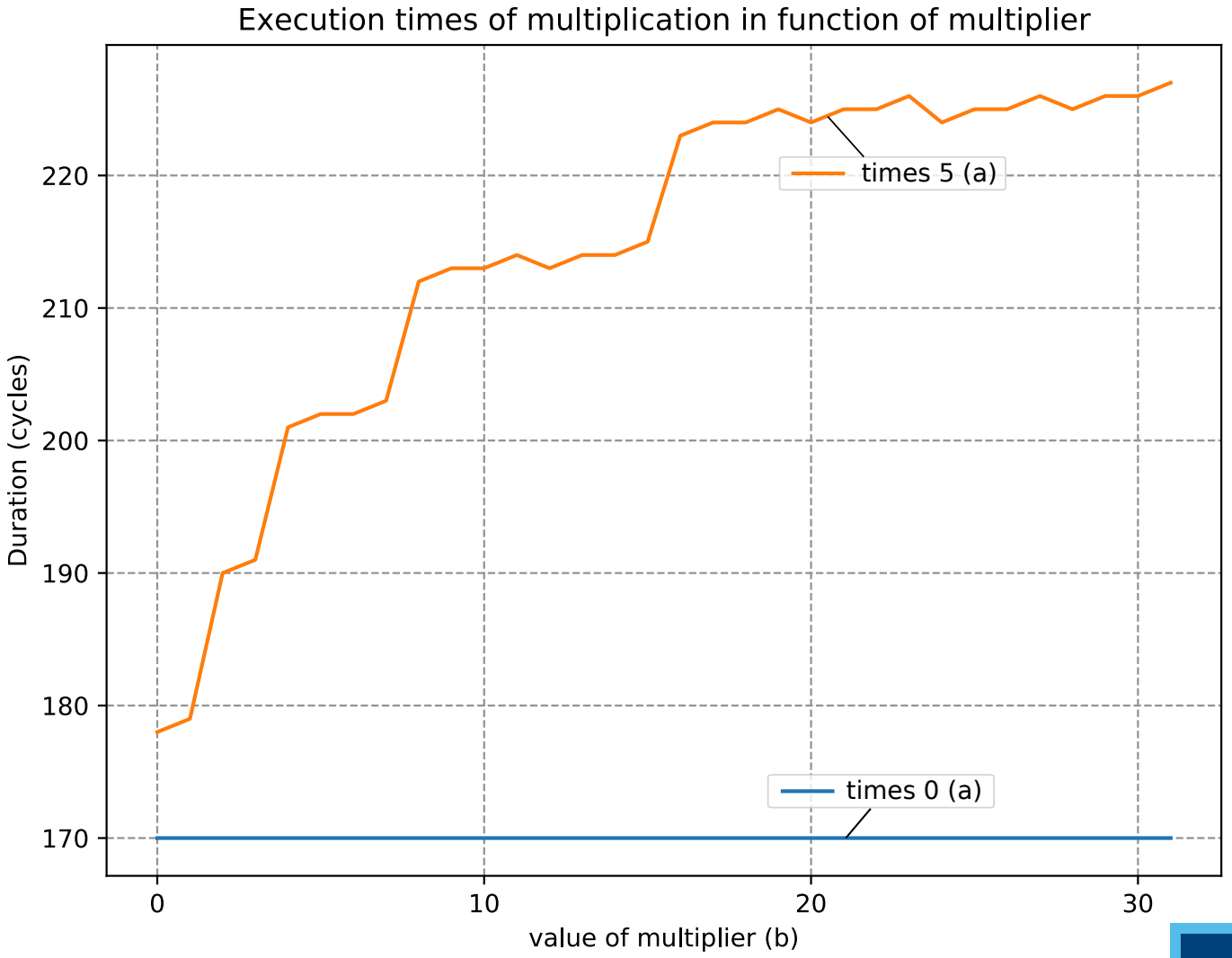
```
#include <sancus/sm_support.h>

DECLARE_SM(SM_mul, 0x1234);

int SM_DATA(SM_mul) a = -63;
int SM_DATA(SM_mul) b = 6;

int SM_ENTRY(SM_mul) sm_multiply(void)
{
    volatile int c = a * b;
    return 0;
}
```

Start-to-End Side-Channel in Multiplication



KU LEUVEN

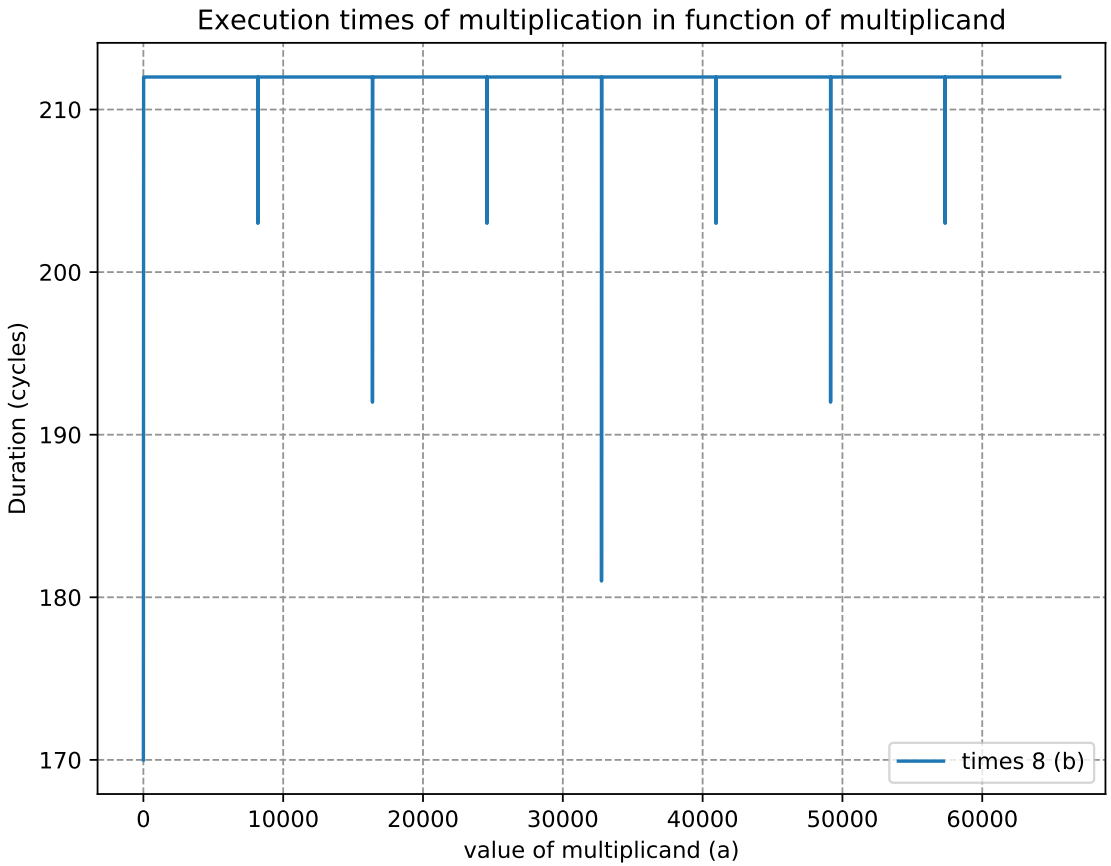
Start-to-End Side-Channel in Multiplication



KU LEUVEN

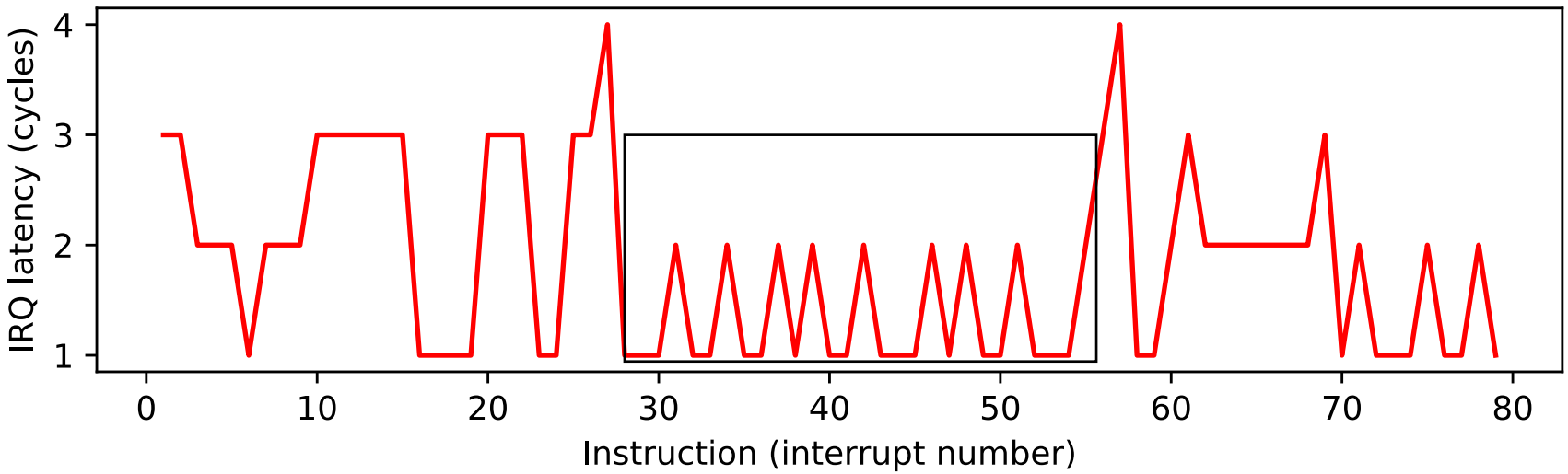
Start-to-End Side-Channel in Multiplication

a\b	...	14	15	...
...	...	...	...	...
120	...	213	214	...
121	...	213	214	...
...	...	...	...	...



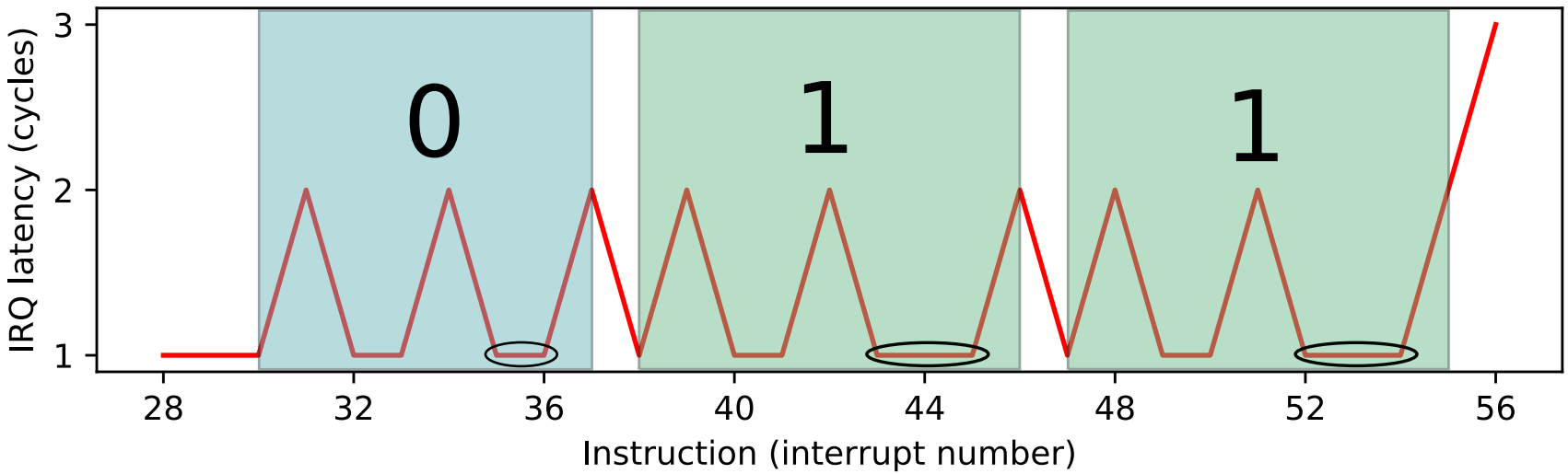
KU LEUVEN

Nemesis Side-Channel in Multiplication



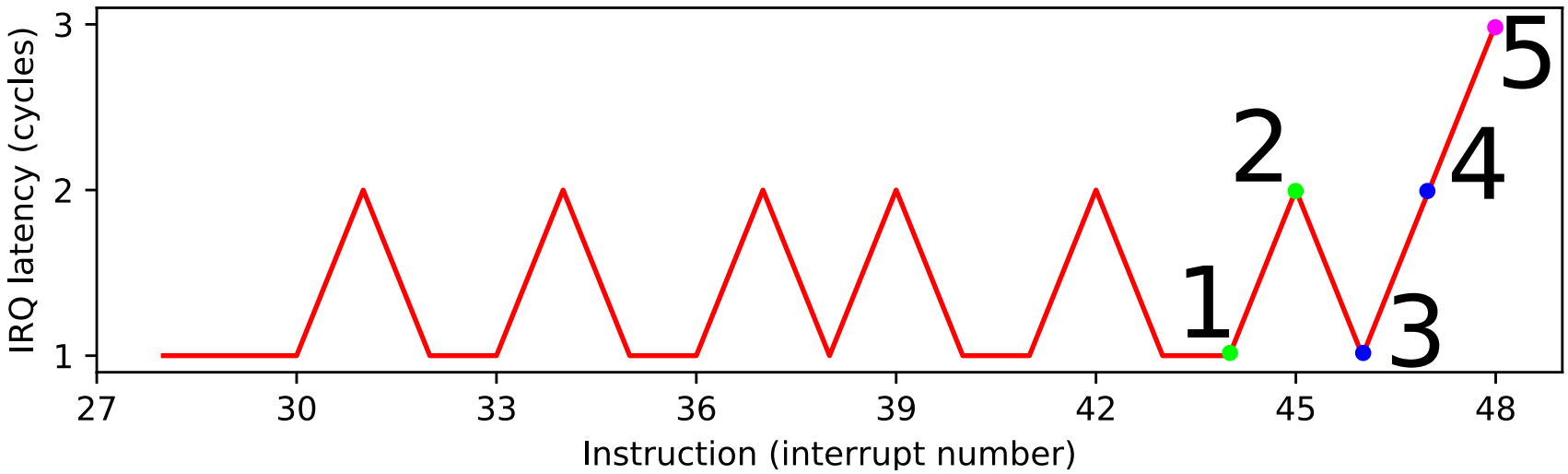
KU LEUVEN

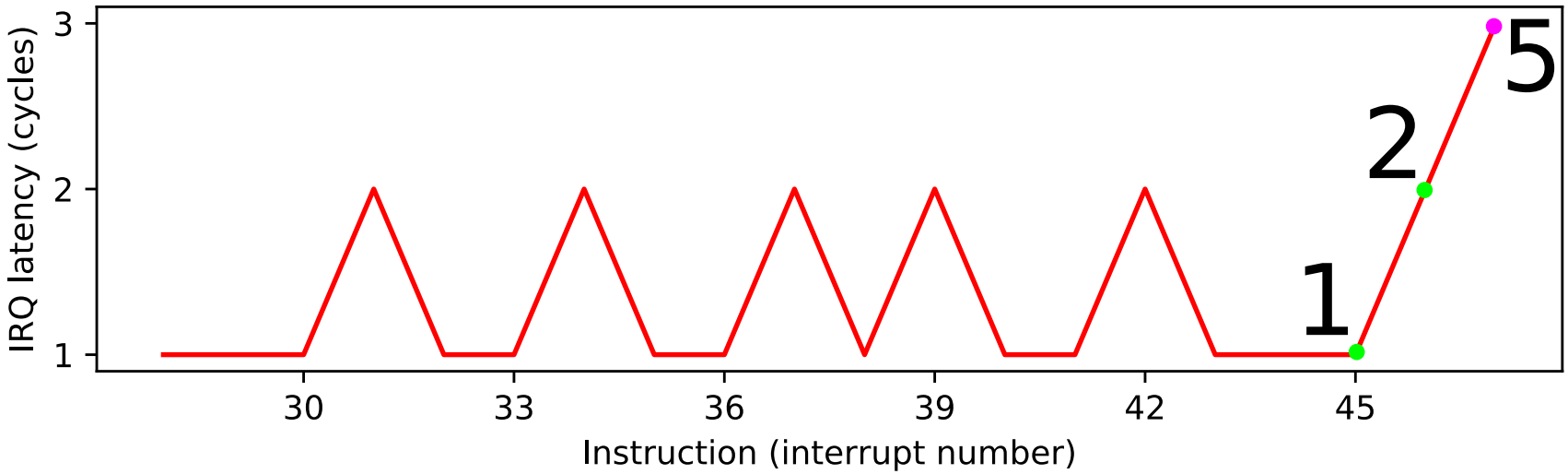
Leaking the Multiplier



KU LEUVEN

Determining the Stop Condition





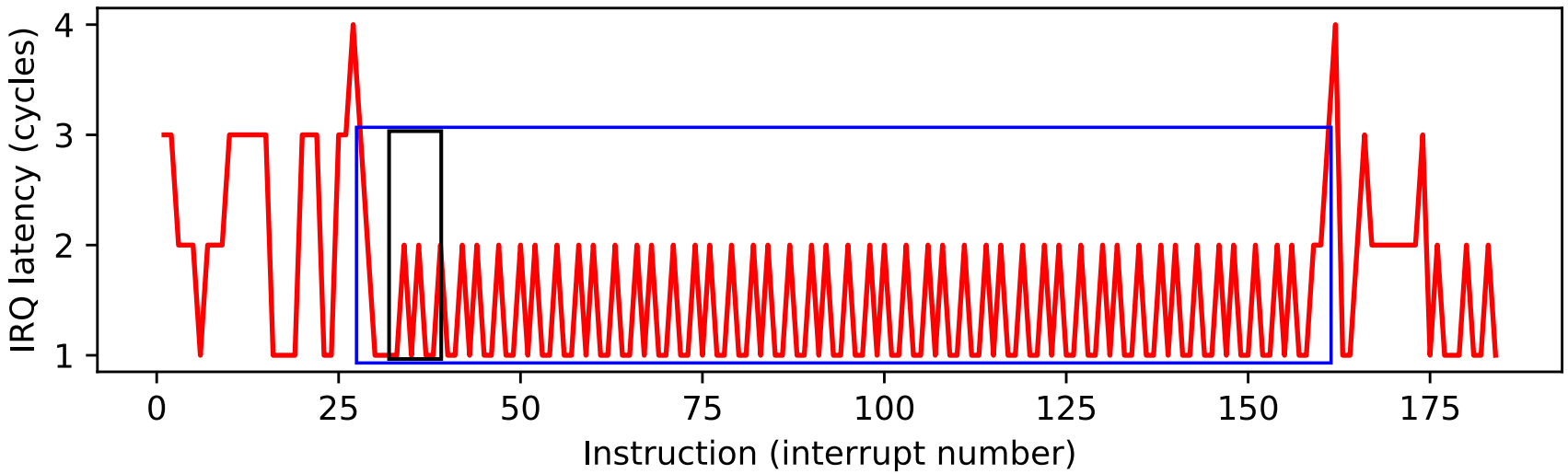
KU LEUVEN

Hardening the Multiplication Stub

```
__sm_mulhi3:
    mov     #16, r12
    mov     r15, r13
    clr     r15
1:  clrc
    rrc     r13
    jnc     2f
    add     r14, r15
    jmp     4f
2:  nop
    jmp     4f
4:  rla     r14
    sub     #1, r12
    jnz     1b
3:  ret
```

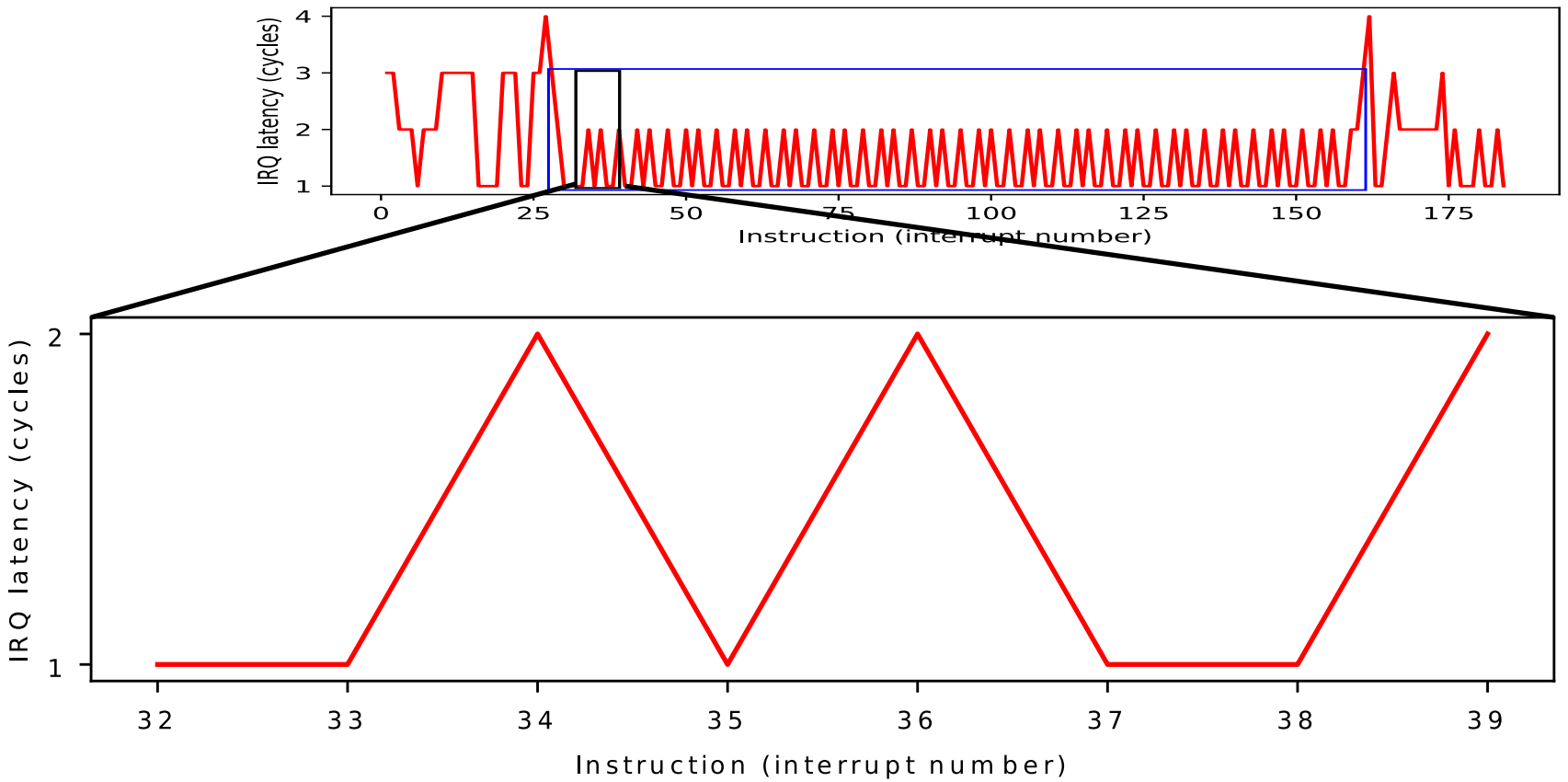
```
unsigned int mul(unsigned  int a,
                 unsigned int b)
{
    unsigned int rv = 0;
    for (int i = 0; i < 16; i++)
    {
        if (b & 1)
            rv += a;
        else
            asm("nop");
        b >>= 1;
        a <<= 1;
    }
    return rv;
}
```

Closing Nemesis

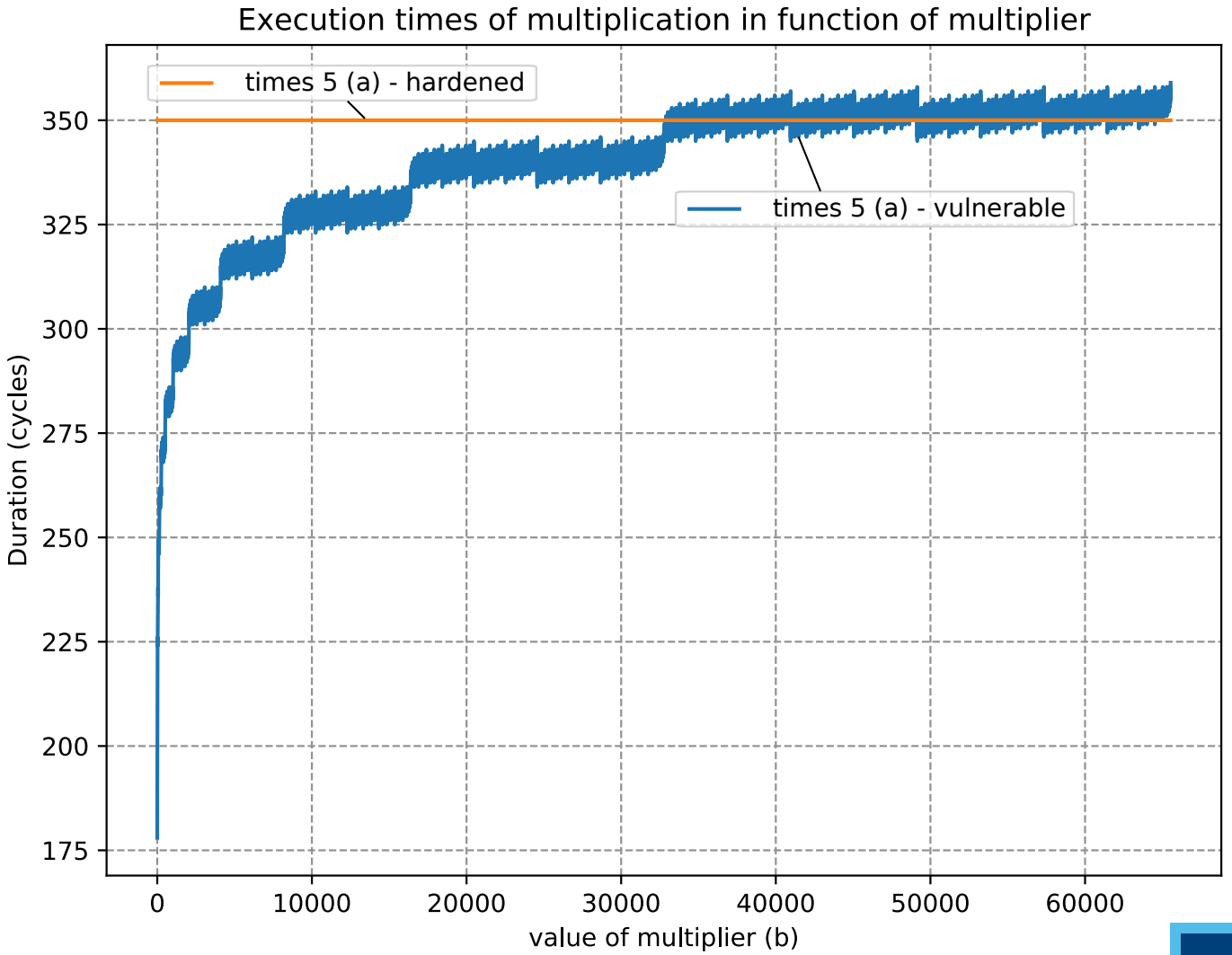


KU LEUVEN

Closing Nemesis



Performance Comparison



KU LEUVEN

Future and Ongoing Work

- Analyse other stubs
  - Side-channel leakage
  - Atomicity issues
- Defense
  - Shortterm $\rightarrow$ software
  - Longterm $\rightarrow$ hardware (ongoing work [6])

Thank you!
Questions?

References I

Pieter Agten et al. "Secure Compilation to Modern Processors". In: *2012 IEEE 25th Computer Security Foundations Symposium*. IEEE, June 2012. DOI: 10.1109/csf.2012.12. URL: https://doi.org/10.1109/csf.2012.12.

Job Noorman et al. "Sancus 2.0: A Low-Cost Security Architecture for IoT Devices". In: *ACM Transactions on Privacy and Security (TOPS)* 20.3 (Sept. 2017), 7:1–7:33.

References II

Raoul Strackx and Frank Piessens. "Fides: Selectively Hardening Software Application Components Against Kernel-level or Process-level Malware". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. Raleigh, North Carolina, USA: ACM, 2012, pp. 2–13. ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382200. URL: http://doi.acm.org/10.1145/2382196.2382200.

Raoul Strackx, Frank Piessens, and Bart Preneel. "Efficient Isolation of Trusted Subsystems in Embedded Systems". In: vol. 50. Sept. 2010, pp. 344–361. DOI: 10.1007/978-3-642-16161-2_20.

KU LEUVEN

References III

Jo Van Bulck, Frank Piessens, and Raoul Strackx. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. CCS '18. Toronto, Canada: ACM, 2018, pp. 178–195. ISBN: 978-1-4503-5693-0. DOI: 10.1145/3243734.3243822. URL: http://doi.acm.org/10.1145/3243734.3243822.

Jo Van Bulck et al. "Towards Availability and Real-Time Guarantees for Protected Module Architectures". In: *Companion Proceedings of the 15th International Conference on Modularity (MASS'16)*. ACM, 2016, pp. 146–151.