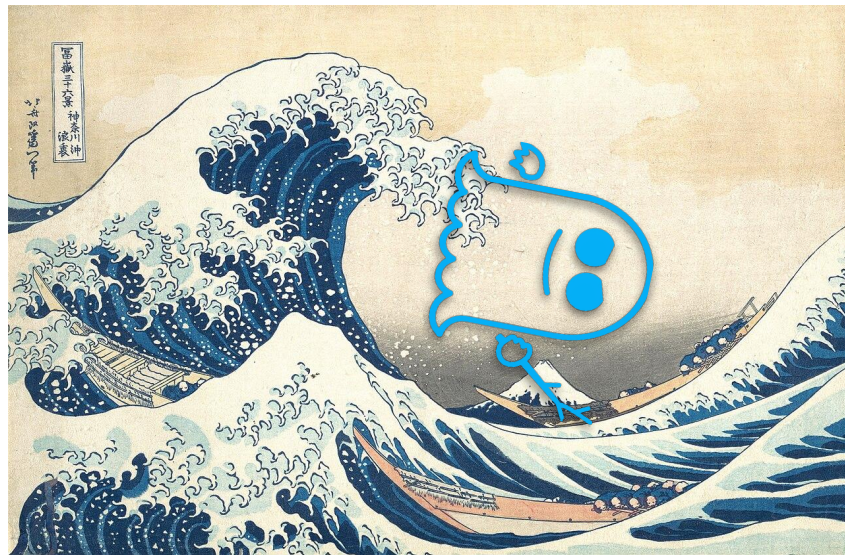


Microarchitectural Attacks and Provable Defenses

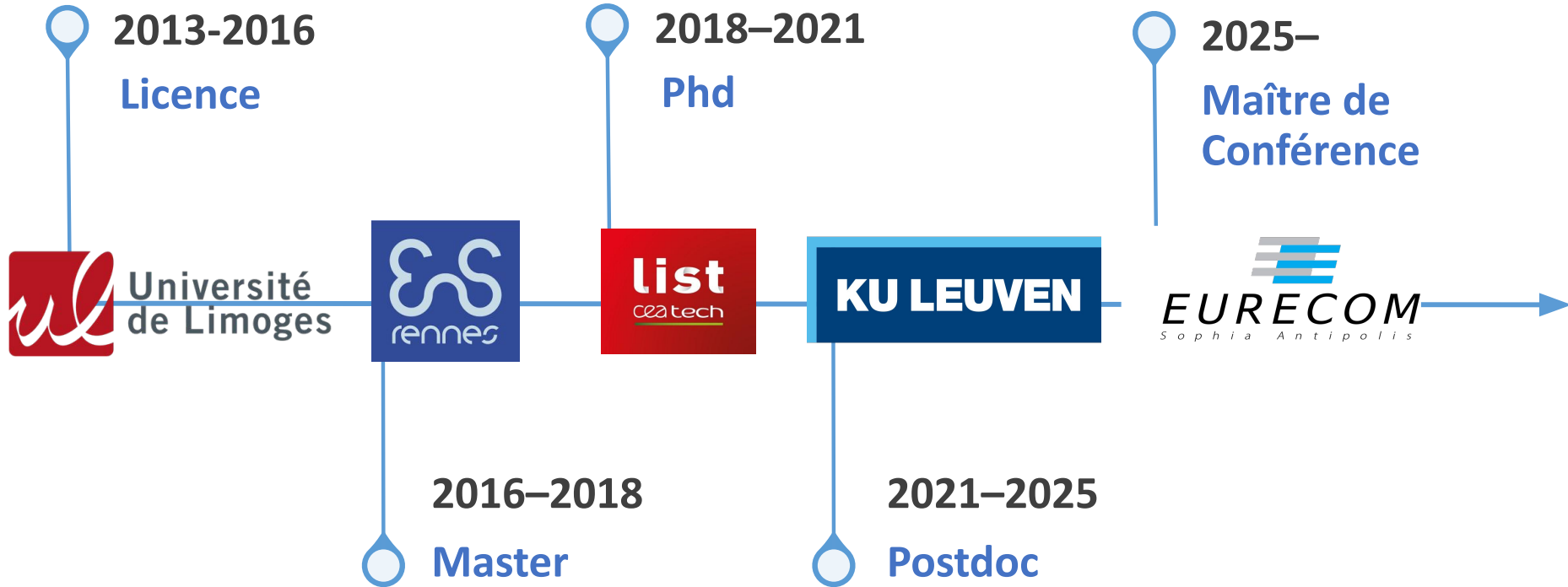
Seminar ENS Rennes

September 24th, 2025

Lesly-Ann Daniel, EURECOM



My background



Something I wish I knew before:

How to live with impostor syndrome

- **Bachelor:** felt like I didn't belong as a woman
- **ENS-Rennes:** weak math background, no classe prépa
- **Academia:** very competitive environment full of smart people
 - Hard to not compare yourself to others



What helped

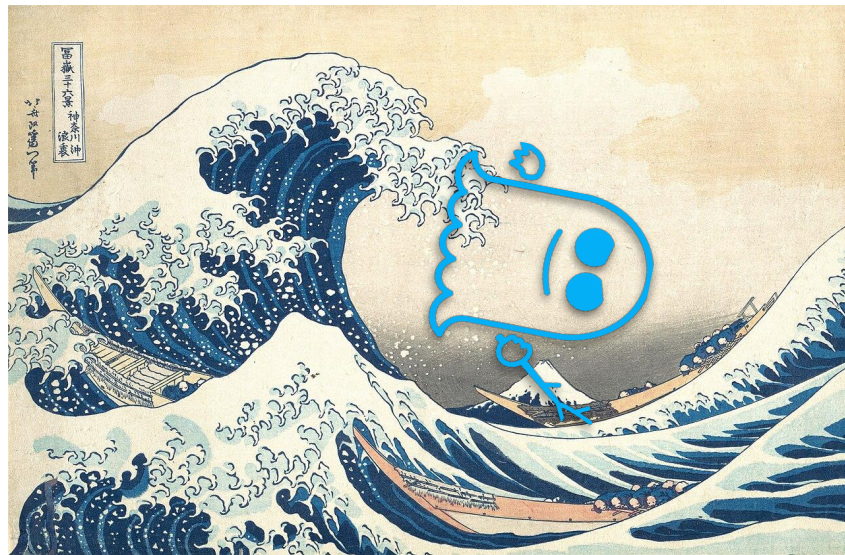
- Know that you're not alone
- Be kind to yourself and others
- Very supportive environments with amazing colleagues
(choose your advisors wisely)
- Do not compare to others: everyone is different!
- [Amy Cuddy TED Talk - Fake it Till You Make it](#)
- For women: [L'Oréal-UNESCO For Women in Science](#)

Microarchitectural Attacks and Provable Defenses

Seminar ENS Rennes

September 24th, 2025

Lesly-Ann Daniel, EURECOM



Security critical software is prevalent

What:

- Secure communications,
- Banking transactions, ...

Where:

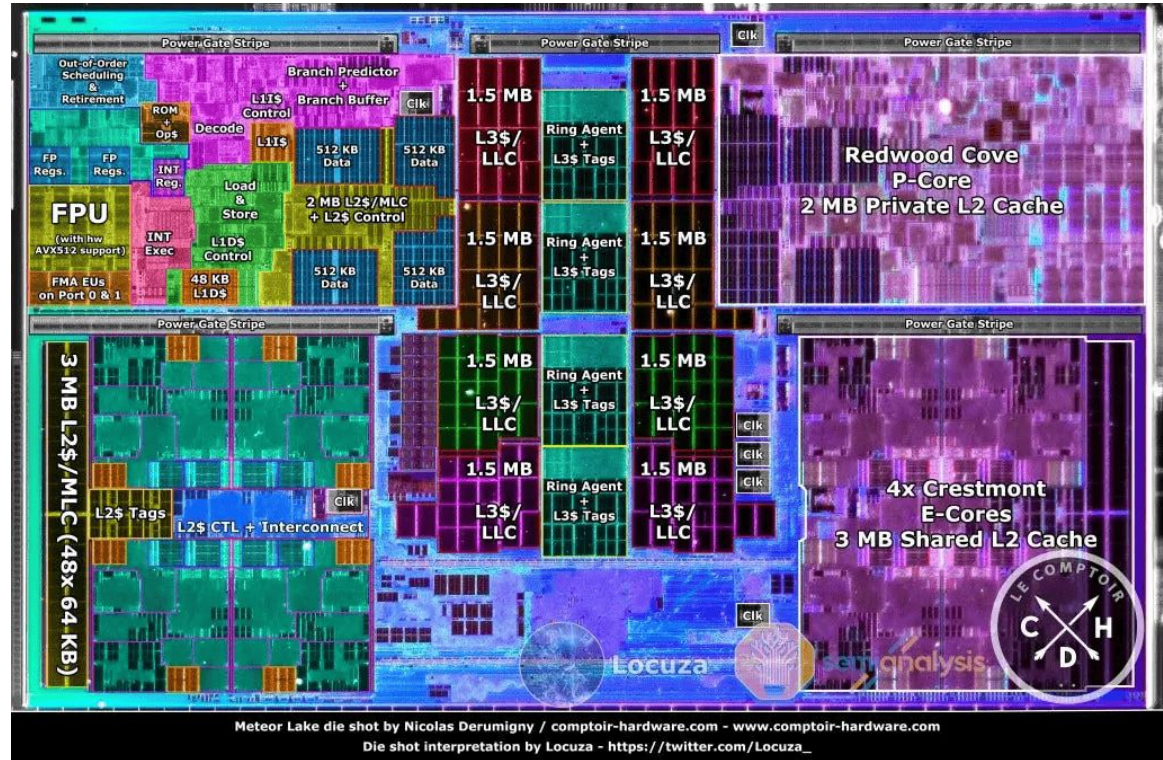
- Servers, smartphones, ...
- Shared by many users (cloud)

How: cryptography

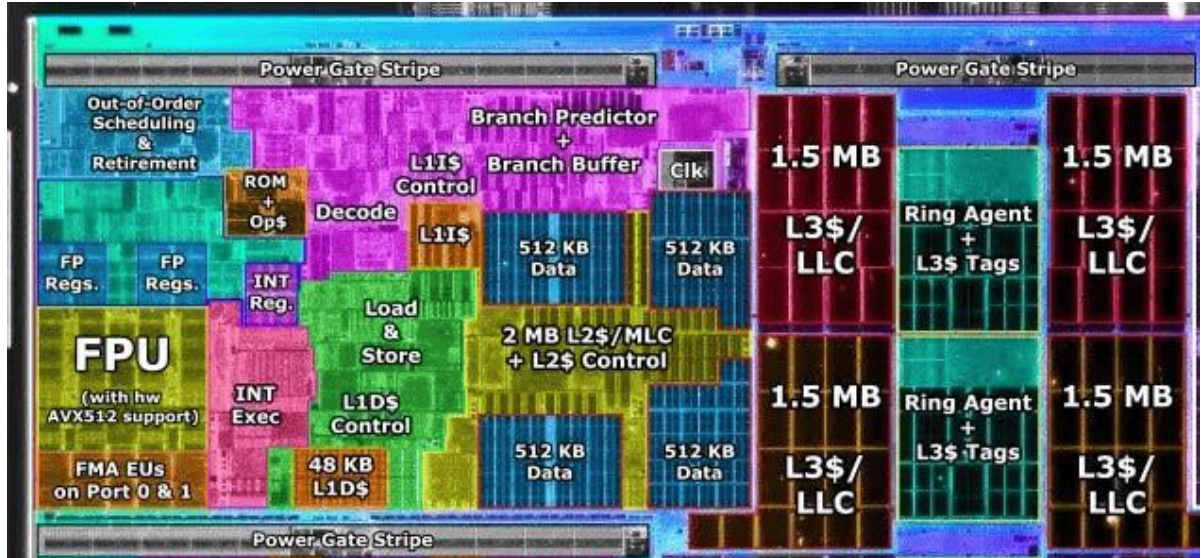
- Mathematical guarantees, verified implems.
- *But what about their execution in the physical world?*



Processors are full of optimizations



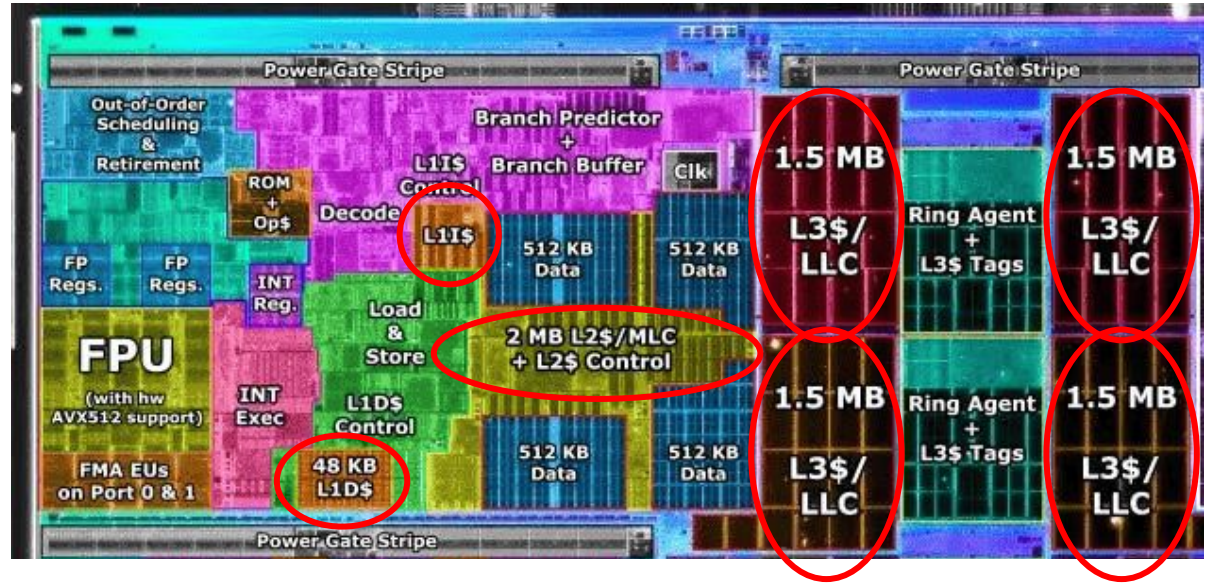
Processors are full of optimizations



Intel Meteor Lake – Credit <https://semianalysis.com/2022/05/26/meteor-lake-die-shot-and-architecture/>

Processors are full of optimizations

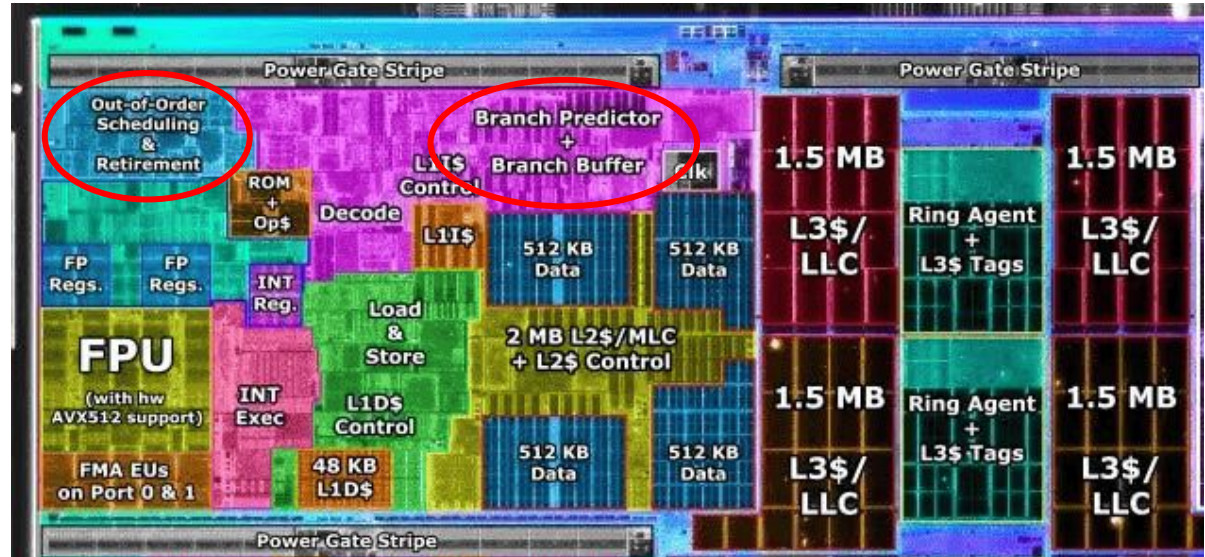
- Caches



Intel Meteor Lake – Credit <https://semianalysis.com/2022/05/26/meteor-lake-die-shot-and-architecture/>

Processors are full of optimizations

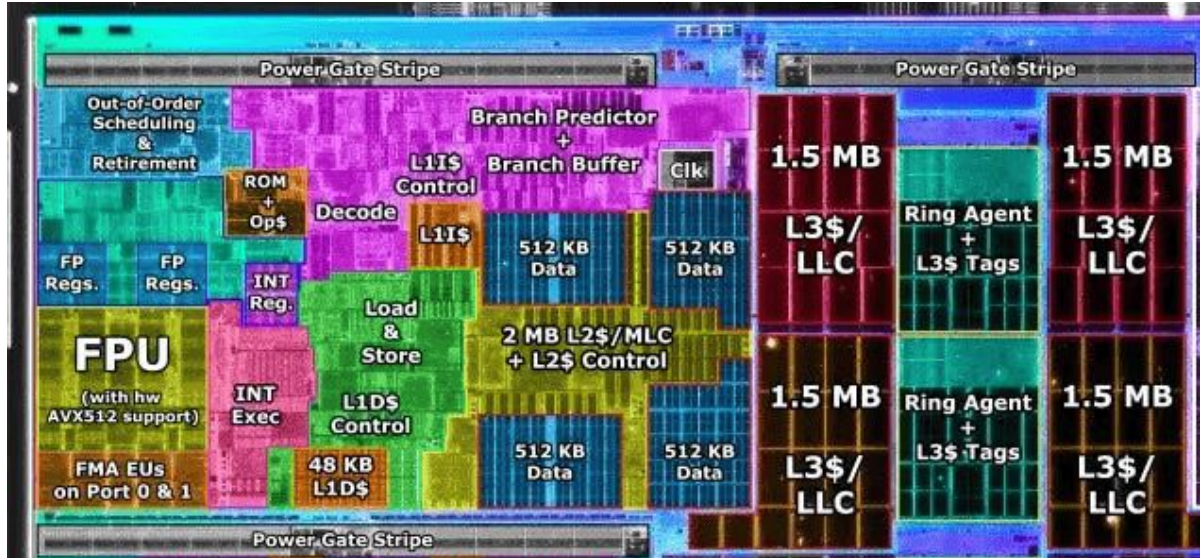
- Caches
- Out-of-order speculative execution



Intel Meteor Lake – Credit <https://semianalysis.com/2022/05/26/meteor-lake-die-shot-and-architecture/>

Processors are full of optimizations

- Caches
- Out-of-order speculative execution
- And more [1]?



[1] Vicarte, Jose Rodrigo Sanchez, et al. "Opening pandora's box: A systematic study of new ways microarchitecture can leak private data." ISCA, 2021

Intel Meteor Lake – Credit <https://semianalysis.com/2022/05/26/meteor-lake-die-shot-and-architecture/>

Processors are full of optimizations

- Caches

What about security?

execution

- And more [1]?



[1] Vicarte, Jose Rodrigo Sanchez, et al. "Opening pandora's box: A systematic study of new ways microarchitecture can leak private data." ISCA, 2021

Intel Meteor Lake – Credit <https://semianalysis.com/2022/05/26/meteor-lake-die-shot-and-architecture/>

... Well security is not good :(

CNN BUSINESS Markets Tech Media Success Video

Major chip flaws affect billions of devices

by Selena Larson @CNNTech

January 4, 2018: 9:44 AM ET

BBC

Home News Sport Business Innovation Culture Arts Travel Earth Audio Video Live

'Foreshadow' attack affects Intel chips

15 August 2018

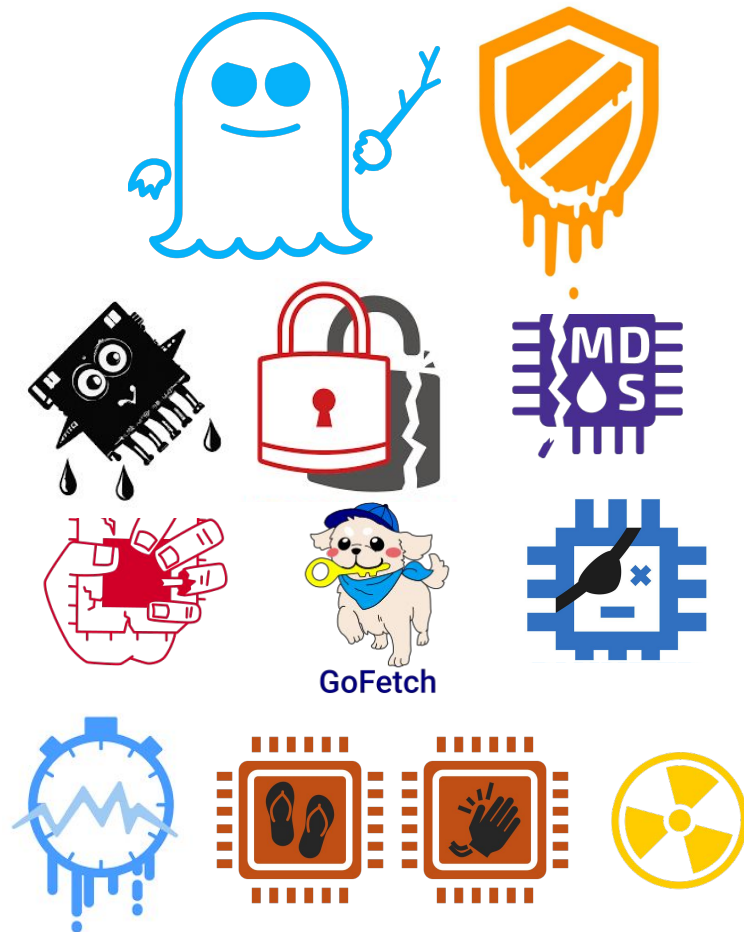
Dave Lee
North America technology reporter

Spectre flaws continue to haunt Intel and AMD as researchers find fresh attack method

The indirect branch predictor barrier is less of a barrier than hoped

Thomas Claburn

Fri 18 Oct 2024 // 14:01 UTC



**non exhaustive list*

Back to the basics

Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems

Paul C. Kocher

Cryptography Research, Inc.
607 Market Street, 5th Floor, San Francisco, CA 94105, USA.
E-mail: paul@cryptography.com.

Abstract. By carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems. Against a vulnerable system, the attack is computationally inexpensive and often requires only known ciphertext. Actual systems are potentially at risk, including cryptographic tokens, network-based cryptosystems, and other applications where attackers can make reasonably accurate timing measurements. Techniques for preventing the attack for RSA and Diffie-Hellman are presented. Some cryptosystems will need to be revised to protect against the attack, and new protocols and algorithms may need to incorporate measures to prevent timing attacks.

1996

Cache-timing attacks on AES

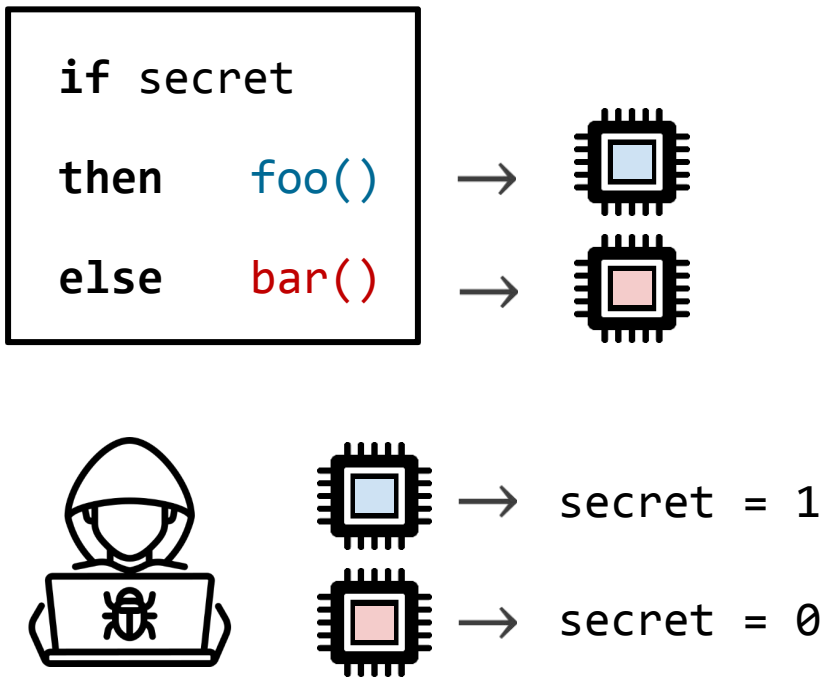
Daniel J. Bernstein *

Department of Mathematics, Statistics, and Computer Science (M/C 249)
The University of Illinois at Chicago
Chicago, IL 60607-7045
djb@cr.yp.to

Abstract. This paper demonstrates complete AES key recovery from known-plaintext timings of a network server on another computer. This attack should be blamed on the AES design, not on the particular AES library used by the server; it is extremely difficult to write constant-time high-speed AES software for common general-purpose computers. This paper discusses several of the obstacles in detail.

2005

Control-flow leaks



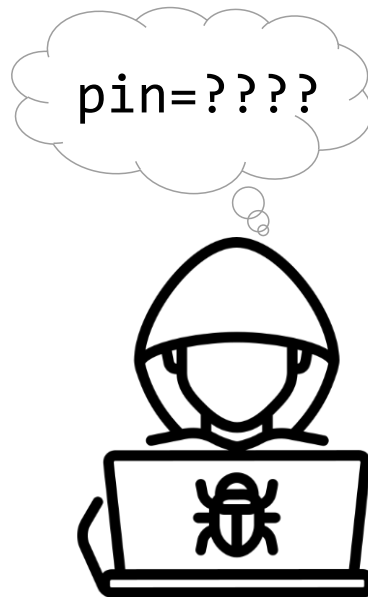
- end-to-end timing
- different resource consumption
- branch predictor state
- instruction cache
- instruction prefetcher
- micro-op cache
- ...

Concrete example

```
bool check_pin(char* guess) {  
    for (i=0; i<4; i++)  
        if (guess[i] != pin[i])  
            return false;  
    return true;  
}
```



pin = 4321



Concrete example

```
bool check_pin(char* guess) {  
    for (i=0; i<4; i++)  
        if (guess[i] != pin[i])  
            return false;  
    return true;  
}
```



pin = 4321

0000 → 1s

1000 → 1s

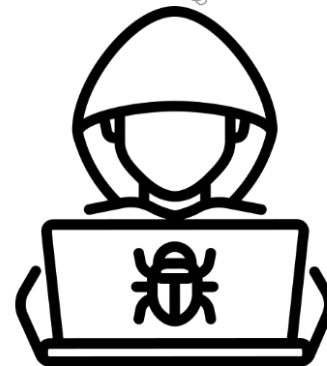
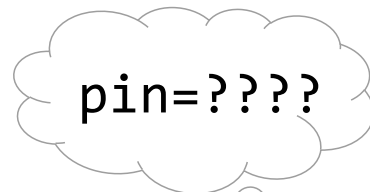
2000 → 1s

3000 → 1s

4000 → 2s

5000 → 1s

...



Concrete example

```
bool check_pin(char* guess) {  
    for (i=0; i<4; i++)  
        if (guess[i] != pin[i])  
            return false;  
    return true;  
}
```



pin = 4321

0000 → 1s

1000 → 1s

2000 → 1s

3000 → 1s

4000 → 2s

5000 → 1s

...

pin=4???



Concrete example

```
bool check_pin(char* guess) {  
    for (i=0; i<4; i++)  
        if (guess[i] != pin[i])  
            return false;  
    return true;  
}
```



pin = 4321

4000 → 2s

4100 → 2s

4200 → 2s

4300 → 3s

4400 → 2s

4500 → 2s

...

pin=43??



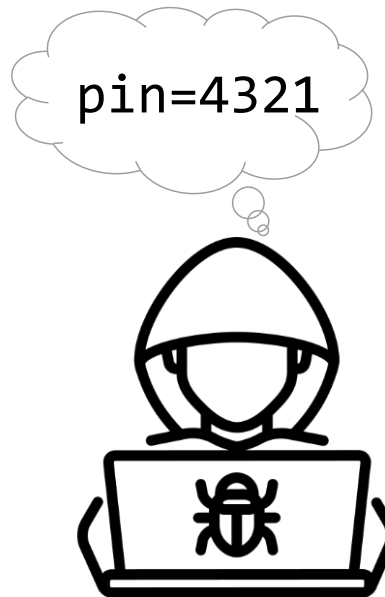
Concrete example

```
bool check_pin(char* guess) {  
    for (i=0; i<4; i++)  
        if (guess[i] != pin[i])  
            return false;  
    return true;  
}
```



pin = 4321

Attack
Complexity:
from 10^4
to 10×4



Concrete example

```
bool check_pin(char* guess) {  
    good = true;  
    for (i=0; i<4; i++)  
        good &= guess[i] == pin[i];  
    return good;  
}
```

Solution

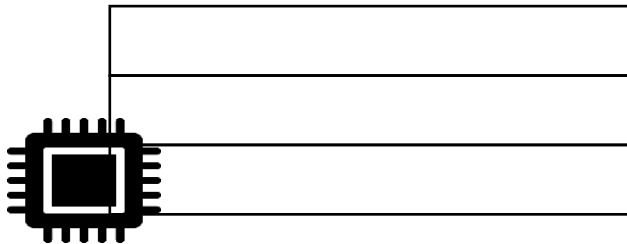
Make timing independent of secret
Remove secret-dependent branch!

Memory accesses leak

Victim program

```
x = tab[secret]
```

Data cache



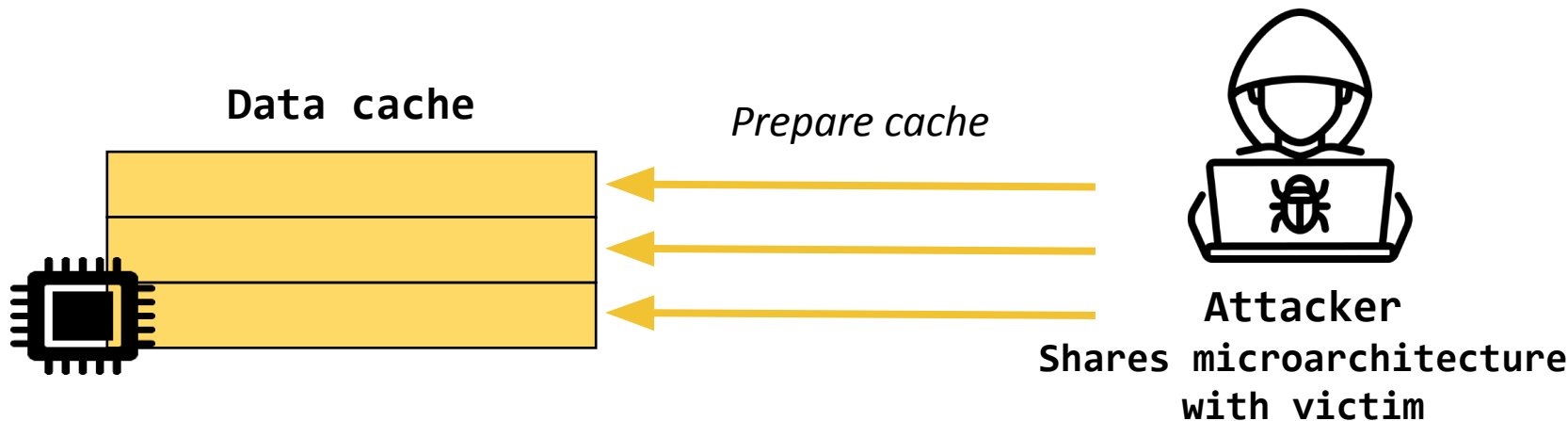
Attacker

Shares microarchitecture
with victim

Memory accesses leak

Victim program

```
x = tab[secret]
```

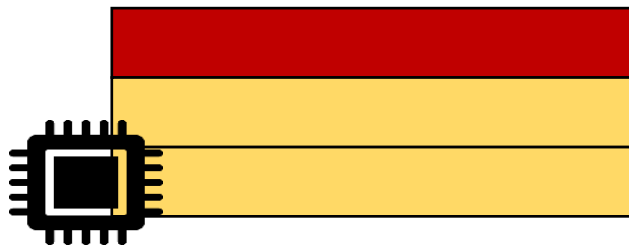


Memory accesses leak

Victim program

```
x = tab[secret]
```

Data cache



Victim executes



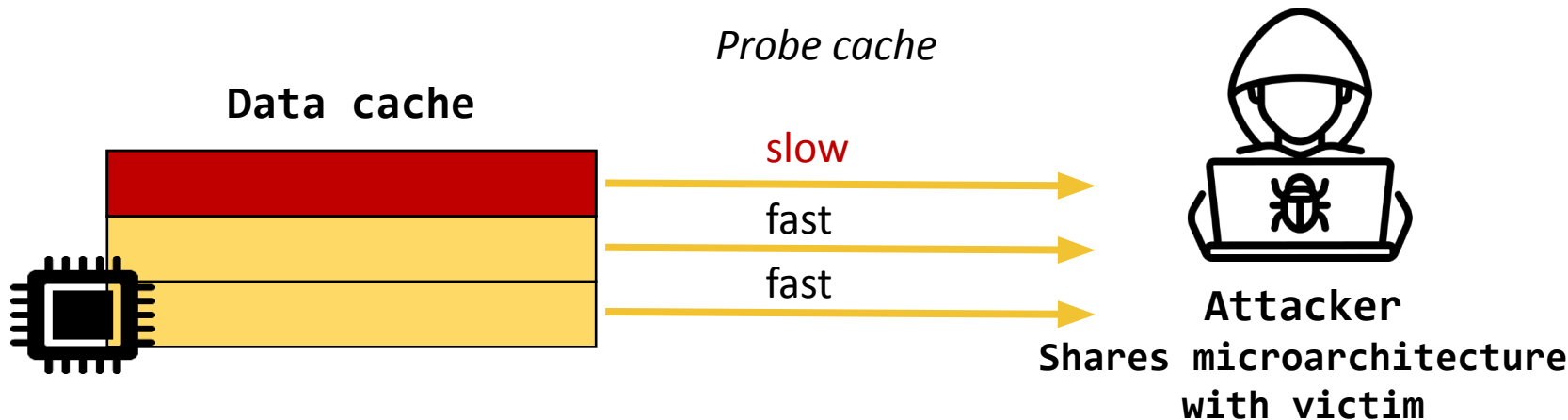
Attacker

Shares microarchitecture
with victim

Memory accesses leak

Victim program

```
x = tab[secret]
```

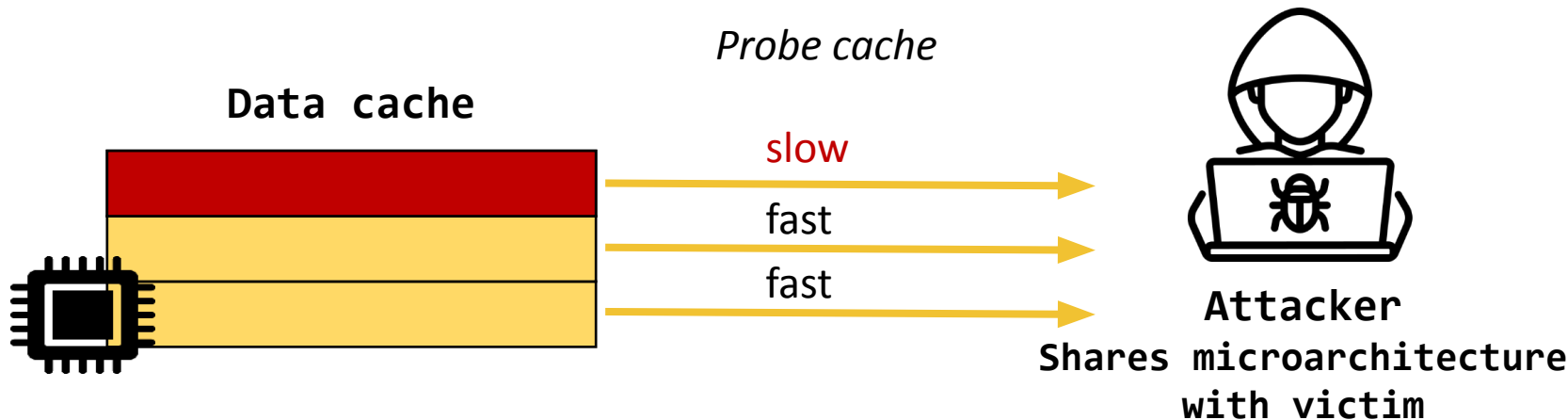


Memory accesses leak

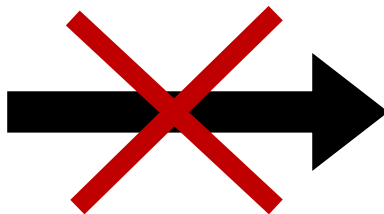
Victim program

```
x = tab[secret]
```

- caches
- data pre-fetchers
- load/store dependencies
- ...



Solution? Constant-time programming!



Unsafe instructions

- Control-Flow
- Memory accesses
- Variable-time instr.

- Full software countermeasure
- De facto standard for crypto: [BearSSL](#), [Libsodium](#), [HACL*](#), etc.
- (Almost) Secure against micro-architectural attacks (hum...)

Constant-time is not easy to implement

```
uint32_t select(uint32_t x, uint32_t y, bool secret) {  
    if (secret) return x;  
    else return y;  
}
```



```
uint32_t ct_select(uint32_t x, uint32_t y, bool secret) {  
    signed b = 0 - secret;  
    return (x & b) | (y & ~b);  
}
```



Compilers can break constant-time!

```
uint32_t ct_select(uint32_t x, uint32_t y, bool secret) {  
    signed b = 0 - secret;  
    return (x & b) | (y & ~b);  
}
```



clang-3.0 -O0

```
public ct_select_u32_v4  
ct_select_u32_v4 proc near  
  
var_14= dword ptr -14h  
var_D= byte ptr -0Dh  
var_C= dword ptr -0Ch  
var_8= dword ptr -8  
arg_0= dword ptr 4  
arg_4= dword ptr 8  
arg_8= byte ptr 0Ch  
  
push     esi  
sub      esp, 10h  
mov      al, [esp+14h+arg_8]  
mov      ecx, [esp+14h+arg_4]  
mov      edx, [esp+14h+arg_0]  
mov      [esp+14h+var_8], edx  
mov      [esp+14h+var_C], ecx  
and      al, 1  
mov      [esp+14h+var_D], al  
mov      al, [esp+14h+var_D]  
and      al, 1  
movzx    ecx, al  
mov      edx, 0  
sub      edx, ecx  
mov      [esp+14h+var_14], edx  
mov      ecx, [esp+14h+var_8]  
and      ecx, [esp+14h+var_14]  
mov      edx, [esp+14h+var_C]  
mov      esi, [esp+14h+var_14]  
xor      esi, 0FFFFFFFh  
and      esi, edx  
or       esi, ecx  
mov      eax, esi  
add      esp, 10h  
pop      esi  
retn  
ct_select_u32_v4 endp
```



clang-3.0 -O3

```
public ct_select_u32_v4  
ct_select_u32_v4 proc near  
  
arg_0= byte ptr 4  
arg_4= byte ptr 8  
arg_8= byte ptr 0Ch  
  
mov      al, [esp+arg_8]  
test     al, al  
jz       short loc_804842F
```

29
lea eax, [esp+arg_0]
mov eax, [eax]
retn

loc_804842F:
lea eax, [esp+arg_4]
mov eax, [eax]
retn
ct_select_u32_v4 endp



What can we do about it?

Constant-time preserving compilers

Secure compilation of side-channel countermeasures:
the case of cryptographic “constant-time”

Gilles Barthe*, Benjamin Grégoire†, Vincent Laporte*

*IMDEA Software Institute, Madrid, Spain

gilles.barthe@imdea.org vlaporte@imdea.org

†Inria, Sophia-Antipolis, France

benjamin.gregoire@inria.fr

Formal Verification of a Constant-Time Preserving C Compiler

GILLES BARTHE, MPI for Security and Privacy, Germany and IMDEA Software Institute, Spain

SANDRINE BLAZY, Univ Rennes, Inria, CNRS, IRISA, France

BENJAMIN GRÉGOIRE, Inria, France

RÉMI HUTIN, Univ Rennes, Inria, CNRS, IRISA, France

VINCENT LAPORTE, Inria, France

DAVID PICHARDIE, Univ Rennes, Inria, CNRS, IRISA, France

ALIX TRIEU, Aarhus University, Denmark

Domain specific languages & compilers for crypto

FaCT: A DSL for Timing-Sensitive Computation

Sunjay Cauligi
UC San Diego, USA

Gary Soeller
UC San Diego, USA

Brian Johannesmeyer
UC San Diego, USA

Fraser Brown
Stanford, USA

Riad S. Wahby
Stanford, USA

John Renner
UC San Diego, USA

Benjamin Grégoire
INRIA Sophia Antipolis, France

Gilles Barthe
MPI for Security and Privacy,
Germany
IMDEA Software Institute, Spain

Ranjit Jhala
UC San Diego, USA

Deian Stefan
UC San Diego, USA

Jasmin: High-Assurance and High-Speed Cryptography

José Bacelar Almeida
INESC TEC and
Universidade do Minho, Portugal

Manuel Barbosa
INESC TEC and FCUP
Universidade do Porto, Portugal

Gilles Barthe
IMDEA Software Institute, Spain

Arthur Blot
ENS Lyon, France

Benjamin Grégoire
Inria Sophia-Antipolis, France

Vincent Laporte
IMDEA Software Institute, Spain

Tiago Oliveira
INESC TEC and FCUP
Universidade do Porto, Portugal

Hugo Pacheco
INESC TEC and
Universidade do Minho, Portugal

Benedikt Schmidt
Google Inc.

Pierre-Yves Strub
École Polytechnique, France

What can we do about it?

Constant-time preserving compilers

Secure compilation of side-channel countermeasures:
the case of cryptographic “constant-time”

Gilles Barthe*, Benjamin Grégoire†, Vincent Laporte*

Formal Verification of a Constant-Time Preserving C Compiler

GILLES BARTHE, MPI for Security and Privacy, Germany and IMDEA Software Institute, Spain
SANDRINE BLAZY, Univ Rennes, Inria, CNRS, IRISA, France
BENJAMIN GRÉGOIRE, Inria, France

**Fight the compiler
And verify binary code**

Sunjay Cauligi
UC San Diego, USA

Fraser Brown
Stanford, USA

Benjamin Grégoire
INRIA Sophia Antipolis, France

Gary Soeller
UC San Diego, USA

Riad S. Wahby
Stanford, USA

Gilles Barthe
MPI for Security and Privacy,
Germany
IMDEA Software Institute, Spain

Deian Stefan
UC San Diego, USA

Brian Johannemeyer
UC San Diego, USA

John Renner
UC San Diego, USA

Ranjit Jhala
UC San Diego, USA

Jose Bacelar Almeida
INESC TEC and
Universidade do Minho, Portugal

Arthur Blot
ENS Lyon, France

Tiago Oliveira
INESC TEC and FCUP
Universidade do Porto, Portugal

Manuel Barbosa
INESC TEC and FCUP
Universidade do Porto, Portugal

Benjamin Grégoire
Inria Sophia-Antipolis, France

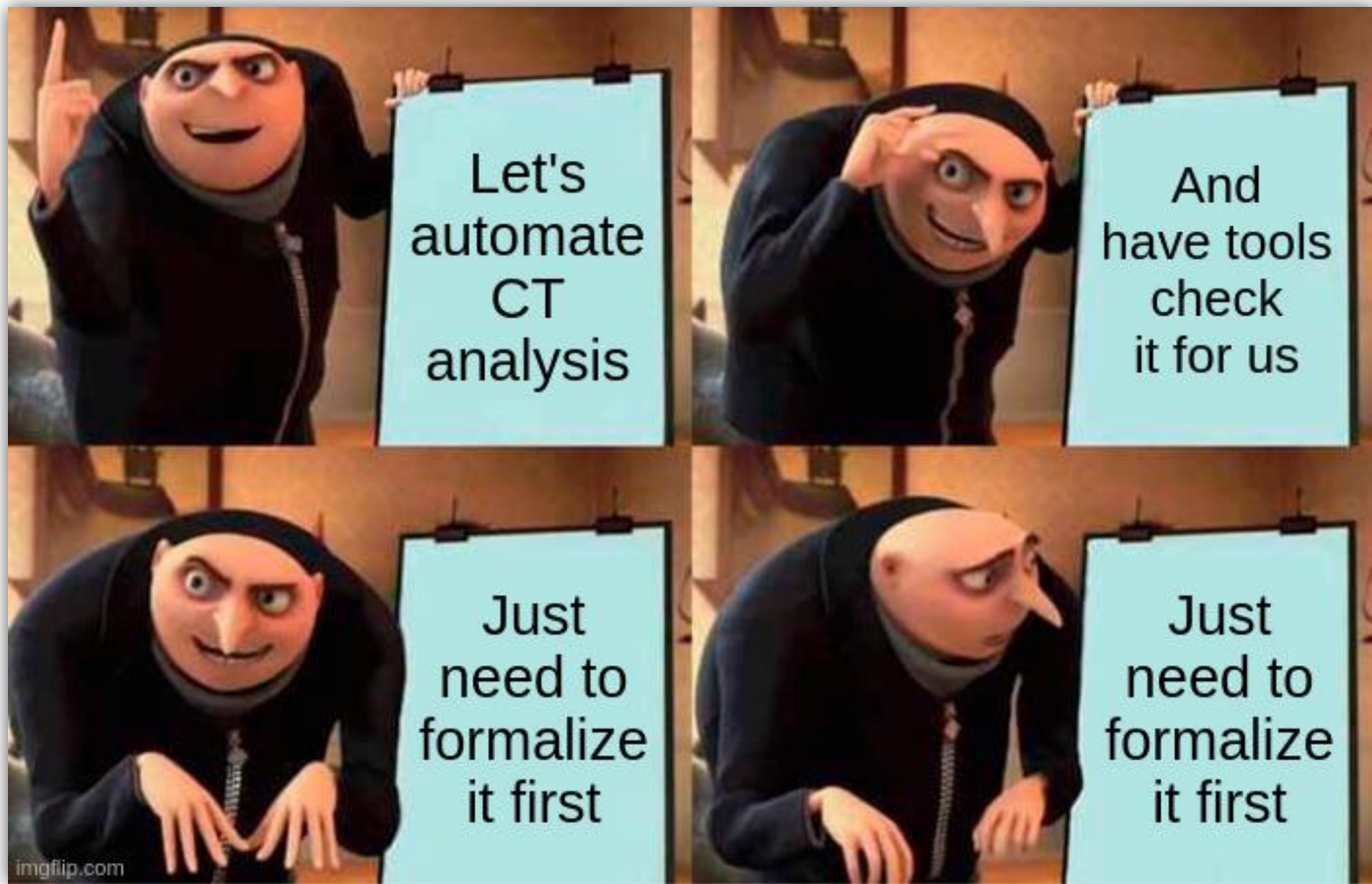
Hugo Pacheco
INESC TEC and
Universidade do Minho, Portugal

Pierre-Yves Strub
École Polytechnique, France

Gilles Barthe
IMDEA Software Institute, Spain

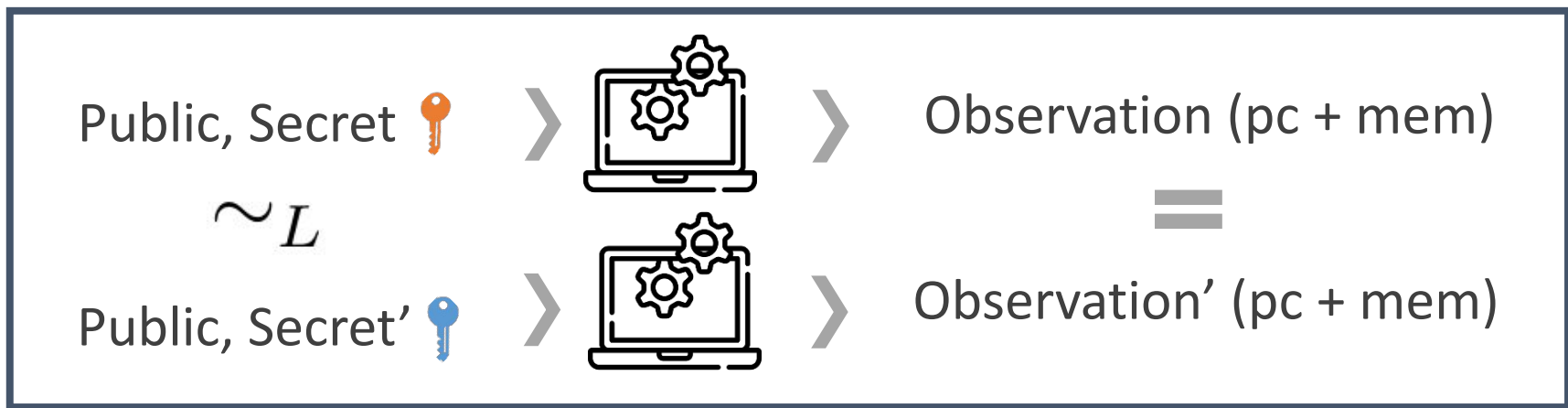
Vincent Laporte
IMDEA Software Institute, Spain

Benedikt Schmidt
Google Inc.



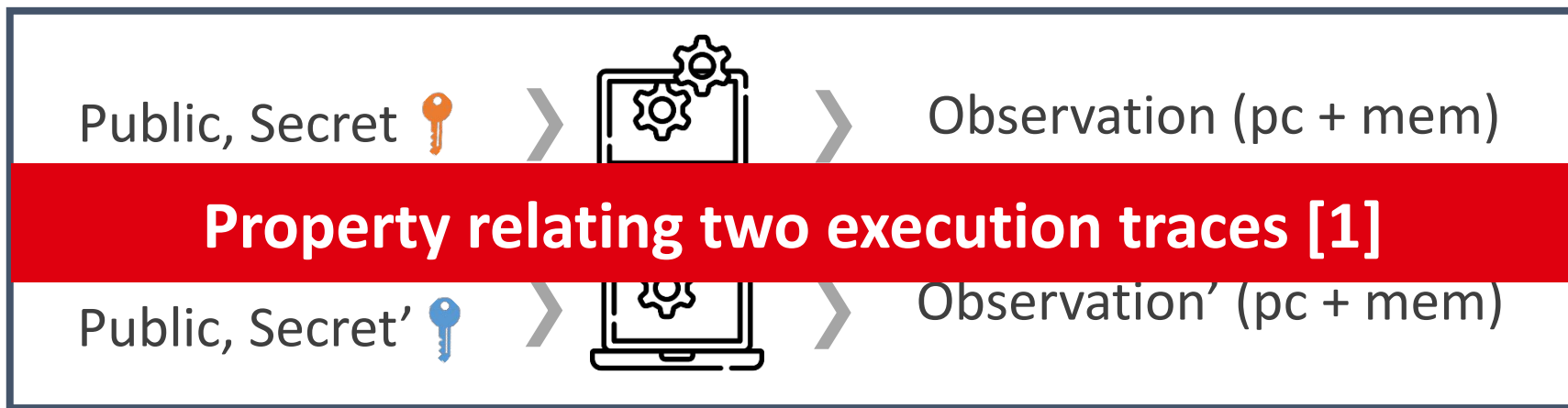
Constant-Time (a bit more) Formally

*2 executions that only differ in their secret input
must be indistinguishable to an observer*



Constant-Time (a bit more) Formally

*2 executions that only differ in their secret input
must be indistinguishable to an observer*



Several approaches [1]

Static

- Type systems
- Abstract interpretation
- **Symbolic execution**

Dynamic

- Record and compare observations
- Statistical tests
- Fuzzing
- Dynamic symbolic execution

[1] Geimer, Antoine, Mathéo Vergnolle, Frédéric Recoules, Lesly-Ann Daniel, Sébastien Bardin, and Clémentine Maurice. "A systematic evaluation of automated tools for side-channel vulnerabilities detection in cryptographic libraries." In *ACM CCS* 2023.

Binary-Level Symbolic Execution for Constant-Time

BINSEC/REL: Efficient Relational Symbolic
Execution for Constant-Time at Binary-Level

Lesly-Ann Daniel*, Sébastien Bardin*, Tamara Rezk†



<https://github.com/binsec/rel>

Background: Symbolic Execution

```
0x0080: mul  t, p, s  
0x0084: add  t, t, 48  
0x0088: beqz t error  
0x008c: div  t, s, t  
0x0090: [...]
```

Can error be reached?



Background: Symbolic Execution

```
0x0080: mul  t, p, s
0x0084: add  t, t, 48
0x0088: beqz t error
0x008c: div  t, s, t
0x0090: [...]
```



Symbolic store

$p \mapsto p_0$

$s \mapsto s_0$

$pc \mapsto 0x0080$

Background: Symbolic Execution

```
0x0080: mul  t, p, s
```

```
0x0084: add  t, t, 48
```

```
0x0088: beqz t error
```

```
0x008c: div  t, s, t
```

```
0x0090: [...]
```



Symbolic store

$p \mapsto p_0$

$s \mapsto s_0$

$t \mapsto p_0 \times s_0$

$pc \mapsto 0x0084$

Background: Symbolic Execution

```
0x0080: mul  t, p, s
0x0084: add  t, t, 48
0x0088: beqz t error
0x008c: div  t, s, t
0x0090: [...]
```



Symbolic store

$p \mapsto p_0$

$s \mapsto s_0$

$t \mapsto p_0 \times s_0 - 48$

$pc \mapsto 0x0088$

Background: Symbolic Execution

```
0x0080: mul  t, p, s
0x0084: add  t, t, 48
0x0088: beqz t error
0x008c: div  t, s, t
0x0090: [...]
```



Symbolic store

$p \mapsto p_0$

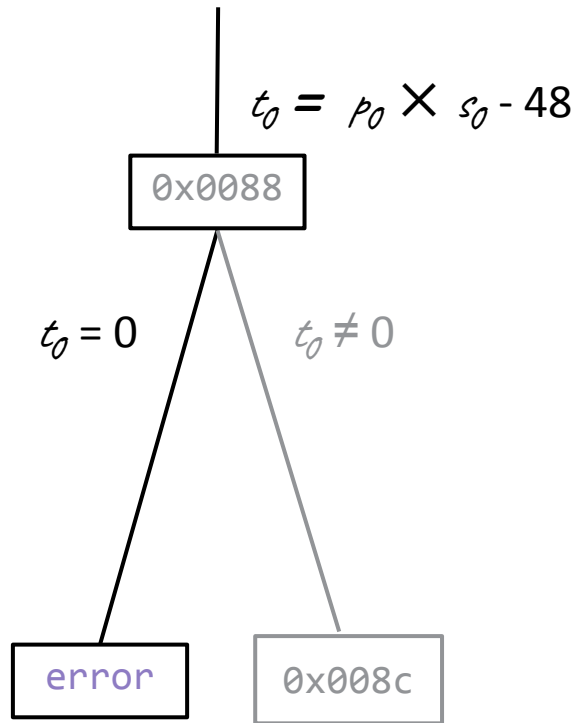
$s \mapsto s_0$

$t \mapsto p_0 \times s_0 - 48$

$pc \mapsto \text{error}$

Path constraint

$t_0 = 0$



Background: Symbolic Execution

```
0x0080: mul  t, p, s
0x0084: add  t, t, 48
0x0088: beqz t error
0x008c: div  t, s, t
0x0090: [...]
```

Symbolic store

$p \mapsto p_0$

$s \mapsto s_0$

$t \mapsto p_0 \times s_0 - 48$

$pc \mapsto \text{error}$

Path constraint

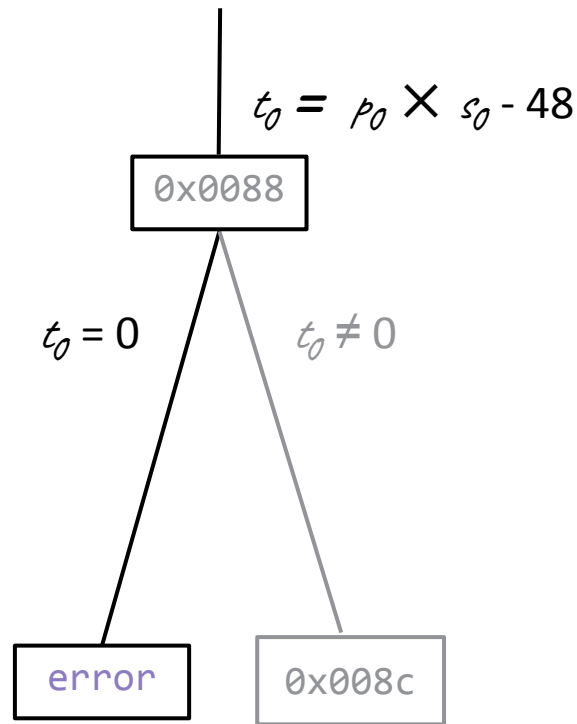
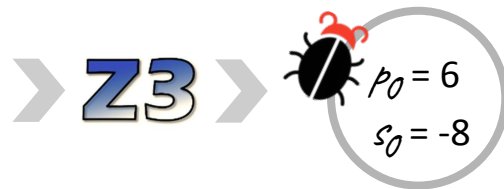
$t_0 = 0$

Can **error** be reached?

=> Query SMT-Solver

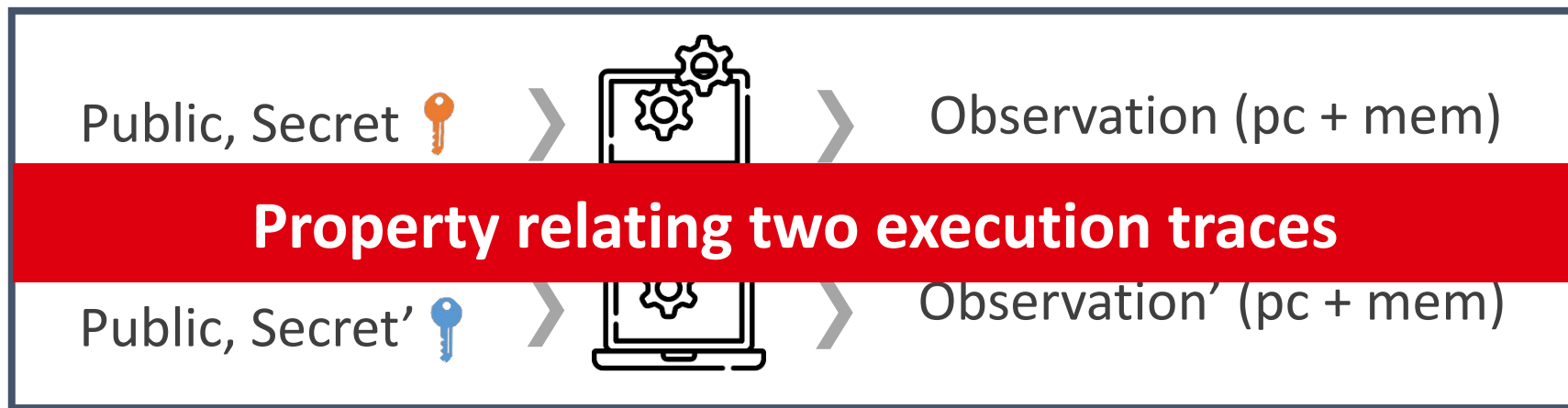
$t_0 = p_0 \times s_0 - 48$

$\wedge t_0 = 0$ is SAT?



Safety vs. 2-Hypersafety

*2 executions that only differ in their secret input
must be indistinguishable to an observer*



Secure Information Flow by Self-Composition*

Gilles Barthe¹ Pedro R. D'Argenio²
Tamara Rezk (corresponding author)³

Key idea: Turn a **2-hypersafety** property of a program **P**
to a **safety** property of a self-composed program **P;P'**

Can re-use verification techniques/tools for safety!

Symbolic Execution for CT

```
0x0080: mul  t, p, s  
0x0084: add  t, t, 48  
0x0088: beqz t error  
0x008c: div  t, s, t  
0x0090: [...]
```

Is program CT?
= Can branch differ in 2 executions?

(1) SE

$$t_0 = p_0 \times s_0 - 48$$

Symbolic Execution for CT

```
0x0080: mul  t, p, s
0x0084: add  t, t, 48
0x0088: beqz t error
0x008c: div  t, s, t
0x0090: [...]
```

Is program CT?

= Can branch differ in 2 executions?

(1) SE

$$t_0 = p_0 \times s_0 - 48$$

(2) Self-composition

$$p_0 = p'_0 \wedge t_0 = p_0 \times s_0 - 48 \wedge t'_0 = p'_0 \times s'_0 - 48 \wedge t_0 \neq t'_0$$

= public

Models 2 executions

Can branch differ?

Symbolic Execution for CT

```
0x0080: mul  t, p, s
0x0084: add  t, t, 48
0x0088: beqz t error
0x008c: div  t, s, t
0x0090: [...]
```

Is program CT?

= Can branch differ in 2 executions?

(1) SE

$$t_0 = p_0 \times s_0 - 48$$

(2) Self-composition

$$p_0 = p'_0 \wedge \begin{matrix} t_0 = p_0 \times s_0 - 48 \\ t'_0 = p'_0 \times s'_0 - 48 \end{matrix} \wedge t_0 \neq t'_0$$

= public

Models 2 executions

Can branch differ?

(3) Solver?



Symbolic Execution for CT

```
0x0080: mul  t, p, s
0x0084: add  t, t, 48
0x0088: beqz t error
0x008c: div  t, s, t
0x0090: [...]
```

Is program CT?

= Can branch differ in 2 executions?

(1) SE

$$t_0 = p_0 \times s_0 - 48$$

(2) Self-composition

$$p_0 = p'_0 \wedge \begin{matrix} t_0 = p_0 \times s_0 - 48 \\ t'_0 = p'_0 \times s'_0 - 48 \end{matrix} \wedge t_0 \neq t'_0$$

= public

Models 2 executions

Can branch differ?

(3) Solver?

$$\begin{matrix} p_0 = 6 & p'_0 = 6 \\ s_0 = 0 & s'_0 = -8 \end{matrix}$$



Beyond Self-Composition: Optimization for SE

Relational Symbolic Execution

Gian Pietro Farina^{*1}, Stephen Chong^{†2} and Marco Gaboardi^{‡1}

¹University at Buffalo, SUNY

²Harvard University

- 2 execution in 1 SE instance
- Maximize sharing
- Spare queries

BINSEC/REL: Efficient Relational Symbolic Execution for Constant-Time at Binary-Level

Lesly-Ann Daniel^{*}, Sébastien Bardin^{*}, Tamara Rezk[†]

^{*} CEA, List, Université Paris-Saclay, France

[†] INRIA Sophia-Antipolis, INDES Project, France

lesly-ann.daniel@cea.fr, sebastien.bardin@cea.fr, tamara.rezk@inria.fr

- RelSE for CT
- Optimization for binary-level

And concretely?



Binsec/Rel

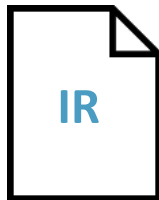
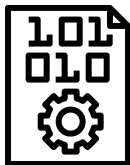
<https://binsec.github.io/>

Binary

X86-32 / 64

RISC-V 32

ARMv7/AARCH64/AMD64



Analysis

SE/RelSE



SMT-Solver

Boolector

Bitwuzla

z3, cvc4, yices



Configuration

Concretize esp, .data,
canaries, ...

Libc stubs



Helpers

Loader for ELF/PE

Build & simplify formulas
[...]



CT-analysis of cryptographic primitives

Preservation of constant-time by compilers

11 compiler versions

- 5 versions of clang for x86
- 5 versions of gcc for x86
- 1 version of gcc for ARM

Optimization setups

- Optimization level O1 ... O3
- Individual optimizations
 - X86-cmov-converter, if-conversion

Programs

- Analyze 34 small programs
- Total: 4148 binaries

Compile
&
Analyze with Binsec/Rel

LLVM ≠ Binary

```
int sort2(int *out2, int *in2) {  
    signed char c;  
    c = (in2[0] < in2[1]) - 1;  
    out2[0] = (~c & in2[0]) | (c & in2[1]);  
    out2[1] = (~c & in2[1]) | (c & in2[0]);  
    return (in2[0] < in2[1]);  
}
```

Source

```
; Function Attrs: nounwind  
define i32 @sort2(i32* nocapture %out2, i32* nocapture readonly %in2) {  
    %1 = load i32* %in2, align 4, !tbaa !1  
    %2 = getelementptr inbounds i32* %in2, i32 1  
    %3 = load i32* %2, align 4, !tbaa !1  
    %4 = select i1 %not., i32 %3, i32 %1  
    store i32 %4, i32* %out2, align 4, !tbaa !1  
    %5 = load i32* %2, align 4, !tbaa !1  
    %6 = load i32* %1, align 4, !tbaa !1  
    %7 = select i1 %not., i32 %6, i32 %5  
    %8 = getelementptr inbounds i32* %out2, i32 1  
    store i32 %7, i32* %8, align 4, !tbaa !1  
    %9 = load i32* %in2, align 4, !tbaa !1  
    %10 = load i32* %2, align 4, !tbaa !1  
    %11 = icmp slt i32 %9, %10  
    %12 = zext i1 %11 to i32  
    ret i32 %12  
}
```

LLVM-IR

Backend passes can still
introduce violations!

```
public sort2  
sort2 proc near  
  
arg_0= dword ptr 4  
arg_4= dword ptr 8  
  
push    esi  
mov     eax, [esp+4+arg_4]  
mov     edx, [eax]  
mov     esi, [eax+4]  
lea     ecx, [eax+4]  
cmp     edx, esi  
jge     short loc_80483B3
```

Binary

```
mov     esi, edx
```

```
loc_80483B3:  
mov     edx, [esp+4+arg_0]  
mov     [edx], esi  
mov     esi, eax  
jge     short loc_80483BF
```

```
mov     esi, ecx
```

53

```
loc_80483BF:  
mov     ecx, [esi]  
mov     [edx+4], ecx  
mov     ecx, [eax]  
cmp     ecx, [eax+4]  
setl    al  
movzx   eax, al  
pop     esi  
retn  
sort2 endp
```

Clang adds secret dependent memory access

```
1 void sort2(i32* out, i32* in) {  
2     a0 = load in[0]  
3     a1 = load in[1]  
4     a = select (a0 < a1) a0 a1  
5     store a out[0]  
6     b1 = load in[1]  
7     b0 = load in[0]  
8     b = select (a0 < a1) b1 b0  
9     store b out[1] }
```


LLVM-IR

```
1 sort2:  
2     esi := load (in+0)  
3     edi := load (in+4)  
4     cmp esi edi  
5     edi := cmovle esi  
6     store (out+0) edi  
7     ecx := in+0  
8     edx := in+4  
9     edx := cmovge ecx  
10    ecx := load edx  
11    store (out+4) ecx
```



clang-9 -m32 -O3 -march=i686

Summary

- **Constant-Time** = de facto standard against microarchitectural SCA
- We can formalize CT as a **2-hypersafety**
- There are tools to **verify crypto** primitives / find bugs
- We can **find cool bugs** introduced by compilers  **Binsec/Rel**

LLVM analysis is not sufficient!



But constant-time is not enough!

CNN BUSINESS Markets Tech Media Success Video

Major chip flaws affect billions of devices

by Selena Larson @CNNTech

January 4, 2018: 9:44 AM ET

BBC

Home News Sport Business Innovation Culture Arts Travel Earth Audio Video Live

'Foreshadow' attack affects Intel chips

15 August 2018

Share Save

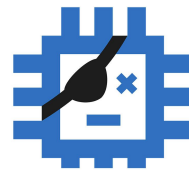
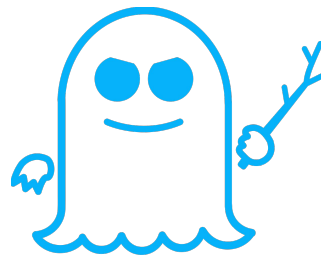
Dave Lee
North America technology reporter

Spectre flaws continue to haunt Intel and AMD as researchers find fresh attack method

The indirect branch predictor barrier is less of a barrier than hoped

Thomas Claburn

Fri 18 Oct 2024 // 14:01 UTC

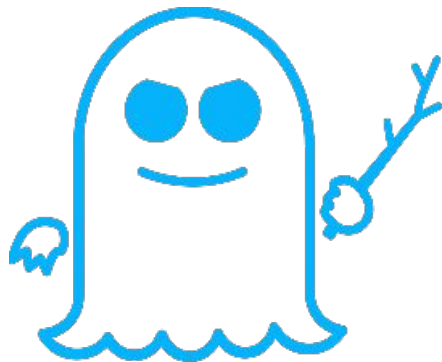


GoFetch



Constant-time is vulnerable to Spectre

```
char array[len]
char mysecret
1:  if (idx < len)
2:      x = array[idx]
3:      load(x)
```



Constant-time is vulnerable to Spectre

```
char array[len]
char mysecret
1:  if (idx < len)
2:      x = array[idx]
3:      load(x)
```

Predict condition true



Consider `idx = len`

Constant-time is vulnerable to Spectre

```
char array[len]
char mysecret
1:  if (idx < len)
2:      x = array[idx]
3:      load(x)
```

Consider `idx = len`

Predict condition true

`x = mysecret`



Constant-time is vulnerable to Spectre

```
char array[len]
char mysecret
1:  if (idx < len)
2:      x = array[idx]
3:      load(x)
```

Consider `idx = len`

Predict condition true



`x = mysecret`

Leak `mysecret` to
microarchitecture!



Mitigate Spectre






Part 1: How to fix existing software?



Fences to block speculative execution

```
char array[len]
char mysecret
1:  if (idx < len)
2:      x = array[idx]
3:      fence
4:      load(x)
```

- Branch is mispredicted to true 
- fence stalls until branch is resolved
- Rollback before **leak**(myscret)



Speculative Constant-Time (SCT)

Constant-Time Foundations for the New Spectre Era

Sunjay Cauligi[†] Craig Disselkoen[†] Klaus v. Gleissenthall[†]
Dean Tullsen[†] Deian Stefan[†] Tamara Rezk^{*} Gilles Barthe^{♦♦}

[†]UC San Diego, USA ^{*}INRIA Sophia Antipolis, France

[♦]MPI for Security and Privacy, Germany ^{♦♦}IMDEA Software Institute, Spain

Idea: Security in the constant-time observation mode
on a *speculative semantics*

Many flavors of microarchitectural semantics / ways to define security (see [1])

[1] Cauligi, S., Disselkoen, C., Moghimi, D., Barthe, G., & Stefan, D. (2022, May). SoK: Practical foundations for software Spectre defenses. *SP'22*

Why is that hard?

Problem. Microarchitectural semantics with **predictions** and **out-of-order** execution

Challenge. Microarchitectural features are complex, often undocumented

Goals. Find suitable **abstraction** to reason about Spectre

- Capture all variants of Spectre
- Keep it simple

Modelling speculative semantics

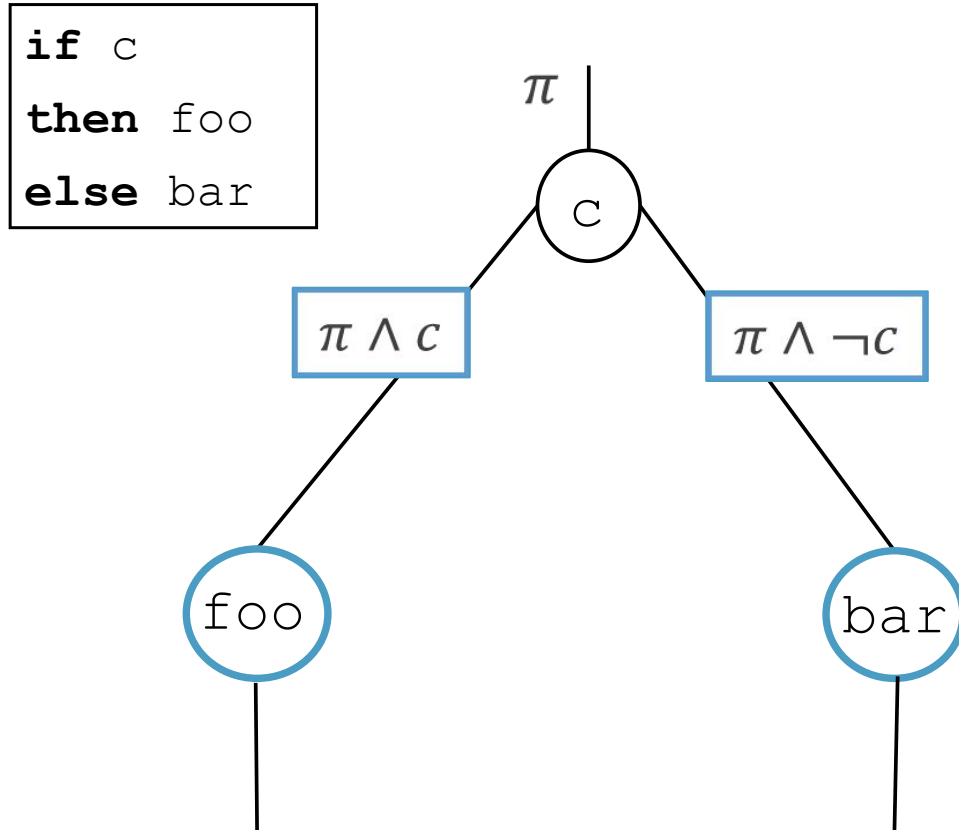
Litmus tests (328 instructions):

- Sequential semantics
→ **14 paths**
- Speculative semantics
→ **37M paths**

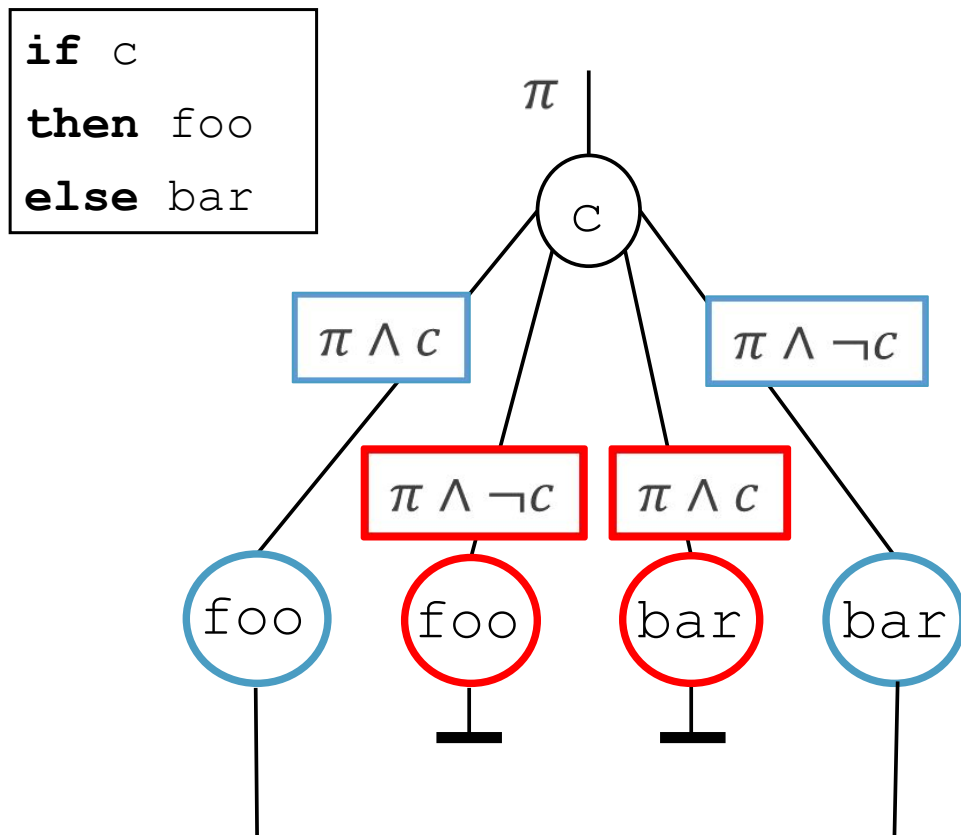


*Modelling all transient paths **explicitly** is intractable
We need to be smarter*

RelSE for architectural semantics



RelSE for Spectre-PHT (naive)

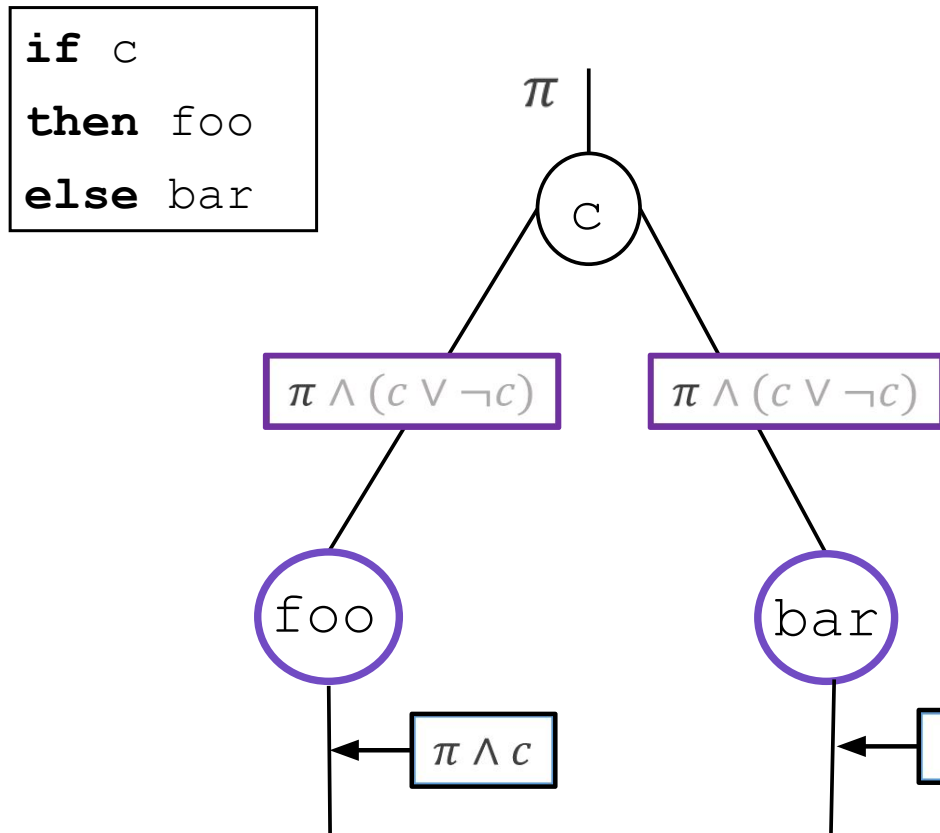


Fork into 4 paths:

- 2 *sequential paths*
- + 2 *extra transient paths*
- and verify constant-time

Wang, G., Chattopadhyay, S., Biswas, A. K., Mitra, T., & Roychoudhury, A. (2020). KLEESpectre: Detecting information leakage through speculative cache attacks via symbolic execution. *ACM TOSEM*

RelSE for Spectre-PHT (but let's be smarter)



Fork into 2 paths:

- 2 *speculative paths*
→ **seq** OR **transient**
- invalidate transient by adding constraint

Daniel, Lesly-Ann, Sébastien Bardin, and Tamara Rezk. "Hunting the Haunter: Efficient Relational Symbolic Execution for Spectre with Haunted RelSE." *NDSS'21*

Experimental evaluation

Benchmark

Litmus tests

Cryptographic primitives:

- tea
- donna
- Libsodium secretbox
- OpenSSL ssl3-digest-record
- OpenSSL mee-cdc-decrypt

Results

Litmus tests

- Paths: 1546 → 370
- Time: 3h → 15s

Libsodium + OpenSSL

- Coverage: 2273 → 8634

Total

- Timeouts: 5 → 1

And concretely?

- Find gadgets in crypto [1,2]
- Find attacks combining Spectre variants [2,3]
- Insert Spectre protections smartly [4,5]
- Type system to protect crypto against Spectre [5]
- Find gadgets in the Linux kernel [6]

[1] Cauligi, Sunjay, et al. "Constant-time foundations for the new spectre era." *PLDI'20*

[2] Daniel, Lesly-Ann, Sébastien Bardin, and Tamara Rezk. "Hunting the haunter-efficient relational symbolic execution for spectre with haunted relse." *NDSS'21*

[3] Fabian, Xaver, Marco Guarnieri, and Marco Patrignani. "Automatic Detection of Speculative Execution Combinations." *CCS'22*

[4] Vassena, Marco, et al. "Automatically eliminating speculative leaks from cryptographic code with blade." *POPL'21*

[5] Shivakumar, Basavesh Ammanaghatta, et al. "Typing High-Speed Cryptography against Spectre v1." *SP'23*

[6] Johannesmeyer, Brian, et al. "Kasper: scanning for generalized transient execution gadgets in the linux kernel." *NDSS'22*

Mitigate Spectre

Speculative constant-time:

- That's hard!
- New speculation mechanisms?



Speculative constant-time:

- That's hard!
- New speculation mechanisms?

Part 2: We want security for CT code Hardware can help

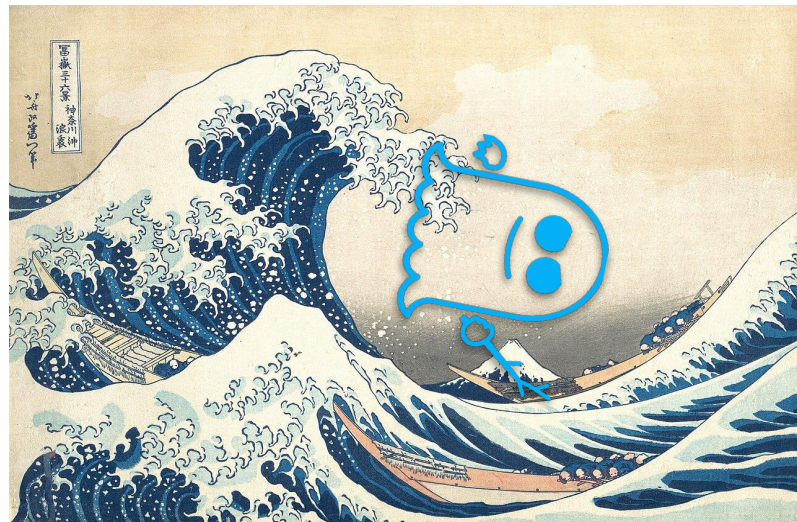


ProSpeCT

Provably Secure Speculation for the Constant-Time Policy

Lesly-Ann Daniel, Marton Boggar, Job Noorman,
Sébastien Bardin, Tamara Rezk, Frank Piessens

KU Leuven, Inria, CEA



USENIX'23

We need Secure Speculation for Constant-Time!



Developers should not care about speculations



Hardware shall not speculatively leak secrets



But still be efficient and enable **speculation**

Hardware Secrecy Tracking

Software side

- Label secrets
- Constant-time program



Hardware side

- Track security labels
- Secrets do not speculatively flow to unsafe instructions

ConTEXT: A Generic Approach for Mitigating Spectre

Michael Schwarz¹, Moritz Lipp¹, Claudio Canella¹
¹Graz University of Technology

SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks

Jacob Ernst

Farzad Farshchi
University of Kansas

Heechul Yun
University of Kansas

Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy

Rutvik Choudhary
UIUC, USA

Christopher W. Fletcher
UIUC, USA

Jiyong Yu
UIUC, USA

Adam Morrison
Tel Aviv University, Israel

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
1:  if (idx < len)
2:      x = array[idx]
3:      load(x)
```

Consider `idx = len`

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
1:  if (idx < len)
2:      x = array[idx]
3:      load(x)
```

Developer marks secrets



Consider `idx = len`

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
1: if (idx < len)
2:     x = array[idx]
3:     load(x)
```

Developer marks secrets

Speculative execution



Consider `idx = len`

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
1: if (idx < len)
2:     x = array[idx]
3:     load(x)
```

Developer marks secrets

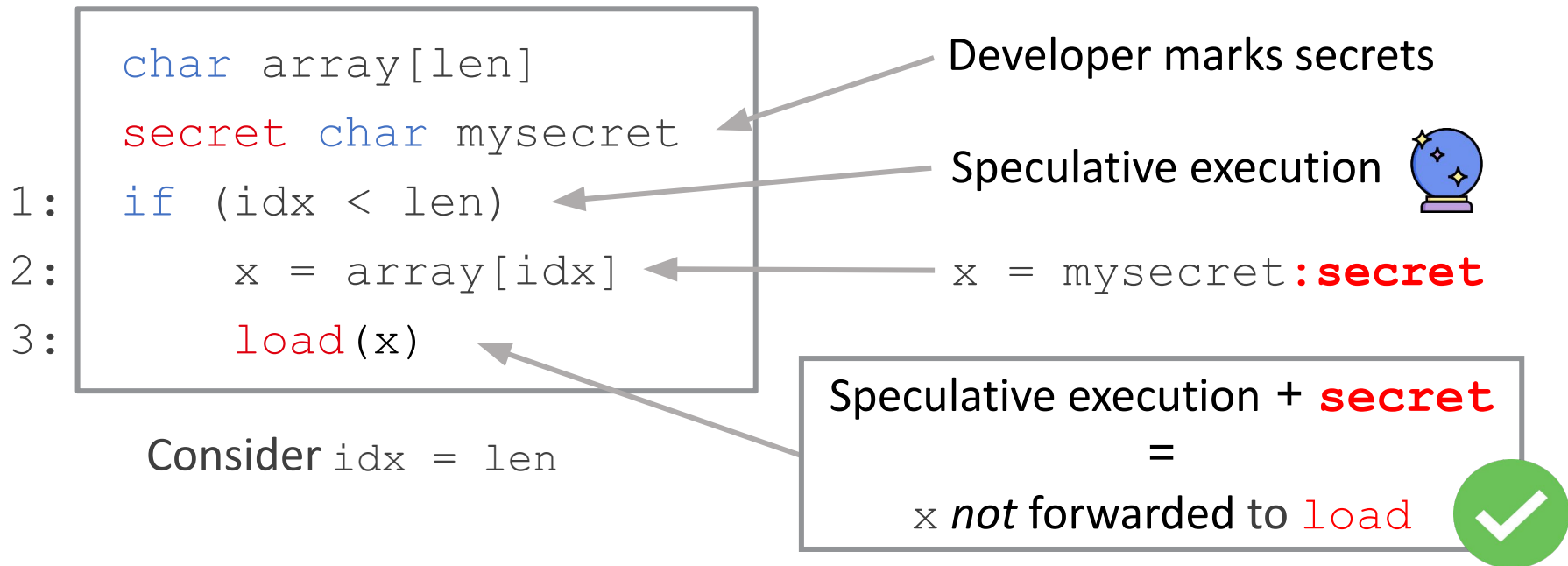
Speculative execution



$x = \text{mysecret} : \text{secret}$

Consider $\text{idx} = \text{len}$

Illustration with Spectre-v1



How do I know that my defense works?



How do I know that my defense works?

YOU BUILT A
HARDWARE DEFENSE?

Hardware-Software Contracts for Secure Speculation

Marco Guarnieri*, Boris Köpf†, Jan Reineke‡, and Pepe Vila*

*IMDEA Software Institute

†Microsoft Research

‡Saarland University

THAT'S CUTE...

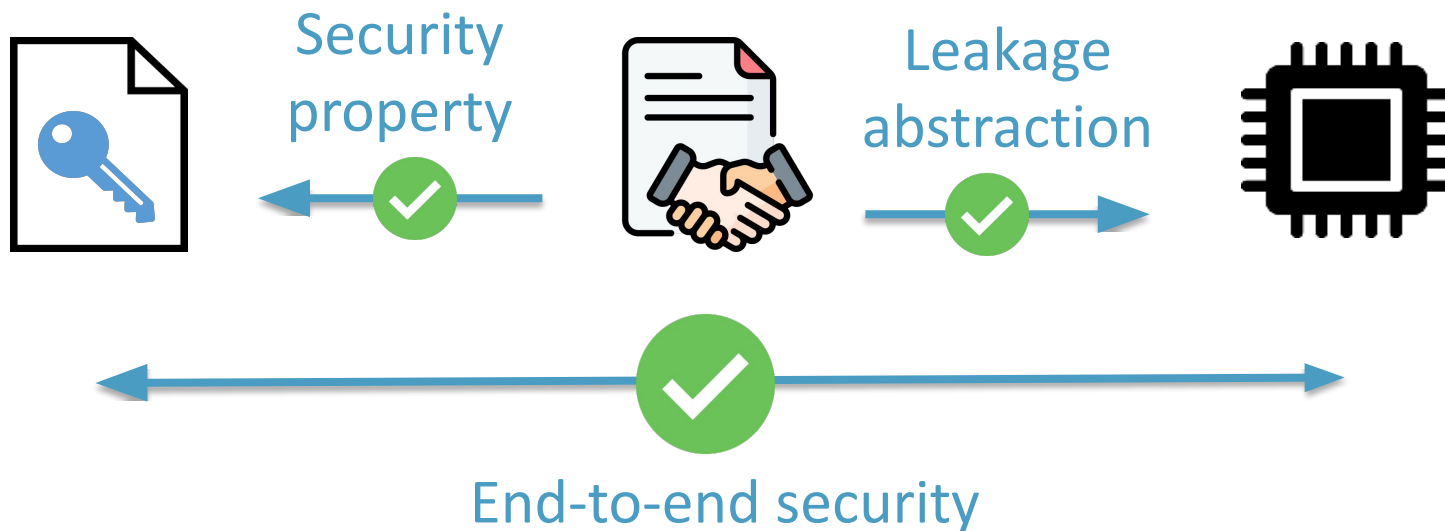
Hardware-Software Contracts for Secure Speculation

Marco Guarnieri^{*}, Boris Köpf[†], Jan Reineke[‡], and Pepe Vila^{*}

^{*}*IMDEA Software Institute*

[†]*Microsoft Research*

[‡]*Saarland University*



ProSpeCT: Generic formal processor model for HST

Semantics of **generic out-of-order speculative** processor with HST

- Abstract microarchitectural context
- Functions *update*, *predict*, *next*

All public values are leaked / influence predictions

- Captures all known variants of Spectre
- And futuristic mechanisms Load Value Prediction



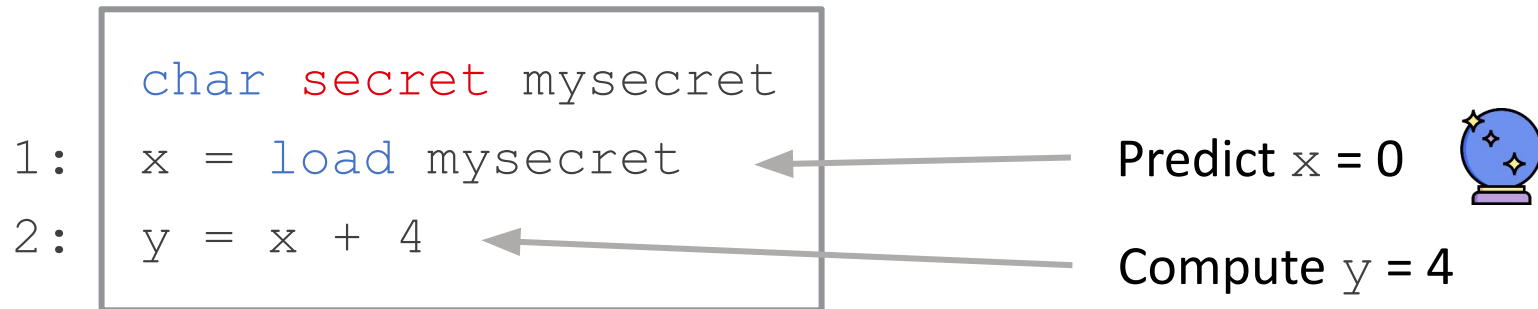
Security proof

Constant-time programs (ISA semantics)
do not leak secrets (microarchitectural semantics)

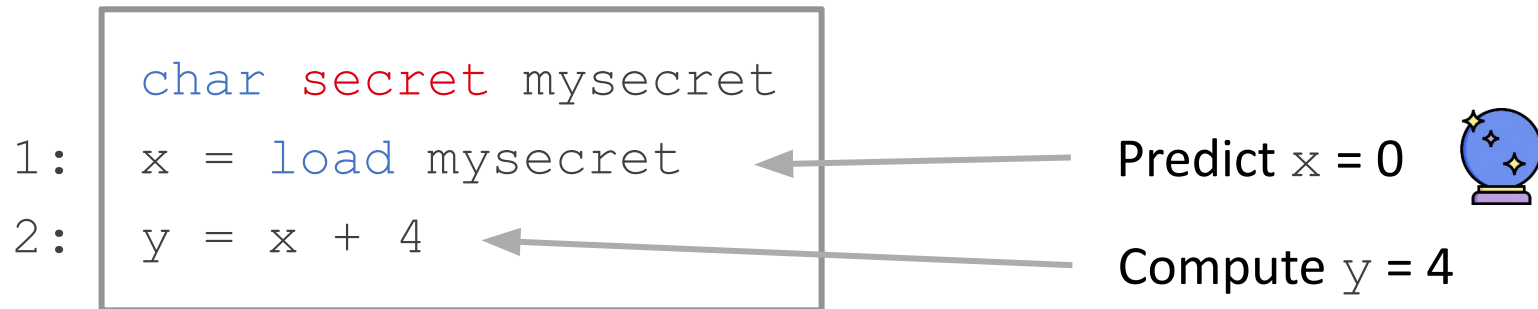
Load Prediction: Rollback correct executions?

```
char secret mysecret  
1: x = load mysecret  
2: y = x + 4
```

Load Prediction: Rollback correct executions?



Load Prediction: Rollback correct executions?

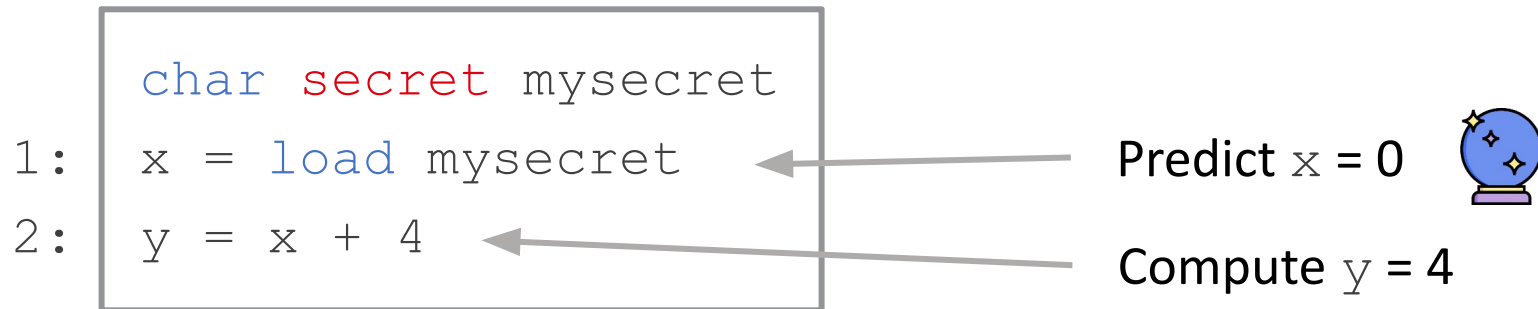


Resolve prediction:

- if `mysecret == 0`: **Commit** and continue to line 3
- if `mysecret != 0`: **Rollback** to line 1

That leaks!

Load Prediction: Rollback correct executions?



Resolve prediction:

- if `mysecret = 0`: **Rollback** to line 1
- if `mysecret != 0`: **Rollback** to line 1

Always rollback when
actual value is secret

Proteus: An Extensible RISC-V Core for Hardware Extensions

(RISC-V Summit '23)

Marton Bogнар, Job Noorman, Frank Piessens



A modular textbook processor to study HW extensions

- **In/Out-of order** pipelines
- **Optimizations**: branch predictors, cache, prefetchers, ...
- **Configurable**: #exec units, ROB size, ...
- **Extensible**: plugin system
- **SpinalHDL** ☐ verilog ☐ FPGA / simulator



Implementation on Proteus and Evaluation



Performance overhead [1]

Speculation/Crypto	25/75	50/50	75/25	90/10
Precise (Key)	0%	0%	0%	0%
Conservative (All)	10%	25%	36%	45%

No overhead in SW for CT code
when secrets are precisely annotated

Hardware Cost:

Synthesized on FPGA

- LUTs: +17%
- Registers: +6%
- Critical path: +2%

[1] Jacob Fustos, Farzad Farshchi, and Heechul Yun. "SpectreGuard: An Efficient Data-Centric Defense Mechanism against Spectre Attacks". In: DAC. 2019

Conclusion



We need to move beyond CT!



Mitigating Spectre in software is hard and costly

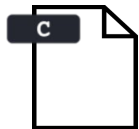


HW-SW co-designs can improve security & performance



My belief: HW-SW contracts are promising for end-to-end security

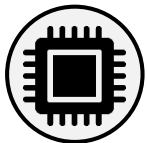
Many remaining challenges!



Software: PL support + parametric in leakage contract



New defenses: new attacks, emerging applications, platforms, etc.



Hardware verification: support defenses and scale existing techniques

Credit icons: <https://www.flaticon.com/>