Microarchitectural side channels: from hardware to software

Clémentine Maurice, CNRS, CRIStAL

March 13, 2025—ARCHI 2025

Attacks on micro-architecture

- hardware usually modeled as an abstract layer behaving correctly

Attacks on micro-architecture

- hardware usually modeled as an abstract layer behaving correctly, but possible attacks

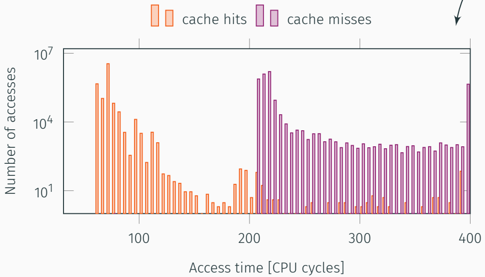
Attacks on micro-architecture

- hardware usually modeled as an abstract layer behaving correctly, but possible attacks
    - faults: bypassing software protections by causing hardware errors
    - side channels: observing side effects of hardware on computations

Attacks on micro-architecture

- hardware usually modeled as an abstract layer behaving correctly, but possible attacks
    - faults: bypassing software protections by causing hardware errors
    - side channels: observing side effects of hardware on computations

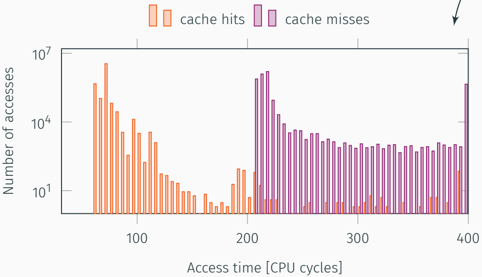
identification



Attacks on micro-architecture

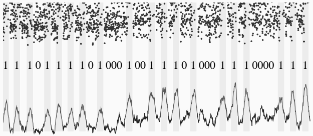
- hardware usually modeled as an abstract layer behaving correctly, but possible attacks
  - faults: bypassing software protections by causing hardware errors
  - side channels: observing side effects of hardware on computations

identification



→

attack

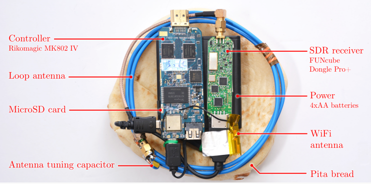


- retrieving secret keys, keystroke timings
- bypassing OS security (ASLR)

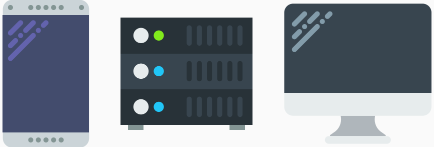
Attacker model

Hardware-based attacks
a.k.a physical attacks

Software-based attacks
a.k.a micro-architectural attacks



vs

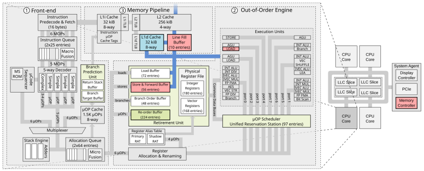


Physical access to hardware
→ embedded devices

Co-located or remote attacker
→ complex systems

Micro-architectural side-channel attacks: Two faces of the same coin

Hardware



&

Implementation

```
Algorithm 1: Square-and-multiply exponentiation
Input: base $b$, exponent $e$, modulus $n$
Output: $b^e \mod n$
$X \leftarrow 1$
for $i \leftarrow bitlen(e)$ downto 0 do
    $X \leftarrow \text{multiply}(X, X)$
    if $e_i = 1$ then
        $X \leftarrow \text{multiply}(X, b)$
    end
end
return $X$
```

Research questions

1. Which hardware component is vulnerable?
2. Which software implementation is vulnerable?

Outline

- **Part 1** Small example: Flush+Reload on GnuPG v 1.4.13
- **Part 2** Which hardware component is vulnerable?
- **Part 3** Which software implementation is vulnerable?

Part 1 Small example: Flush+Reload on GnuPG v 1.4.13

GnuPG 1.4.13 RSA square-and-multiply exponentiation

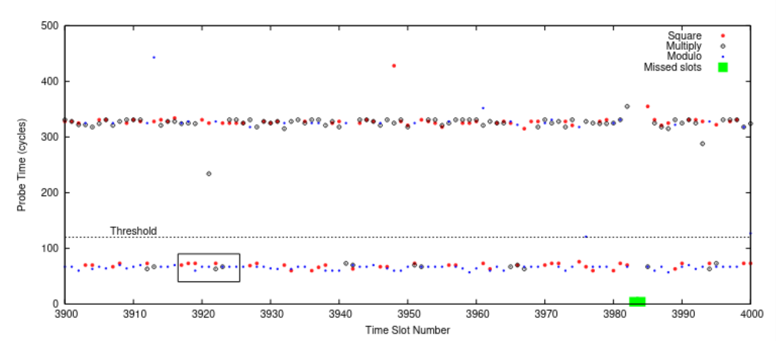
GnuPG version 1.4.13 (2013)

Algorithm 1: GnuPG 1.4.13 Square-and-multiply exponentiation

```
Input: base $c$, exponent $d$, modulus $n$
Output: $c^d \mod n$
$X \leftarrow 1$
for $i \leftarrow bitlen(d)$ downto 0 do
    $X \leftarrow \mathrm{square}(X)$
    $X \leftarrow X \bmod n$
    if $d_i = 1$ then
        $X \leftarrow \mathrm{multiply}(X, c)$
        $X \leftarrow X \bmod n$
    end
end
return $X$
```

Attacking GnuPG 1.4.13 RSA exponentiation

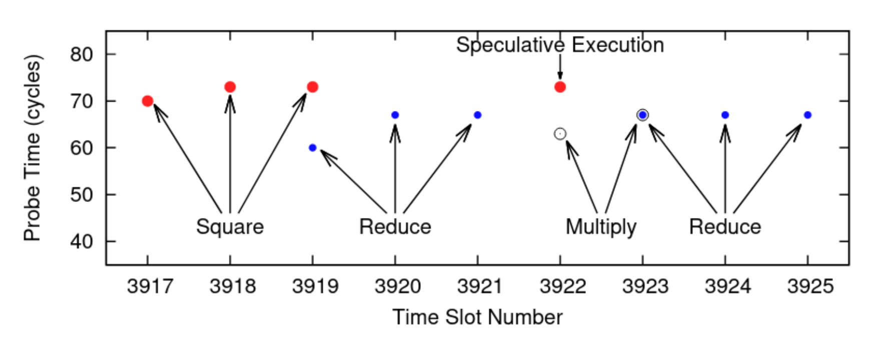
- monitor the square and multiply functions with Flush+Reload to recover the bits of the secret exponent



Y. Yarom and K. Falkner. "Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack". In: *USENIX Security Symposium*. 2014.

Attacking GnuPG 1.4.13 RSA exponentiation

- monitor the square and multiply functions with Flush+Reload to recover the bits of the secret exponent

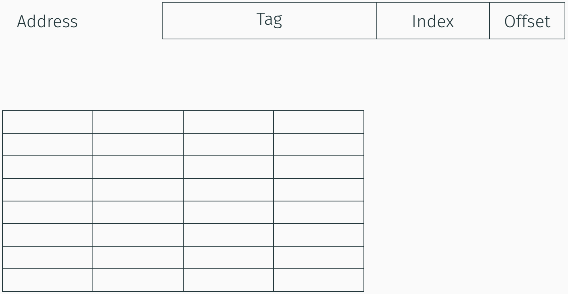


Y. Yarom and K. Falkner. “Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *USENIX Security Symposium*. 2014.

Part 2 Which hardware component is vulnerable?

Cache side-channel attacks

Set-associative caches

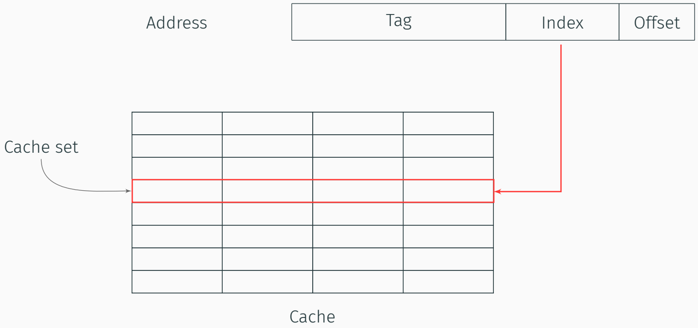


Address

Tag	Index	Offset

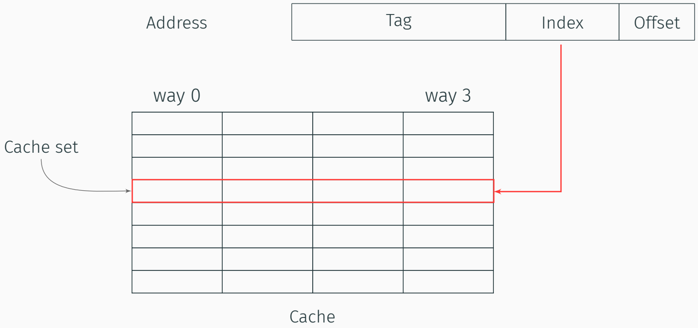
Cache

Set-associative caches



Data loaded in a specific set depending on its address

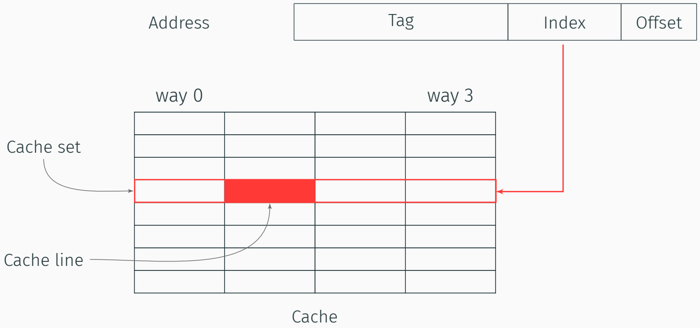
Set-associative caches



Data loaded in a specific set depending on its address

Several ways per set

Set-associative caches



Data loaded in a specific set depending on its address

Several ways per set

Cache line loaded in a specific way depending on the replacement policy

Cache attacks

- cache attacks → exploit timing differences of memory accesses

Cache attacks

- cache attacks → exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content

Cache attacks

- cache attacks → exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
    - not allowed to do so, e.g., across VMs

Cache attacks

- cache attacks → exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
    - not allowed to do so, e.g., across VMs
- side-channel attack: one malicious process spies on benign processes
    - e.g., steals crypto keys, spies on keystrokes

Timing attacks

How every timing attack works:

- learn timing of different corner cases
- later, we recognize these corner cases by timing only

Timing attacks

How every timing attack works:

- learn timing of different corner cases
- later, we recognize these corner cases by timing only
- here, corner cases: hits and misses

First step: building the histogram

1. build two cases: cache hits and cache misses
2. time each case many times (get rid of noise)

First step: building the histogram

1. build two cases: cache hits and cache misses
2. time each case many times (get rid of noise)
3. we have a histogram!

First step: building the histogram

1. build two cases: cache hits and cache misses
2. time each case many times (get rid of noise)
3. we have a histogram!
4. find a threshold to distinguish the two cases

Building the histogram: cache hits

Loop:

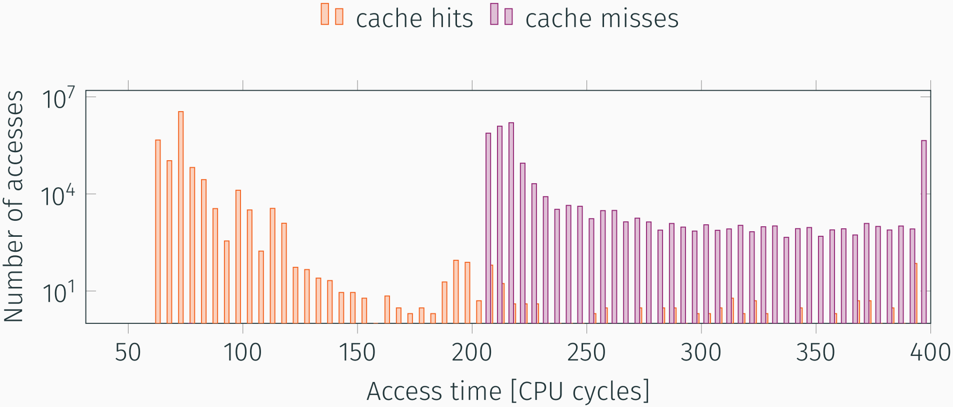
1. measure time
2. access variable (always cache hit)
3. measure time
4. update histogram with delta

Building the histogram: cache misses

Loop:

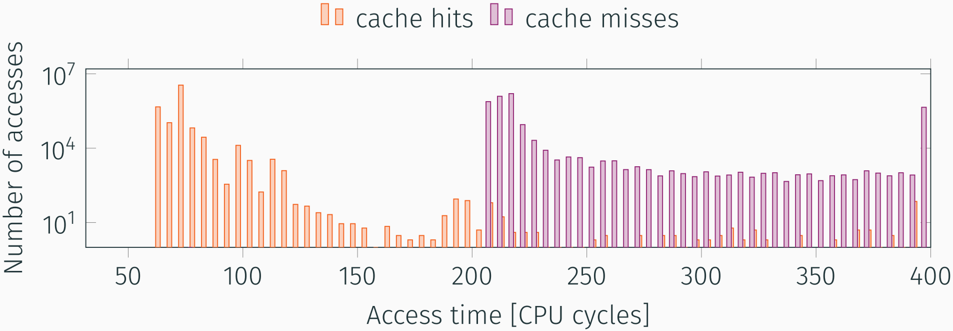
1. flush variable (`clflush` instruction)
2. measure time
3. access variable (always cache miss)
4. measure time
5. update histogram with delta

Timing differences



Finding the threshold

- as high as possible → most cache hits are below
- no cache miss below



How to measure time accurately? (1/3)

- very short timings
- `rdtsc` instruction: cycle-accurate timestamps

How to measure time accurately? (1/3)

- very short timings
- `rdtsc` instruction: cycle-accurate timestamps

```
[...]
rdtsc
function()
rdtsc
[...]
```

How to measure time accurately? (2/3)

- do you measure what you think you measure?

How to measure time accurately? (2/3)

- do you measure what you think you measure?
- out-of-order execution

How to measure time accurately? (2/3)

- do you measure what you think you measure?
- out-of-order execution → what is really executed

```
rdtsc
function()
[...]
rdtsc
```

```
rdtsc
[...]
rdtsc
function()
```

```
rdtsc
rdtsc
function()
[...]
```

How to measure time accurately? (3/3)

- use pseudo-serializing instruction `rdtscp` (recent CPUs)

How to measure time accurately? (3/3)

- use pseudo-serializing instruction **`rdtscp`** (recent CPUs)
- and/or use serializing instructions like **`cpuid`**

How to measure time accurately? (3/3)

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
- and/or use fences like `mfence`

How to measure time accurately? (3/3)

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
- and/or use fences like `mfence`

Intel, *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures White Paper*, December 2010.

Cache attacks techniques

- two (main) techniques
  1. Flush+Reload (Gullasch et al., Osvik et al., Yarom et al.)
  2. Prime+Probe (Percival, Osvik et al., Liu et al.)
- exploitable on x86 and ARM
- used for both covert channels and side-channel attacks
- many variants: Flush+Flush, Evict+Reload, Prime+Scope, Prime+Abort...

D. Gullasch, E. Bangerter, and S. Krenn. "Cache Games – Bringing Access-Based Cache Attacks on AES to Practice". In: *S&P'11*. 2011.

Y. Yarom and K. Falkner. "Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack". In: *USENIX Security Symposium*. 2014.

D. A. Osvik, A. Shamir, and E. Tromer. "Cache Attacks and Countermeasures: the Case of AES". In: *CT-RSA 2006*. 2006.

C. Percival. "Cache missing for fun and profit". In: *Proceedings of BSDCan*. 2005.

F. Liu et al. "Last-Level Cache Side-Channel Attacks are Practical". In: *S&P'15*. 2015.

Spatial and temporal resolution

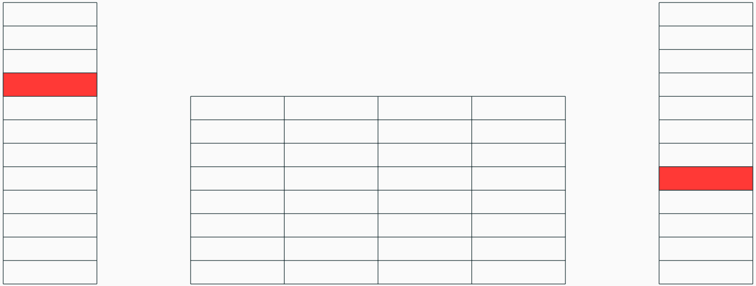
- spatial resolution: what can I monitor? A page? A set? A line?
  → a spatial resolution of a 4KB page means that you cannot distinguish two memory accesses within a 4KB page
- temporal resolution: how often can I perform a monitoring operation?
  → a temporal resolution of 1ms means that you cannot monitor more than one event every 1ms: if an event happens every 1μs, you can only capture 0.1% of events

Spatial and temporal resolution

- spatial resolution: what can I monitor? A page? A set? A line?
  → a spatial resolution of a 4KB page means that you cannot distinguish two memory accesses within a 4KB page
- temporal resolution: how often can I perform a monitoring operation?
  → a temporal resolution of 1ms means that you cannot monitor more than one event every 1ms: if an event happens every 1μs, you can only capture 0.1% of events

Both influence the type of attacks that you can perform: an attacker that can only monitor a 4KB page every minute obtains less information than an attacker that can monitor a cache line every 100ns.

Cache attack: Flush+Reload



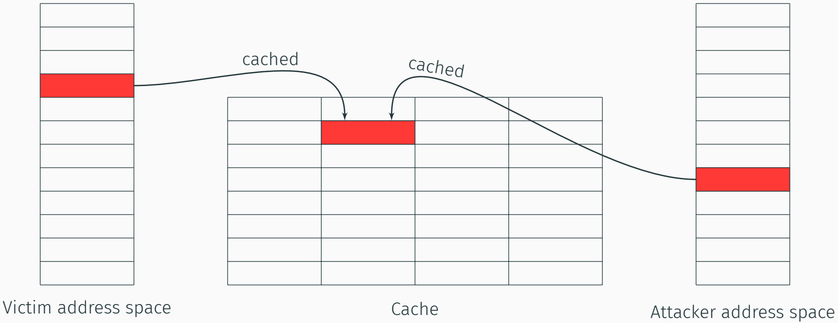
Victim address space

Cache

Attacker address space

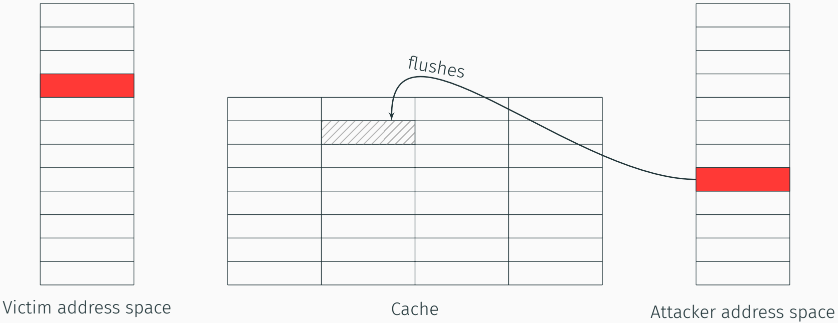
Step 1: Attacker maps shared library (shared memory, in cache)

Cache attack: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

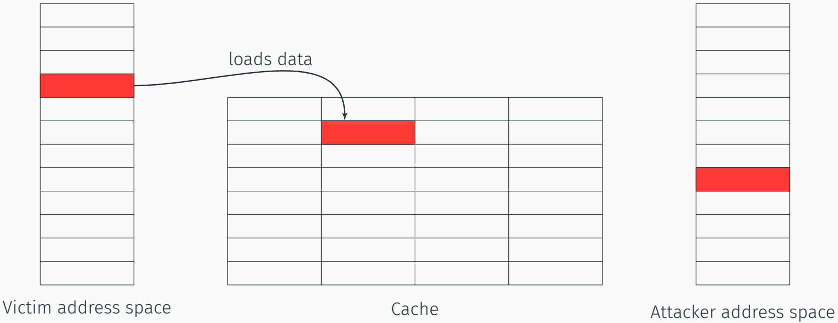
Cache attack: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Cache attack: Flush+Reload

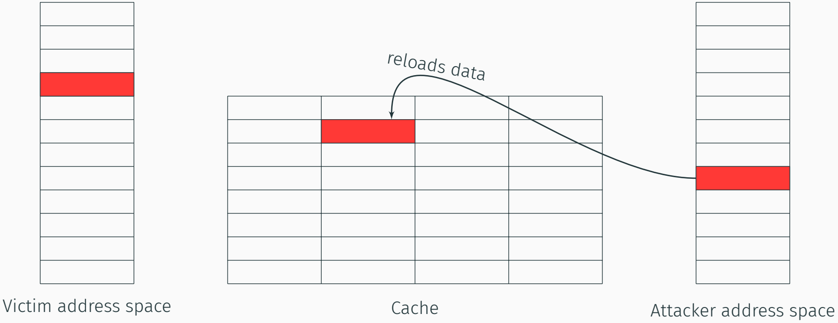


Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Cache attack: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Step 4: Attacker reloads the data

Flush+Reload: Applications

- cross-VM (memory-deduplication enabled) side channel attacks on cryptographic primitives:
  - RSA: 96.7% of secret key bits in a single signature
  - AES: full key recovery in 30000 dec. (a few seconds)
- attacks against pseudorandom number generators
- attacks against RSA key generation
- revival of Bleichenbacher attacks on TLS

Y. Yarom and K. Falkner. “Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: *USENIX Security Symposium*. 2014.

B. Gülmezoglu et al. “A Faster and More Realistic Flush+Reload Attack on AES”. In: *COSADE*. 2015.

S. Cohney et al. “Pseudorandom Black Swans: Cache Attacks on CTR_DRBG”. In: *S&P*. 2020.

A. C. Aldaya et al. “Cache-Timing Attacks on RSA Key Generation”. In: *TCHES* (2019).

E. Ronen et al. “The 9 Lives of Bleichenbacher’s CAT: New Cache ATtacks on TLS Implementations”. In: *S&P*. 2019.

Flush+Reload: Pros and cons

Pros

high spatial resolution: 1 line

high temporal resolution

Cons

restrictive

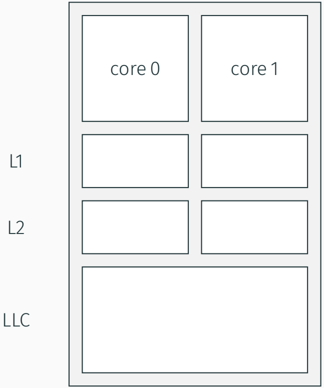
1. needs `clflush` instruction (not available e.g., on ARM-v7)
2. needs shared memory

What if there is **no shared memory**?

What if there is **no shared memory**?

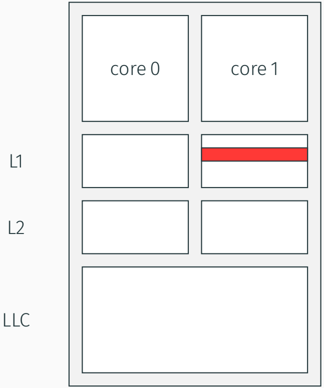
E.g., there is no memory deduplication and no accessible shared library

Inclusive property



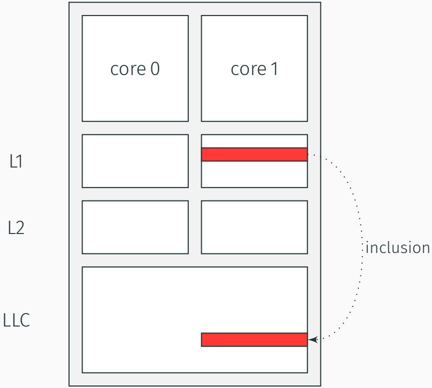
- inclusive LLC: superset of L1 and L2

Inclusive property



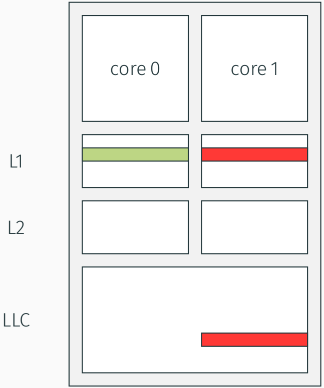
- inclusive LLC: superset of L1 and L2

Inclusive property



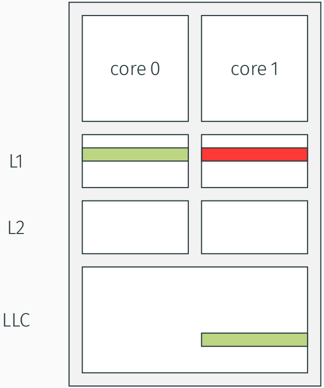
- inclusive LLC: superset of L1 and L2

Inclusive property



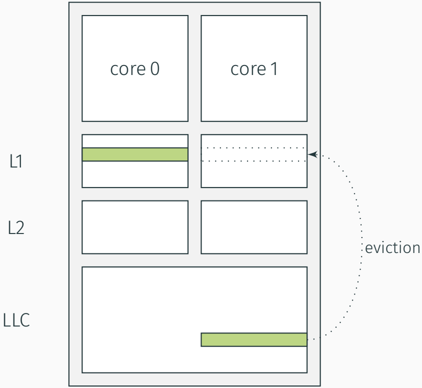
- inclusive LLC: superset of L1 and L2

Inclusive property



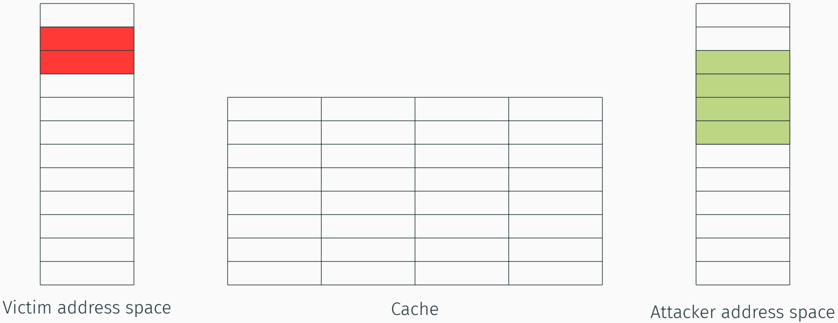
- inclusive LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2

Inclusive property

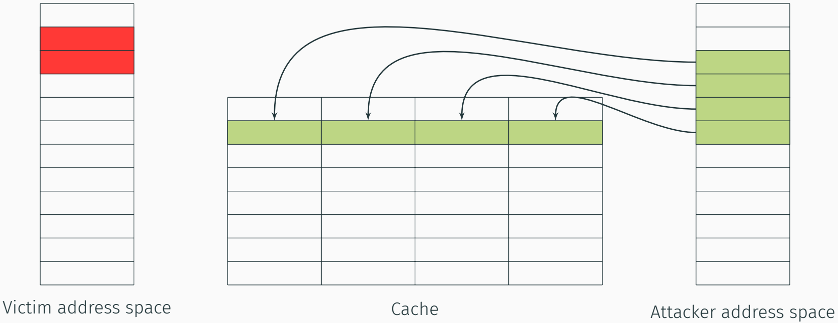


- inclusive LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2
- a core can evict lines in the private L1 of another core

Cache attacks: Prime+Probe

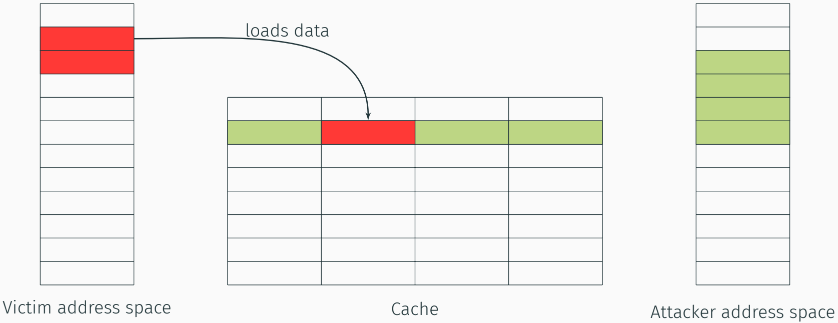


Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

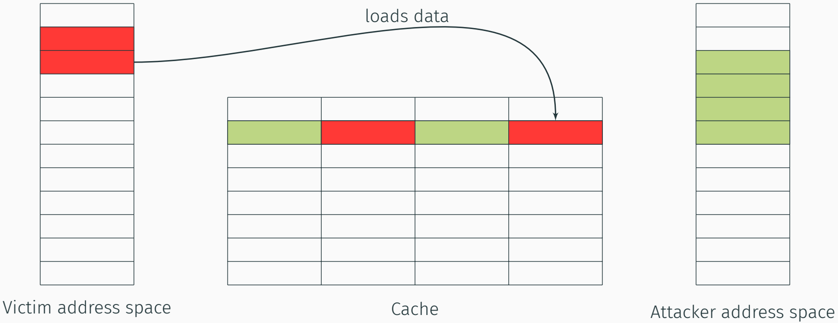
Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

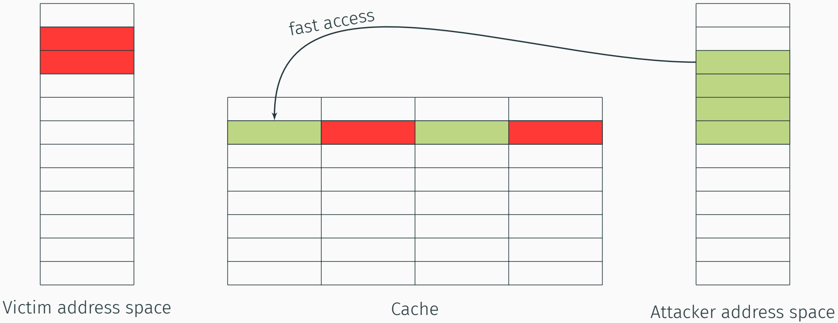
Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Cache attacks: Prime+Probe

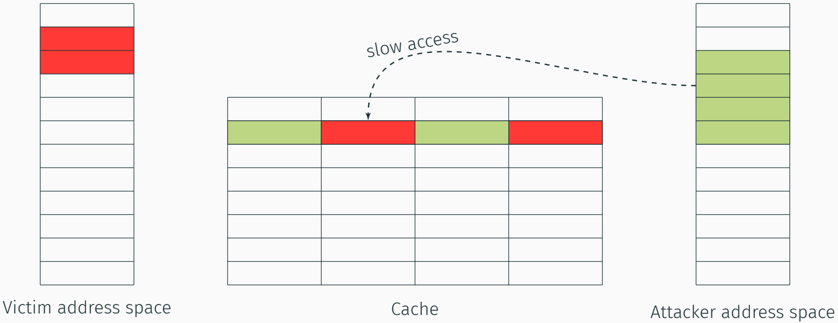


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Step 3: Attacker probes data to determine if set has been accessed

Cache attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Step 3: Attacker probes data to determine if set has been accessed

Prime+Probe: Applications

- cross-VM side channel attacks on crypto implementations:
  - El Gamal (sliding window): full key recovery in 12 min.
- tracking user behavior in the browser, in JavaScript
- covert channels between virtual machines in the cloud

F. Liu et al. "Last-Level Cache Side-Channel Attacks are Practical". In: *S&P'15*. 2015.

Y. Oren et al. "The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications". In: *CCS'15*. 2015.

C. Maurice et al. "Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud". In: *NDSS'17*. 2017.

Prime+Probe: Pros and cons

Pros

less restrictive

1. no need for `clflush`
2. no need for shared memory

Cons

- lower spatial resolution: 1 set
- lower temporal resolution: probe *n* addresses to evict 1 line
- prone to noise

Prime+Probe in practice

We need to evict caches lines without `clflush` or shared memory:

1. which addresses do we access to have congruent cache lines?
2. without any privilege?
3. and in which order do we access them?

P. Vila, B. Köpf, and J. F. Morales. “Theory and Practice of Finding Eviction Sets”. In: *S&P*. 2019.

C. Maurice et al. “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: *RAID'15*. 2015.

P. Vila et al. “CacheQuery: learning replacement policies from hardware caches”. In: *PLDI*. 2020.

Prime+Probe in practice

We need to evict caches lines without `clflush` or shared memory:

1. which addresses do we access to have congruent cache lines?
2. without any privilege?
3. and in which order do we access them?

We need:

1. an eviction set: addresses in the same set and same slice (issues #1 and #2)
2. an eviction strategy: the order in which we access the eviction set (issue #3)

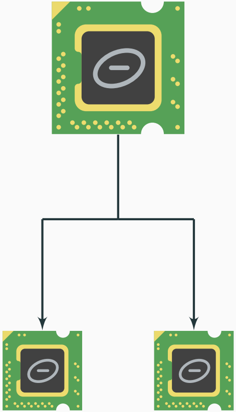
P. Vila, B. Köpf, and J. F. Morales. "Theory and Practice of Finding Eviction Sets". In: *S&P*. 2019.

C. Maurice et al. "Reverse Engineering Intel Complex Addressing Using Performance Counters". In: *RAID'15*. 2015.

P. Vila et al. "CacheQuery: learning replacement policies from hardware caches". In: *PLDI*. 2020.

Port contention side-channel attacks

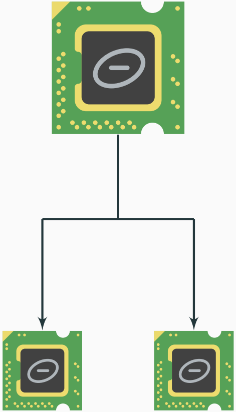
Background: Hyper-threading



Simultaneous computation technology of Intel.

- physical cores are shared between logical cores
- abstraction at the OS level

Background: Hyper-threading



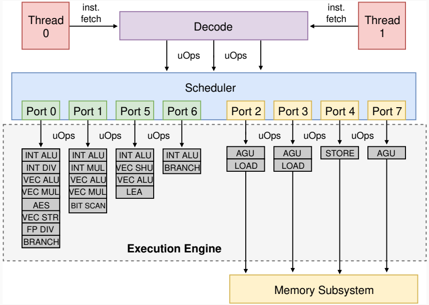
Simultaneous computation technology of Intel.

- physical cores are shared between logical cores
- abstraction at the OS level

→ hardware resources are shared between logical cores

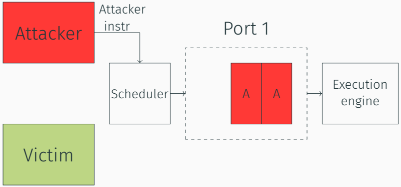
Background: Execution pipeline

- instructions are decomposed in uops to optimize Out-of-Order execution
- uops are dispatched to specialized execution units through CPU ports
- deterministic decomposition of instructions into uops



Port contention

No contention

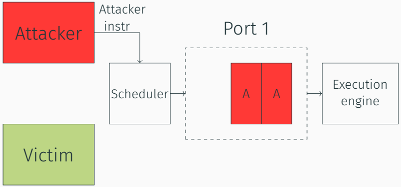


All attacker instructions are executed in a row
→ fast execution time

A. C. Aldaya et al. “Port Contention for Fun and Profit”. In: *S&P*. 2019.

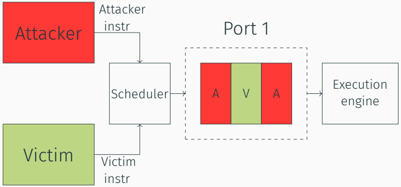
Port contention

No contention



All attacker instructions are executed in a row
→ fast execution time

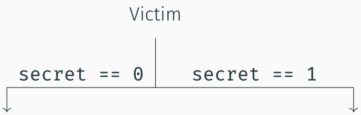
Contention



Victim instructions delay the attacker instructions
→ slow execution time

A. C. Aldaya et al. “Port Contention for Fun and Profit”. In: *S&P*. 2019.

Port contention side-channel attack



Victim

```
secret == 0
```

```
secret == 1
```

```
POPCNT %r8,%r8
POPCNT %r8,%r8
...
POPCNT %r8,%r8
POPCNT %r8,%r8
```

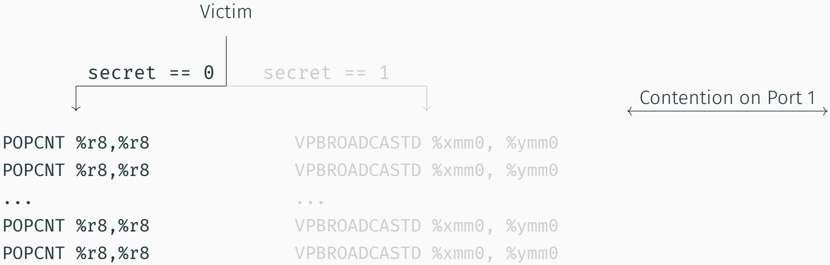
Contention on Port 1

```
VPBROADCASTD %xmm0, %ymm0
VPBROADCASTD %xmm0, %ymm0
...
VPBROADCASTD %xmm0, %ymm0
VPBROADCASTD %xmm0, %ymm0
```

Contention on Port 5

Monitors port usage

Port contention side-channel attack



Victim

`secret == 0`

`secret == 1`

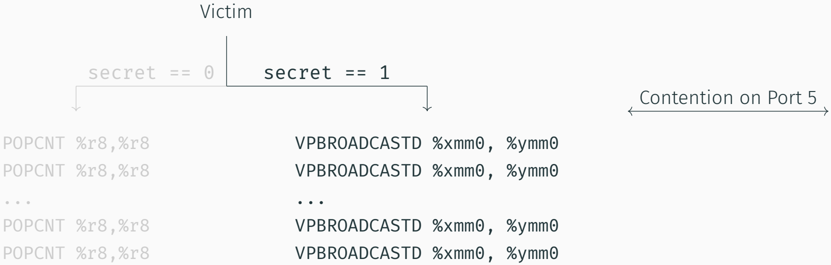
```
POPCNT %r8,%r8
POPCNT %r8,%r8
...
POPCNT %r8,%r8
POPCNT %r8,%r8
```

```
VPBROADCASTD %xmm0, %ymm0
VPBROADCASTD %xmm0, %ymm0
...
VPBROADCASTD %xmm0, %ymm0
VPBROADCASTD %xmm0, %ymm0
```

Contention on Port 1

Secret is 0!

Port contention side-channel attack



Victim

```
          secret == 0          secret == 1

POPCNT %r8,%r8             VPBROADCASTD %xmm0, %ymm0
POPCNT %r8,%r8             VPBROADCASTD %xmm0, %ymm0
...                        ...
POPCNT %r8,%r8             VPBROADCASTD %xmm0, %ymm0
POPCNT %r8,%r8             VPBROADCASTD %xmm0, %ymm0
```

Contention on Port 5

Secret is 1!

Port contention: applications

- end-to-end attack on a TLS server (OpenSSL 1.1.0h): recovers a P-384 ECDSA private key
  - → secret dependent on double-and-add operations of `ec_wNAF_mul` point multiplication
- SMoTherSpectre, a speculative code-reuse attack

A. C. Aldaya et al. “Port Contention for Fun and Profit”. In: *S&P*. 2019.

A. Bhattacharyya et al. “SMoTherSpectre: Exploiting Speculative Execution through Port Contention”. In: *CCS*. 2019.

Port contention: Pros and cons

Pros

- very high spatial resolution: 1 instruction!
- high temporal resolution
- more resistant to noise if processes do not share a physical core
- no offline phase of creating an eviction set

Cons

- restrictive: requires SMT enabled + co-location on the same physical core
- mapping from instructions to port can change from one generation to another

Conclusion: We are more or less doomed on the hardware side

Translation look-aside buffer

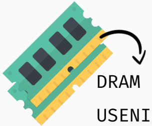
USENIX Sec'18

CPU Ports

S&P'19

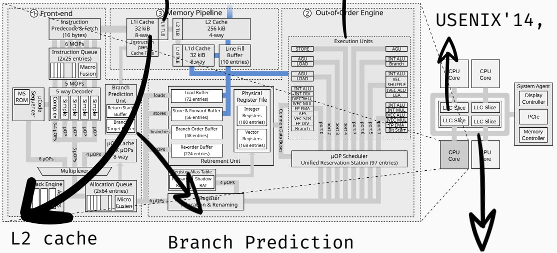
LLC attacks

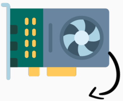
USENIX'14, S&P'15



DRAM

USENIX Sec'16





GPU

S&P'18

L1d, L1i, L2 cache

BSDCon'05, CT-RSA'06, ASIACCS'20

Branch Prediction

CT-RSA'07

Ring Interconnect

USENIX Sec'21, DIMVA'21

State of the art today: each component shared by two processes is a potential micro-architectural side-channel vector

Part 3 Which software implementation is vulnerable?

Side-channel vulnerabilities and constant-time programming

💣 Problem?

Side-channel vulnerability

Any branch or memory access that depends on a secret

Side-channel vulnerabilities and constant-time programming

💣 Problem?

💡 Solution!

Side-channel vulnerability

Any branch or memory access that depends on a secret

→

Constant-time programming

No branch or memory access depends on a secret!

Side-channel vulnerabilities and constant-time programming

💣 Problem?

💡 Solution!

Side-channel vulnerability

Any branch or memory access that depends on a secret

→

Constant-time programming

No branch or memory access depends on a secret!

That's easy, right?

Side-channel vulnerabilities and constant-time programming

Problem?

Side-channel vulnerability

Any branch or memory access that depends on a secret

→

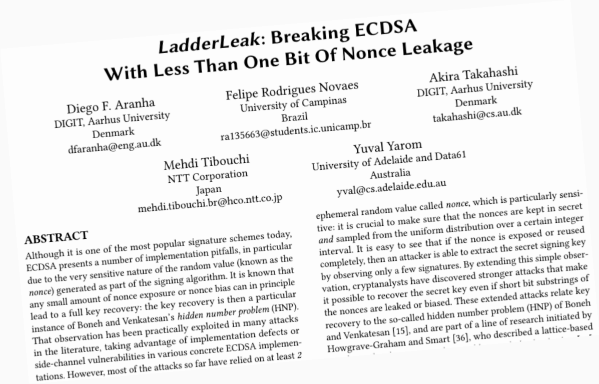
Solution!

Constant-time programming

No branch or memory access depends on a secret!

That's easy, right?... right?

So many attacks...



LadderLeak: Breaking ECDSA With Less Than One Bit Of Nonce Leakage

Diego F. Aranha
DIGIT, Aarhus University
Denmark
dfaranha@eng.au.dk

Felipe Rodrigues Novaes
University of Campinas
Brazil
ra135663@students.ic.unicamp.br

Akira Takahashi
DIGIT, Aarhus University
Denmark
takahashi@cs.au.dk

Mehdi Tibouchi
NTT Corporation
Japan
mehdi.tibouchi.br@hco.ntt.co.jp

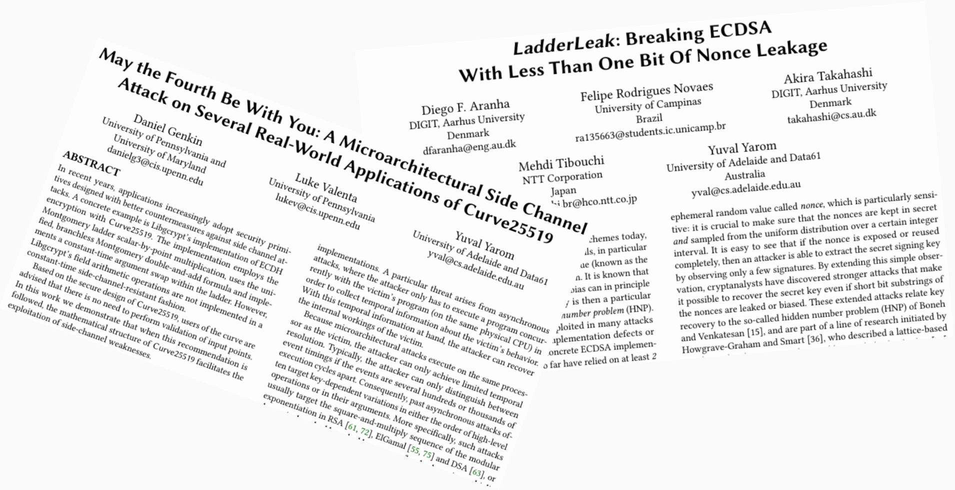
Yuval Yarom
University of Adelaide and Data61
Australia
yval@cs.adelaide.edu.au

ABSTRACT

Although it is one of the most popular signature schemes today, ECDSA presents a number of implementation pitfalls, in particular due to the very sensitive nature of the random value (known as the *nonce*) generated as part of the signing algorithm. It is known that any small amount of nonce exposure or nonce bias can in principle lead to a full key recovery: the key recovery is then a particular instance of Boneh and Venkatesan's *hidden number problem* (HNP). That observation has been practically exploited in many attacks in the literature, taking advantage of implementation defects or side-channel vulnerabilities in various concrete ECDSA implementations. However, most of the attacks so far have relied on at least 2

ephemeral random value called *nonce*, which is particularly sensitive: it is crucial to make sure that the nonces are kept in secret *and* sampled from the uniform distribution over a certain integer interval. It is easy to see that if the nonce is exposed or reused completely, then an attacker is able to extract the secret signing key by observing only a few signatures. By extending this simple observation, cryptanalysts have discovered stronger attacks that make it possible to recover the secret key even if short bit substrings of the nonces are leaked or biased. These extended attacks relate key recovery to the so-called hidden number problem (HNP) of Boneh and Venkatesan [15], and are part of a line of research initiated by Howgrave-Graham and Smart [36], who described a lattice-based

So many attacks...



May the Fourth Be With You: A Microarchitectural Side Channel Attack on Several Real-World Applications of Curve25519

Daniel Genkin
University of Pennsylvania and
University of Maryland
danielg3@cis.upenn.edu

Luke Valenta
University of Pennsylvania
lukev@cis.upenn.edu

Yuval Yarom
University of Adelaide and Data61
yval@cs.adelaide.edu.au

ABSTRACT

In recent years, applications increasingly adopt security primitives designed with better countermeasures against side channel attacks. A concrete example is Libgcrypt's implementation of ECDH encryption with Curve25519. The implementation employs the Montgomery ladder scalar-by-point multiplication, uses the unified, branchless Montgomery double-and-add formula and implements a constant-time argument swap within the ladder. However, Libgcrypt's field arithmetic operations are not implemented in a constant-time side-channel-resistant fashion.

Based on the secure design of Curve25519, users of the curve are advised that there is no need to perform validation of input points. In this work we demonstrate that when this recommendation is followed, the mathematical structure of Curve25519 facilitates the exploitation of side-channel weaknesses.

implementations. A particular threat arises from asynchronous attacks, where the attacker only has to execute a program concurrently with the victim's program (on the same physical CPU) in order to collect temporal information about the victim's behavior. With this temporal information at hand, the attacker can recover the internal workings of the victim.

Because microarchitectural attacks execute on the same processor as the victim, the attacker can only achieve limited temporal resolution. Typically, the attacker can only distinguish between event timings if the events are several hundreds or thousands of execution cycles apart. Consequently, past asynchronous attacks often target key-dependent variations in either the order of high-level operations or in their arguments. More specifically, such attacks usually target the square-and-multiply sequence of the modular exponentiation in RSA [61, 72], ElGamal [55, 75] and DSA [63], or

LadderLeak: Breaking ECDSA With Less Than One Bit Of Nonce Leakage

Diego F. Aranha
DIGIT, Aarhus University
Denmark
dfaranha@eng.au.dk

Felipe Rodrigues Novaes
University of Campinas
Brazil
ra135663@students.ic.unicamp.br

Akira Takahashi
DIGIT, Aarhus University
Denmark
takahashi@cs.au.dk

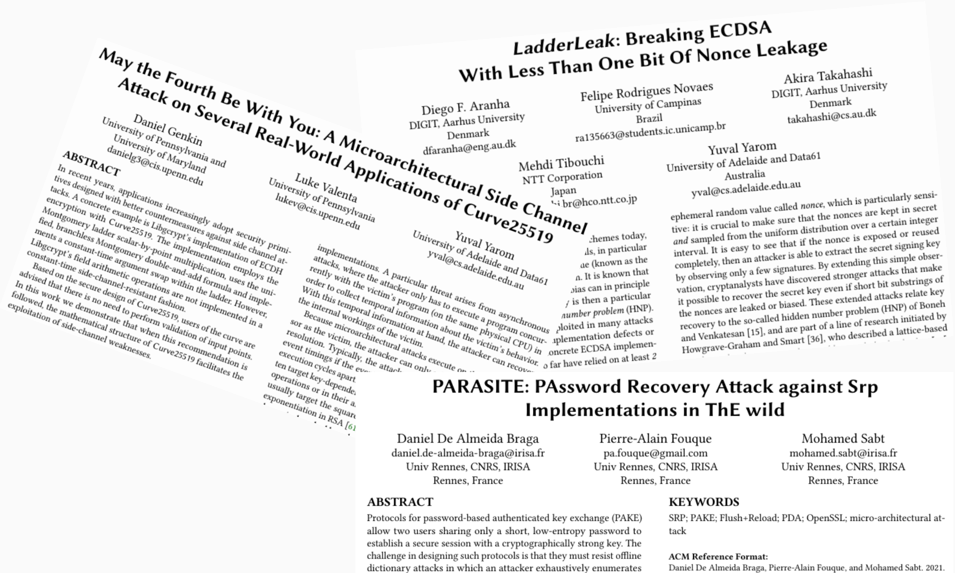
Mehdi Tibouchi
NTT Corporation
Japan
br@hco.ntt.co.jp

Yuval Yarom
University of Adelaide and Data61
Australia
yval@cs.adelaide.edu.au

chemes today,
ls, in particular
ue (known as the
n. It is known that
bias can in principle
is then a particular
number problem (HNP).
ploited in many attacks
plementation defects or
oncrete ECDSA implemen-
o far have relied on at least 2

ephemeral random value called *nonce*, which is particularly sensitive: it is crucial to make sure that the nonces are kept in secret *and* sampled from the uniform distribution over a certain integer interval. It is easy to see that if the nonce is exposed or reused completely, then an attacker is able to extract the secret signing key by observing only a few signatures. By extending this simple observation, cryptanalysts have discovered stronger attacks that make it possible to recover the secret key even if short bit substrings of the nonces are leaked or biased. These extended attacks relate key recovery to the so-called hidden number problem (HNP) of Boneh and Venkatesan [15], and are part of a line of research initiated by Howgrave-Graham and Smart [36], who described a lattice-based

So many attacks…



May the Fourth Be With You: A Microarchitectural Side Channel Attack on Several Real-World Applications of Curve25519

Daniel Genkin
University of Pennsylvania and
University of Maryland
danielg3@cis.upenn.edu

Luke Valenta
University of Pennsylvania
lukev@cis.upenn.edu

Yuval Yarom
University of Adelaide and Data61
yval@cs.adelaide.edu.au

ABSTRACT

In recent years, applications increasingly adopt security primitives designed with better countermeasures against side channel attacks. A concrete example is Libgcrypt's implementation of ECDH encryption with Curve25519. The implementation employs the Montgomery ladder scalar-by-point multiplication, uses the unified, branchless Montgomery double-and-add formula and implements a constant-time argument swap within the ladder. However, Libgcrypt's field arithmetic operations are not implemented in a constant-time side-channel-resistant fashion.

Based on the secure design of Curve25519, users of the curve are advised that there is no need to perform validation of input points. In this work we demonstrate that when this recommendation is followed, the mathematical structure of Curve25519 facilitates the exploitation of side-channel weaknesses.

implementations. A particular threat arises from asynchronous attacks, where the attacker only has to execute a program concurrently with the victim's program (on the same physical CPU) in order to collect temporal information about the victim's behavior. With this temporal information at hand, the attacker can recover the internal workings of the victim.

Because microarchitectural attacks execute on
sor as the victim, the attacker can only
resolution. Typically, the attac
event timings if the eve
execution cycles apart
ten target key-depende
operations or in their a
usually target the square
exponentiation in RSA [61

LadderLeak: Breaking ECDSA With Less Than One Bit Of Nonce Leakage

Diego F. Aranha
DIGIT, Aarhus University
Denmark
dfaranha@eng.au.dk

Felipe Rodrigues Novaes
University of Campinas
Brazil
ra135663@students.ic.unicamp.br

Akira Takahashi
DIGIT, Aarhus University
Denmark
takahashi@cs.au.dk

Mehdi Tibouchi
NTT Corporation
Japan
br@hco.ntt.co.jp

Yuval Yarom
University of Adelaide and Data61
Australia
yval@cs.adelaide.edu.au

chemes today,
ds, in particular
ue (known as the
n. It is known that
bias can in principle
is then a particular
number problem (HNP).
ploited in many attacks
nplementation defects or
oncrete ECDSA implemen-
o far have relied on at least 2

ephemeral random value called *nonce*, which is particularly sensitive: it is crucial to make sure that the nonces are kept in secret *and* sampled from the uniform distribution over a certain integer interval. It is easy to see that if the nonce is exposed or reused completely, then an attacker is able to extract the secret signing key by observing only a few signatures. By extending this simple observation, cryptanalysts have discovered stronger attacks that make it possible to recover the secret key even if short bit substrings of the nonces are leaked or biased. These extended attacks relate key recovery to the so-called hidden number problem (HNP) of Boneh and Venkatesan [15], and are part of a line of research initiated by Howgrave-Graham and Smart [36], who described a lattice-based

PARASITE: PAssword Recovery Attack against Srp Implementations in ThE wild

Daniel De Almeida Braga
daniel.de-almeida-braga@irisa.fr
Univ Rennes, CNRS, IRISA
Rennes, France

Pierre-Alain Fouque
pa.fouque@gmail.com
Univ Rennes, CNRS, IRISA
Rennes, France

Mohamed Sabt
mohamed.sabt@irisa.fr
Univ Rennes, CNRS, IRISA
Rennes, France

ABSTRACT

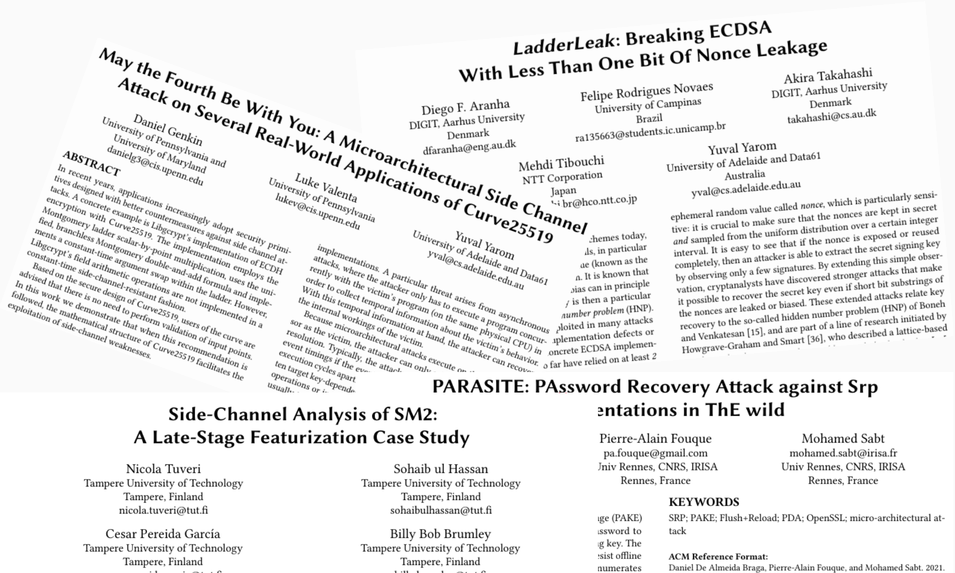
Protocols for password-based authenticated key exchange (PAKE) allow two users sharing only a short, low-entropy password to establish a secure session with a cryptographically strong key. The challenge in designing such protocols is that they resist offline dictionary attacks in which an attacker exhaustively enumerates

KEYWORDS

SRP; PAKE; Flush+Reload; PDA; OpenSSL; micro-architectural attack

ACM Reference Format:
Daniel De Almeida Braga, Pierre-Alain Fouque, and Mohamed Sabt. 2021.

So many attacks...



May the Fourth Be With You: A Microarchitectural Side Channel Attack on Several Real-World Applications of Curve25519

Daniel Genkin
University of Pennsylvania and
University of Maryland
danielg3@cis.upenn.edu

Luke Valenta
University of Pennsylvania
lukev@cis.upenn.edu

Yuval Yarom
University of Adelaide and Data61
yval@cs.adelaide.edu.au

ABSTRACT

In recent years, applications increasingly adopt security primitives designed with better countermeasures against side channel attacks. A concrete example is Libgcrypt's implementation of ECDH encryption with Curve25519. The implementation employs the Montgomery ladder scalar-by-point multiplication, uses the unified, branchless Montgomery double-and-add formula and implements a constant-time argument swap within the ladder. However, Libgcrypt's field arithmetic operations are not implemented in a constant-time side-channel-resistant fashion.

Based on the secure design of Curve25519, users of the curve are advised that there is no need to perform validation of input points. In this work we demonstrate that when this recommendation is followed, the mathematical structure of Curve25519 facilitates the exploitation of side-channel weaknesses.

implementations. A particular threat arises from asynchronous attacks, where the attacker only has to execute a program concurrently with the victim's program (on the same physical CPU) in order to collect temporal information about the victim's behavior. With this temporal information at hand, the attacker can recover the internal workings of the victim.

Because microarchitectural attacks execute on
sor as the victim, the attacker can only
resolution. Typically, the attack
event timings if the eve
execution cycles apart
ten target key-depende
operations or i
usually

LadderLeak: Breaking ECDSA With Less Than One Bit Of Nonce Leakage

Diego F. Aranha
DIGIT, Aarhus University
Denmark
dfaranha@eng.au.dk

Felipe Rodrigues Novaes
University of Campinas
Brazil
ra135663@students.ic.unicamp.br

Akira Takahashi
DIGIT, Aarhus University
Denmark
takahashi@cs.au.dk

Mehdi Tibouchi
NTT Corporation
Japan
br@hco.ntt.co.jp

Yuval Yarom
University of Adelaide and Data61
Australia
yval@cs.adelaide.edu.au

chemes today,
ls, in particular
ue (known as the
n. It is known that
bias can in principle
is then a particular
number problem (HNP).
ploited in many attacks
nplementation defects or
oncrete ECDSA implemen-
far have relied on at least 2

ephemeral random value called *nonce*, which is particularly sensitive: it is crucial to make sure that the nonces are kept in secret *and* sampled from the uniform distribution over a certain integer interval. It is easy to see that if the nonce is exposed or reused completely, then an attacker is able to extract the secret signing key by observing only a few signatures. By extending this simple observation, cryptanalysts have discovered stronger attacks that make it possible to recover the secret key even if short bit substrings of the nonces are leaked or biased. These extended attacks relate key recovery to the so-called hidden number problem (HNP) of Boneh and Venkatesan [15], and are part of a line of research initiated by Howgrave-Graham and Smart [36], who described a lattice-based

Side-Channel Analysis of SM2: A Late-Stage Featurization Case Study

Nicola Tuveri
Tampere University of Technology
Tampere, Finland
nicola.tuveri@tut.fi

Sohaib ul Hassan
Tampere University of Technology
Tampere, Finland
sohaibulhassan@tut.fi

Cesar Pereida García
Tampere University of Technology
Tampere, Finland

Billy Bob Brumley
Tampere University of Technology
Tampere, Finland

PARASITE: PAssword Recovery Attack against Srp entations in ThE wild

Pierre-Alain Fouque
pa.fouque@gmail.com
Univ Rennes, CNRS, IRISA
Rennes, France

Mohamed Sabt
mohamed.sabt@irisa.fr
Univ Rennes, CNRS, IRISA
Rennes, France

ige (PAKE)
ssword to
g key. The
sist offline
numerates

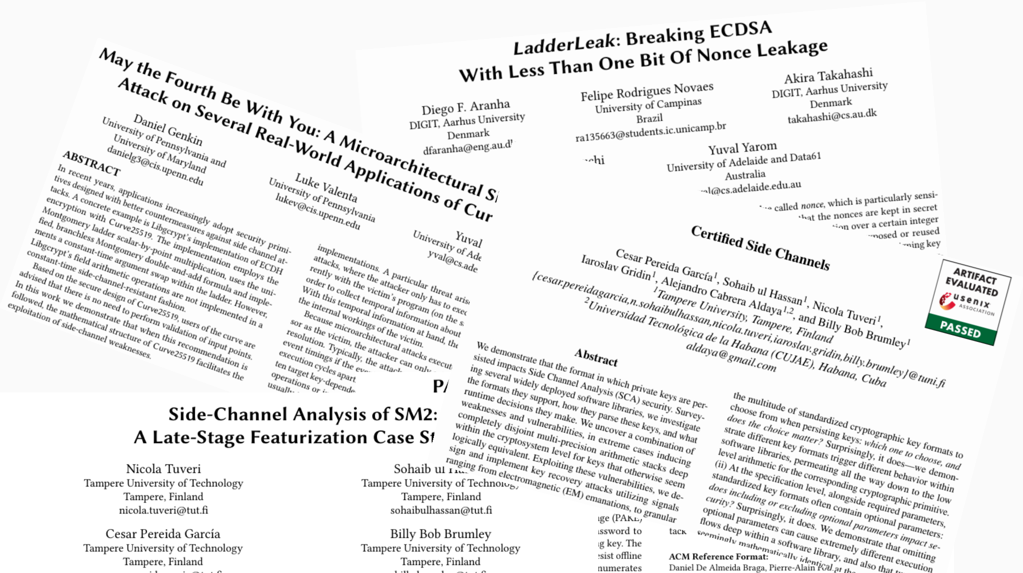
KEYWORDS

SRP; PAKE; Flush+Reload; PDA; OpenSSL; micro-architectural attack

ACM Reference Format:

Daniel De Almeida Braga, Pierre-Alain Fouque, and Mohamed Sabt. 2021.

So many attacks...



May the Fourth Be With You: A Microarchitectural S
Attack on Several Real-World Applications of Cur

Daniel Genkin
University of Pennsylvania and
University of Maryland
danielg3@cis.upenn.edu

Luke Valenta
University of Pennsylvania
lukev@cis.upenn.edu

Yuval
University of Ade
yval@cs.ade

ABSTRACT

In recent years, applications increasingly adopt security primitives designed with better countermeasures against side channel attacks. A concrete example is Libgcrypt's implementation of ECDH encryption with Curve25519. The implementation employs the Montgomery ladder scalar-by-point multiplication, uses the unified, branchless Montgomery double-and-add formula and implements a constant-time argument swap within the ladder. However, Libgcrypt's field arithmetic operations are not implemented in a constant-time side-channel-resistant fashion.

Based on the secure design of Curve25519, users of the curve are advised that there is no need to perform validation of input points. In this work we demonstrate that when this recommendation is followed, the mathematical structure of Curve25519 facilitates the exploitation of side-channel weaknesses.

implementations. A particular threat aris attacks, where the attacker only has to exec rently with the victim's program (on the s order to collect temporal information about With this temporal information at hand, the the internal workings of the victim.

Because microarchitectural attacks execut sor as the victim, the attacker can only resolution. Typically, the attack event timings if the ev execution cycles apart ten target key-depende operations or i usuall

LadderLeak: Breaking ECDSA
With Less Than One Bit Of Nonce Leakage

Diego F. Aranha
DIGIT, Aarhus University
Denmark
dfaranha@eng.au.d

Felipe Rodrigues Novaes
University of Campinas
Brazil
ra135663@students.ic.unicamp.br

Akira Takahashi
DIGIT, Aarhus University
Denmark
takahashi@cs.au.dk

chi

Yuval Yarom
University of Adelaide and Data61
Australia
l@cs.adelaide.edu.au

called *nonce*, which is particularly sensi- hat the nonces are kept in secret ion over a certain integer posed or reused ning key

Certified Side Channels

Cesar Pereida García[1], Sohaib ul Hassan[1], Nicola Tuveri[1],
Iaroslav Gridin[1], Alejandro Cabrera Aldaya[1,2], and Billy Bob Brumley[1]
[1]*Tampere University, Tampere, Finland*
[2]*Universidad Tecnológica de la Habana (CUJAE), Habana, Cuba*
{cesar.pereidagarcia,n.sohaibulhassan,nicola.tuveri,iaroslav.gridin,billy.brumley}@tuni.fi
aldaya@gmail.com

ARTIFACT EVALUATED
usenix ASSOCIATION
PASSED

Abstract

We demonstrate that the format in which private keys are persisted impacts Side Channel Analysis (SCA) security. Surveying several widely deployed software libraries, we investigate the formats they support, how they parse these keys, and what runtime decisions they make. We uncover a combination of weaknesses and vulnerabilities, in extreme cases inducing completely disjoint multi-precision arithmetic stacks deep within the cryptosystem level for keys that otherwise seem logically equivalent. Exploiting these vulnerabilities, we design and implement key recovery attacks utilizing signals ranging from electromagnetic (EM) emanations, to granular

the multitude of standardized cryptographic key formats to choose from when persisting keys: *which one to choose, and does the choice matter?* Surprisingly, it does—we demonstrate different key formats trigger different behavior within software libraries, permeating all the way down to the low level arithmetic for the corresponding cryptographic primitive. (ii) At the specification level, alongside required parameters, standardized key formats often contain optional parameters: *does including or excluding optional parameters impact security?* Surprisingly, it does. We demonstrate that omitting optional parameters can cause extremely different execution flows deep within a software library, and also that seemingly mathematically identical at the

P

ige (PAKE) ssword to g key. The sist offline numerates

tack

ACM Reference Format:
Daniel De Almeida Braga, Pierre-Alain Fou

Side-Channel Analysis of SM2:
A Late-Stage Featurization Case St

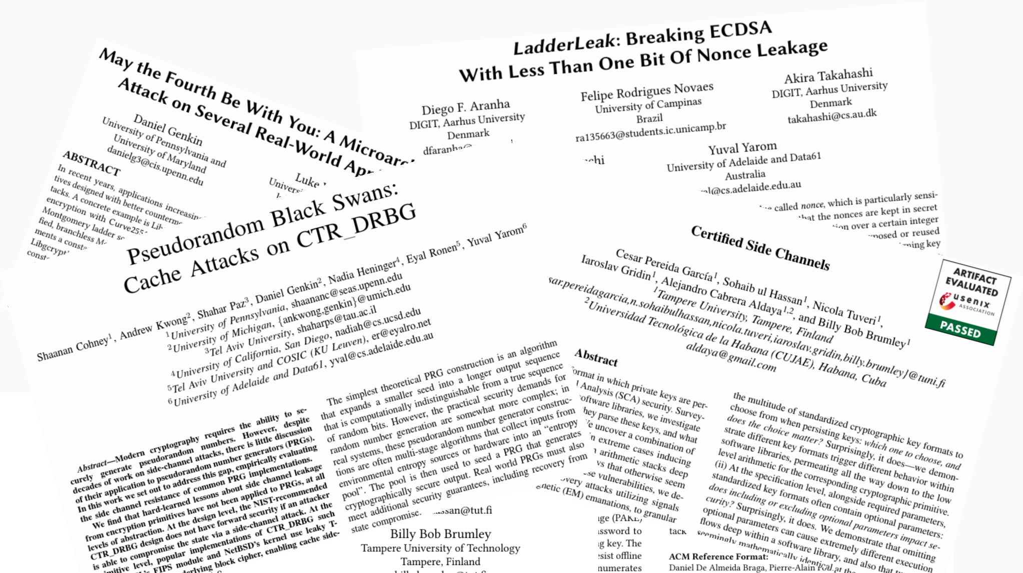
Nicola Tuveri
Tampere University of Technology
Tampere, Finland
nicola.tuveri@tut.fi

Sohaib ul Ha
Tampere University of Technology
Tampere, Finland
sohaibulhassan@tut.fi

Cesar Pereida García
Tampere University of Technology
Tampere, Finland

Billy Bob Brumley
Tampere University of Technology
Tampere, Finland

So many attacks...



LadderLeak: Breaking ECDSA With Less Than One Bit Of Nonce Leakage

Diego F. Aranha
DIGIT, Aarhus University
Denmark
dfaranha@

Felipe Rodrigues Novaes
University of Campinas
Brazil
ra135663@students.ic.unicamp.br

Akira Takahashi
DIGIT, Aarhus University
Denmark
takahashi@cs.au.dk

…chi

Yuval Yarom
University of Adelaide and Data61
Australia
…l@cs.adelaide.edu.au

… called *nonce*, which is particularly sensi- …hat the nonces are kept in secret …ion over a certain integer …posed or reused …ning key

May the Fourth Be With You: A Microar… Attack on Several Real-World Ap…

Daniel Genkin
University of Pennsylvania and
University of Maryland
danielg3@cis.upenn.edu

Luke …
Univers…

ABSTRACT

In recent years, applications increasin… tives designed with better counterm… tacks. A concrete example is Lib… encryption with Curve255… Montgomery ladder sc… fied, branchless M… ments a const… Libgcrypt'… const…

Pseudorandom Black Swans: Cache Attacks on CTR_DRBG

Shaanan Cohney[1], Andrew Kwong[2], Shahar Paz[3], Daniel Genkin[2], Nadia Heninger[4], Eyal Ronen[5], Yuval Yarom[6]
[1]*University of Pennsylvania*, shaananc@seas.upenn.edu
[2]*University of Michigan*, {ankwong,genkin}@umich.edu
[3]*Tel Aviv University*, shaharps@tau.ac.il
[4]*University of California, San Diego*, nadiah@cs.ucsd.edu
[5]*Tel Aviv University and COSIC (KU Leuven)*, er@eyalro.net
[6]*University of Adelaide and Data61*, yval@cs.adelaide.edu.au

***Abstract*—Modern cryptography requires the ability to securely generate pseudorandom numbers. However, despite decades of work on side-channel attacks, there is little discussion of their application to pseudorandom number generators (PRGs). In this work we set out to address this gap, empirically evaluating the side channel resistance of common PRG implementations.**

We find that hard-learned lessons about side channel leakage from encryption primitives have not been applied to PRGs, at all levels of abstraction. At the design level, the NIST-recommended CTR_DRBG design does not have forward security if an attacker is able to compromise the state via a side-channel attack. At the …imitive level, popular implementations of CTR_DRBG such …L's FIPS module and NetBSD's kernel use leaky T- …derlying block cipher, enabling cache side-

The simplest theoretical PRG construction is an algorithm that expands a smaller seed into a longer output sequence that is computationally indistinguishable from a true sequence of random bits. However, the practical security demands for random number generation are somewhat more complex; in real systems, these pseudorandom number generator constructions are often multi-stage algorithms that collect inputs from environmental entropy sources or hardware into an "entropy pool". The pool is then used to seed a PRG that generates cryptographically secure output. Real world PRGs must also meet additional security guarantees, including recovery from state compromise.

…ssan@tut.fi

Billy Bob Brumley
Tampere University of Technology
Tampere, Finland

Certified Side Channels

Cesar Pereida García[1], Sohaib ul Hassan[1], Nicola Tuveri[1], Iaroslav Gridin[1], Alejandro Cabrera Aldaya[1,2], and Billy Bob Brumley[1]
[1]*Tampere University, Tampere, Finland*
[2]*Universidad Tecnológica de la Habana (CUJAE), Habana, Cuba*
…sar.pereidagarcia,n.sohaibulhassan,nicola.tuveri,iaroslav.gridin,billy.brumley}@tuni.fi
aldaya@gmail.com

ARTIFACT EVALUATED usenix ASSOCIATION PASSED

Abstract

…ormat in which private keys are per- …l Analysis (SCA) security. Survey- …oftware libraries, we investigate …hey parse these keys, and what …e uncover a combination of …n extreme cases inducing …n arithmetic stacks deep …vs that otherwise seem …se vulnerabilities, we de- …overy attacks utilizing signals …netic (EM) emanations, to granular

the multitude of standardized cryptographic key formats to choose from when persisting keys: *which one to choose, and does the choice matter?* Surprisingly, it does—we demonstrate different key formats trigger different behavior within software libraries, permeating all the way down to the low level arithmetic for the corresponding cryptographic primitive. (ii) At the specification level, alongside required parameters, standardized key formats often contain optional parameters: *does including or excluding optional parameters impact security?* Surprisingly, it does. We demonstrate that omitting optional parameters can cause extremely different execution flows deep within a software library, and also that … seemingly mathematically identical at the …

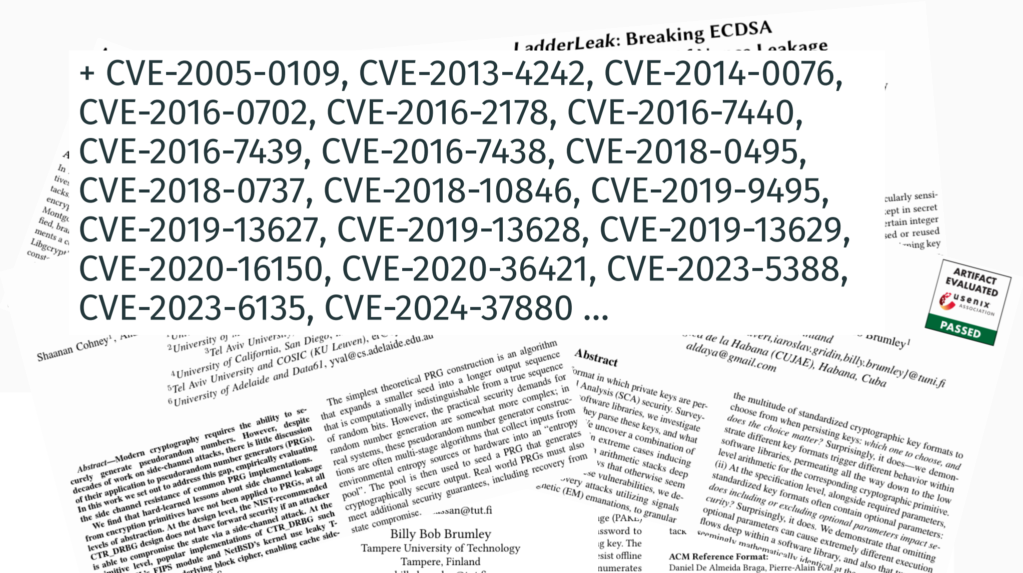
…ge (PAKE) …ssword to …g key. The …sist offline …numerates

tack

ACM Reference Format:
Daniel De Almeida Braga, Pierre-Alain Fouque, …

So many attacks...

+ CVE-2005-0109, CVE-2013-4242, CVE-2014-0076, CVE-2016-0702, CVE-2016-2178, CVE-2016-7440, CVE-2016-7439, CVE-2016-7438, CVE-2018-0495, CVE-2018-0737, CVE-2018-10846, CVE-2019-9495, CVE-2019-13627, CVE-2019-13628, CVE-2019-13629, CVE-2020-16150, CVE-2020-36421, CVE-2023-5388, CVE-2023-6135, CVE-2024-37880 ...



LadderLeak: Breaking ECDSA

Leakage

Shaanan Cohney[1]

[2] University of

[3] Tel Aviv University

[4] University of California, San Diego

[5] Tel Aviv University and COSIC (KU Leuven)

[6] University of Adelaide and Data61, yval@cs.adelaide.edu.au

Abstract—Modern cryptography requires the ability to securely generate pseudorandom numbers. However, despite decades of work on side-channel attacks, there is little discussion of their application to pseudorandom number generators (PRGs). In this work we set out to address this gap, empirically evaluating the side channel resistance of common PRG implementations.

We find that hard-learned lessons about side channel leakage from encryption primitives have not been applied to PRGs, at all levels of abstraction. At the design level, the NIST-recommended CTR_DRBG design does not have forward security if an attacker is able to compromise the state via a side-channel attack. At the primitive level, popular implementations of CTR_DRBG such FIPS module and NetBSD's kernel use leaky T- block cipher, enabling cache side-

The simplest theoretical PRG construction is an algorithm that expands a smaller seed into a longer output sequence that is computationally indistinguishable from a true sequence of random bits. However, the practical security demands for random number generation are somewhat more complex; in real systems, these pseudorandom number generator constructions are often multi-stage algorithms that collect inputs from environmental entropy sources or hardware into an "entropy pool". The pool is then used to seed a PRG that generates cryptographically secure output. Real world PRGs must also meet additional security guarantees, including recovery from state compromise.

Billy Bob Brumley
Tampere University of Technology
Tampere, Finland

Abstract

format in which private keys are per- Analysis (SCA) security. Survey- software libraries, we investigate they parse these keys, and what We uncover a combination of in extreme cases inducing arithmetic stacks deep that otherwise seem these vulnerabilities, we de- recovery attacks utilizing signals netic (EM) emanations, to granular

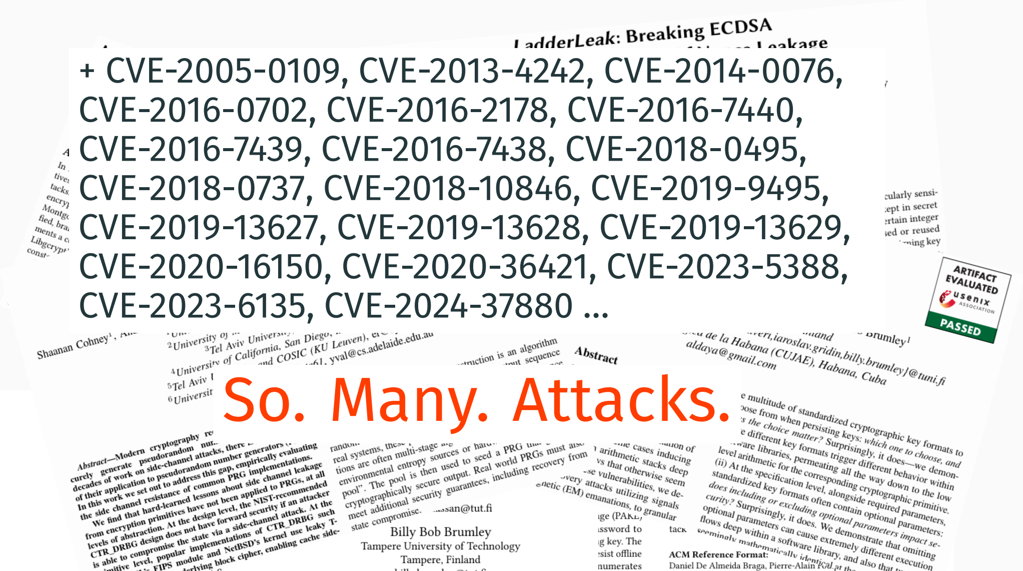
de la Habana (CUJAE), Habana, Cuba
aldaya@gmail.com
iaroslav.gridin,billy.brumley}@tuni.fi

the multitude of standardized cryptographic key formats to choose from when persisting keys: *which one to choose, and does the choice matter?* Surprisingly, it does—we demonstrate different key formats trigger different behavior within software libraries, permeating all the way down to the low level arithmetic for the corresponding cryptographic primitive. (ii) At the specification level, alongside required parameters, standardized key formats often contain optional parameters: *does including or excluding optional parameters impact security?* Surprisingly, it does. We demonstrate that omitting optional parameters can cause extremely different execution flows deep within a software library, and also that

ACM Reference Format:
Daniel De Almeida Braga, Pierre-Alain Fouque

ARTIFACT EVALUATED usenix ASSOCIATION PASSED

So many attacks...



LadderLeak: Breaking ECDSA

...Leakage

A

In

tives

tacks

encry

Montgo

fied, bra

ments a c

Libgcrypt

const

cularly sensi-

ept in secret

ertain integer

sed or reused

ning key

ARTIFACT EVALUATED usenix ASSOCIATION PASSED

+ CVE-2005-0109, CVE-2013-4242, CVE-2014-0076, CVE-2016-0702, CVE-2016-2178, CVE-2016-7440, CVE-2016-7439, CVE-2016-7438, CVE-2018-0495, CVE-2018-0737, CVE-2018-10846, CVE-2019-9495, CVE-2019-13627, CVE-2019-13628, CVE-2019-13629, CVE-2020-16150, CVE-2020-36421, CVE-2023-5388, CVE-2023-6135, CVE-2024-37880 ...

Shaanan Cohney1, Al

2University of

3Tel Aviv Univers

University of California, San Diego,

4University of ... and COSIC (KU Leuven)

5Tel Aviv U

6Universit

yval@cs.adelaide.edu.au

struction is an algorithm

input sequence

Abstract

eda de la Habana (CUJAE), Habana, Cuba

aldaya@gmail.com

veri,iaroslav.gridin,billy.brumley}@tuni.fi

Brumley1

So. Many. Attacks.

Abstract—Modern cryptography re... curely generate pseudorandom num... decades of work on side-channel attacks, there ... of their application to pseudorandom number generators (... In this work we set out to address this gap, empirically evaluating the side channel resistance of common PRG implementations. We find that hard-learned lessons about side channel leakage from encryption primitives have not been applied to PRGs, at all levels of abstraction. At the design level, the NIST-recommended CTR_DRBG design does not have forward security if an attacker is able to compromise the state via a side-channel attack. At the primitive level, popular implementations of CTR_DRBG such ... FIPS module and NetBSD's kernel use leaky T-... underlying block cipher, enabling cache side-

random... real systems, these ... tions are often multi-stage ... environmental entropy sources or hard... pool". The pool is then used to seed a PRG ... cryptographically secure output. Real world PRGs must also meet additional security guarantees, including recovery from state compromise.

ssan@tut.fi

Billy Bob Brumley
Tampere University of Technology
Tampere, Finland

cases inducing ... arithmetic stacks deep ... s that otherwise seem ... vulnerabilities, we de... very attacks utilizing signals ... netic (EM) emanations, to granular

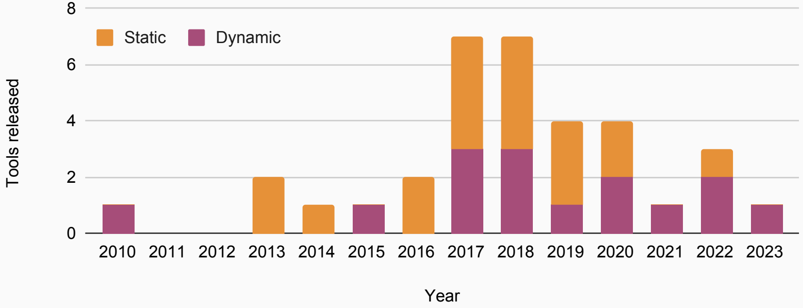
ge (PAKE) ... ssword to ... g key. The ... sist offline ... numerates

tack

e multitude of standardized cryptographic key formats to ... pose from when persisting keys: which one to choose, and ... s the choice matter? Surprisingly, it does—we demon... e different key formats trigger different behavior within ... ware libraries, permeating all the way down to the low level arithmetic for the corresponding cryptographic primitive. (ii) At the specification level, alongside required parameters, standardized key formats often contain optional parameters. does including or excluding optional parameters impact security? Surprisingly, it does. We demonstrate that omitting optional parameters can cause extremely different execution flows deep within a software library, and also that ... seemingly mathematically identical at the

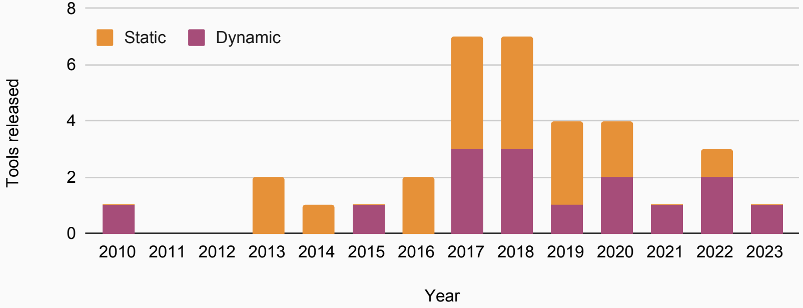
ACM Reference Format:
Daniel De Almeida Braga, Pierre-Alain Fouque

So many detection frameworks, yet so many attacks... Why?



Many tools published from 2017, 67% of tools are open source (23 over 34)

So many detection frameworks, yet so many attacks... Why?

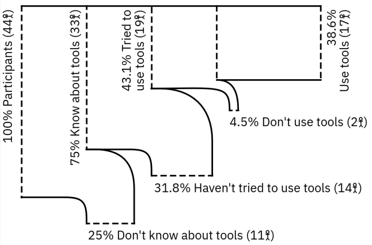


Many tools published from 2017, 67% of tools are open source (23 over 34)

Why are so many attacks still manually found?

Related Work

- do developers use CT tools? [S&P 2022]
  → most developers do not use them, or do not know about them
- how to improve the tool usability? [USENIX Sec 2024]
  → most developers find them really hard to use

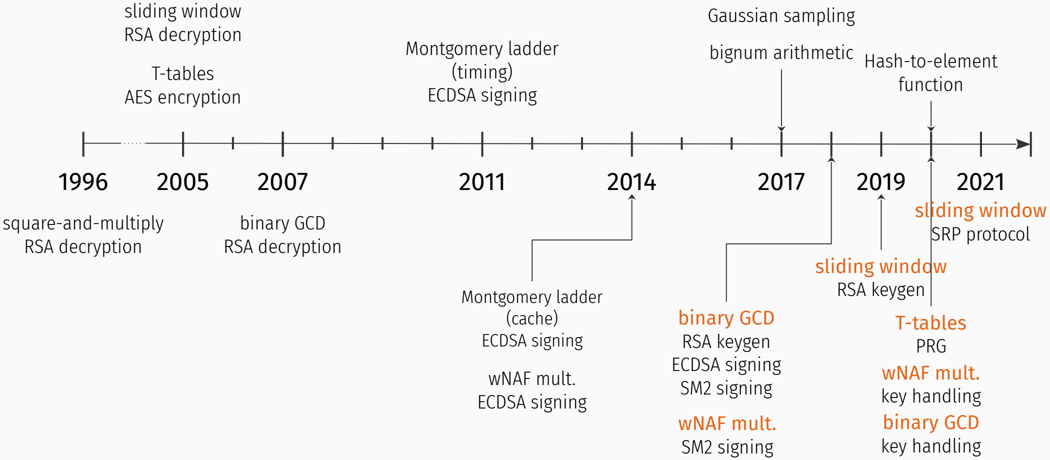


J. Jancar et al. “”They’re not that hard to mitigate”: What Cryptographic Library Developers Think About Timing Attacks”. In: *S&P*. 2022.

M. Fourné et al. ““These results must be false”: A usability evaluation of constant-time analysis tools”. In: *USENIX Security Symposium*. 2024.

Would the tools actually work to automatically find recent vulnerabilities?

Comparing recent vulnerabilities (2017-2022) with past vulnerabilities



1996

square-and-multiply
RSA decryption

2005

sliding window
RSA decryption

T-tables
AES encryption

2007

binary GCD
RSA decryption

2011

Montgomery ladder
(timing)
ECDSA signing

2014

Montgomery ladder
(cache)
ECDSA signing

wNAF mult.
ECDSA signing

2017

Gaussian sampling

bignum arithmetic

binary GCD
RSA keygen
ECDSA signing
SM2 signing

wNAF mult.
SM2 signing

2019

sliding window
RSA keygen

Hash-to-element
function

T-tables
PRG

wNAF mult.
key handling

binary GCD
key handling

2021

sliding window
SRP protocol

The SAME vulnerabilities keep resurfacing. Why? (1/2)

New contexts:

- Key generation
- Key parsing and handling
- Random number generation

(Mostly OpenSSL) Vulnerable code stays in the library
and the CT flag is not correctly set

A. C. Aldaya et al. “Cache-Timing Attacks on RSA Key Generation”. In: *TCHES* (2019)

C. P. García et al. “Certified Side Channels”. In: *USENIX Security Symposium*. 2020

S. Cohney et al. “Pseudorandom Black Swans: Cache Attacks on CTR_DRBG”. In: *S&P*. 2020

The SAME vulnerabilities keep resurfacing. Why? (2/2)

New libraries

- MbedTLS sliding window RSA implementation
- Bleichenbacher-like attacks in MbedTLS, s2n, or NSS

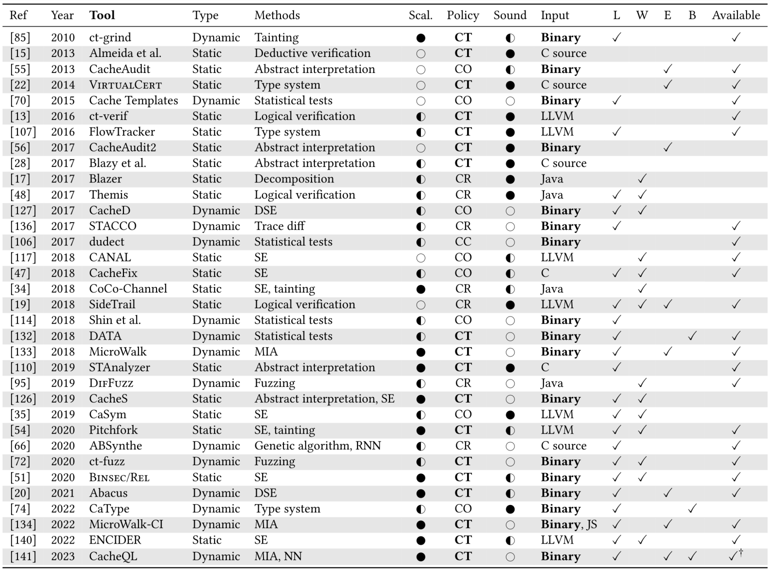
Vulnerability is found in OpenSSL but
patches are not propagated to other libraries

M. Schwarz et al. “Malware Guard Extension: Using SGX to Conceal Cache Attacks”. In: *DIMVA*. 2017

E. Ronen et al. “The 9 Lives of Bleichenbacher’s CAT: New Cache ATtacks on TLS Implementations”. In: *S&P*. 2019

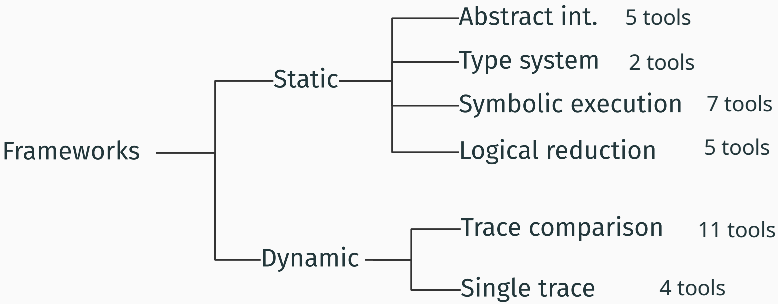
Most vulnerabilities stem from code
already known to be vulnerable

Side-channel vulnerability detection tools (1/2)



Ref	Year	**Tool**	Type	Methods	Scal.	Policy	Sound	Input	L	W	E	B	Available
[85]	2010	ct-grind	Dynamic	Tainting	●	**CT**	◐	**Binary**	✓				✓
[15]	2013	Almeida et al.	Static	Deductive verification	○	**CT**	●	C source					
[55]	2013	CacheAudit	Static	Abstract interpretation	○	CO	◐	**Binary**			✓		✓
[22]	2014	VirtualCert	Static	Type system	○	**CT**	●	C source			✓		✓
[70]	2015	Cache Templates	Dynamic	Statistical tests	○	CO	○	**Binary**	✓				✓
[13]	2016	ct-verif	Static	Logical verification	◐	**CT**	●	LLVM					✓
[107]	2016	FlowTracker	Static	Type system	◐	**CT**	●	LLVM	✓				✓
[56]	2017	CacheAudit2	Static	Abstract interpretation	○	**CT**	●	**Binary**			✓		
[28]	2017	Blazy et al.	Static	Abstract interpretation	◐	**CT**	●	C source					
[17]	2017	Blazer	Static	Decomposition	◐	CR	●	Java		✓			
[48]	2017	Themis	Static	Logical verification	◐	CR	●	Java	✓	✓			
[127]	2017	CacheD	Dynamic	DSE	◐	CO	○	**Binary**	✓	✓			
[136]	2017	STACCO	Dynamic	Trace diff	◐	CR	○	**Binary**	✓				✓
[106]	2017	dudect	Dynamic	Statistical tests	◐	CC	○	**Binary**					✓
[117]	2018	CANAL	Static	SE	○	CO	◐	LLVM		✓			✓
[47]	2018	CacheFix	Static	SE	◐	CO	◐	C	✓	✓			✓
[34]	2018	CoCo-Channel	Static	SE, tainting	●	CR	◐	Java		✓			
[19]	2018	SideTrail	Static	Logical verification	○	CR	●	LLVM	✓	✓	✓		✓
[114]	2018	Shin et al.	Dynamic	Statistical tests	◐	CO	○	**Binary**	✓				
[132]	2018	DATA	Dynamic	Statistical tests	◐	**CT**	○	**Binary**	✓			✓	✓
[133]	2018	MicroWalk	Dynamic	MIA	●	**CT**	○	**Binary**	✓		✓		✓
[110]	2019	STAnalyzer	Static	Abstract interpretation	●	**CT**	●	C	✓				✓
[95]	2019	DifFuzz	Dynamic	Fuzzing	◐	CR	○	Java		✓			✓
[126]	2019	CacheS	Static	Abstract interpretation, SE	●	**CT**	○	**Binary**	✓	✓			
[35]	2019	CaSym	Static	SE	◐	CO	●	LLVM	✓	✓			
[54]	2020	Pitchfork	Static	SE, tainting	●	**CT**	◐	LLVM	✓	✓			✓
[66]	2020	ABSynthe	Dynamic	Genetic algorithm, RNN	◐	CR	○	C source	✓				✓
[72]	2020	ct-fuzz	Dynamic	Fuzzing	◐	**CT**	○	**Binary**	✓	✓			✓
[51]	2020	Binsec/Rel	Static	SE	●	**CT**	◐	**Binary**	✓	✓			✓
[20]	2021	Abacus	Dynamic	DSE	●	**CT**	◐	**Binary**	✓		✓		✓
[74]	2022	CaType	Dynamic	Type system	◐	CO	●	**Binary**	✓			✓	
[134]	2022	MicroWalk-CI	Dynamic	MIA	●	**CT**	○	**Binary**, JS	✓		✓		✓
[140]	2022	ENCIDER	Static	SE	●	**CT**	◐	LLVM	✓	✓			✓
[141]	2023	CacheQL	Dynamic	MIA, NN	●	**CT**	○	**Binary**	✓		✓	✓	✓[†]

Side-channel vulnerability detection tools (2/2)



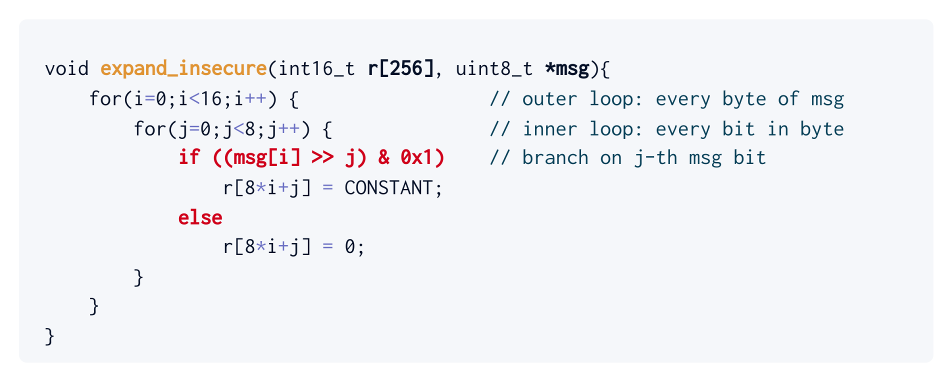
Side-note: Why you want to detect vulnerabilities at the binary level (1/4)

- the compiler is not your friend, it just wants to make stuff fast
- recent example: Kyber implementation, CVE-2024-37880, June 03, 2024

`https://pqshield.com/pqshield-plugs-timing-leaks-in-kyber-ml-kem-to-improve-pqc-implementation-maturity/`

Side-note: Why you want to detect vulnerabilities at the binary level (2/4)

Expanding a string into an array of integer, the wrong way

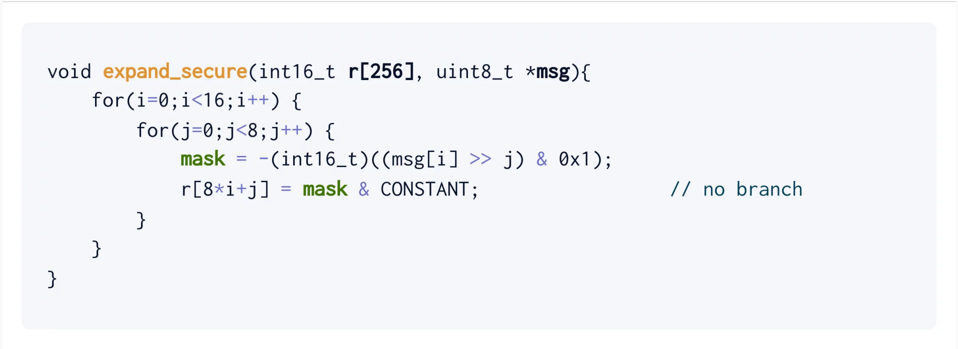


```
void expand_insecure(int16_t r[256], uint8_t *msg){
    for(i=0;i<16;i++) {                   // outer loop: every byte of msg
        for(j=0;j<8;j++) {                // inner loop: every bit in byte
            if ((msg[i] >> j) & 0x1)    // branch on j-th msg bit
                r[8*i+j] = CONSTANT;
            else
                r[8*i+j] = 0;
        }
    }
}
```

https://pqshield.com/pqshield-plugs-timing-leaks-in-kyber-ml-kem-to-improve-pqc-implementation-maturity/

Side-note: Why you want to detect vulnerabilities at the binary level (3/4)

Expanding a string into an array of integer, the right way



```
void expand_secure(int16_t r[256], uint8_t *msg){
    for(i=0;i<16;i++) {
        for(j=0;j<8;j++) {
            mask = -(int16_t)((msg[i] >> j) & 0x1);
            r[8*i+j] = mask & CONSTANT;                  // no branch
        }
    }
}
```

https://pqshield.com/pqshield-plugs-timing-leaks-in-kyber-ml-kem-to-improve-pqc-implementation-maturity/

Side-note: Why you want to detect vulnerabilities at the binary level (4/4)

Now, what does the compiler do with your code?

```
expand_insecure:    // x86 assembly
       xor     eax, eax
.outer:
       xor     ecx, ecx
.inner:
       movzx   r8d, byte ptr [rsi + rax]
       xor     edx, edx
       bt      r8d, ecx   // LSB test on (m[i] >> j)
       jae     .skip      // unsafe branch
       mov     edx, 1665  // load of CONSTANT (may be skipped)
.skip:
       mov     word ptr [rdi + 2*rcx], dx
       inc     rcx
       cmp     rcx, 8
       jne     .inner     // safe branch: inner loop
       inc     rax
       add     rdi, 16
       cmp     rax, 32
       jne     .outer     // safe branch: outer loop
       ret
```

https://pqshield.com/pqshield-plugs-timing-leaks-in-kyber-ml-kem-to-improve-pqc-implementation-maturity/

Side-note: Why you want to detect vulnerabilities at the binary level (4/4)

Now, what does the compiler do with your code?

```
expand_insecure:    // x86 assembly
       xor     eax, eax
.outer:
       xor     ecx, ecx
.inner:
       movzx   r8d, byte ptr [rsi + rax]
       xor     edx, edx
       bt      r8d, ecx    // LSB test on (m[i] >> j)
       jae     .skip       // unsafe branch
       mov     edx, 1665   // load of CONSTANT (may be skipped)
.skip:
       mov     word ptr [rdi + 2*rcx], dx
       inc     rcx
       cmp     rcx, 8
       jne     .inner      // safe branch: inner loop
       inc     rax
       add     rdi, 16
       cmp     rax, 32
       jne     .outer      // safe branch: outer loop
       ret
```

```
expand_secure:  // x86 assembly
         [...]
.outer:
         [...]
.inner:
         movzx    r8d, byte ptr [rsi + rax]
         xor      edx, edx
         bt       r8d, ecx
         jae      .skip       // still here :(
         mov      edx, 1665
.skip:
         [...]
         ret
```

https://pqshield.com/pqshield-plugs-timing-leaks-in-kyber-ml-kem-to-improve-pqc-implementation-maturity/

Side-note: Why you want to detect vulnerabilities at the binary level (4/4)

Now, what does the compiler do with your code? Yes, it ✨ optimizes it ✨

```
expand_insecure:    // x86 assembly
        xor     eax, eax
.outer:
        xor     ecx, ecx
.inner:
        movzx   r8d, byte ptr [rsi + rax]
        xor     edx, edx
        bt      r8d, ecx    // LSB test on (m[i] >> j)
        jae     .skip       // unsafe branch
        mov     edx, 1665   // load of CONSTANT (may be skipped)
.skip:
        mov     word ptr [rdi + 2*rcx], dx
        inc     rcx
        cmp     rcx, 8
        jne     .inner      // safe branch: inner loop
        inc     rax
        add     rdi, 16
        cmp     rax, 32
        jne     .outer      // safe branch: outer loop
        ret
```

```
expand_secure:  // x86 assembly
        [...]
.outer:
        [...]
.inner:
        movzx   r8d, byte ptr [rsi + rax]
        xor     edx, edx
        bt      r8d, ecx
        jae     .skip       // still here :(
        mov     edx, 1665
.skip:
        [...]
        ret
```

https://pqshield.com/pqshield-plugs-timing-leaks-in-kyber-ml-kem-to-improve-pqc-implementation-maturity/

Benchmark: cryptographic operations

Unified benchmark representative of cryptographic operations:

- 5 tools: Binsec/Rel, Abacus, ctgrind, dudect, Microwalk-CI
- 25 benchmarks from 3 libraries (OpenSSL, MbedTLS, BearSSL)
- cryptographic primitives: symmetric, AEAD schemes, asymmetric

L. Daniel, S. Bardin, and T. Rezk. “Binsec/Rel: Efficient Relational Symbolic Execution for Constant-Time at Binary-Level”. In: *S&P*. 2020.

Q. Bao et al. “Abacus: Precise Side-Channel Analysis”. In: *ICSE*. 2021.

https://github.com/agl/ctgrind

O. Reparaz, J. Balasch, and I. Verbauwhede. “Dude, is my code constant time?” In: *DATE*. 2017.

J. Wichelmann et al. “Microwalk-CI: Practical Side-Channel Analysis for JavaScript Applications”. In: *CCS*. 2022.

Benchmark results: cryptographic operations (selection)

	Binsec/Rel2	Abacus	ctgrind	Microwalk
	#V	#V	#V	#V
AES-CBC-bearssl (T)	36	36	36	36
AES-CBC-bearssl (BS)	0	0	0	0
AES-GCM-openssl (EVP)	0	0	70	8
RSA-bearssl (OAEP)	2 (⌛)	💣	87	0
RSA-openssl (PKCS)	1 (⌛)	0	321	46
RSA-openssl (OAEP)	1 (⌛)	💣	546	61

- timeout limit (⌛): 1 hour
- tools generally agree on symmetric crypto, but disagree on asymmetric crypto
- takeaway: support for vector instructions is essential

Benchmark: recent vulnerabilities

Replication of published vulnerabilities:

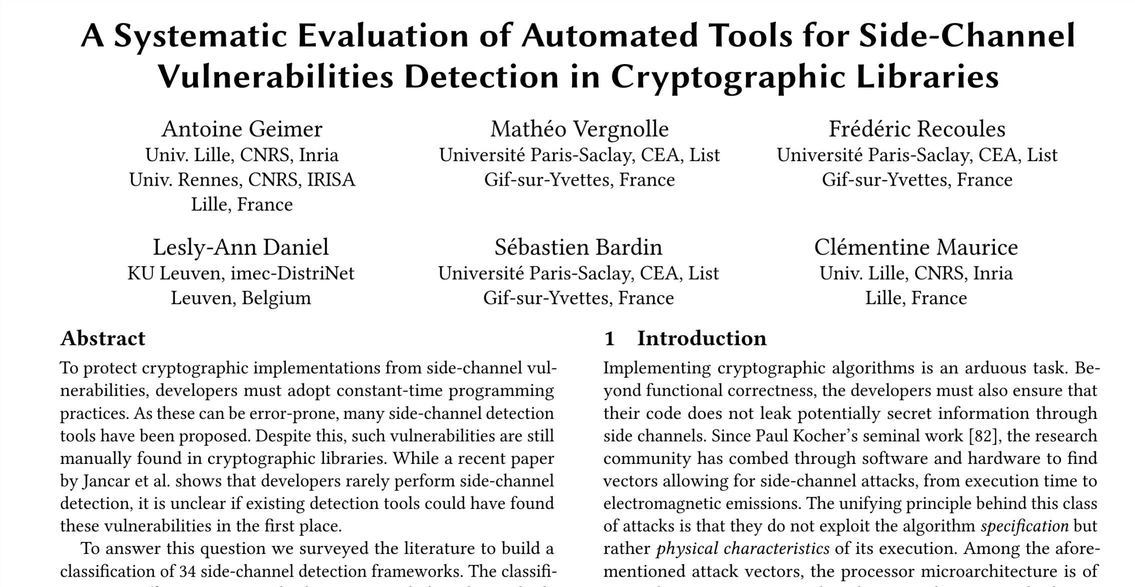
- 7 vulnerable functions from 3 publications
- both the function itself and its context are targeted
- total: 11 additional benchmarks

Benchmark results: recent vulnerabilities (selection)

	Binsec/Rel2		Abacus		ctgrind		Microwalk	
	V	T(s)	V	T(s)	V	T(s)	V	T(s)
RSA valid. (MbedTLS)		⌛		490.01	✓	0.40	✓	278.94
GCD		⌛		37.74		0.21	✓	22.96
modular inversion		⌛		242.10	✓	0.24	✓	141.82
RSA keygen (OpenSSL)		0.17	💣	8.66		6.36	✓	842.02
GCD	✓	⌛		⌛	✓	0.19	✓	3.61
modular inversion		⌛		⌛	✓	0.21	✓	5.96

- some vulnerabilities are missed because of implicit flows
- most tools do not support tainting internal secrets

More details in our CCS 2023 paper!



A Systematic Evaluation of Automated Tools for Side-Channel Vulnerabilities Detection in Cryptographic Libraries

Antoine Geimer
Univ. Lille, CNRS, Inria
Univ. Rennes, CNRS, IRISA
Lille, France

Mathéo Vergnolle
Université Paris-Saclay, CEA, List
Gif-sur-Yvettes, France

Frédéric Recoules
Université Paris-Saclay, CEA, List
Gif-sur-Yvettes, France

Lesly-Ann Daniel
KU Leuven, imec-DistriNet
Leuven, Belgium

Sébastien Bardin
Université Paris-Saclay, CEA, List
Gif-sur-Yvettes, France

Clémentine Maurice
Univ. Lille, CNRS, Inria
Lille, France

Abstract

To protect cryptographic implementations from side-channel vulnerabilities, developers must adopt constant-time programming practices. As these can be error-prone, many side-channel detection tools have been proposed. Despite this, such vulnerabilities are still manually found in cryptographic libraries. While a recent paper by Jancar et al. shows that developers rarely perform side-channel detection, it is unclear if existing detection tools could have found these vulnerabilities in the first place.

To answer this question we surveyed the literature to build a classification of 34 side-channel detection frameworks. The classifi-

1 Introduction

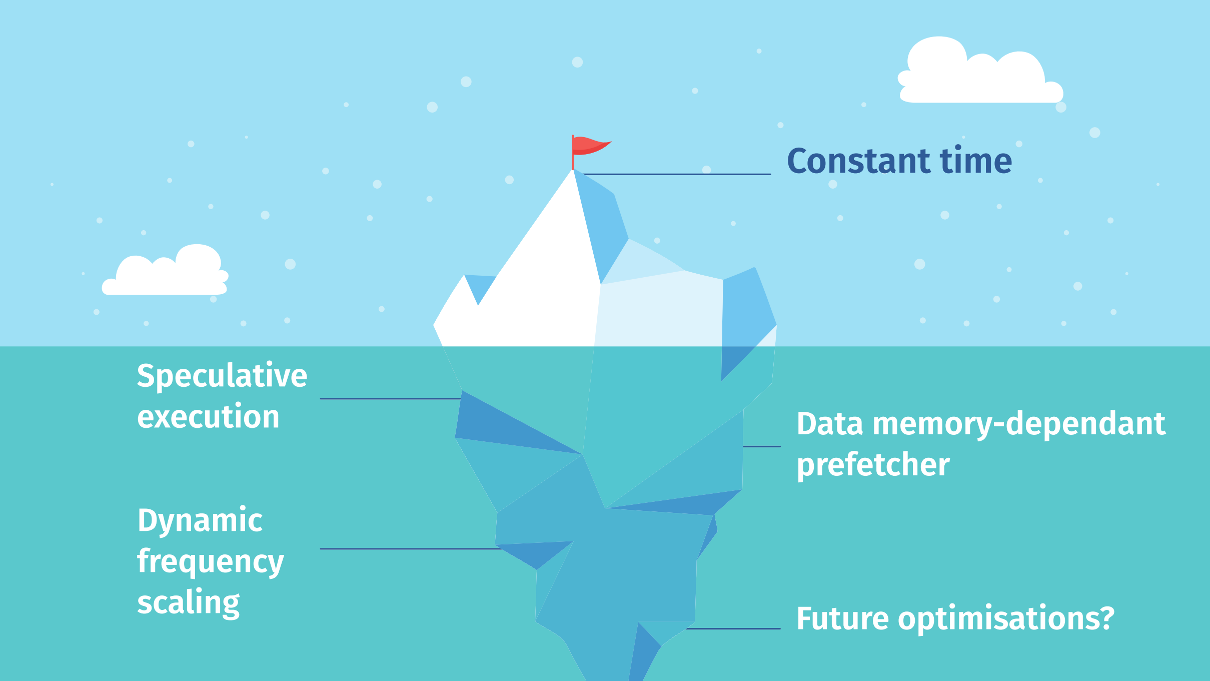
Implementing cryptographic algorithms is an arduous task. Beyond functional correctness, the developers must also ensure that their code does not leak potentially secret information through side channels. Since Paul Kocher's seminal work [82], the research community has combed through software and hardware to find vectors allowing for side-channel attacks, from execution time to electromagnetic emissions. The unifying principle behind this class of attacks is that they do not exploit the algorithm *specification* but rather *physical characteristics* of its execution. Among the aforementioned attack vectors, the processor microarchitecture is of

Perspectives & Conclusion

tl;dr

Side-channel free software, are we there yet?

Nope!



Constant time

Speculative execution

Data memory-dependant prefetcher

Dynamic frequency scaling

Future optimisations?

Beyond constant time

Code that is ”constant-time” (and considered secure until recently) can be vulnerable too!

Conclusions

- first paper by Kocher in 1996: almost 30 years of research in this area

Conclusions

- first paper by Kocher in 1996: almost 30 years of research in this area
- domain still in expansion: increasing number of papers published since 2015

Conclusions

- first paper by Kocher in 1996: almost 30 years of research in this area
- domain still in expansion: increasing number of papers published since 2015
- micro-architectural attacks require a:
  - low-level understanding of hardware → micro-architecture, reverse-engineering
  - low-level understanding of software → program analysis, compilation, cryptography...

→ work across all abstraction layers!

Thank you!

Contact

✉ clementine.maurice@inria.fr

Microarchitectural side channels: from hardware to software

Clémentine Maurice, CNRS, CRIStAL

March 13, 2025—ARCHI 2025