

# Side-channel-free software, are we there yet?

---

Clémentine Maurice, CNRS, CRIStAL

17 June 2024—MPI-SP Symposium

## Attacks on micro-architecture

- **hardware** usually modeled as an abstract layer behaving correctly

# Attacks on micro-architecture

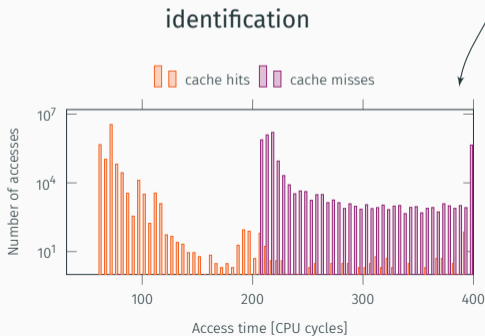
- **hardware** usually modeled as an abstract layer behaving correctly, but possible attacks

# 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

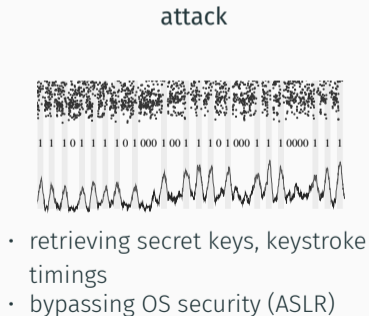
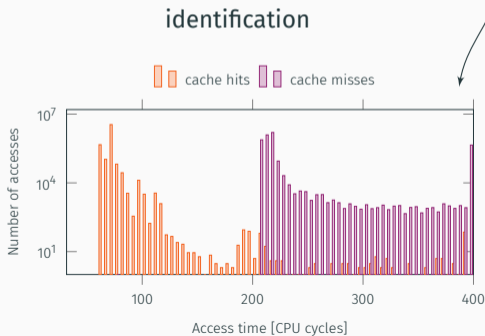
# Attacks on micro-architecture

- **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



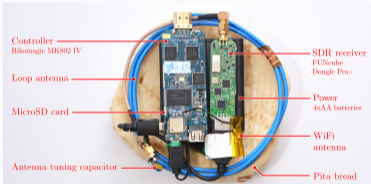
# Attacks on micro-architecture

- **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



# Attacker model

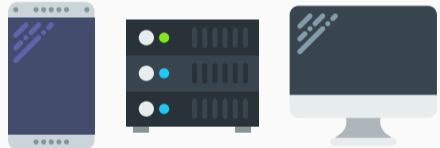
## Hardware-based attacks a.k.a physical attacks



Physical access to hardware  
→ embedded devices

VS

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



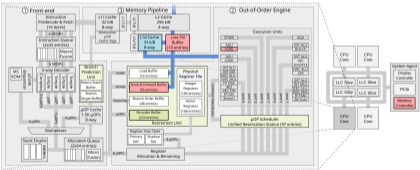
Co-located or remote attacker  
→ complex systems

# 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:**  $b^e \bmod n$

$X \leftarrow 1$

for  $i \leftarrow \text{bitlen}(e)$  downto 0 do

$X \leftarrow \text{multiply}(X, X)$

    if  $e_i = 1$  then

$X \leftarrow \text{multiply}(X, b)$

    end

end

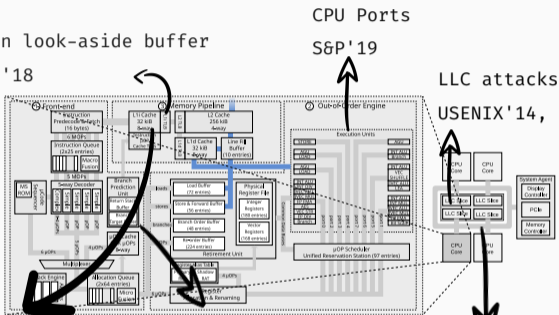
return  $X$



# We are more or less doomed on the hardware side

Translation look-aside buffer

USENIX Sec'18



L1d, L1i, L2 cache

BSDCon'05, CT-RSA'06,

ASIACCS'20

Branch Prediction

CT-RSA'07

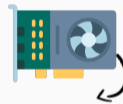
Ring Interconnect

USENIX Sec'21, DIMVA'21



DRAM

USENIX Sec'16



GPU

S&P'18

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

# 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:**  $c^d \bmod n$

$X \leftarrow 1$

**for**  $i \leftarrow \text{bitlen}(d)$  **downto** 0 **do**

$X \leftarrow \text{square}(X)$

$X \leftarrow X \bmod n$

**if**  $d_i = 1$  **then**

$X \leftarrow \text{multiply}(X, c)$

$X \leftarrow X \bmod n$

**end**

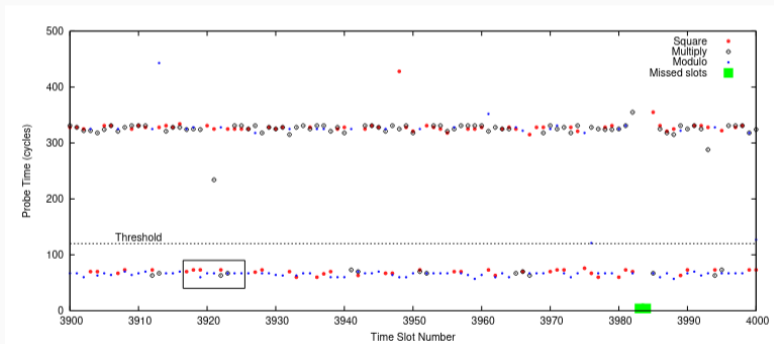
**end**

**return**  $X$

---

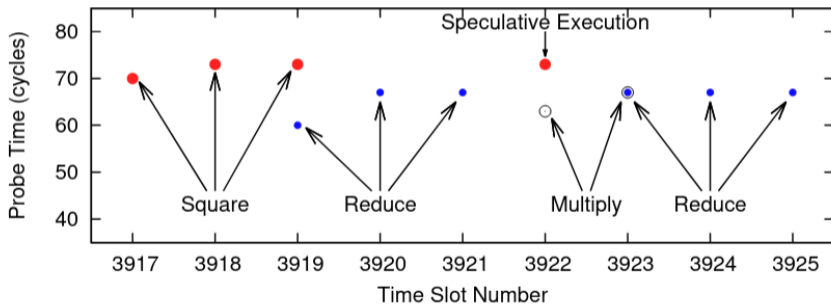
# Attacking GnuPG 1.4.13 RSA exponentiation

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



# Attacking GnuPG 1.4.13 RSA exponentiation

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



# mbedTLS 2.3.0 RSA square-and-multiply exponentiation

mbedTLS version 2.3.0 (2017), “fixes” the issue with a single operation multiply

---

**Algorithm 2:** mbedTLS 2.3.0 Square-and-multiply exponentiation

---

**Input:** base  $c$ , **exponent**  $d$ , modulus  $n$

**Output:**  $c^d \bmod n$

$X \leftarrow 1$

**for**  $i \leftarrow \text{bitlen}(d)$  **downto** 0 **do**

$X \leftarrow \text{multiply}(X, X)$

$X \leftarrow X \bmod n$

**if**  $d_i = 1$  **then**

$X \leftarrow \text{multiply}(X, c)$

$X \leftarrow X \bmod n$

**end**

**end**

**return**  $X$

---

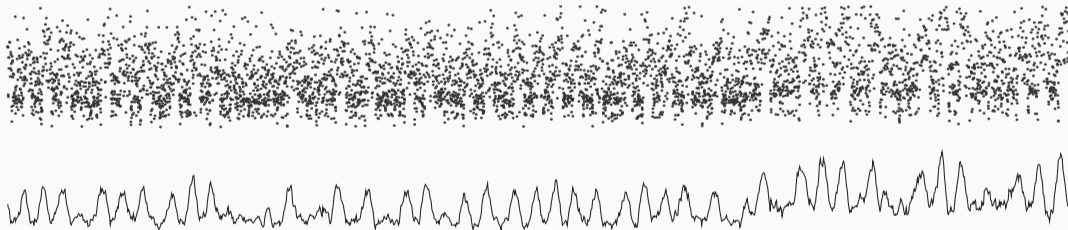
# Attacking mbedTLS 2.3.0 RSA exponentiation

- raw Prime+Probe trace on the buffer holding the multiplier  $c$



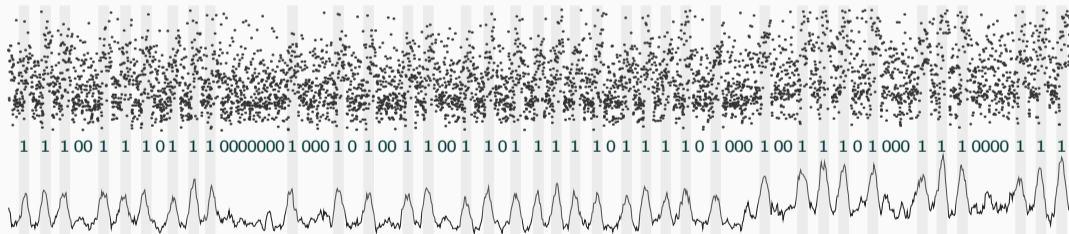
# Attacking mbedTLS 2.3.0 RSA exponentiation

- raw Prime+Probe trace on the buffer holding the multiplier  $c$
- processed with a simple moving average



# Attacking mbedTLS 2.3.0 RSA exponentiation

- raw Prime+Probe trace on the buffer holding the multiplier  $c$
- processed with a simple moving average
- allows to clearly recover the **bits of the secret exponent**





## Problem?

Side-channel vulnerability

Any **branch or memory access**  
that depends on a **secret**

# 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!**

# 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?

# 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?

## 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 *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

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

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

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

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

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

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

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

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

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

### ABSTRACT

In recent years, applications increasingly adopt security primitives designed with better countermeasures against side channel attacks. A concrete example is Libcrypt's implementation of ECDSA 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, Libcrypt's field arithmetic operations are not implemented in 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 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, 73] and DSA [63], or

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

## 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

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

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

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

## 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

### ABSTRACT

In recent years, applications increasingly adopt security primitives designed with better countermeasures against side channel attacks. A concrete example is Libcrypt's implementation of ECDSA 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, Libcrypt's field arithmetic operations are not implemented in 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 reconstruct the internal workings of the victim. Because microarchitectural attacks execute on the processor as the victim, the attacker can observe the execution event timings if the execution cycles are separated by ten target key-dependent operations or in their usual timing. This attack usually targets the squaring operation in RSA [6].

### 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 must resist offline dictionary attacks in which an attacker exhaustively enumerates

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

### KEYWORDS

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

### ACM Reference Format:

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

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

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

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

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

## 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

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

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 Libcrypt's implementation of ECDSA 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, Libcrypt'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 reconstruct the internal workings of the victim. Because microarchitectural attacks execute on the processor as the victim, the attacker can observe the execution event timings if the execution cycles are separated by ten target key-dependent operations or more.

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 presentations 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

## 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 Garcia  
Tampere University of Technology  
Tampere, Finland

Billy Bob Brumley  
Tampere University of Technology  
Tampere, Finland

### KEYWORDS

SRP; PAKE; password to g key. The 'sist offline numerates

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

### ACM Reference Format:

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



## 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

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

Yuval Yarom  
University of Adelaide  
yyal@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 Libcrypt'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, Libcrypt's field arithmetic operations are not implemented in 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.

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

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

Cesar Pereida García  
Tampere University of Technology  
Tampere, Finland  
cesar.pereida@tut.fi

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

Billy Bob Brumley  
Tampere University of Technology  
Tampere, Finland  
billybob@tut.fi

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

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

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

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

... called *nonce*, which is particularly sensitive that the nonces are kept in integer form over a certain integer range or reused during key

### Certified Side Channels

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

### 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 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 side-channel leakage (PVA) to cache-based attacks (LTA) to password to g key. The 'sist offline numerates

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, permuting 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 algorithms at the



ACM Reference Format

Daniel De Almeida Braga, Pierre-Alain Fouquet, et al.

## May the Fourth Be With You: A Micro-Attack on Several Real-World Applications

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

### ABSTRACT

In recent years, applications increasingly designed with better countermeasures against side-channel attacks. A concrete example is LadderLeak, an encryption with Curve25519. Montgomery ladder is a fast, branchless Montgomery ladder implementation. LadderLeak implements a constant-time Montgomery ladder.

## Pseudorandom Black Swans: Cache Attacks on CTR\_DRBG

Shaanan Cohn<sup>1</sup>, Andrew Kwong<sup>2</sup>, Shahar Paz<sup>3</sup>, Daniel Genkin<sup>2</sup>, Nadia Heninger<sup>4</sup>, Eyal Ronen<sup>5</sup>, Yuval Yarom<sup>6</sup>

<sup>1</sup>University of Pennsylvania, shaananc@seas.upenn.edu  
<sup>2</sup>University of Michigan, {ankwong,genkin}@umich.edu  
<sup>3</sup>Tel Aviv University, shaharps@tau.ac.il  
<sup>4</sup>University of California, San Diego, nadiah@cs.ucsd.edu  
<sup>5</sup>Tel Aviv University and COSIC (KU Leuven), er@eyalro.net  
<sup>6</sup>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 implementation level, popular implementations of CTR\_DRBG use a non-constant-time PRNG module and NetBSD's kernel use leaky T-DES, a non-constant-time 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 numbers that collect inputs from various 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  
bbrumley@tut.fi

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

Diego F. Aranha  
DIGIT, Aarhus University  
Denmark  
dfaranh@acm.org

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

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

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

## Certified Side Channels

Cesar Pereira Garcia<sup>1</sup>, Sohaib ul Hassan<sup>1</sup>, Nicola Tuveri<sup>1</sup>, Jaroslav Gridin<sup>1</sup>, Alejandro Cabrera Aldaya<sup>1,2</sup>, and Billy Bob Brumley<sup>1</sup>

### Abstract

**Abstract** in which private keys are performed by software libraries, we investigate how to parse these keys, and what we can learn from them. In extreme cases inducing vulnerabilities, we demonstrate how to recover private keys from side-channel attacks utilizing signals emanating from granular side-channel attacks (EM) emanations, to granularly recover private keys. The side-channel attacks are implemented using a constant-time PRG module and NetBSD's kernel use leaky T-DES, a non-constant-time block cipher, enabling cache side-

ACM Reference Format:  
Daniel De Almeida Braga, Pierre-Alain Fouquet, and Sohaib ul Hassan. 2018. Certified Side Channels. In Proceedings of the ACM Conference on Computer and Communications Security (CCS '18), October 15–17, 2018, Toronto, Ontario, Canada. ACM, New York, NY, USA, 1–11. <https://doi.org/10.1145/3243746.3243754>



# So many attacks...

## LadderLeak: Breaking ECDSA

+ 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 ...

In  
five  
attacks  
(entropy)  
Montg  
fied, bra  
ments a c  
LibGcryp  
const

cularly sensi-  
cept in secret  
ertain integer  
sed or reused  
-ring key



Shaanan Cooney<sup>1</sup>, ...  
<sup>2</sup>University of ...  
<sup>3</sup>Tel Aviv University ...  
<sup>4</sup>University of California, San Diego, ...  
<sup>5</sup>Tel Aviv University and COSIC (KU Leuven), ...  
<sup>6</sup>University of Adelaide and Data61, yval@cs.adelaide.edu.au

... de la Habana (CUJAE), Habana, Cuba  
aldaya@gmail.com  
...ert, ianoslav.gridin, billy.bramley}@tuni.fi

**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 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 implementation level, popular implementations of CTR\_DRBG such as FIPS module and NetBSD's kernel use leaky T-DES-like underlying 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.

### Abstract

Format in which private keys are per-  
software libraries, we investigate  
key parse these keys, and what  
to uncover a combination of  
extreme cases inducing  
arithmetic stacks deep  
that otherwise seem  
vulnerabilities, we de-  
very attacks utilizing signals  
-etic (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, permuting 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 algorithms at the

Billy Bob Brumley  
Tampere University of Technology  
Tampere, Finland

### ACM Reference Format

Daniel De Almeida Braga, Pierre-Alain Fouquet, and Billy Bob Brumley

So many attacks...

LadderLeak: Breaking ECDSA

+ 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 ...

In  
tive  
acks  
enry)  
Montg  
fied, bra  
ments a c  
Libgcryp  
const

cularly sensi-  
cept in secret  
ertain integer  
sed or reused  
-ving key



Shaanan Cooney<sup>1</sup>, ...  
<sup>1</sup>University of ...  
<sup>2</sup>Tel Aviv University ...  
<sup>3</sup>University of California, San Diego, ...  
<sup>4</sup>University of ...  
<sup>5</sup>Tel Aviv U  
<sup>6</sup>Universit

Abstract  
... is an algorithm  
... sequence  
... de la Habana (CUJAE), Habana, Cuba  
aldaya@gmail.com  
... bert, ianoslav.gridin, billy.bramley}@tuni.fi

So. Many. Attacks.

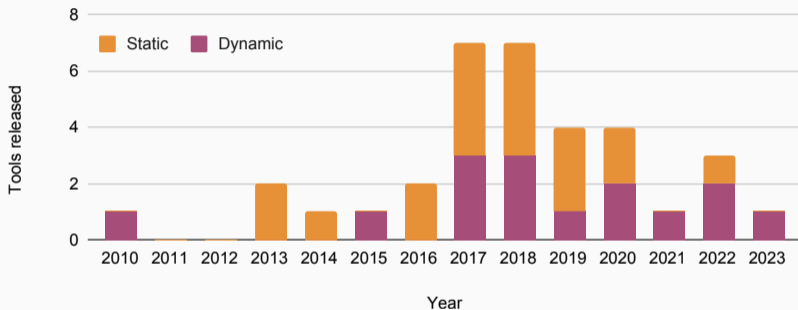
Abstract—Modern cryptography re-  
curely generate pseudorandom non-  
decades of work on side-channel attacks, there  
of their application to pseudorandom number generators  
In this work we set out to address this gap, empirically evaluating  
the side channel resistance of common PRG implementations.  
We find that hard-learned lessons about side channel leakage  
from encryption primitives have not been applied to an attacker  
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  
implementative level, popular implementations of CTR\_DRBG such  
as FIPS module and NetBSD's kernel use leaky T-  
state compromise. ...an@tut.fi

Billy Bob Bramley  
Tampere University of Technology  
Tampere, Finland

e multitude of standardized cryptographic key formats to  
ose from when persisting keys: which one to choose, and  
u the choice matter? Surprisingly, it does—we demon-  
e different key formats trigger different behavior within  
ware libraries, permuting 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 se-  
curity? Surprisingly, it does. We demonstrate that omitting  
optional parameters can cause extremely different execution  
flows deep within a software library, and also that  
verminally mathematically identical at the

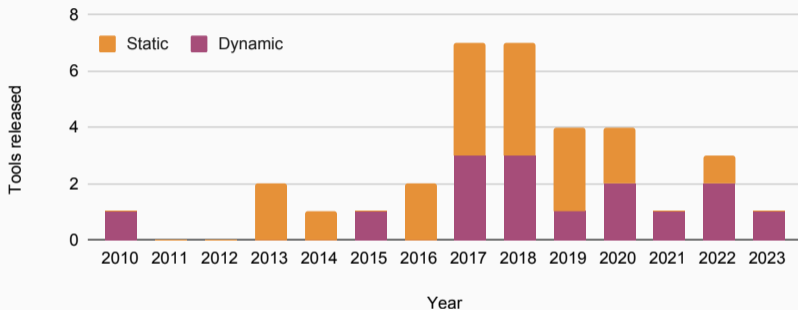
ACM Reference Format  
Daniel De Almeida Braga, Pierre-Alain Fouquet, ...

## 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?

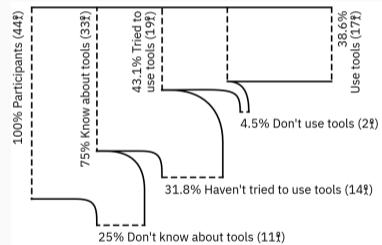


Many tools published from 2017, 67% of tools are open source (23 over 34)

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



Would the tools **actually work** to automatically  
**find recent vulnerabilities?**



# Research questions

RQ1 How can we **compare** these tools?

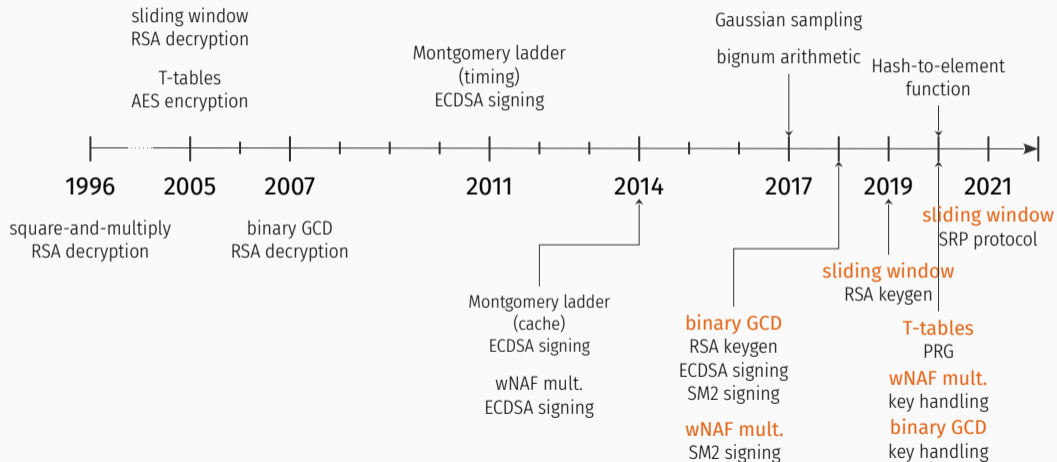
RQ2 Could an existing one have **detected** these vulnerabilities?

RQ3 What **features** might be missing from existing tools?

## Recent side-channel attacks

---

# Comparing recent vulnerabilities (2017-2022) with past vulnerabilities



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

## New contexts:

- Key generation [AsiaCCS 2018]
- Key parsing and handling [USENIX Sec 2020, S&P 2019]
- Random number generation [S&P 2020]

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

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

### New libraries

- MbedTLS sliding window RSA implementation [DIMVA 2017]
- Bleichenbacher-like attacks in MbedTLS, s2n, or NSS [S&P 2019]

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

Most vulnerabilities stem from code  
already known to be vulnerable

# Side-channel vulnerability detection tools

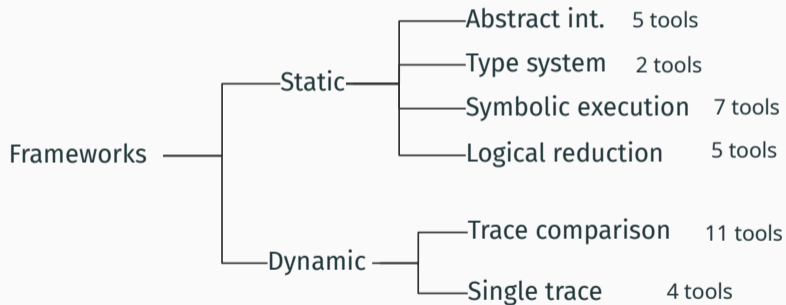
---

# 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	DIFUZZ	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

## 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;
        }
    }
}
```

## 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
        }
    }
}
```

## 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
```

## 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
```

## 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
```

# Benchmarks

---



# 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)
<b>RSA valid. (MbedTLS)</b>				490.01	✓	0.40	✓	278.94
GCD				37.74		0.21	✓	22.96
modular inversion				242.10	✓	0.24	✓	141.82
<b>RSA keygen (OpenSSL)</b>		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**

## Recommendations

---

**#1** Support for vector instructions

# Recommendations

#1 Support for vector instructions

#2 Support for indirect flows

# Recommendations

**#1** Support for vector instructions

**#2** Support for indirect flows

**#3** Support for internally generated secrets (e.g. key generation)



# Recommendations

**#1** Support for vector instructions

**#2** Support for indirect flows

**#3** Support for internally generated secrets (e.g. key generation)

**#4** Promote usage of a standardized benchmark

# Recommendations

**#1** Support for vector instructions

**#2** Support for indirect flows

**#3** Support for internally generated secrets (e.g. key generation)

**#4** Promote usage of a standardized benchmark

**#5** Improve usability for static tools (e.g. core-dump initialization)

# Recommendations

**#1** Support for vector instructions

**#2** Support for indirect flows

**#3** Support for internally generated secrets (e.g. key generation)

**#4** Promote usage of a standardized benchmark

**#5** Improve usability for static tools (e.g. core-dump initialization)

**#6** Make libraries more static analysis friendly

## Perspectives & Conclusion

---

Side-channel free software, are we there yet?

Nope!

Other microarchitectural vulnerabilities:

- transient execution, e.g., Spectre, LVI
- data memory-dependent prefetchers, e.g., GoFetch
- dynamic voltage and frequency scaling (DVFS), e.g., Hertzbleed

→ code that is "constant-time" (and considered secure until recently) can be vulnerable too!

# Conclusion

- first paper by Kocher in 1996: 25 years of research in this area
- so many detection tools, yet, **so many vulnerabilities** (manually) found
- most vulnerabilities stem from code already known to be vulnerable
- we introduced a **benchmark** for fair tool comparison
- we identified limitations in the current literature and issued **recommendations** for the community

 <https://github.com/ageimer/sok-detection/>

# 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 classification we offer compares multiple criteria, including the methods used, the scalability of the analysis or the threat model considered.

## 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 particular interest, as it is a shared resource between multiple programs. By observing the target execution through microarchitec-



# Thank you!

Contact

 `clementine.maurice@cncrs.fr`

 @BloodyTangerine

# Side-channel-free software, are we there yet?

---

Clémentine Maurice, CNRS, CRIStAL

17 June 2024—MPI-SP Symposium