All paths lead to Rome: Polymorphic Runtime Code Generation for Embedded Systems

CS2 2018 – Manchester
2018-01-24

Damien Couroussé, Thierno Barry, Bruno Robisson, Nicolas Belleville, Philippe Jaillon, Olivier Potin, Hélène Le Bouder, Jean-Louis Lanet, Karine Heydemann

Damien Couroussé, CEA – LIST / LIALP; Grenoble Université Alpes
damien.courousse@cea.fr

PROJET FINANCÉ PAR L'ANR

CEA: French Atomic & Alternative Energies Commission

- » 15 700 employees
- » 10 research centers
- » Budget of 3.9 billion €
- » 580 patents/year
- » 4000 publications/year
- » 120 startup created since 1984

Low Carbon Energy

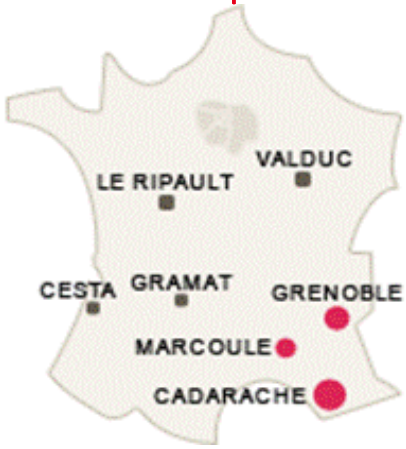
DEN

Defense & Security

DAM

Health & Information Technologies

DRT



Fundamental research

DSM

©2018 CEA France. All rights reserved

CEA TECH DACLE

leti Grenoble

list Saclay

DACLE
Architectures, IC Design & Embedded Software Division

300 members
160 permanent researchers

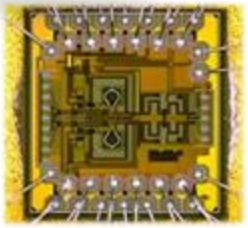
60 PhD students & postdocs

> 150 scientific papers per year

45 patents per year

Digital design

Programming



Analog & MEMs

Signal processing

Imaging

Test

PhD and post-doc offers:
http://www-instn.cea.fr/formations/formation-par-la-recherche/doctorat/liste-des-sujets-de-these.html
http://www-instn.cea.fr/formations/formation-par-la-recherche/post-doctorat/liste-des-propositions-de-post-doctorat.html

©2018 CEA France. All rights reserved

FROM RESEARCH TO INDUSTRY
cea tech

NOT SECURED!

SECURITY & FRAUD

At-Home Holes Th

By PYMNTS

Posted on April 14

Ransomware

BY GRAHAM CLULEY POSTED 27

CYBERCRIME

Worldwide IoT ~~Security~~ Cyber-Security Spending Forecast

(Millions of Dollars)

SOURCE: GARTNER (APRIL 2016)

...er be Secure

FACEBOOK

...ut on by the National Highway Traffic S...
...Washington, DC, surfaced a wide range ...
...car industry. One clear takeaway from ...

...will be a major step forward for the aut...
...problem by setting up an ISAC (Informa...
...ith the Alliance for Automobile Manufa...
...ne stiff challenges in an industry consist...

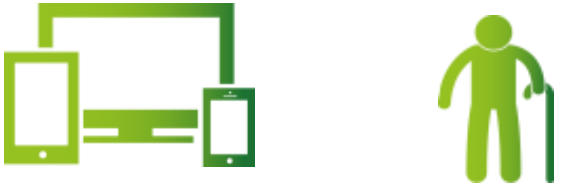
...participated on multiple panels at yesterday's ...

BESTIARY OF EMBEDDED SYSTEMS

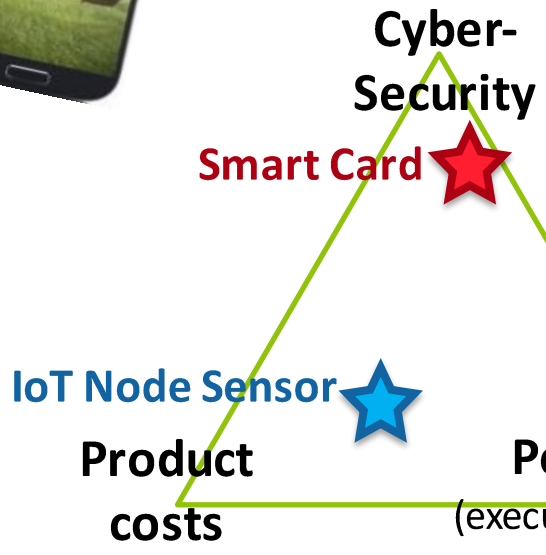
… IN NEED FOR ~~SECURITY~~ Cyber-Security CAPABILITES

Smart Card

Secure Element inside…



… And many other things



… SUPPORTED BY TOOLS

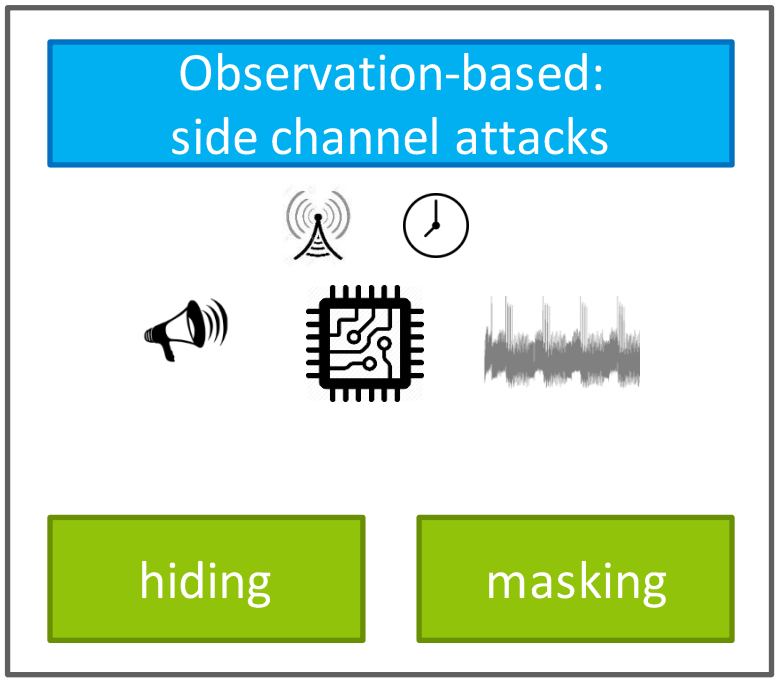
- flexibility
- automated application of protections
- deploy at large scale

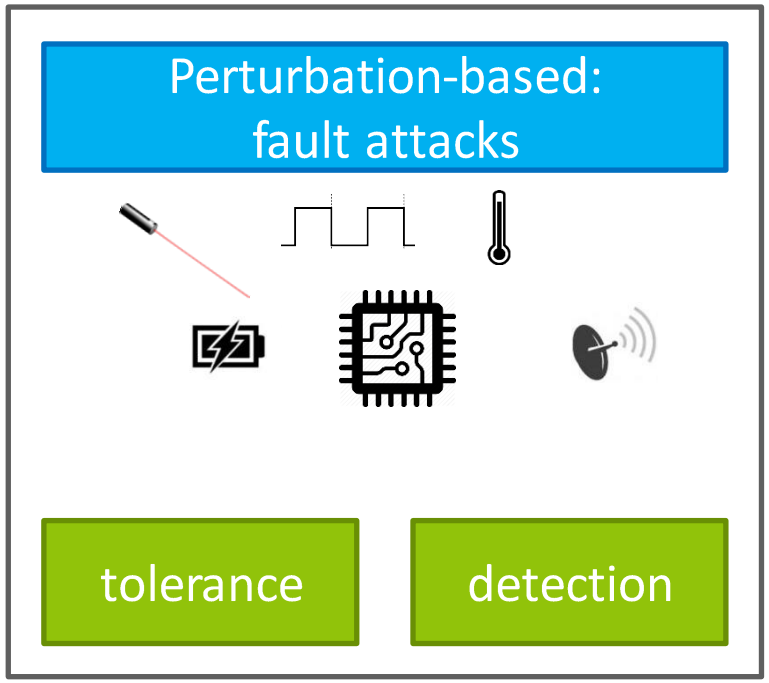
©2018 CEA France. All rights reserved

PHYSICAL ATTACKS

One of the major threats against secure embedded systems

- The only effective class of attacks against implementations of cryptography
- Relevant in many cases against cyber-physical systems: bootloaders, firmware upgrade, reverse-engineering, user authentication, etc.

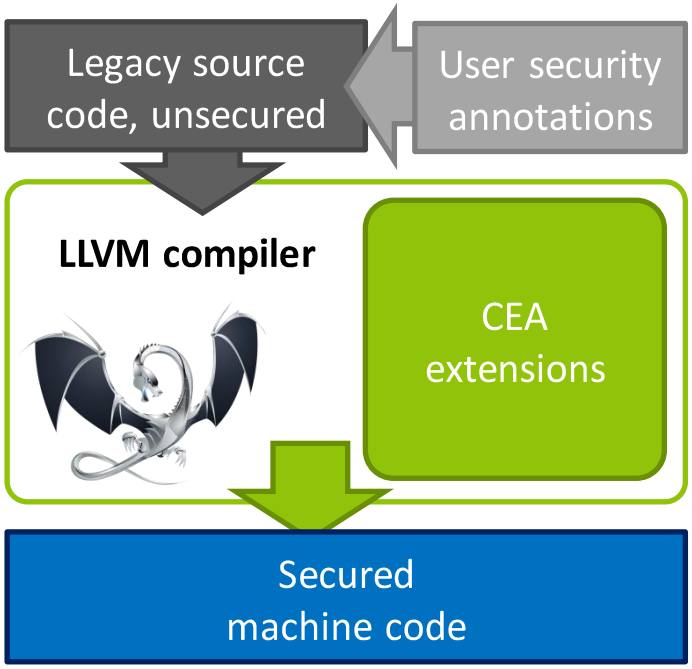




©2018 CEA France. All rights reserved

Automated application of software countermeasures against physical attacks

=> A toolchain for the compilation of secured programs



- Countermeasures supported:
  - **Fault tolerance**, including multiple fault injections
  - **Fault detection**
  - **Control-Flow Integrity**
    - Combined with integrity of execution pathes at the granularity of a single machine instruction
  - **Polymorphism**
- **LLVM**: an industry-grade, state-of-the art compiler (competitive with GCC)

©2018 CEA France. All rights reserved

HIDING

Information leakage: information related to secret data and secret operations "sneaks" outside of the secured component (via a side channel)

Hiding: "reducing the SNR", where

- Signal -> information leakage
- Noise -> everything else

- Temporal dispersion: spread leakage at different computation times
  - Shuffle independent operations
  - Insert «dummy» operations to randomly delay the secret computation
- Spatial dispersion:
  - Move the leaky computation at different places in the circuit
    - E.g. use different registers
  - Modify the "appearance" of information leakage
    - E.g. use different operations

In practice, a secured product combines masking and hiding countermeasures.

©2018 CEA France. All rights reserved

SOFTWARE PROTECTIONS BASED ON HIDING

The many ways to reach the same program results!

- **Insertion of dummy operations**
  - Same computation, with fake data
  - Random delays
  - Broken by recent filtering or re-synchronization attacks
- **Multi-versionning**
  - Increases the code size,
    - also increases the attack surface
  - The path can be computed from an authentication key
    - Code traps can detect malicious usages
- **Polymorphic runtime code generation**
  - Opens the door to many code transformation opportunities, e.g. random allocation of registers
  - The protected code is not available before runtime, i.e. for static analysis
  - The main overhead comes from runtime code generation
  - Can reduce the frequency of code generation to reduce the overhead

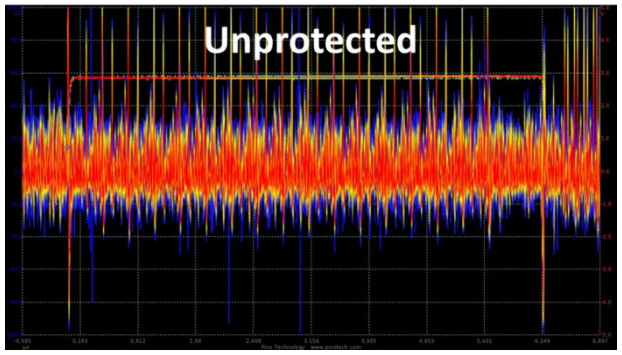
©2017 CEA France. All rights reserved

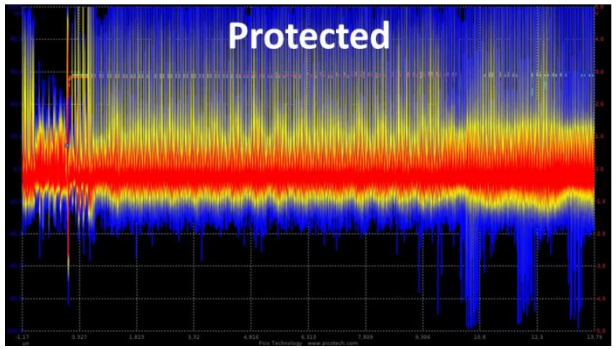
CODE POLYMORPHISM

Code polymorphism: regularly changing the behavior of a (secured) component, at runtime, while maintaining unchanged its functional properties, with runtime code generation

- **Protection against physical attacks: side channel & fault attacks**
  - polymorphism changes the spatial and temporal properties of the secured code
  - Can be combined with other state-of-the-Art HW & SW Countermeasures
- Implementation with runtime code generation

(patented)

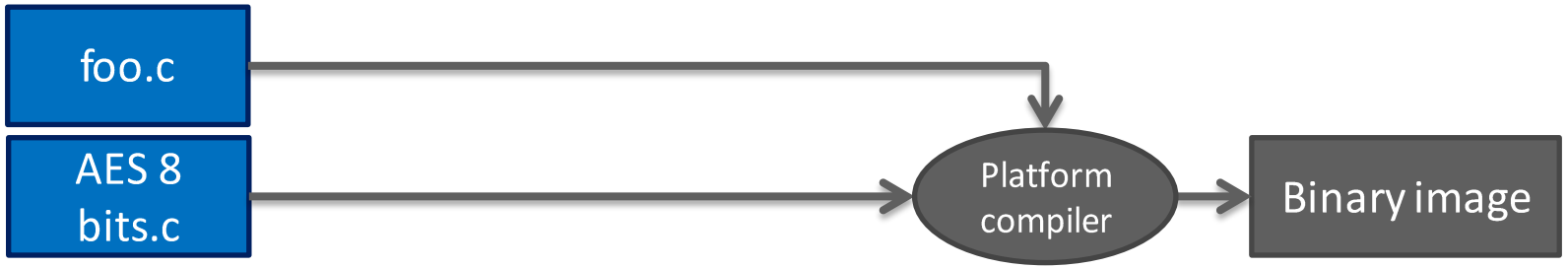




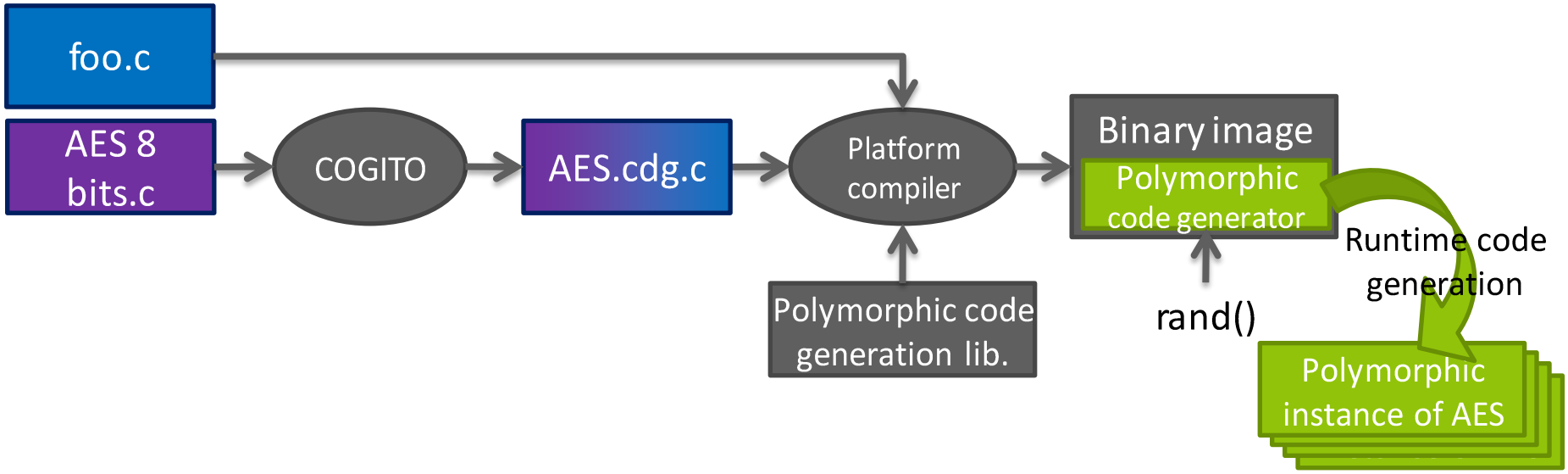
©2018 CEA France. All rights reserved

Runtime code generation for embedded systems

Reference version:



Polymorphic version, with COGITO:



©2018 CEA France. All rights reserved

AES 8-BIT. PERFORMANCE OVERHEAD

$$k = \frac{t_{\text{gen}} + \omega \times t_{\text{poly}}}{\omega \times t_{\text{ref}}}$$

k: performance overhead factor
ω: runtime code generation interval

	AddRoundKey			SubBytes			All round functions		
	k Min.	k Avg.	k Max.	k Min.	k Avg.	k Max.	k Min.	k Avg.	k Max.
ω=1	3.16	4.91	6.37	5.81	7.27	8.94	20.10	22.94	26.16
ω=10	1.32	1.50	1.66	1.59	1.76	1.92	3.86	4.36	4.85
ω=100	1.09	1.16	1.22	1.16	1.21	1.25	2.17	2.50	2.78
ω=1000	1.09	1.13	1.18	1.16	1.15	1.20	2.17	2.32	2.59
ω=10000	1.05	1.12	1.18	1.11	1.15	1.19	1.99	2.30	2.58

- **Variable performance results** according to
  - Settings of the polymorphic code generator
    - model of noise insertion
- Code is slower when executed in RAM (memory accesses)
- Room for performance improvements
  - The non-polymorphic generated code is slower than the reference

AES 8 BIT. MEMORY FOOTPRINT

	text	data	bss	total
Unprotected	4857	52	1168	6077
AddRoundKey only	8785	56	2980	11821
SubBytes only	7833	56	2980	10869
Full polymorphic	14913	56	6052	21021

x3.45

A POLYMORPHIC SUBBYTES

```
void subBytes_compilette (
     cdg_insn_t* code, unsigned char* sbox_addr, unsigned char * state_addr)
{
    #[
    Begin code Prelude

        Type uint32 int 32
        Alloc uint32 rstate
        Alloc uint32 rsbox
        Alloc uint32 rstatei
        Alloc uint32 rsboxi
        Alloc uint32 index

        mv rstate, #((unsigned int)state_addr)
        mv rsbox, #((unsigned int)sbox_addr)

        mv index, #(16)
        loop:
            sub index, index, #(1)
            lb rstatei, rstate, index     //statei = state[i]
            lb rsboxi, rsbox, rstatei     //sboxi = sbox[statei]
            sb rstate, index, rsboxi      //state[i] = sboxi
            bneq loop, index, #(0)

        rtn
    End
    ]#;
}
```

All the polymorphic code transformations are automatically handled by the toolchain:

- **Random register allocation**
- **Semantic variants**
- **Instruction shuffling**
- **Noise instructions**
- **Execution of loops in random order**

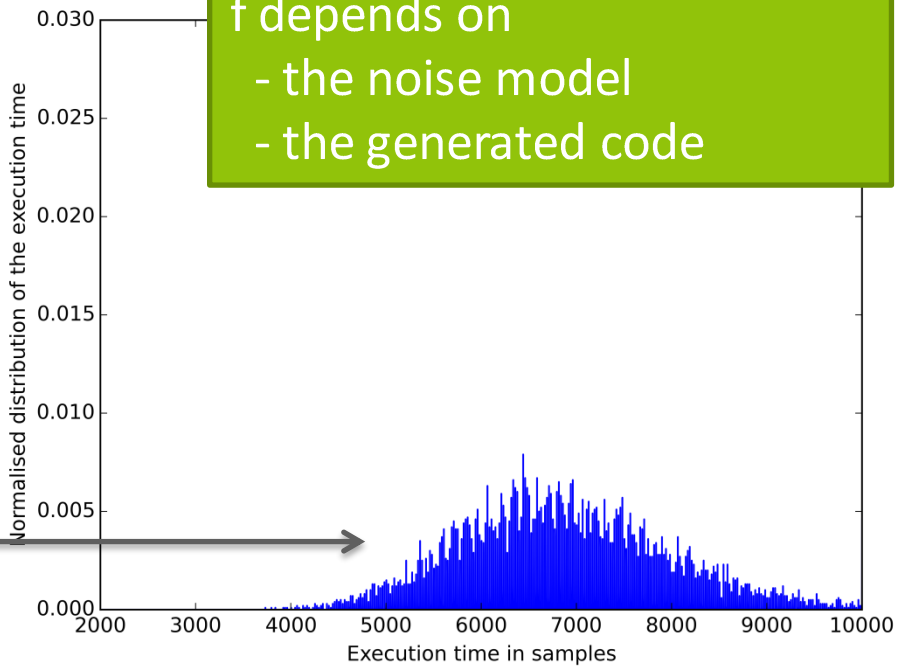
©2017 CEA France. All rights reserved

INSERTION OF NOISE INSTRUCTIONS

- Noise instructions have no effect on the result of the program
- Parametrable model of the inserted delay ~ program execution time
  - Goal:
    Maximize standard deviation **σ**
    Minimize mean **E**
- Can insert any instruction:
  - nop
  - Arithmetic (add, xor…)
  - *Memory accesses* (lw, lb, …)
  - Power hungry instructions
    (mul, mac…)
  - Etc.

N: number of insertions
(E, σ) = f(N)
f depends on
- the noise model
- the generated code

The leaky instruction is spread over ~300 CPU cycles



©2018 CEA France. All rights reserved

Finding a needle in a haystack…

- Relationship between the different components of power consumption:

$$P_{total} = P_{operations} + P_{data} + P_{noise}$$

$$P_{total} = P_{exploitable} + P_{switching.noise} + P_{electronic.noise} + P_{const}$$

needle haystack

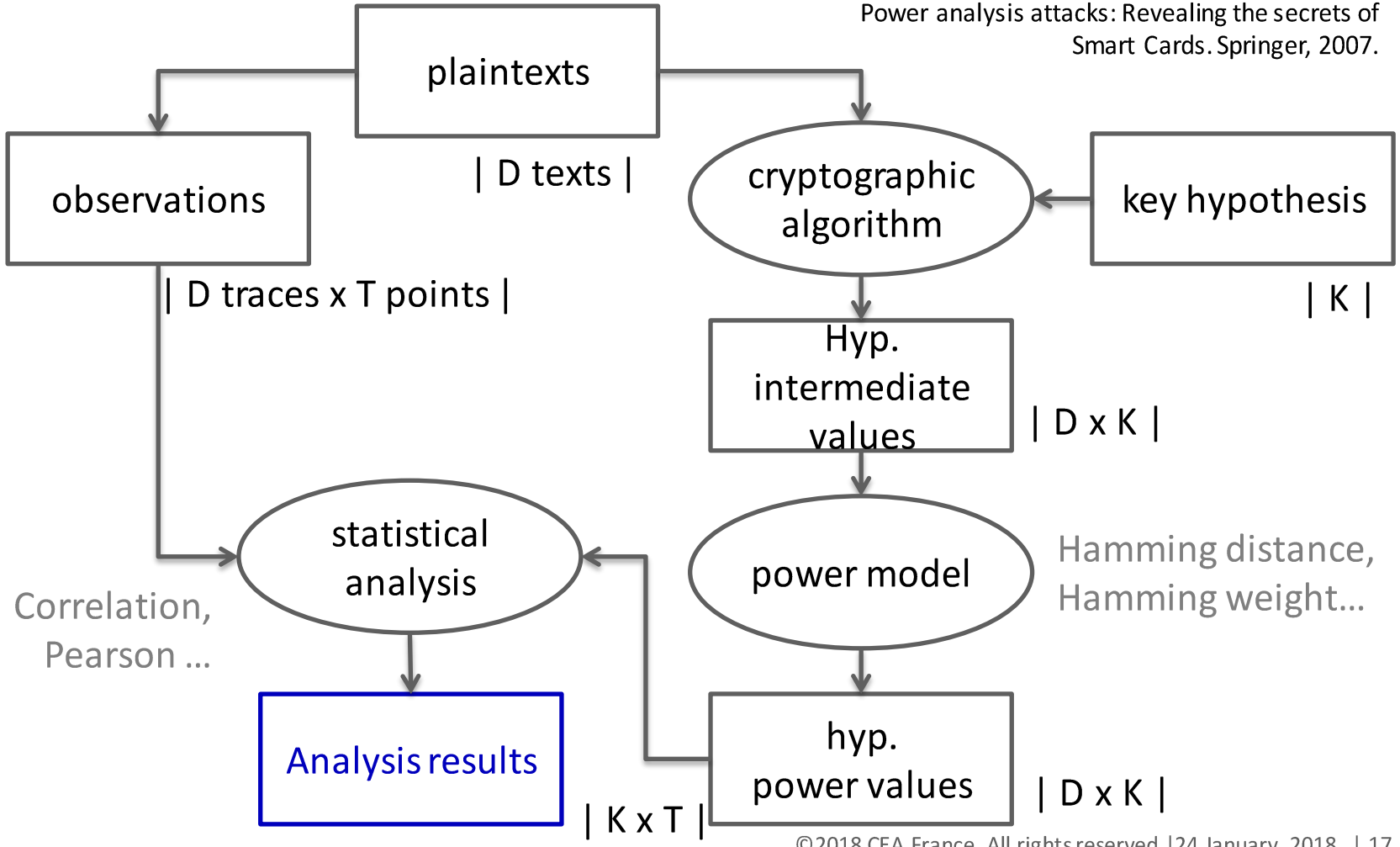
- Power signal: a static and a dynamic component.
  - Static component: power consumption of the gate states → a * HW(state)
  - Dynamic component: power consumption of transitions in gate states → b * HD(state[i], state[i-1])
- Other needles & stacks
  - Electromagnetic emissions
  - Execution time
  - Chip temperature
  - Etc.

©2018 CEA France. All rights reserved

CPA – ANALYSIS

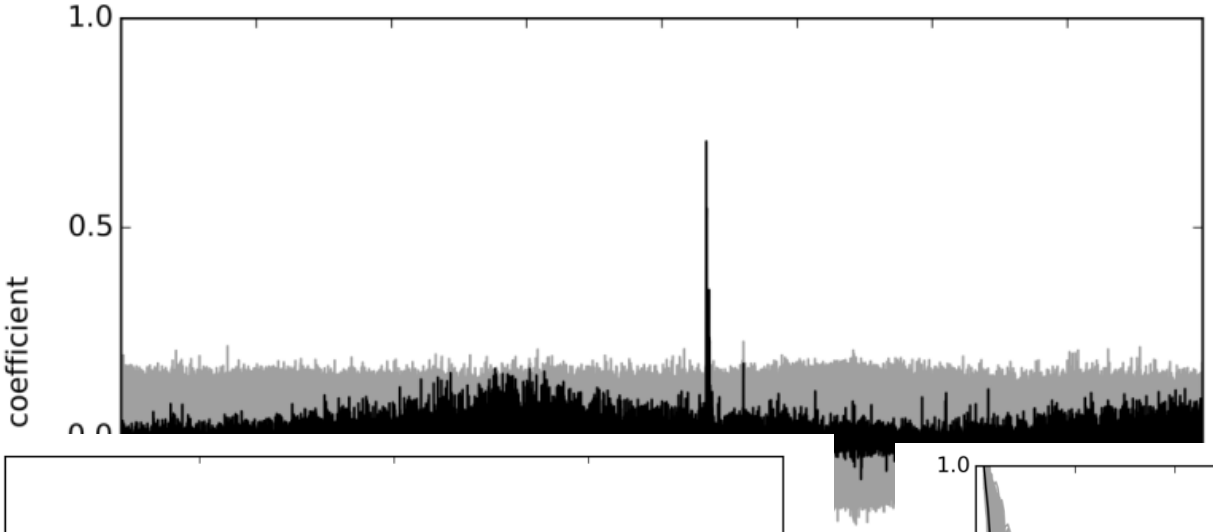
m: plaintext -> controlled by the attacker or observable
k: cipher key -> unknown to the attacker

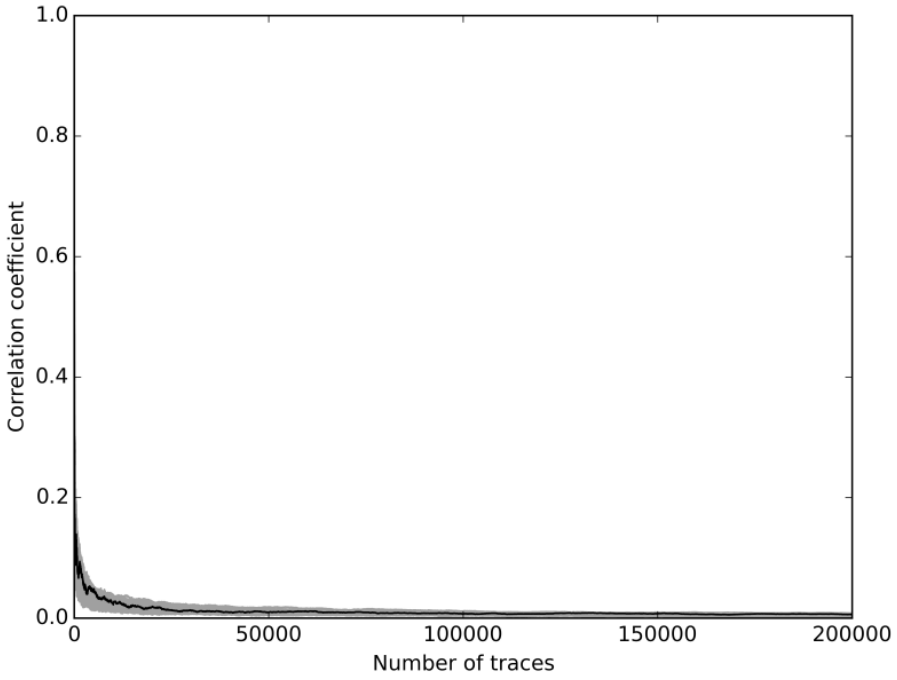
S. Mangard, E. Oswald, and T. Popp,
Power analysis attacks: Revealing the secrets of
Smart Cards. Springer, 2007.

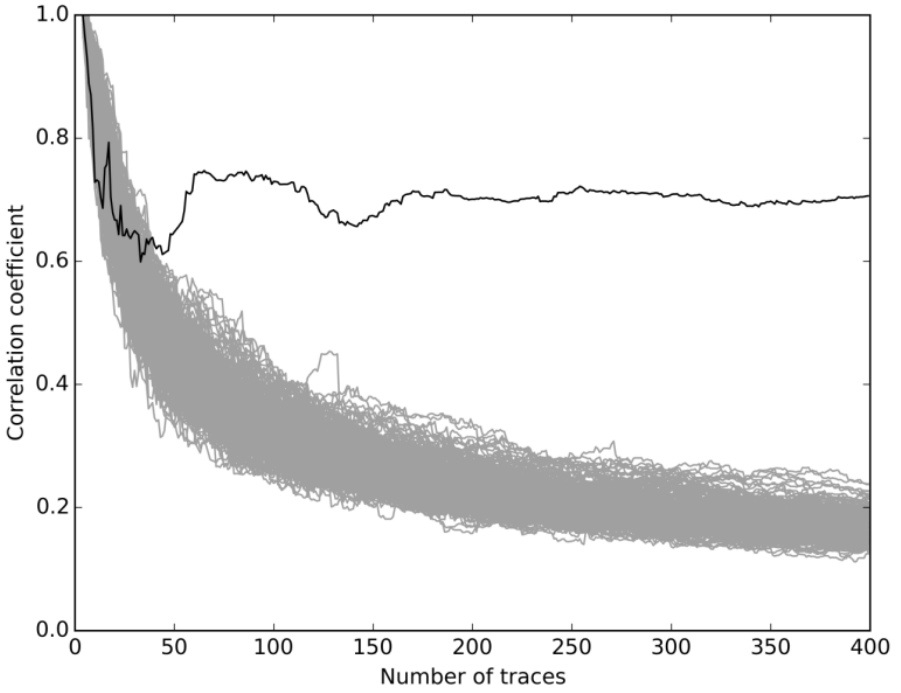


©2018 CEA France. All rights reserved

CPA RESULTS



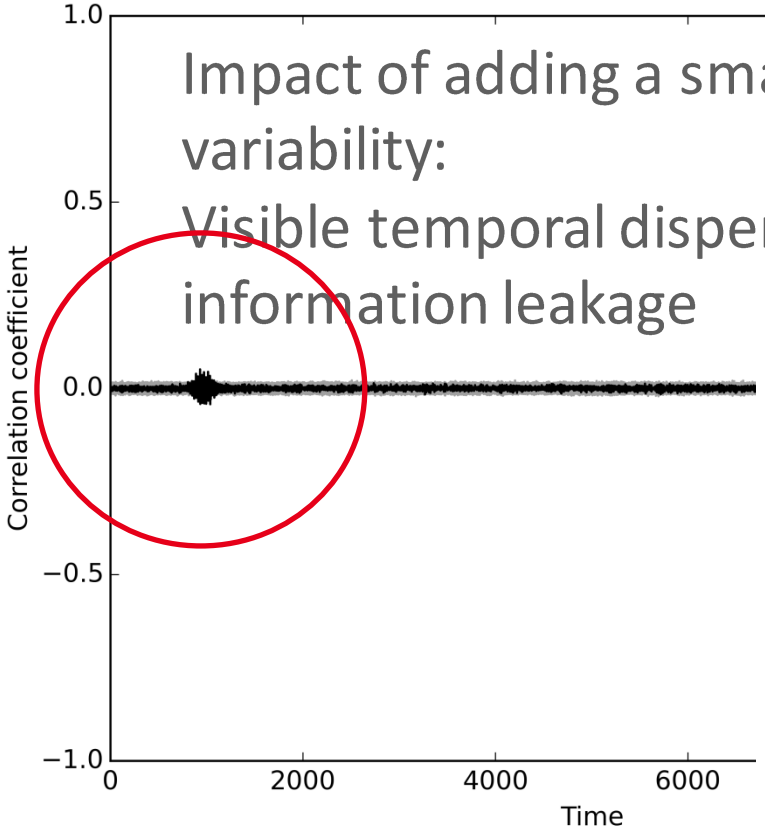


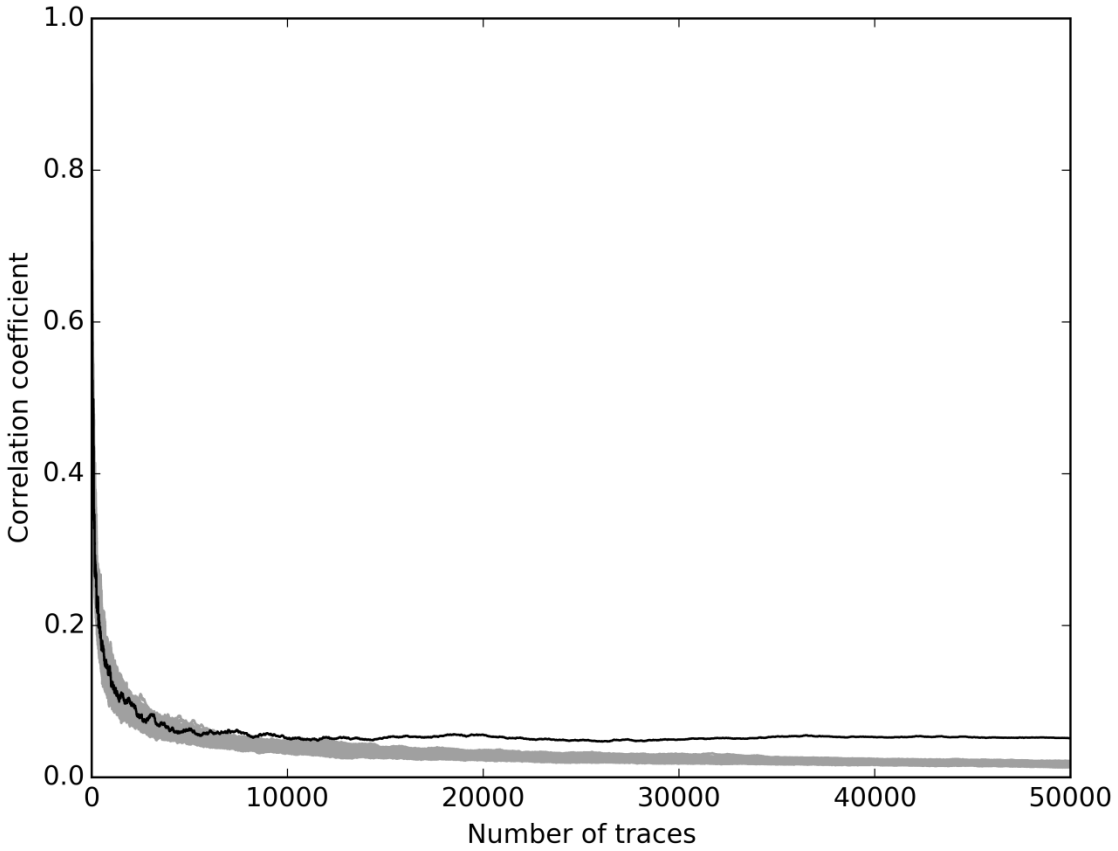


©2018 CEA France. All rights reserved

IMPACT OF POLYMORPHISM ON CPA

Impact of adding a small variability:
Visible temporal dispersion of information leakage



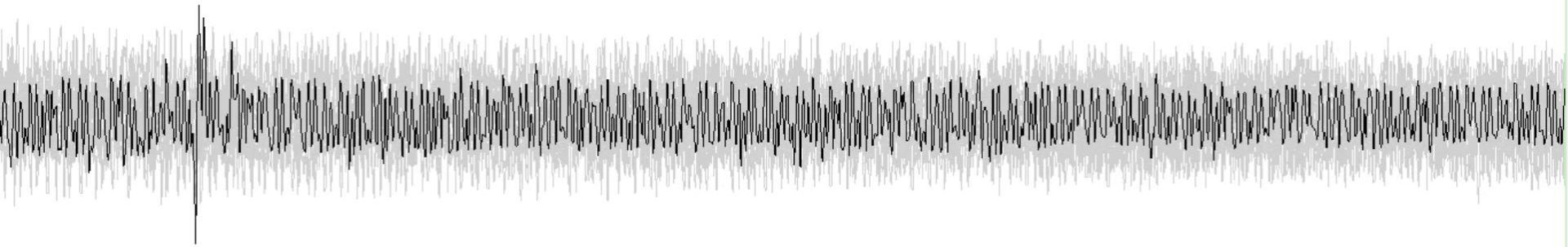


©2018 CEA France. All rights reserved

CPA RESULTS

Correlation(EM, key hypothesis)

Unprotected AES



Key found!

#265

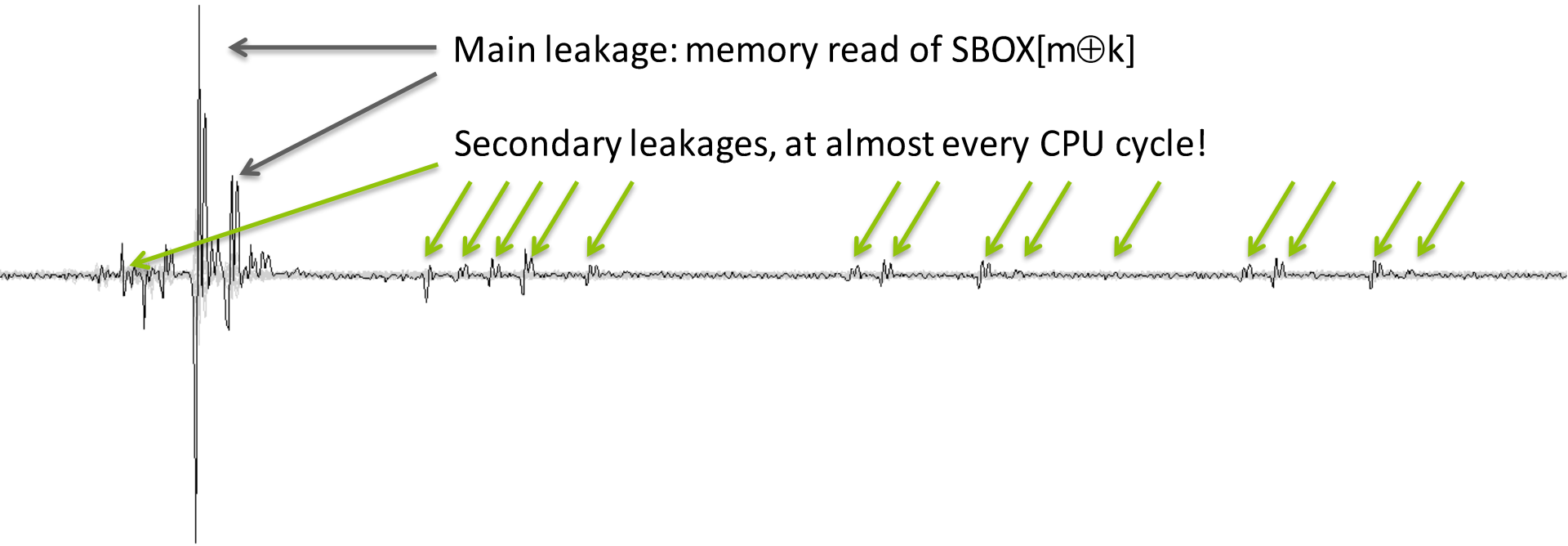
AES, unprotected implementation
EM traces
Attack on the output of the 1st SBOX

©2018 CEA France. All rights reserved |

CPA RESULTS

Correlation(EM, key hypothesis)

Unprotected AES



#121278

AES, unprotected implementation
EM traces
Attack on the output of the 1st SBOX

©2018 CEA France. All rights reserved

CPA RESULTS

Correlation(EM, key hypothesis)

#304667

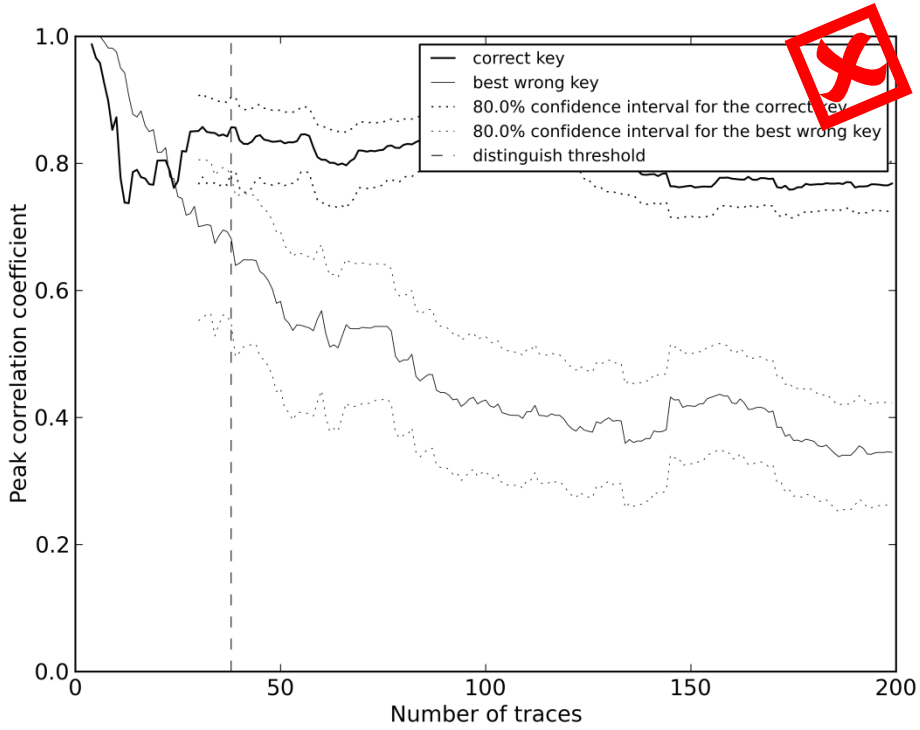
Same AES, **polymorphic implementation**
EM traces
Attack on the output of the 1st SBOX

©2018 CEA France. All rights reserved

IMPACT OF POLYMORPHISM ON CPA

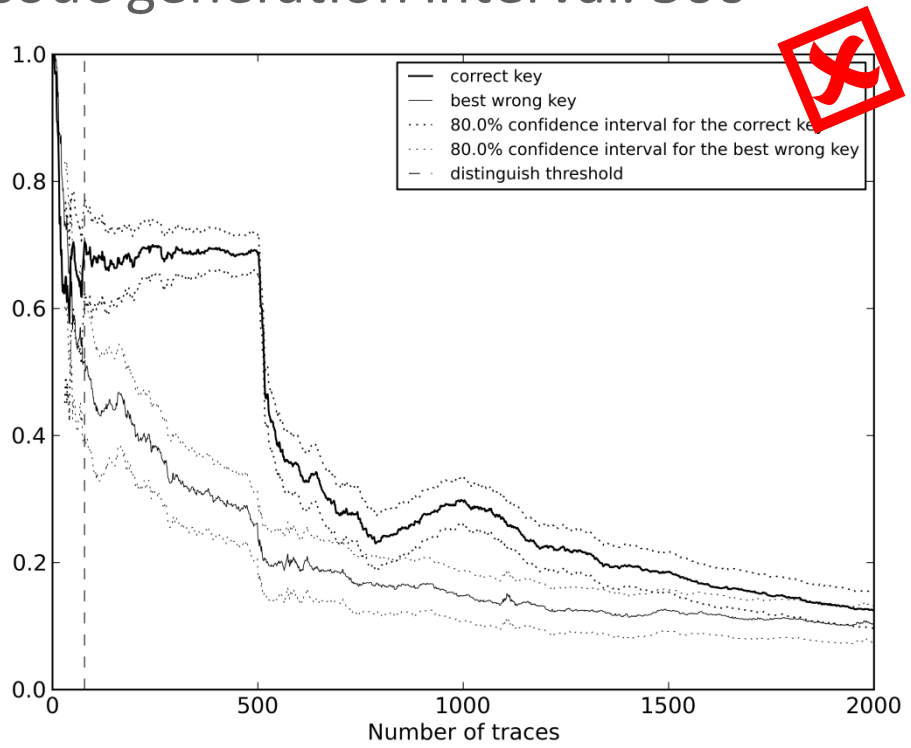
Effect of the code generation interval

Reference implementation



Distinguish threshold = 39 traces

Polymorphic version,
code generation interval: **500**

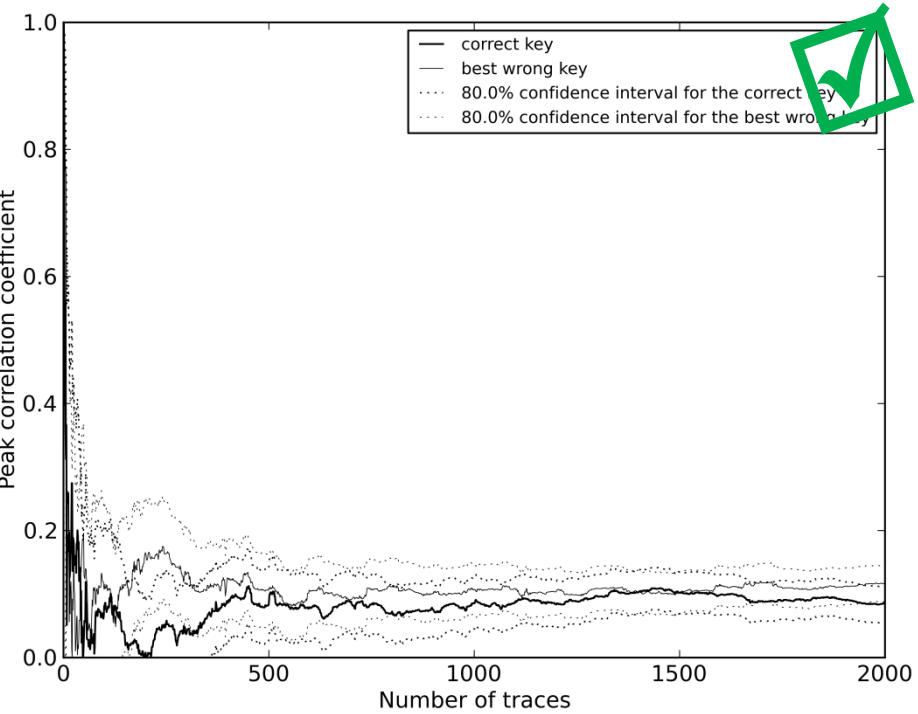


Distinguish threshold = 89 traces

©2018 CEA France. All rights reserved

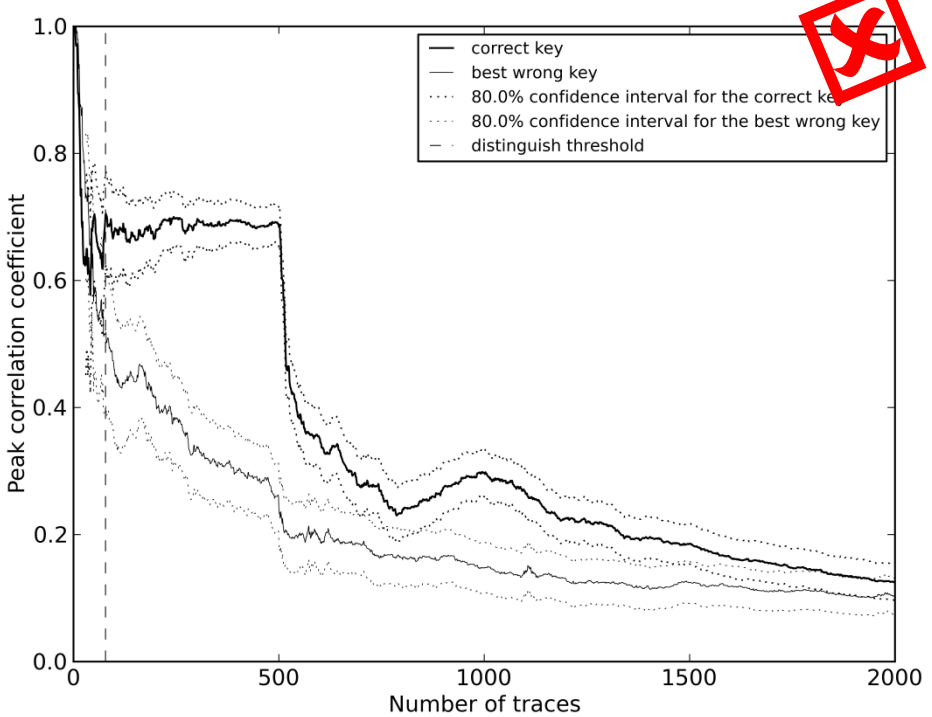
IMPACT OF POLYMORPHISM ON CPA

Polymorphic version
code generation interval: **20**



Distinguish threshold > 10000 traces

Polymorphic version,
code generation interval: **500**



Distinguish threshold = 89 traces

©2018 CEA France. All rights reserved

SECURITY EVALUATION

- **CPA / DPA … attacks do not constitute a security evaluation.**
- **Playing the role of the attacker is great, but the attacker**
  - is focused on a potential vulnerability
  - Follows a specific attack path
- **Starting from the previous attack, we could change**
  - The hypothetical intermediate values: output of 1st SubBytes, output of 1st AddRoundKey, input of the10th SubBytes…
  - The power model: Hamming Weight, Hamming Distance, no power model…
  - The distinguisher: Pearson Correlation, Mutual Information…
  - There are many other attacks!
- **Our evaluation target is very "leaky" (less than 1000 traces is enough)**
  - Unprotected components executed on more complex targets (i.e. ARM Cortex A9) will require 100.000 to 10^6 traces.
  - What about attacking a counter-measure in this case?
- **As a security designer, you need to cover all the possible attack passes**

©2018 CEA France. All rights reserved

TLVA: Test Leakage Vector Assessment

- Exploit Welch's t-test to assess the amount of information leakage
- Extract two populations of side-channel observations (traces)
- Test the null hypothesis: the two populations are not statistically distinguishable → no information leakage

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{{s_0}^2}{n_0} + \frac{{s_1}^2}{n_1}}},$$

t > 4.5 → confidence of 99.999% that the null hypothesis is rejected

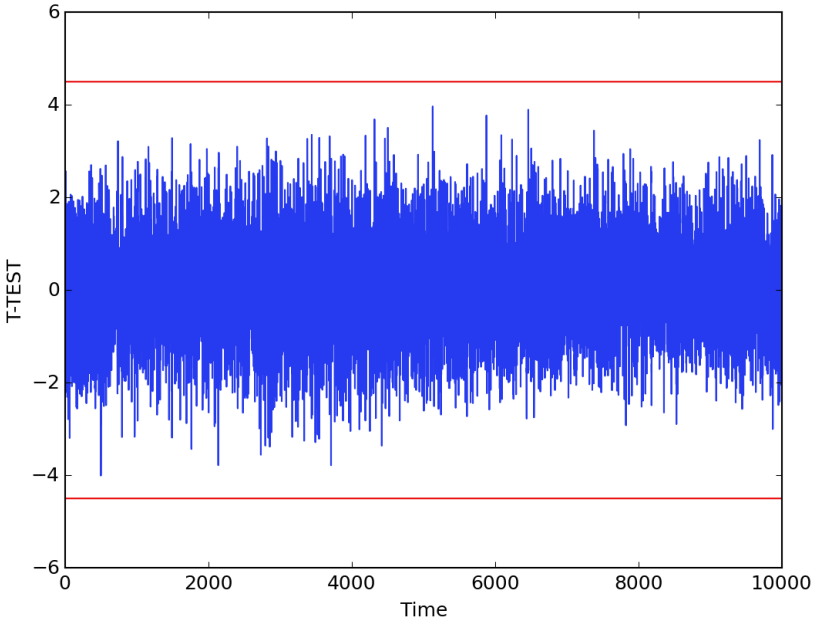
©2018 CEA France. All rights reserved

TLVA: Test Leakage Vector Assessment

- Exploit Welch's t-test to assess the amount of information leakage
- Extract two populations of side-channel observations (traces)
- Test the null hypothesis: the two populations are not statistically distinguishable → no information leakage

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{{s_0}^2}{n_0} + \frac{{s_1}^2}{n_1}}},$$

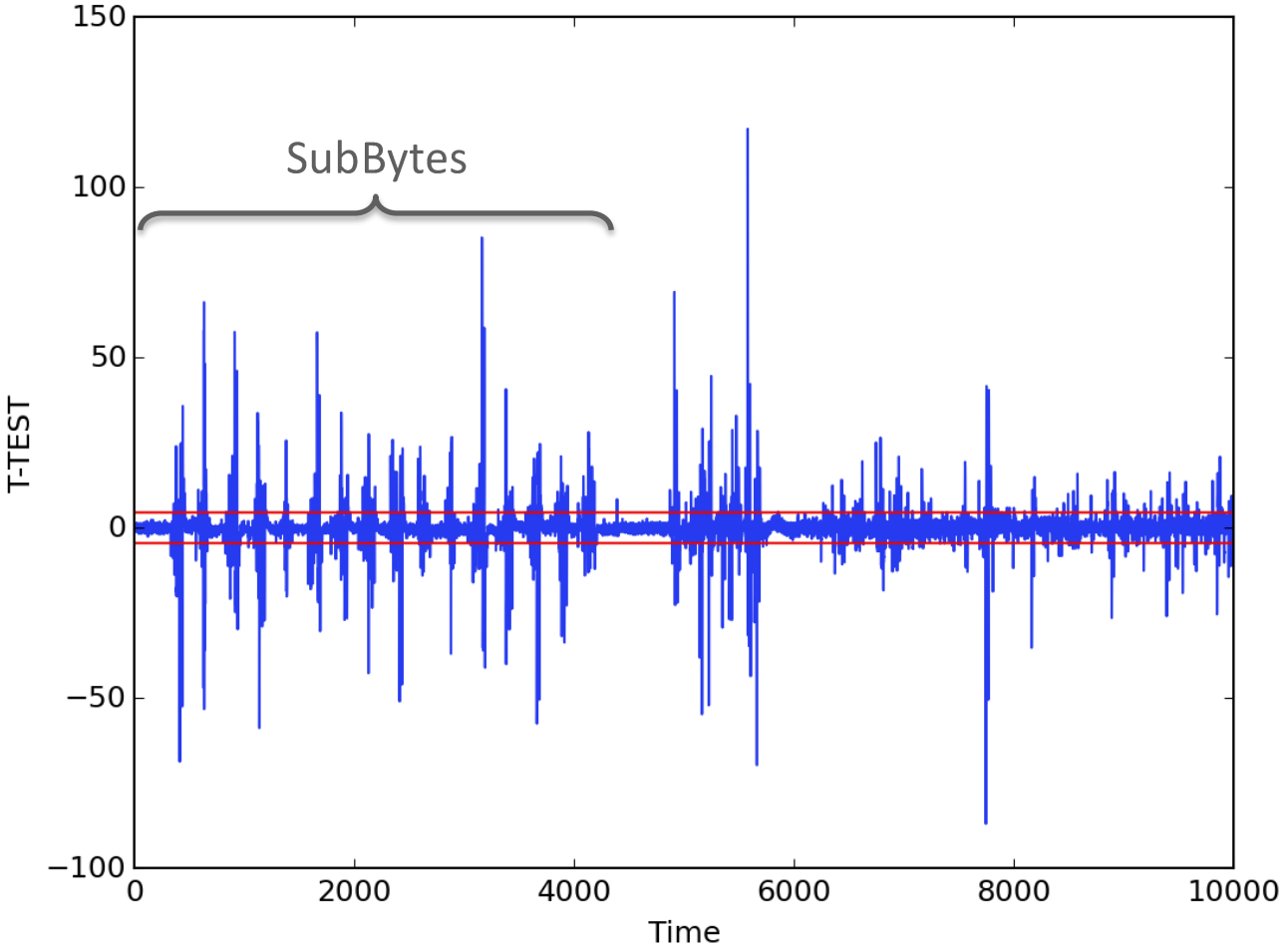
-> confidence o



©2018 CEA France. All rights reserved

NON-SPECIFIC T-TEST

Q0: fixed input plaintext
Q1: random input plaintext

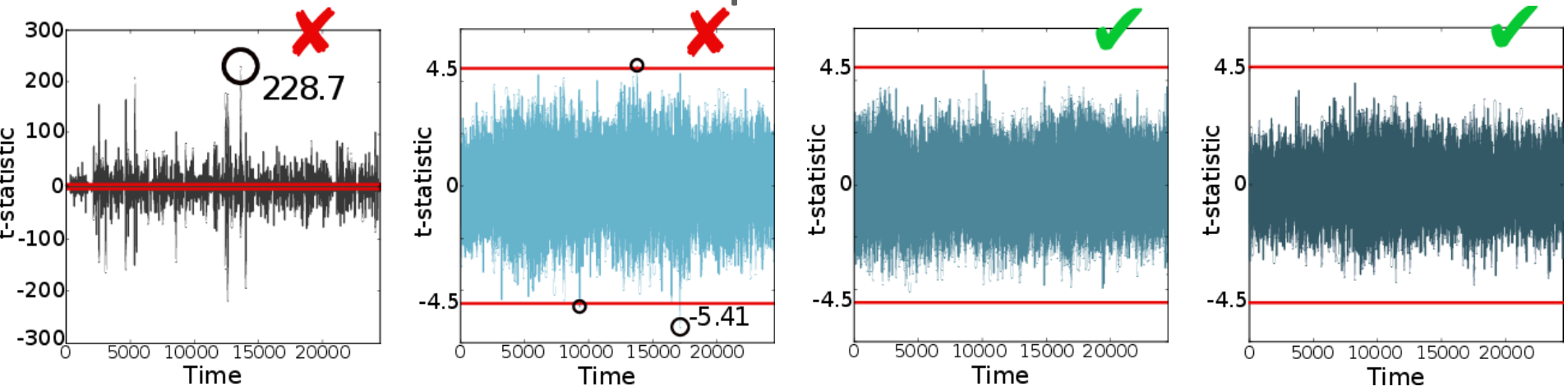


©2018 CEA France. All rights reserved

SECURITY EVALUATION

- Polymorphism is a *hiding* countermeasure against side-channel attacks
  - Does not *remove* information leakage; *reduces* SNR only
- The t-test is usually used to verify implementations of *masking*
- **With polymorphism, information leakage is sufficiently blurred such that it is *not found* in observation traces, with a confidence level of 99.999%**
- Configurable level of polymorphism for security-performance trade-offs

Non-specific t-test

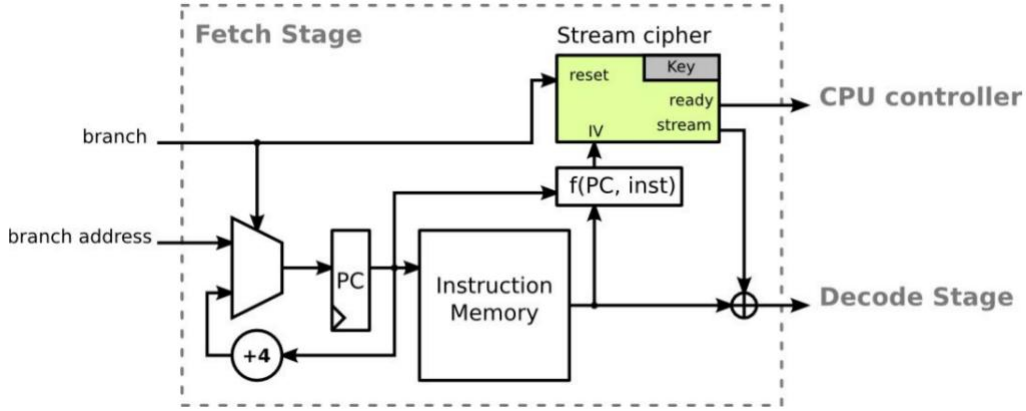


reference | "Low" variability | "Medium" variability | "High" variability

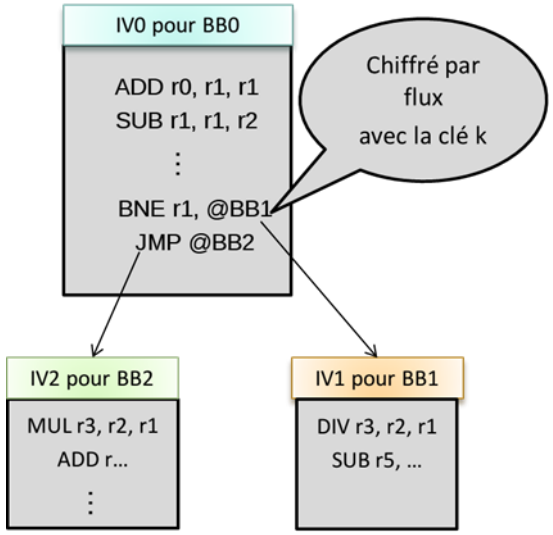
Attack complexity increasing

©2018 CEA France. All rights reserved

PROGRAM CONFIDENTIALITY

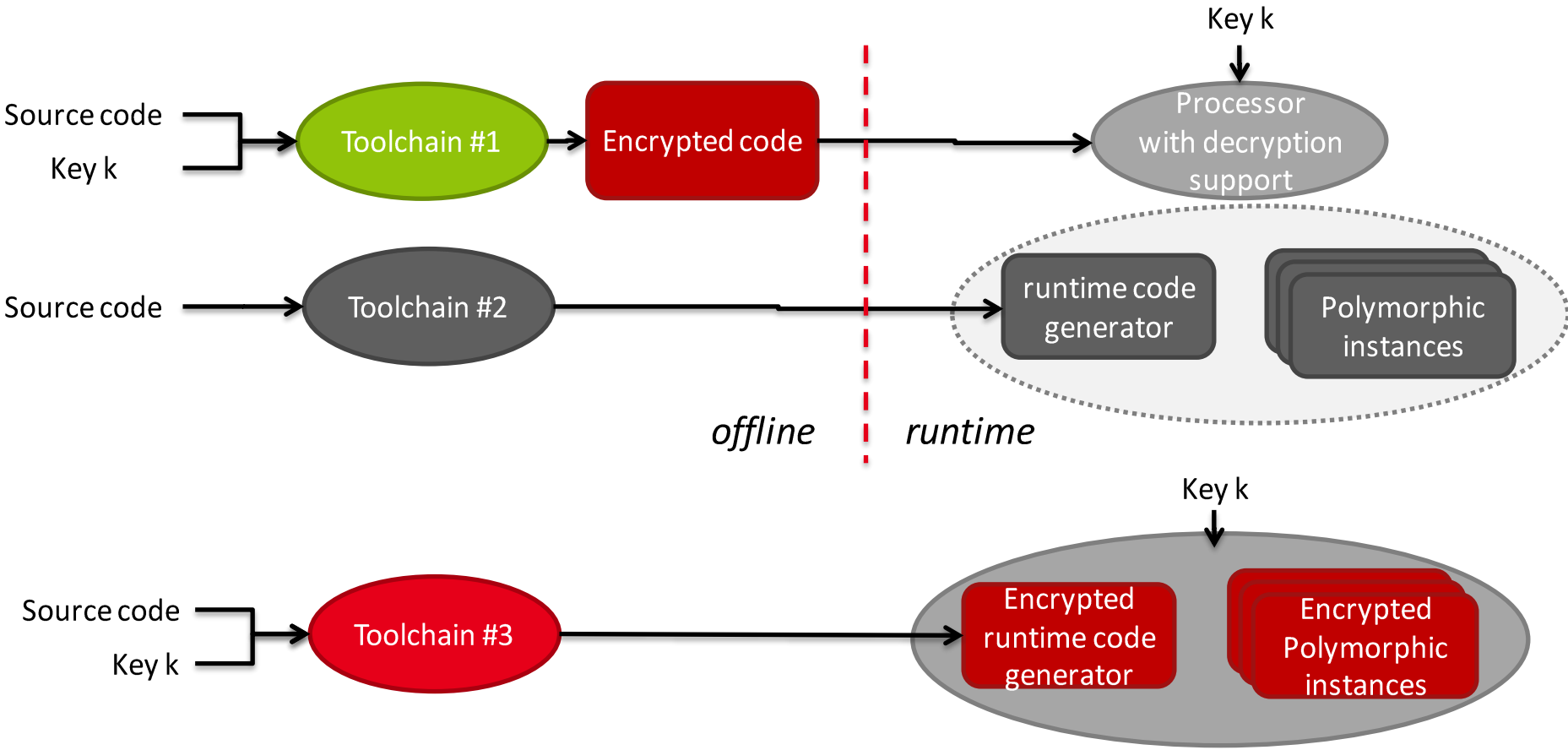


Hiscock, T., Savry, O. and Goubin, L. (2017) 'Lightweight Software Encryption for Embedded Processors' Euromicro DSD



	Side-channel attacks	Static reverse-engineering	Dynamic reverse (SCARE)
Encrypted program	✗	✓	✗
Code polymorphism	✓	✗	(✓)
Encrypted code polymorphism	✓	✓	✓

©2017 CEA France. All rights reserved



(patented)

©2017 CEA France. All rights reserved

BUILDING BETTER TOOLS FOR SECURITY

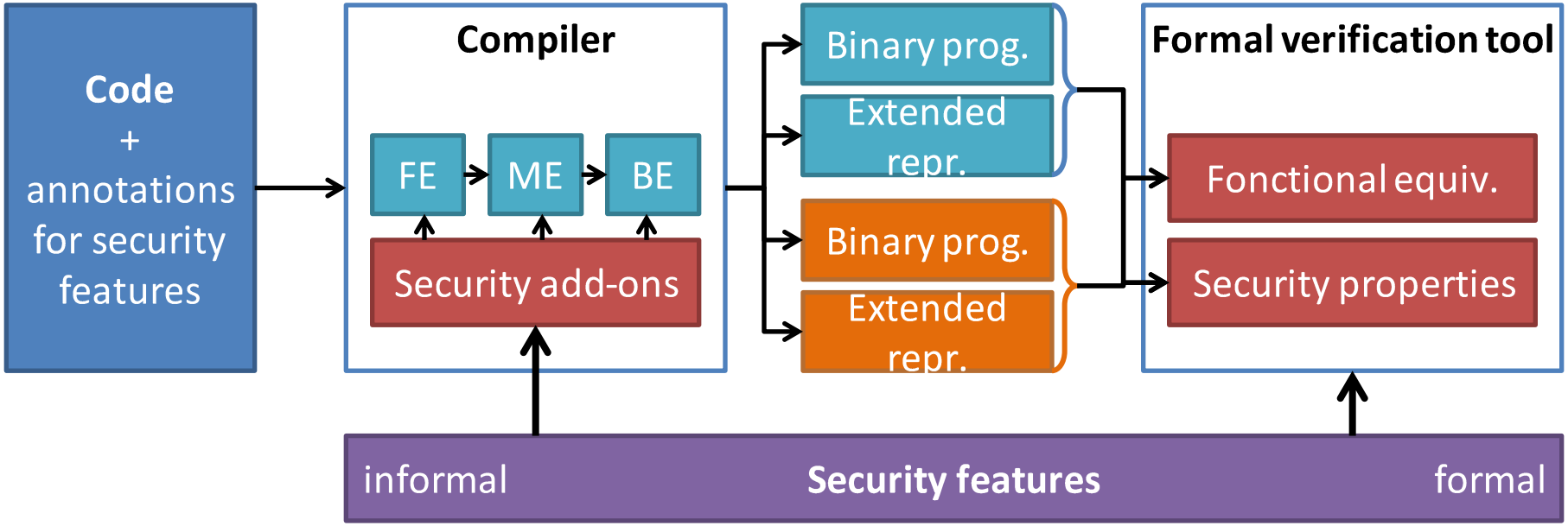
- **Resistance of polymorphism to side-channel attacks**
  - Hiding was shown to be an effective protection
    - … against textbook attacks
  - What about more recent attacks?
- **Mitigation of new vulnerabilities provided by runtime code generation**
  - W access to program memory! → X⊕W permissions
  - JIT spraying is not applicable to our implementation
- **Determinism and reproducibility**
  - Inherent to our implementation
- **Debug**
- **Functional validation**
- **Formal verification of the secured code**

©2017 CEA France. All rights reserved

BUILDING BETTER TOOLS FOR SECURITY

Formal verification of the secured code ➔ project PROSECCO

- **Compilation**: automation of the application of software countermeasures against fault attacks and side-channel attacks
- **Functional verification**: of the secured machine code (equivalence with an unprotected version of the same program)
- **Security verification**: correctness of the applied countermeasures w.r.t a security model



©2017 CEA France. All rights reserved

All paths lead to Rome: Polymorphic Runtime Code Generation for Embedded Systems

CS2 2018 – Manchester
2018-01-24

Damien Couroussé, CEA – LIST / LIALP; Grenoble Université Alpes
damien.courousse@cea.fr

leti
Centre de Grenoble
17 rue des Martyrs
38054 Grenoble Cedex

list
Centre de Saclay
Nano-Innov PC 172
91191 Gif sur Yvette Cedex

INSTITUT CARNOT CEA LETI
INSTITUT CARNOT CEA LIST