Side-Channel Attacks

ISSISP 2017 – Gif-sur-Yvette
2017-07-21

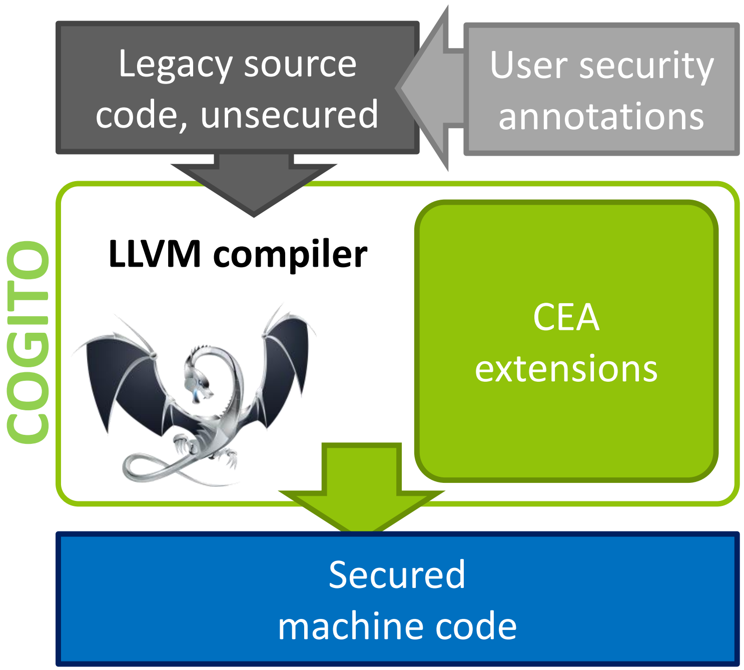
Damien Couroussé, CEA – LIST / LIALP; Grenoble Université Alpes
damien.courousse@cea.fr

COMPILATION OF COUNTER-MEASURES

CODE POLYMORPHISM

Automated application of software countermeasures against physical attacks

=> A toolchain for the compilation of secured programs



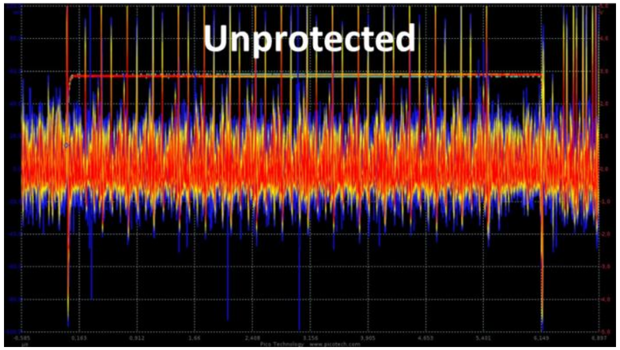
- Countermeasures supported:
  - **Fault tolerance**, including multiple fault injections
  - **Fault detection**
  - **Control-Flow Integrity**
    - Combined with integrity of execution pathes at the granularity of a single machine instruction
  - **Polymorphism**
- **LLVM**: an industry-grade, state-of-the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

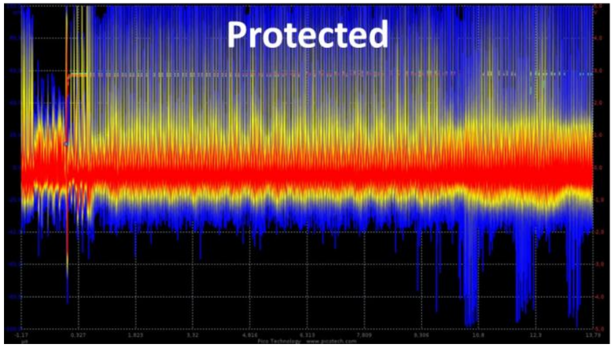
CODE POLYMORPHISM

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

(patented techno.)

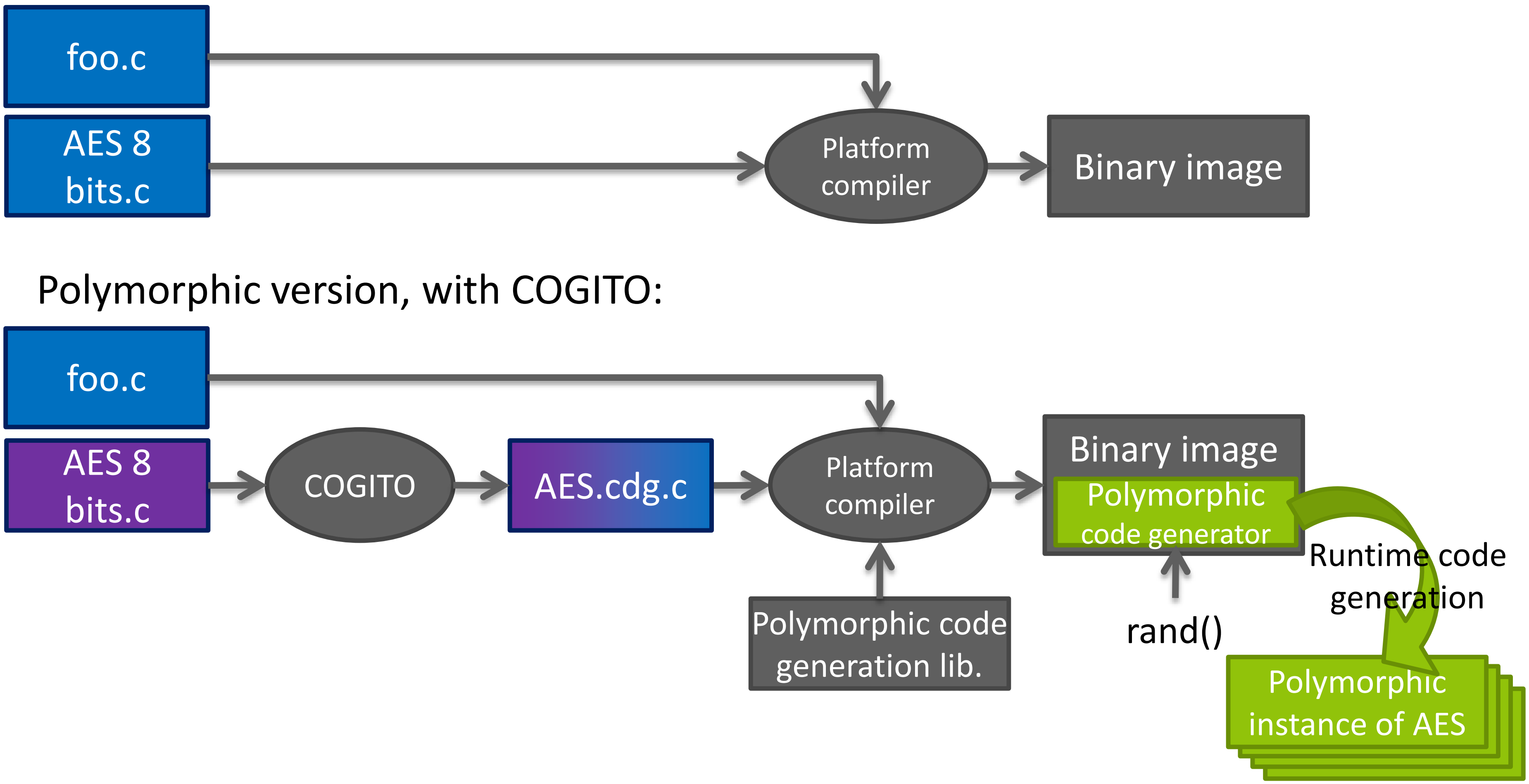




Runtime code generation for embedded systems

Reference version:

Polymorphic version, with COGITO:



VARIABILITY MECHANISMS

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

RANDOM REGISTER ALLOCATION

- Greedy algorithm: each register is allocated among one of the free registers remaining
- Has an impact on:
  - The management of the context (ABI)
  - Instruction selection

- Replace an instruction by a semantically equivalent sequence of one or several instructions
- Select the sequence in a list of equivalences
- Examples:

```
c := a xor b <=> c := ((a xor r) xor b) xor r
c := a xor b <=> c := (a or b) xor (a and b)
c := a - b   <=> k := 1 ; c:= (a + k) + (not b)
c := a - b   <=> c := ((a + r) - b) - r
```

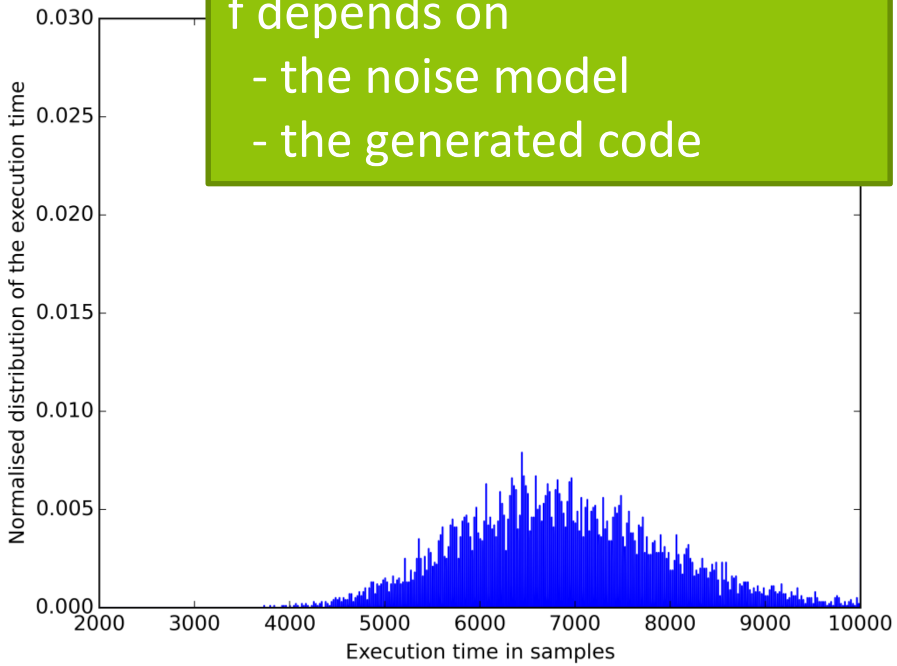

INSTRUCTION SHUFFLING

- Randomly reorder instructions
- … but do not break the semantics of the code!
  - Defs – read registers
  - Uses – modified registers
  - *Do not* take into account result latency and issue latency
  - Special treatments for… special instructions. E.g. branch instructions

INSERTION OF NOISE INSTRUCTIONS

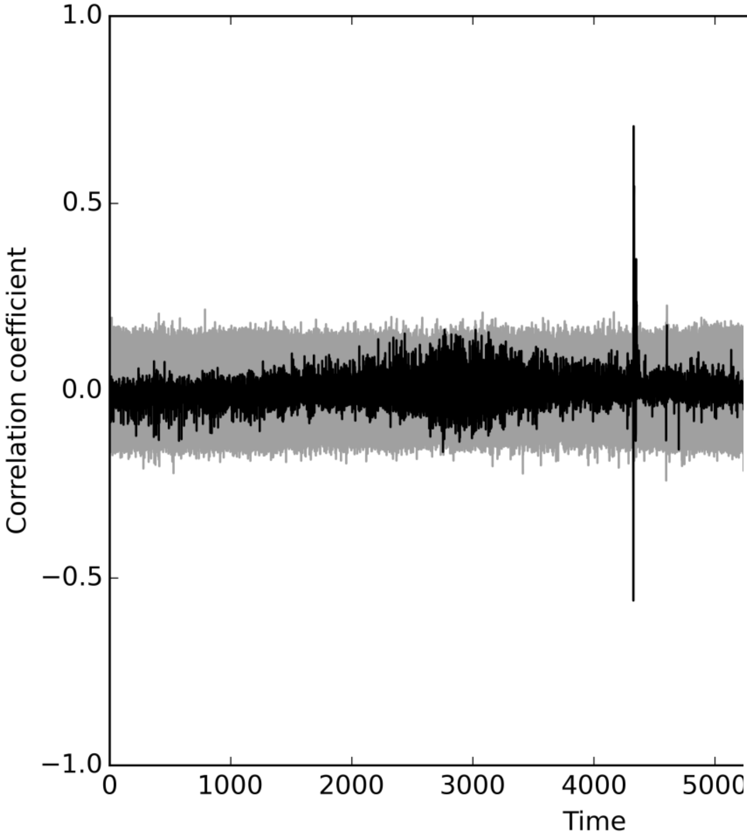
- Noise instructions have no effect on the result of the program
- Parametrable model of the inserted delay ~ program execution time
  - Goal:
    Maximize standard deviation **σ**
    Minimize mean **E**
- Can insert any instruction:
  - nop
  - Arithmetic (add, xor…)
  - *Memory accesses* (lw, lb,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

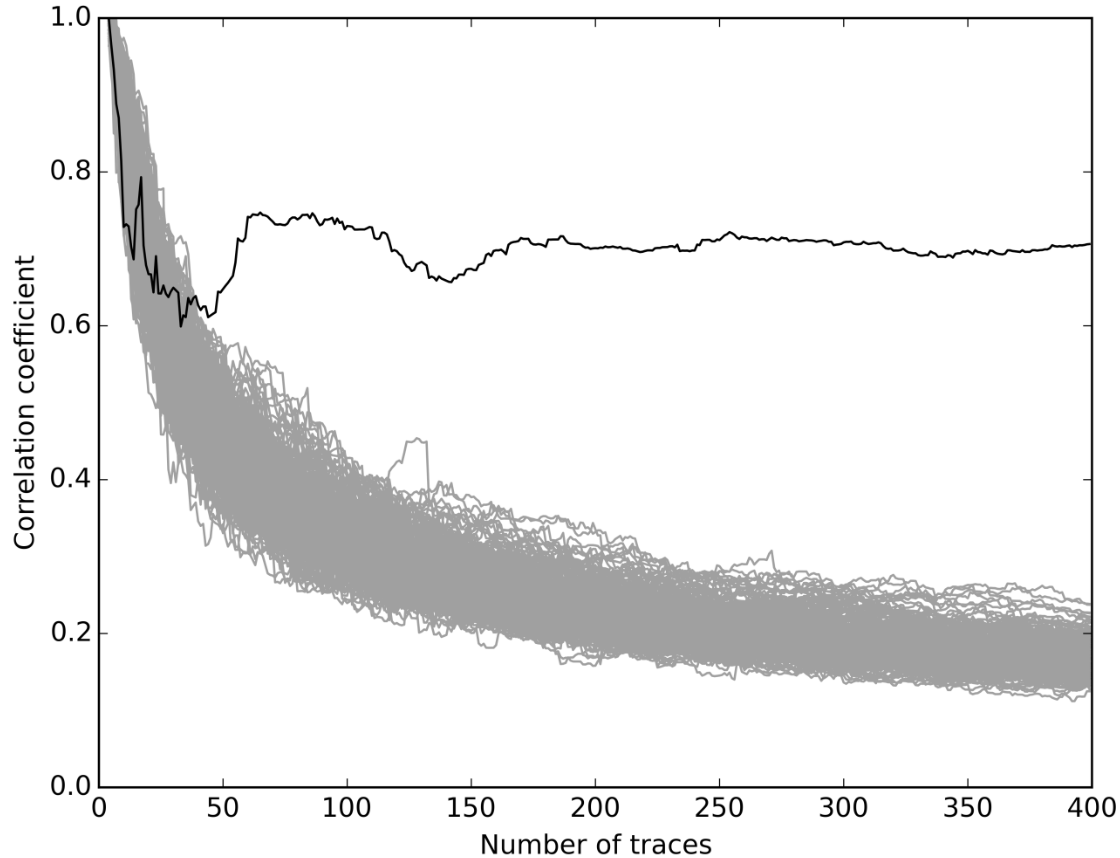
N: number of insertions
$(E, \sigma) = f(N)$
f depends on
- the noise model
- the generated code



IMPACT OF POLYMORPHISM ON 1ST ORDER CPA

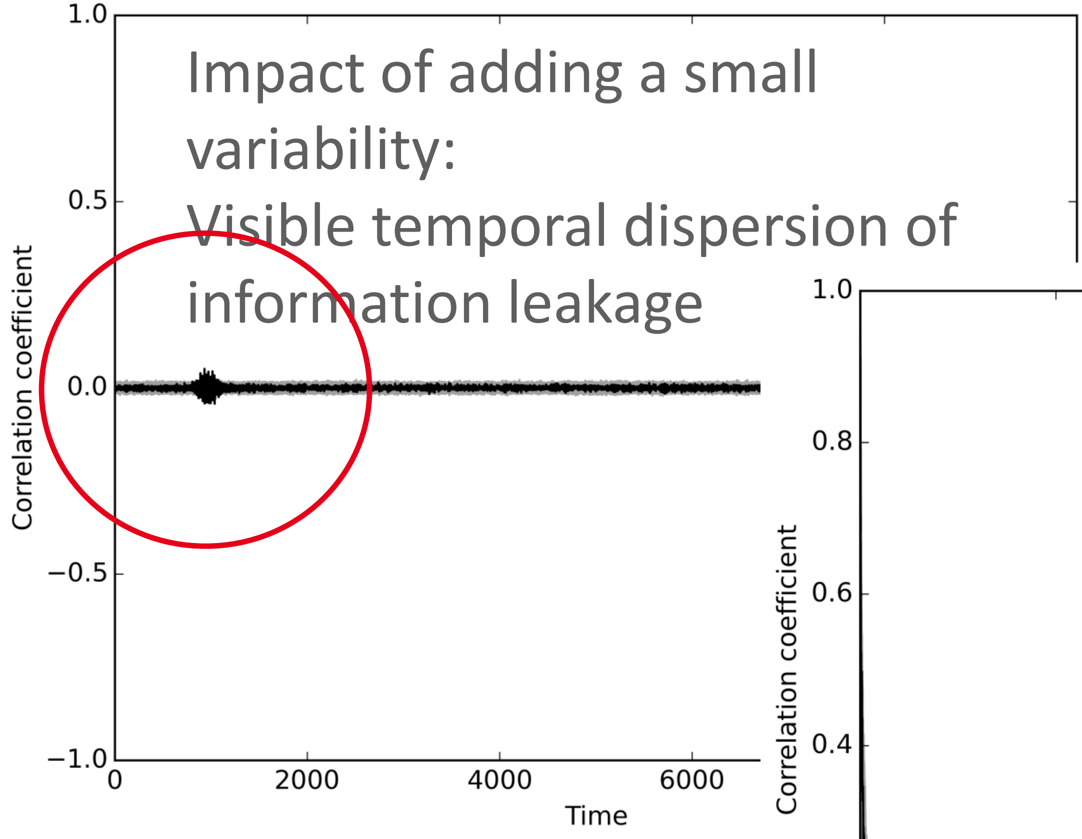
Reference version:
unprotected AES-8

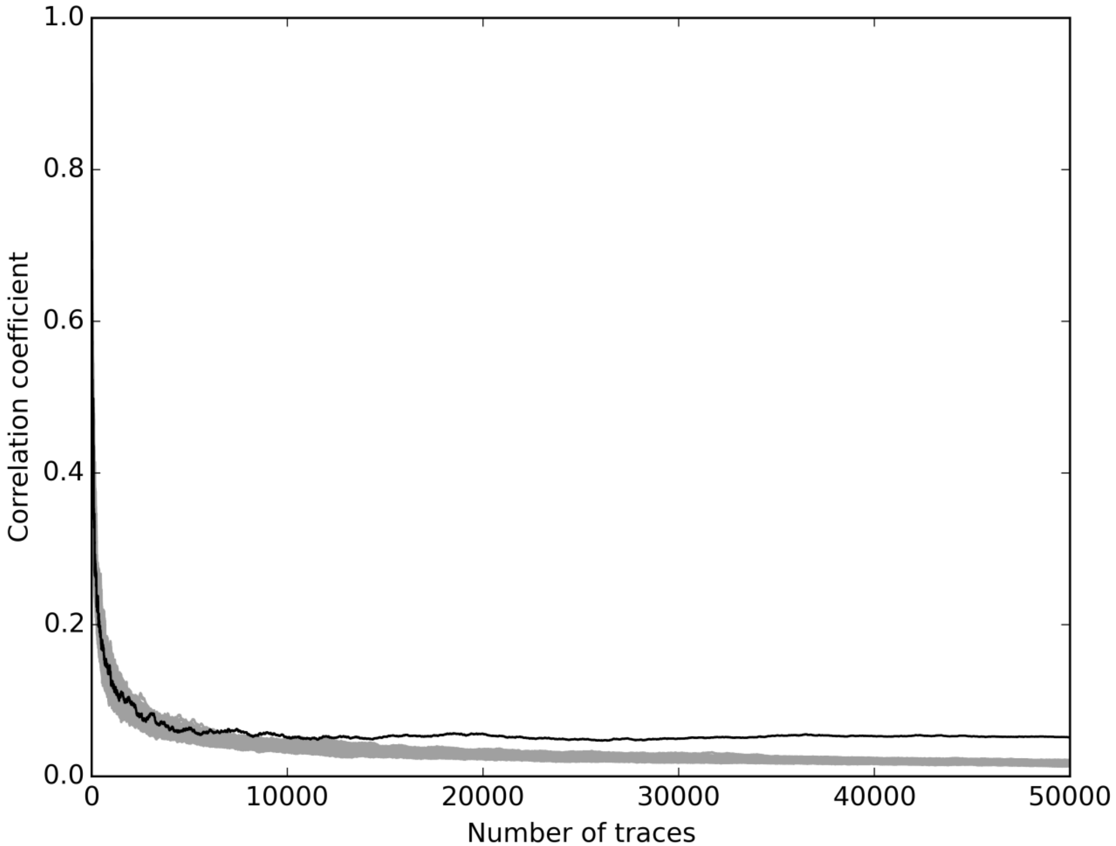




IMPACT OF POLYMORPHISM ON CPA

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



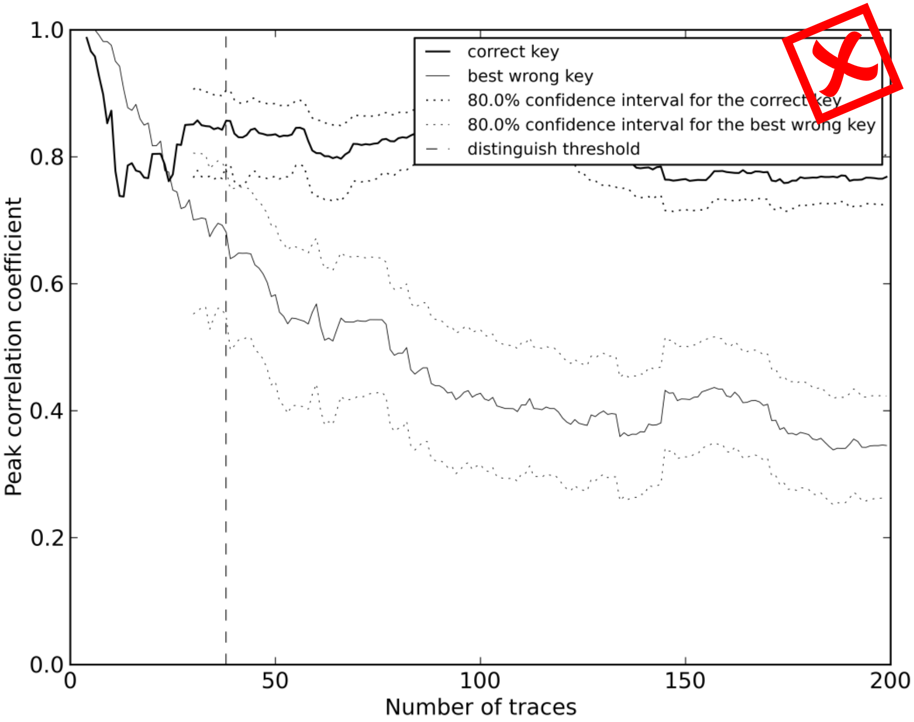


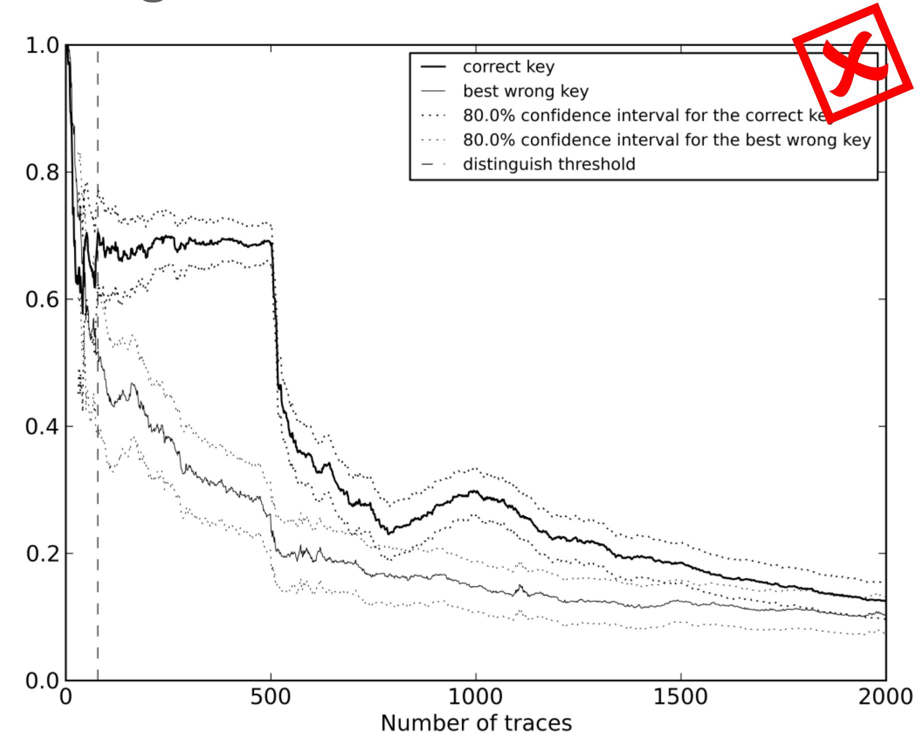
IMPACT OF POLYMORPHISM ON CPA

Effect of the code generation interval

Reference implementation

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





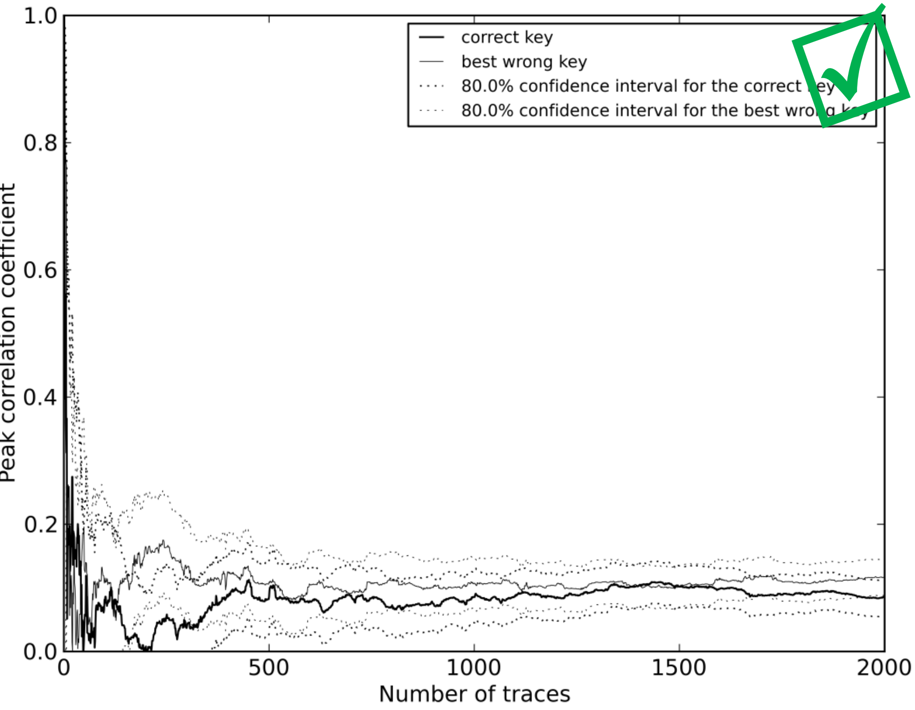
Distinguish threshold = 39 traces

Distinguish threshold = 89 traces

IMPACT OF POLYMORPHISM ON CPA

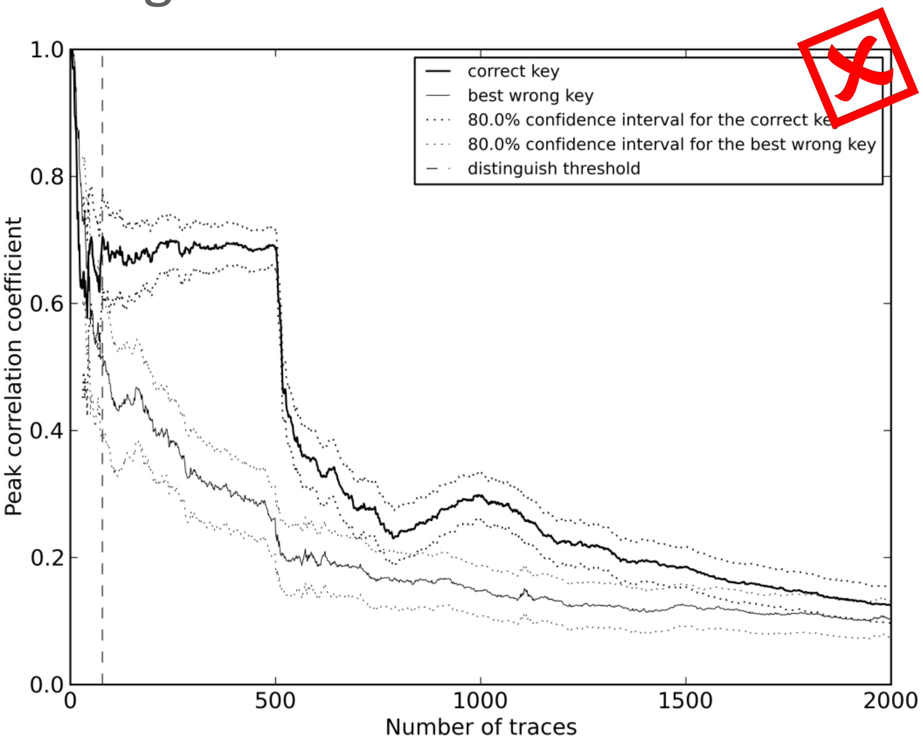
Polymorphic version

code generation interval: **20**



Polymorphic version,

code generation intervall: **500**



Distinguish threshold > 10000 traces

Distinguish threshold = 89 traces

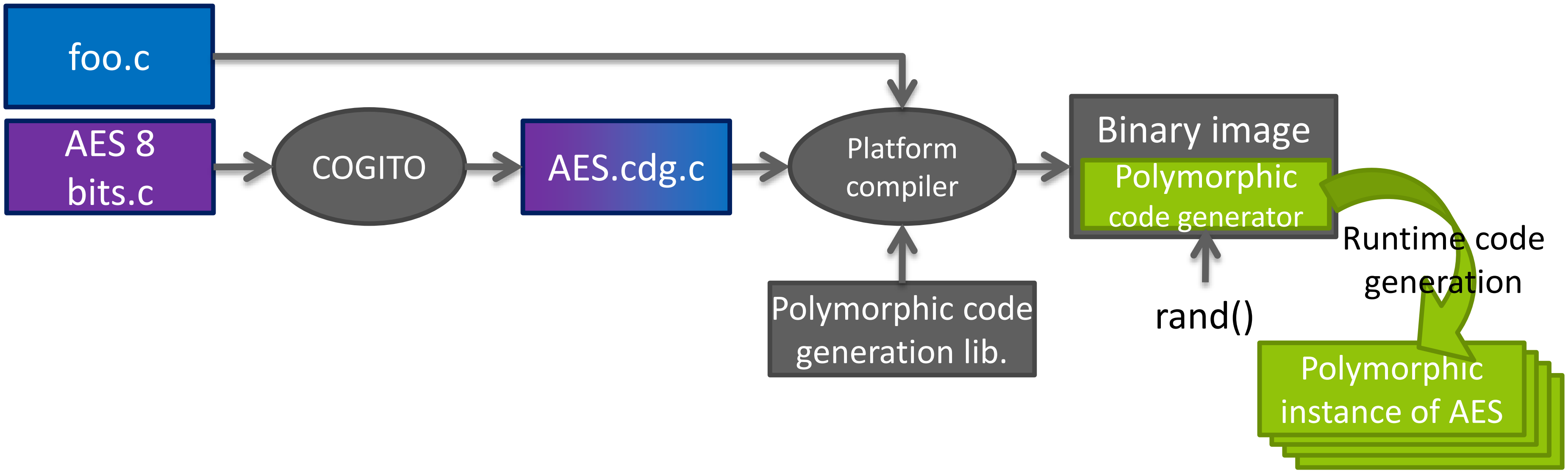
AUTOMATED APPLICATION OF POLYMORPHISM

Automated application using LLVM

- **Declaration of polymorphism with a source code annotation**

/* unsecured */	/* secured */
	#pragma polymorphic (…)
void AES_encrypt(…)	void AES_encrypt(…)
{ /* … */	{ /* … */

- **Configurable levels of polymorphic transformations => security/performance tradeoff**
  - Nature of the code transformations: random allocation of registers, semantic variants, instruction shuffling, insertion of noise instructions.
  - Degree of polymorphic variability inserted



AUTOMATED APPLICATION OF POLYMORPHISM

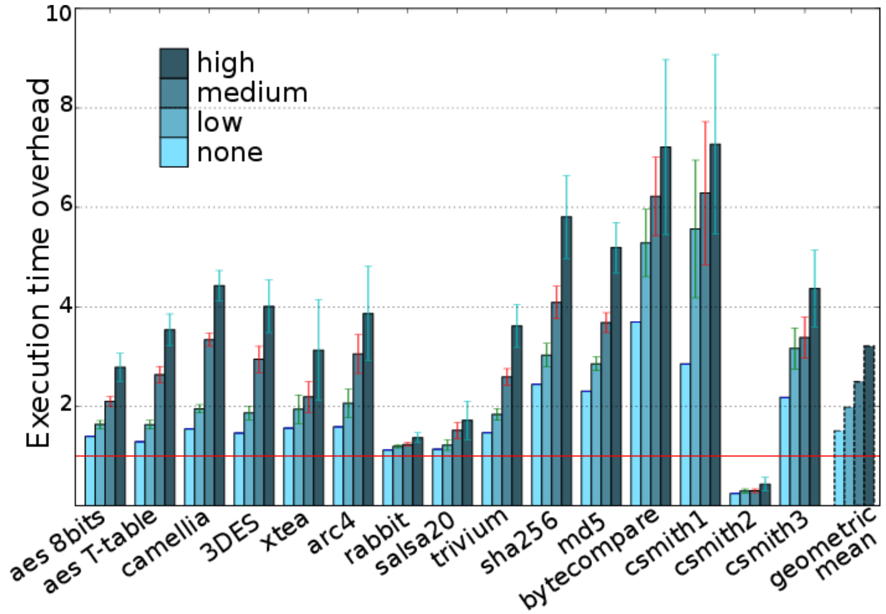
Automated application using LLVM

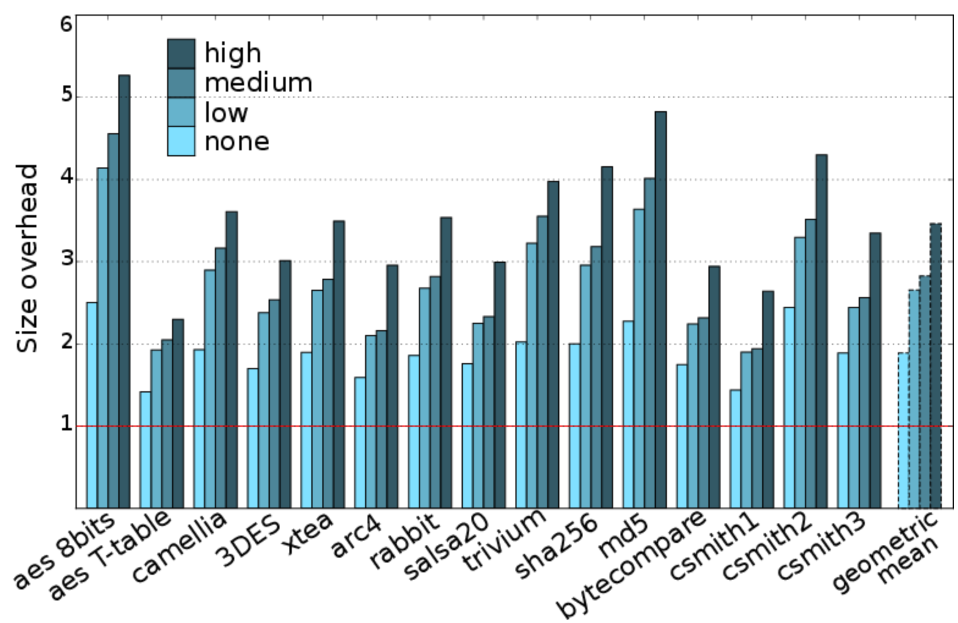
- **Declaration of polymorphism with a source code annotation**

/* unsecured */	/* secured */
	#pragma polymorphic (…)
void AES_encrypt(…)	void AES_encrypt(…)
{ /* … */	{ /* … */

- **Configurable levels of polymorphic transformations => security/performance tradeoff**
  - Nature of the code transformations: random allocation of registers, semantic variants, instruction shuffling, insertion of noise instructions.
  - Degree of polymorphic variability inserted

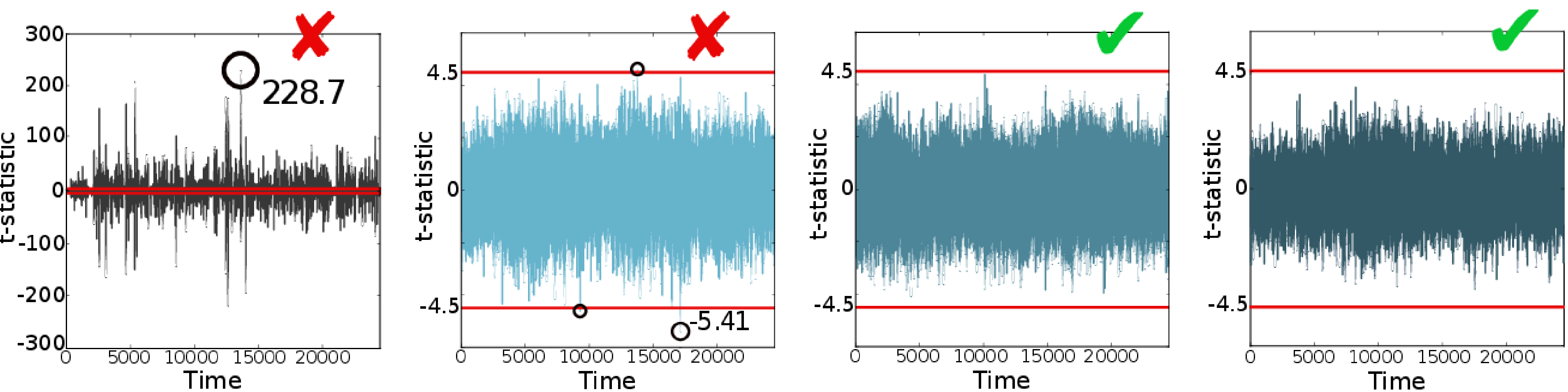
Components evaluated: ciphers, hash functions, simple authentication, random generated codes





- **Polymorphism is a hiding countermeasure against side-channel attacks**
  - Does not *remove* information leakage; *reduces* SNR only
- **However, 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

TAKE HOME MESSAGES

TAKE HOME MESSAGES

- **Physical attacks are currently the most effective way to break cryptography**
  - Also applicable to other classes of applications
- **Side-channel attacks**
  - Secured products involve a combination of hiding and masking protections
  - Advanced attacks use a combination of side-channel and fault injection attacks
- **Do not trust the compiler, unless it is specifically designed for security purposes**
  - You can workaround compiler optimisations,
  - but this is tricky, and **fragile**
- **Even if the compiler is specifically designed for security purposes, do not trust the compiler**
  - A security compiler is not enough if used alone

Side-Channel Attacks

ISSISP 2017 – Gif-sur-Yvette
2017-07-21

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

Centre de Grenoble
17 rue des Martyrs
38054 Grenoble Cedex

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