FROM RESEARCH TO INDUSTRY

cea tech

Génération de code au runtime pour la sécurité des systèmes embarqués

COGITO

RESSI – Toulouse
10/05/2016

Damien Couroussé
CEA – LIST / LIALP – Grenoble Université Alpes

PROJET FINANCÉ PAR - PROJECT FUNDED BY THE ANR

THE COGITO CONSORTIUM

CEA LIST-DACLE

Abderahmanne Seriai, Damien Couroussé , Hassan Noura, Nicolas Belleville, Thierno Barry

- Bringing the deGoal framework
- Compilation & runtime code generation

INRIA Rennes, Tamis team

Hélène Le Bouder, Jean-Louis Lanet

- JavaCards
- Physical & logical attacks, software countermeasures

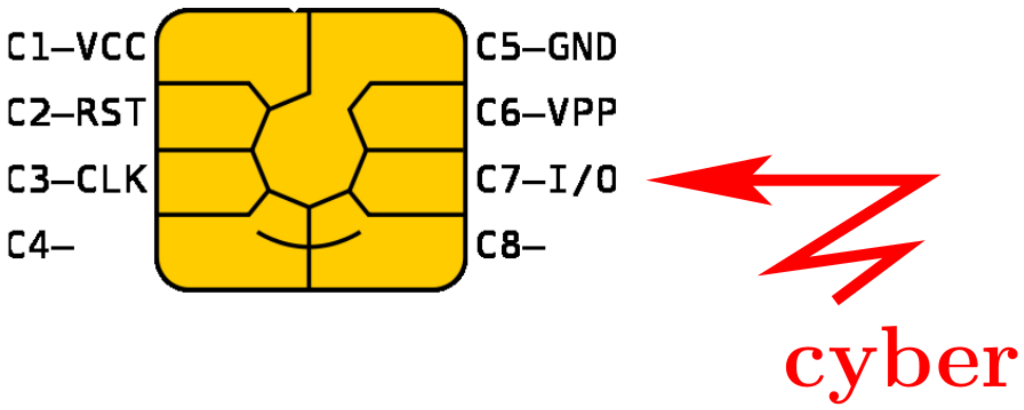
ENSMSE / CEA Tech DPACA - SAS laboratory

Bruno Robisson, Olivier Potin, Karim Abdelatif, Philippe Jaillon

- Physical attacks, HW/SW countermeasures
- Experimental validation

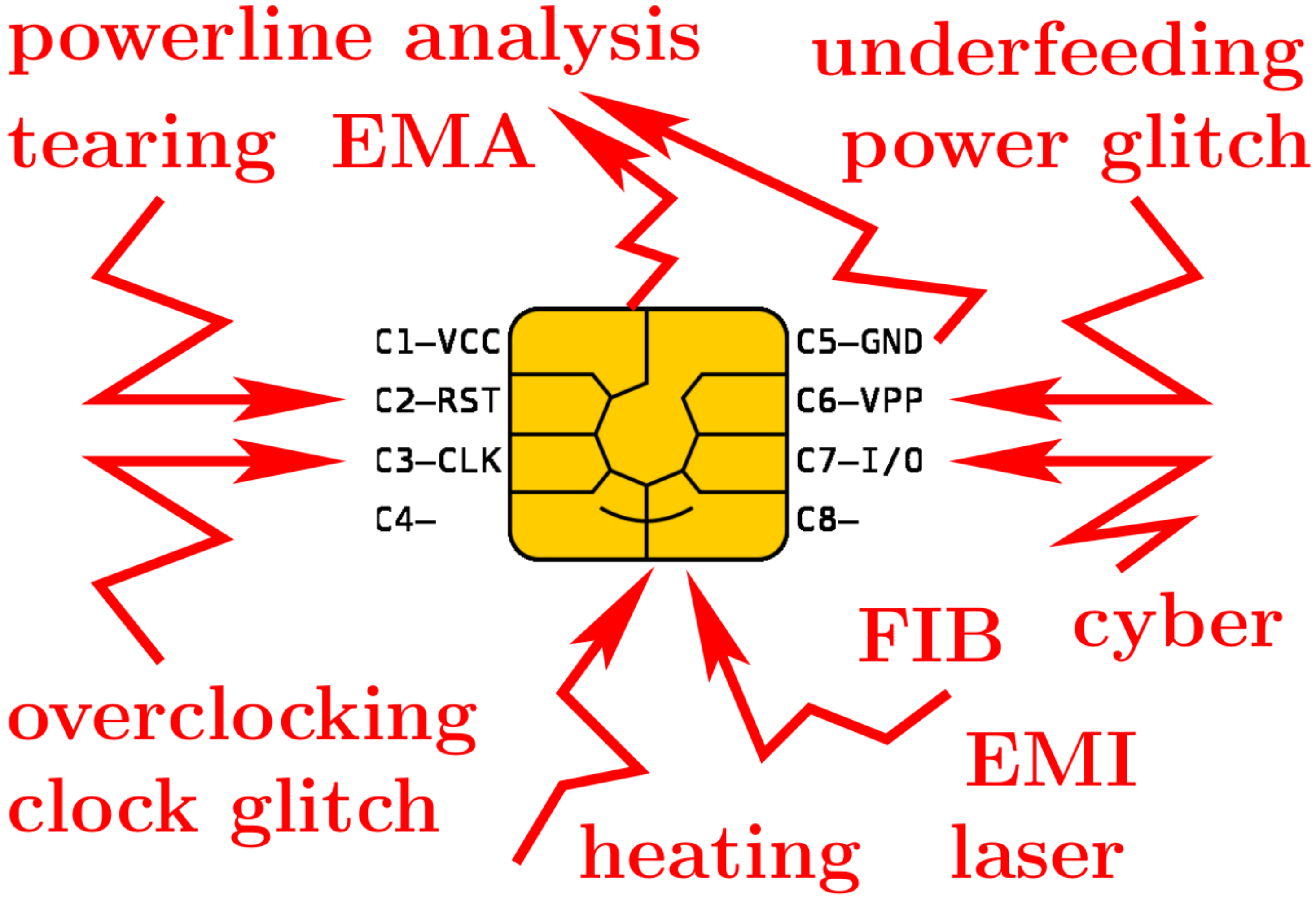
Public website: http://www.cogito-anr.fr

CYBER-PHYSICAL ATTACKS



Courtesy of Sylvain Guilley, Télécom ParisTech - Secure-IC

CYBER-PHYSICAL ATTACKS



Courtesy of Sylvain Guilley, Télécom ParisTech - Secure-IC

BESTIARY OF EMBEDDED SYSTEMS

… IN NEED FOR SECURITY CAPABILITES

Smart Card

Secure Element inside…

… And many other things

APPROXIMATE TYPOLOGY OF PHYSICAL ATTACKS

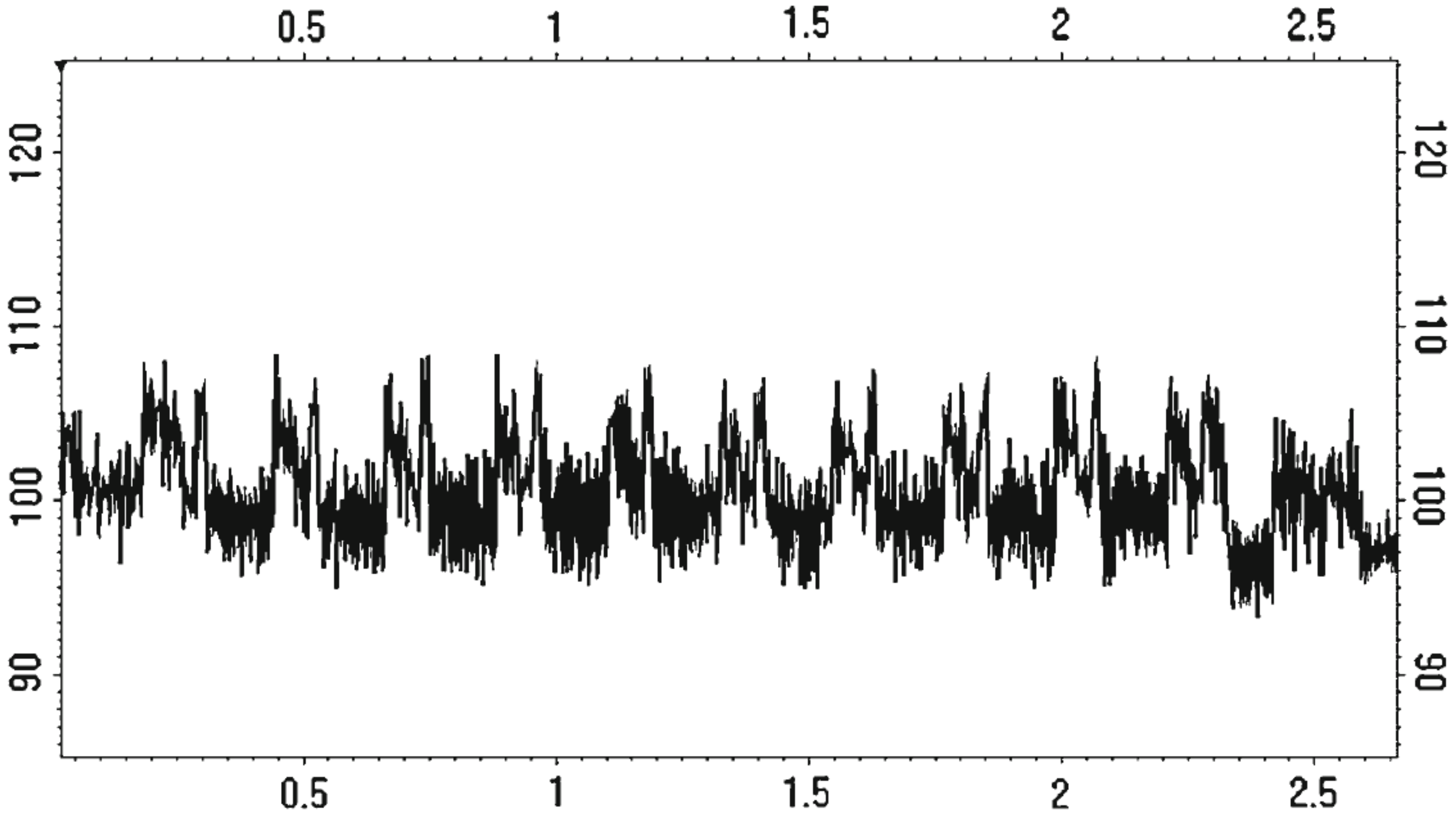
- **Target inspection**
  - HW inspection: decapsulation, abrasion, chemical etching, memory extraction, etc.
  - SW inspection: debug, memory dumps, code analysis, etc.

- **Intrusive / active attacks: fault injection**
  - under/over voltage drops
  - iom / laser beam, optical illumination
  - glitch attacks
  - …

- **Observation attacks: side channel attacks**
  - Electromagnetic analysis
  - Power analysis
  - Timing attacks
  - Acoustic analysis
  - …

SIMPLE POWER ANALYSIS (SPA)

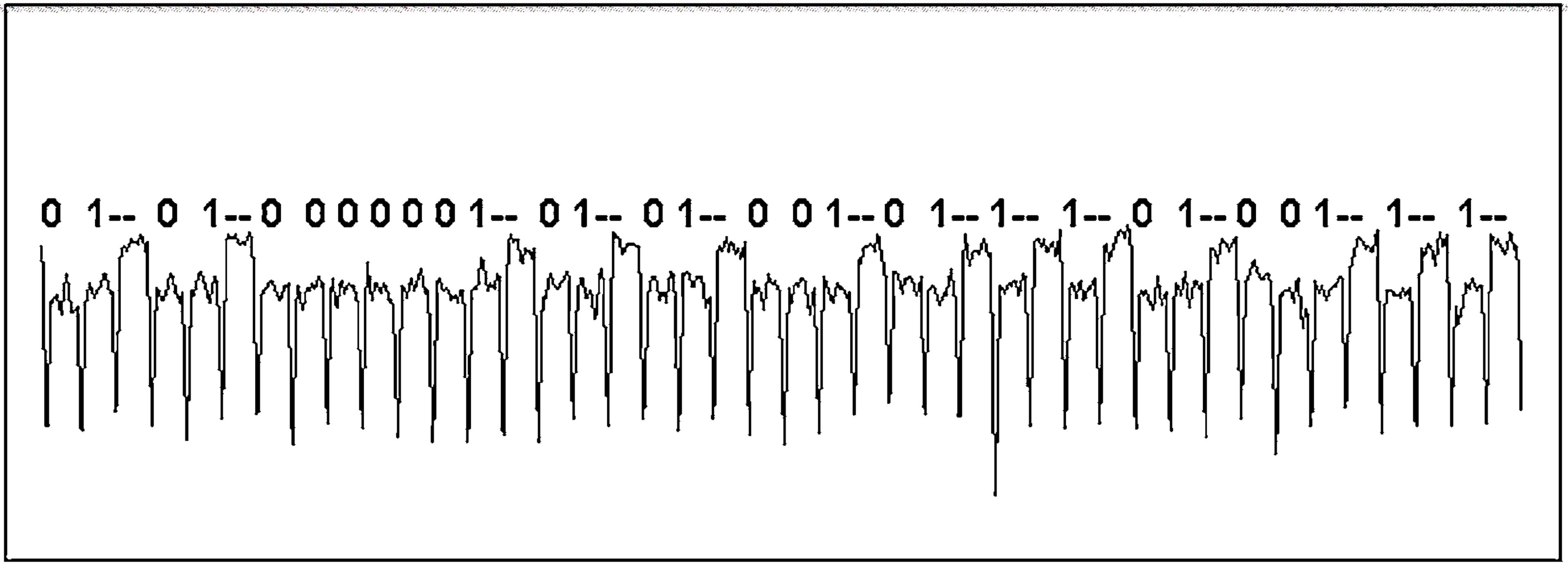
SPA on AES [Kocher, 2011]



The AES rounds are « clearly » visible

SIMPLE POWER ANALYSIS (SPA)

SPA on RSA [Kocher, 2011]



Direct access to key contents:

- bit 0 = square
- bit 1 = square, multiply

DIFFERENTIAL/CORRELATION POWER ANALYSIS (DPA/CPA)

- select *n* clear inputs => record *n* observations from the target
- compute *n* intermediate values, for each possible key values
- compute {power/EM/timing… } estimation from the intermediate values
- compute the correlation with the observation traces, for each observation sample

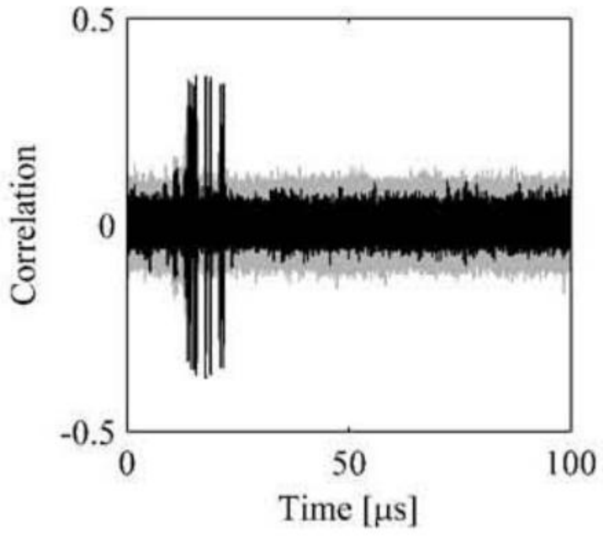


Figure 6.3. All rows of R. Key hypothesis 225 is plotted in black, while all other key hypotheses are plotted in gray.

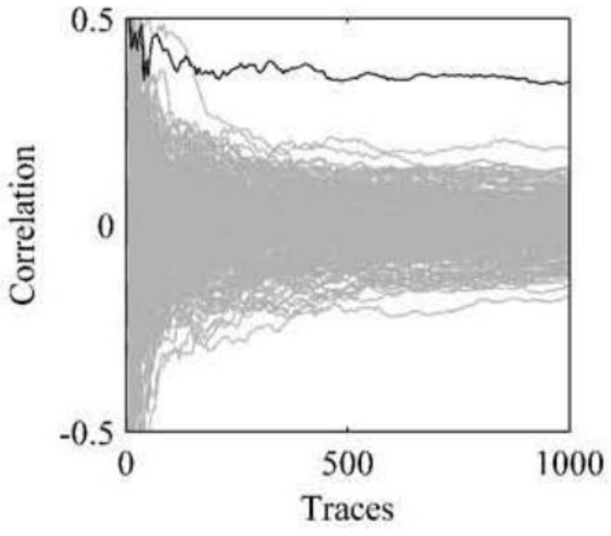


Figure 6.4. The column of R at 13.8 µs for different numbers of traces. Key hypotheses 225 is plotted in black.

[Mangard, 2007]

APPROXIMATE TYPOLOGY OF PHYSICAL ATTACKS

- **Target inspection**
  - HW inspection: decapsulation, abrasion, chemical etching, memory extraction, etc.
  - SW inspection: debug, memory dumps, code analysis, etc.

- **Fault injection attacks**
  - under/over voltage drops
  - ion / laser beam, EM perturbation **=> spatial and temporal sensibility**
  - optical illumination
  - glitch attacks
  - ...

- **Side channel attacks**
  - Electromagnetic analysis
  - Power analysis
  - Acoustic analysis
  - Timing attacks

=> spatial and temporal sensibility

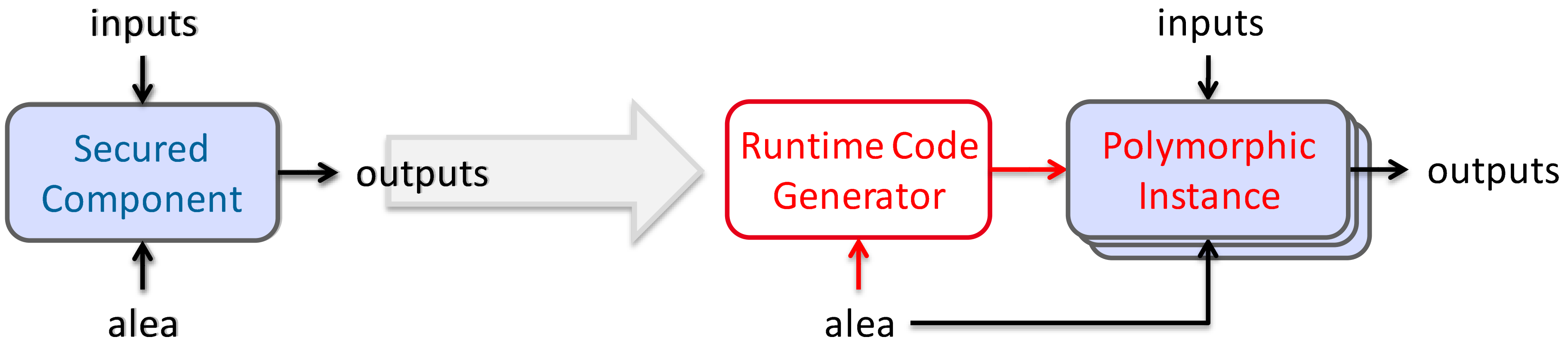
«REAL-LIFE» PHYSICAL ATTACKS

An attack is usually split between:

1. **A first step attack:**
   - global inspection of the target
   - identification of the security components involved (HW/SW)
   - identification of weaknesses
2. **A second step attack:**
   - focused attack
   - on an identified potential weakness

Definition

- **Regularly changing the behavior of a (secured) component, at runtime, while maintaining unchanged its functional properties, with runtime code generation**



POLYMORPHIC RUNTIME CODE GENERATION

Definition

- **Regularly changing the behavior of a (secured) component, at runtime, while maintaining unchanged its functional properties, with runtime code generation**

- **Protection against reverse engineering of SW**
  - the secured code is not available before runtime
  - the secured code regularly changes its form (code generation interval ω)

- **Protection against physical attacks**
  - polymorphism changes the spatial and temporal properties of the secured code: side channel & fault attacks
  - Compatible with State-of-the-Art HW & SW Countermeasures

- **deGoal: runtime code generation for embedded systems**
  - fast code generation
  - tiny memory footprint: proof of concept on TI's MSP430 (512 B RAM)
  - Easy targeting of application-specific instrutions or HW features

COUNTERMEASURES & POLYMORPHISM

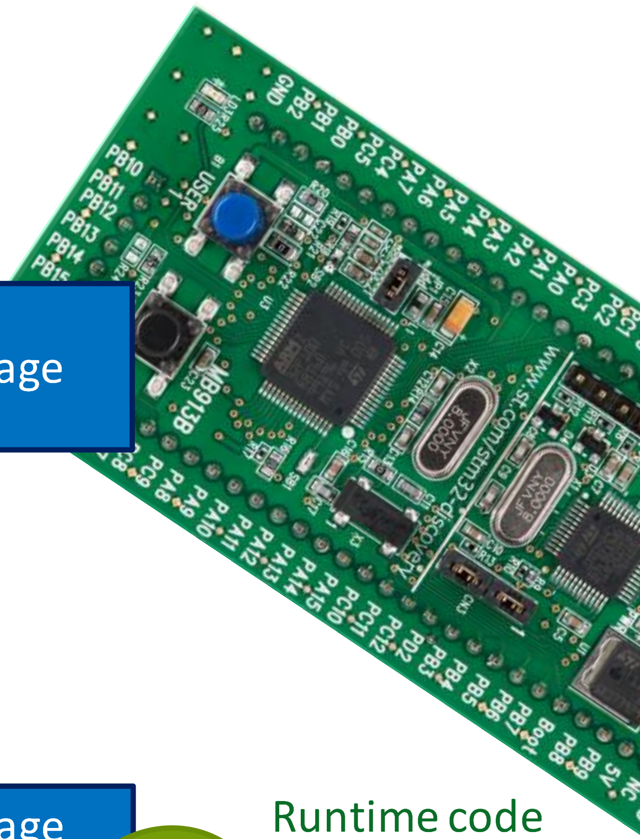
State of the Art

- Random register renaming [May 2011a, Agosta 2012]
- Out-of-Order execution
  - At the instruction level [May 2011b, Bayrak 2012]
  - At the control flow level [Agosta 2014, Crane 2015]
- Execution of dummy instructions [Ambrose 2007, Coron 2009, Coron 2010]
- A few proof-of-concept implementations, not suitable for embedded devices [Amarilli 2011, Amarilli 2011, Agosta 2012]

Our approach

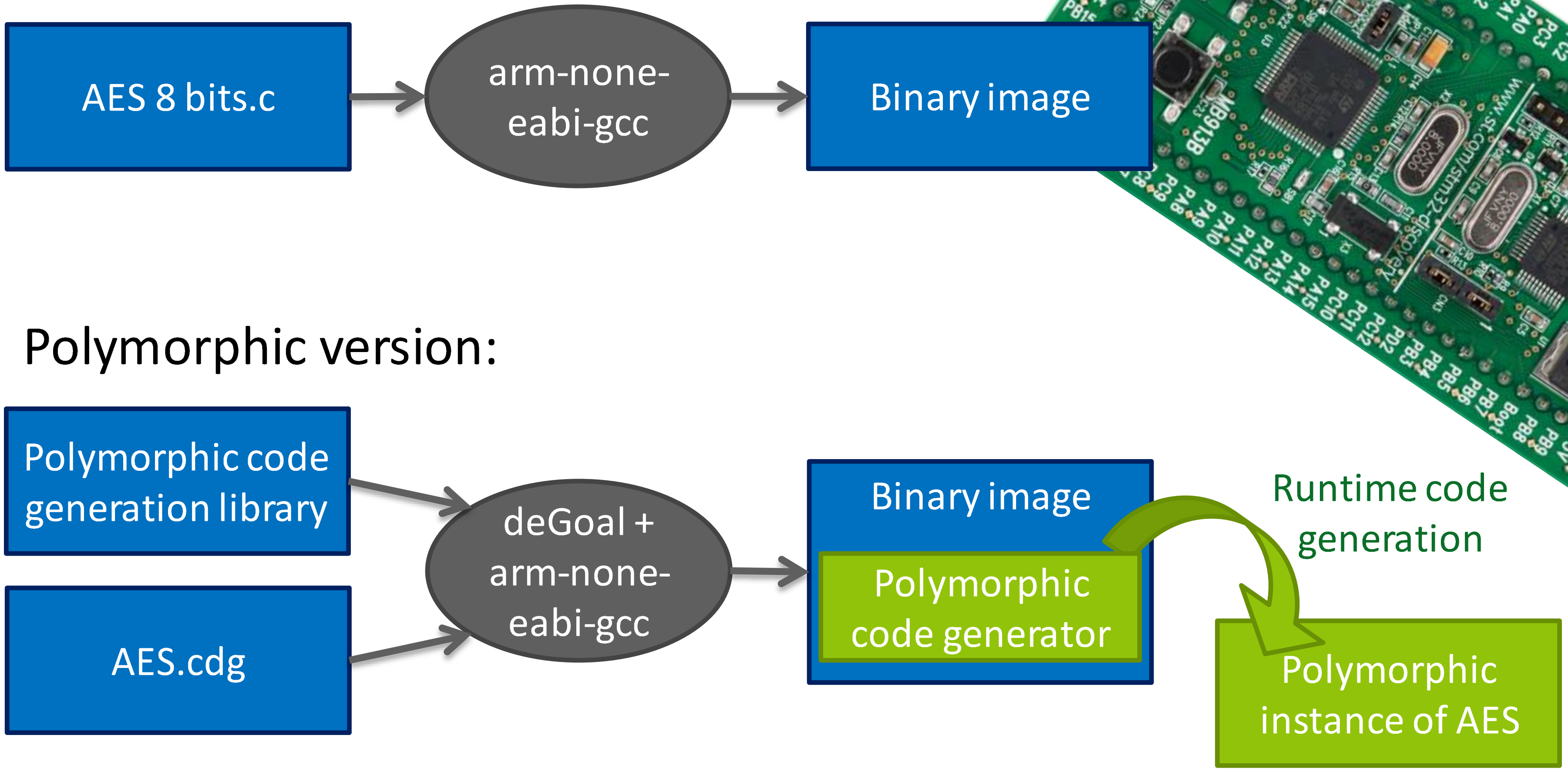
- Pure software → portability, genericity
- Combination of *all* the polymorphic levers found in the state of the art,
  - Currently at the basic block level
- Modest overhead (execution time & memory footprint)
- With runtime code generation

APPLICATION TO AES

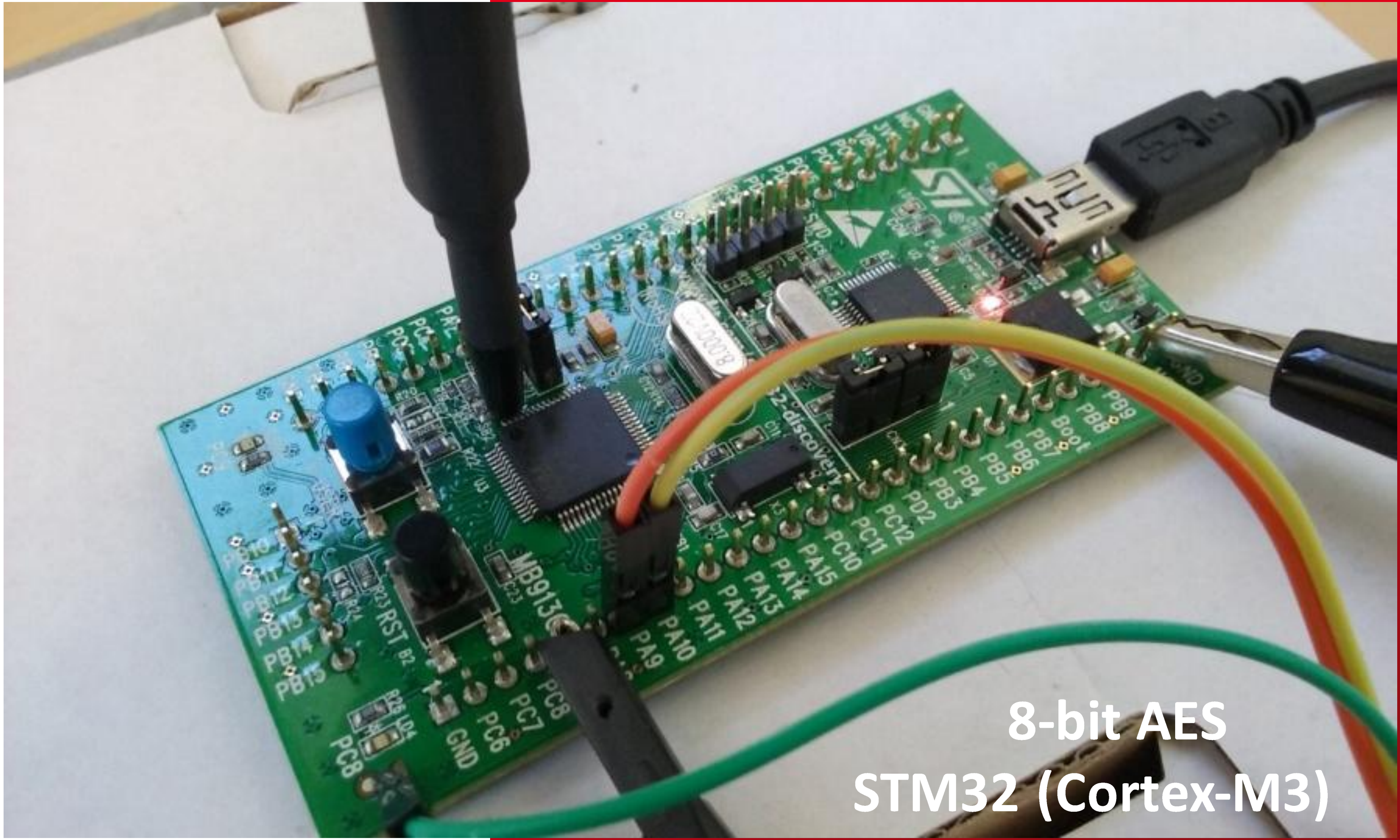


Reference version:

Polymorphic version:


AES 8 bits.c
arm-none-eabi-gcc
Binary image
Polymorphic code generation library
AES.cdg
deGoal + arm-none-eabi-gcc
Binary image
Polymorphic code generator
Runtime code generation
Polymorphic instance of AES

DEMO

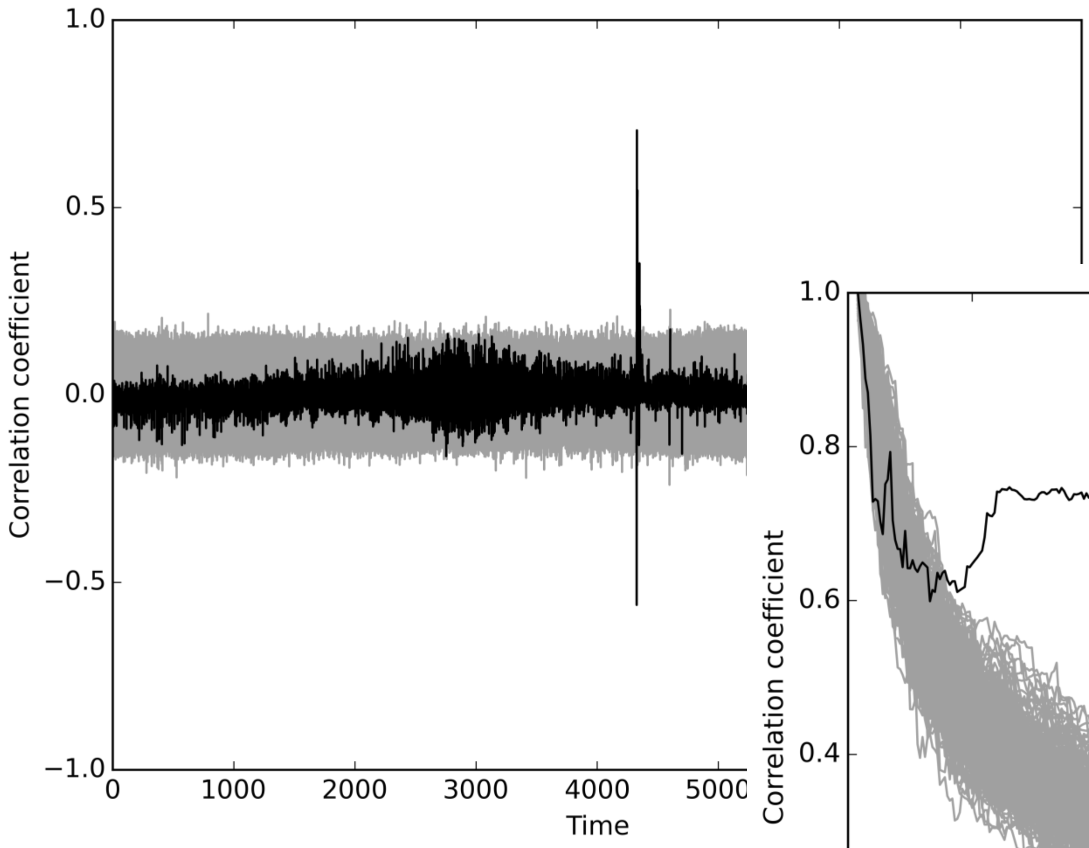


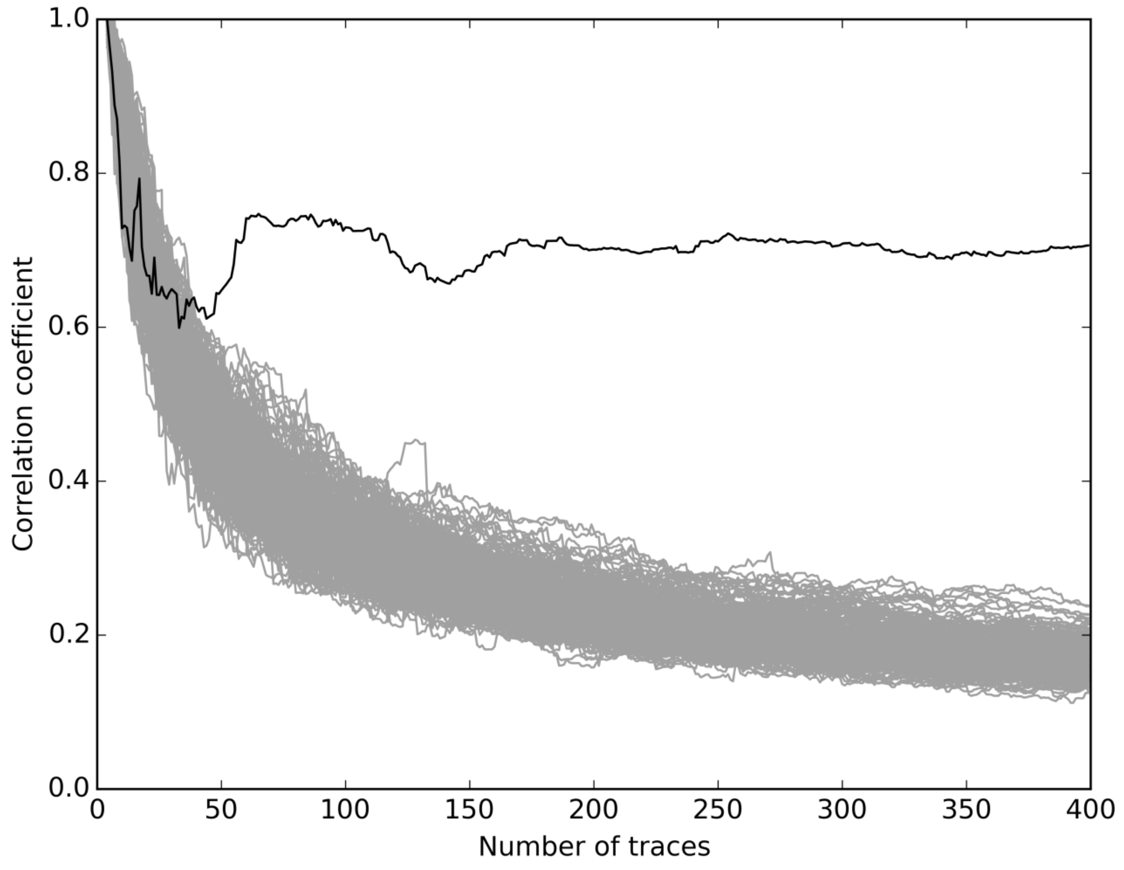
8-bit AES

STM32 (Cortex-M3)

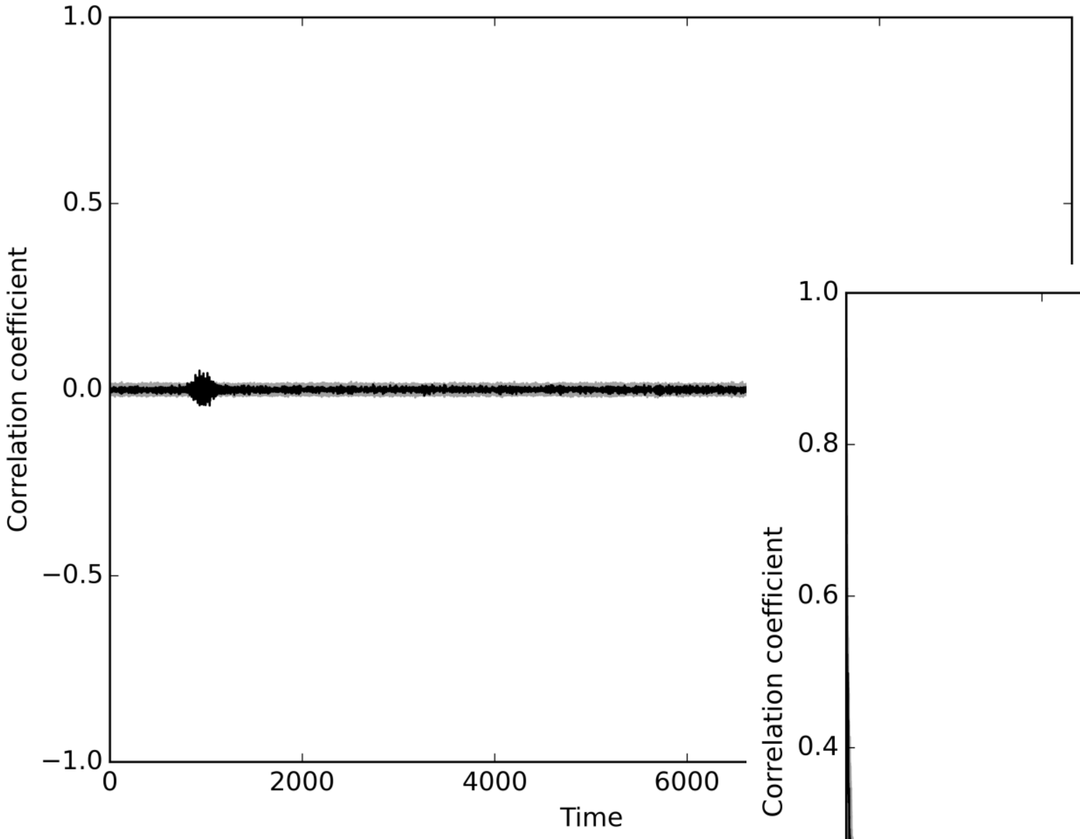
IMPACT OF POLYMORPHISM ON CPA

Reference version

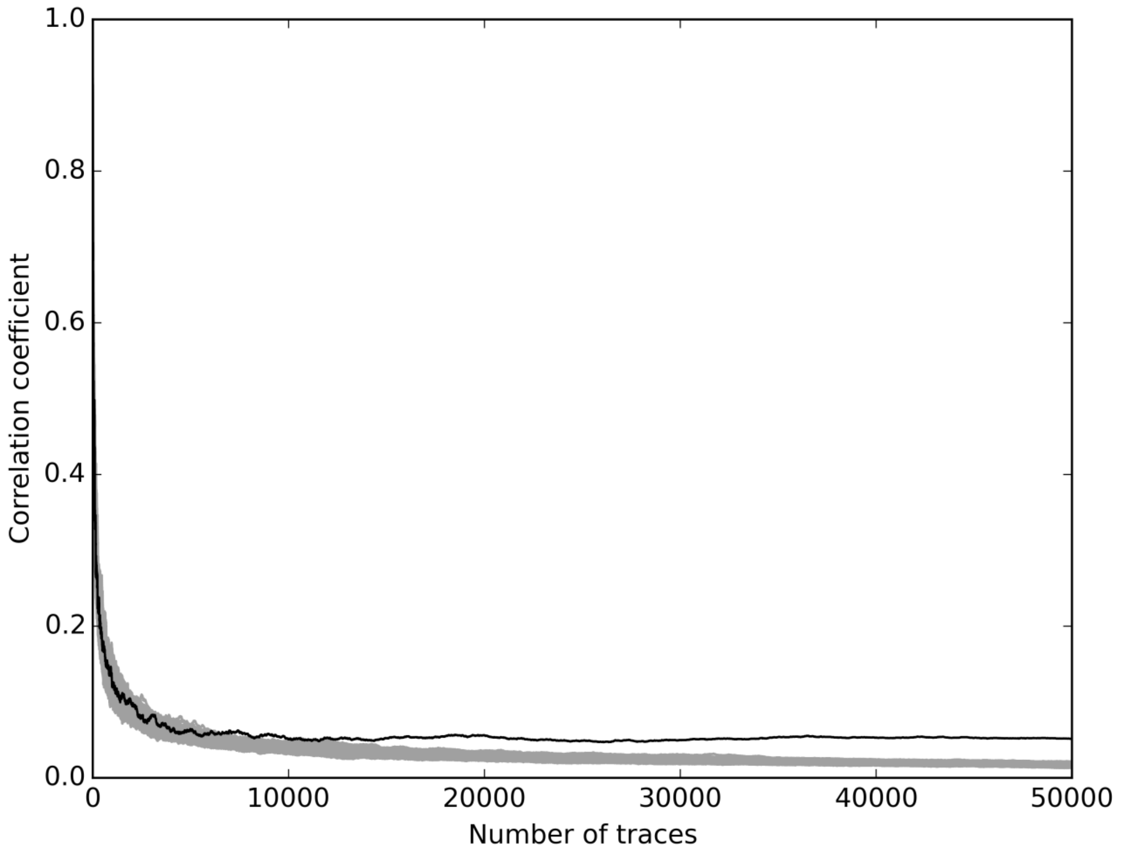




IMPACT OF POLYMORPHISM ON CPA



Adding a bit of temporal dispersion



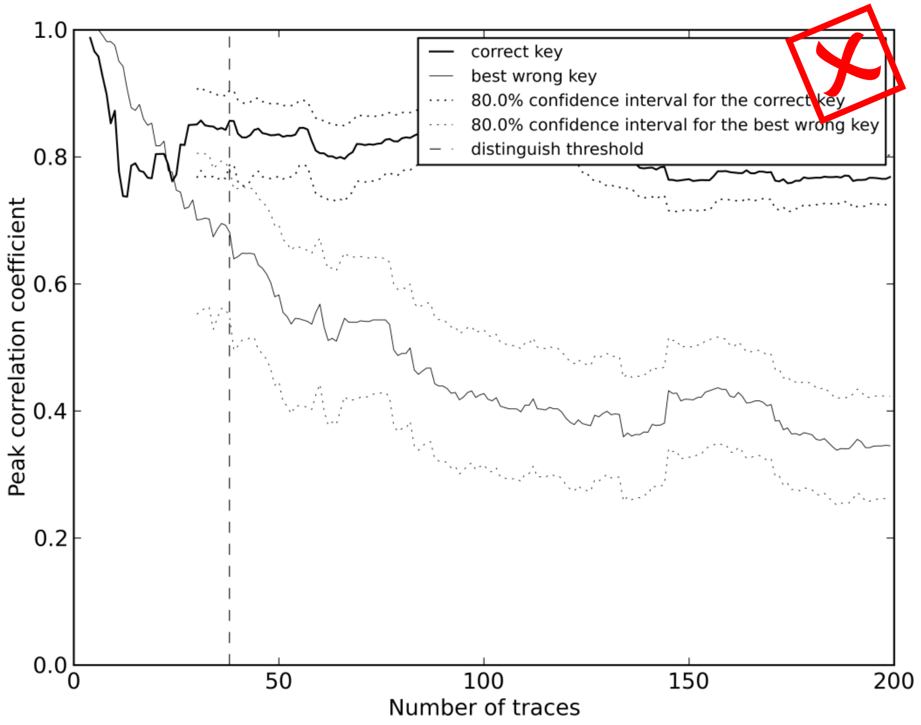
IMPACT OF POLYMORPHISM ON CPA

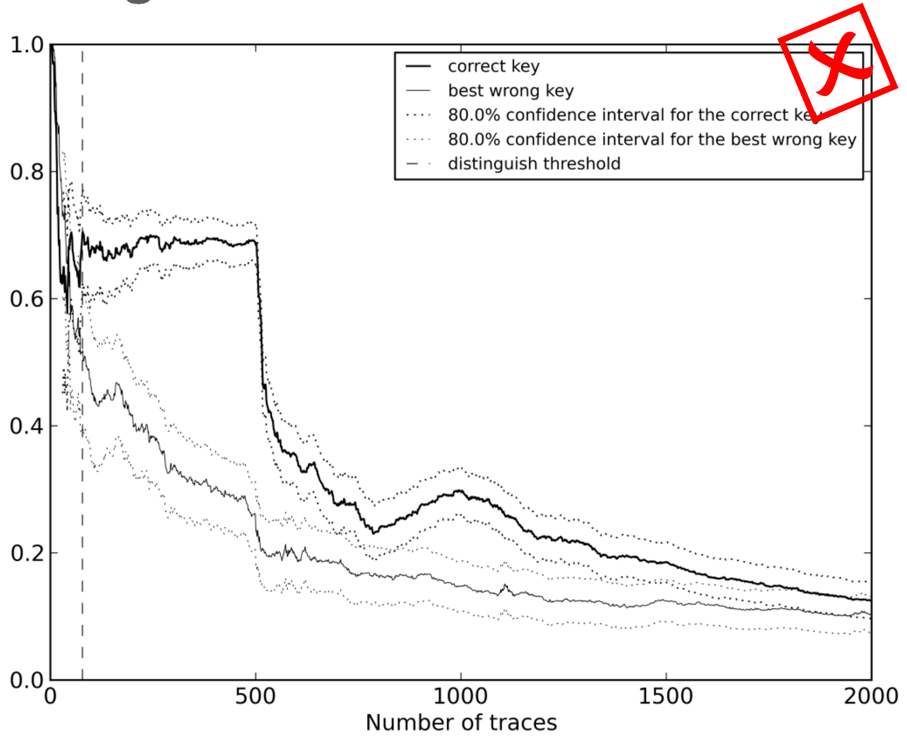
Effect of the code generation interval

Reference implementation

Polymorphic version,

code generation intervall: **500**





Distinguish threshold = 39 traces

Key byte 10

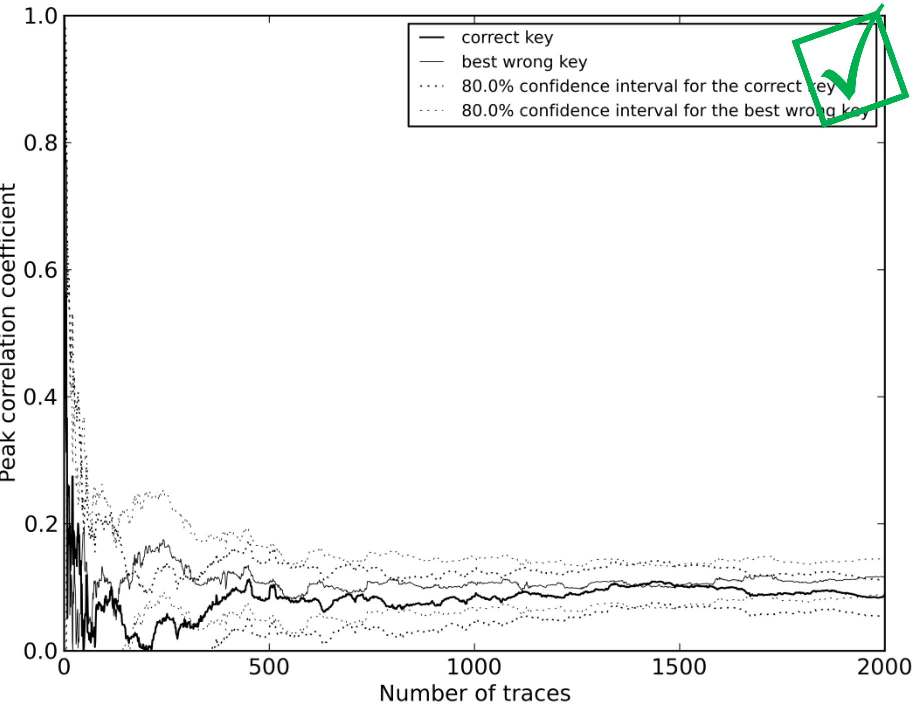
Distinguish threshold = 89 traces

Key byte 02

IMPACT OF POLYMORPHISM ON CPA

Polymorphic version

code generation interval: **20**

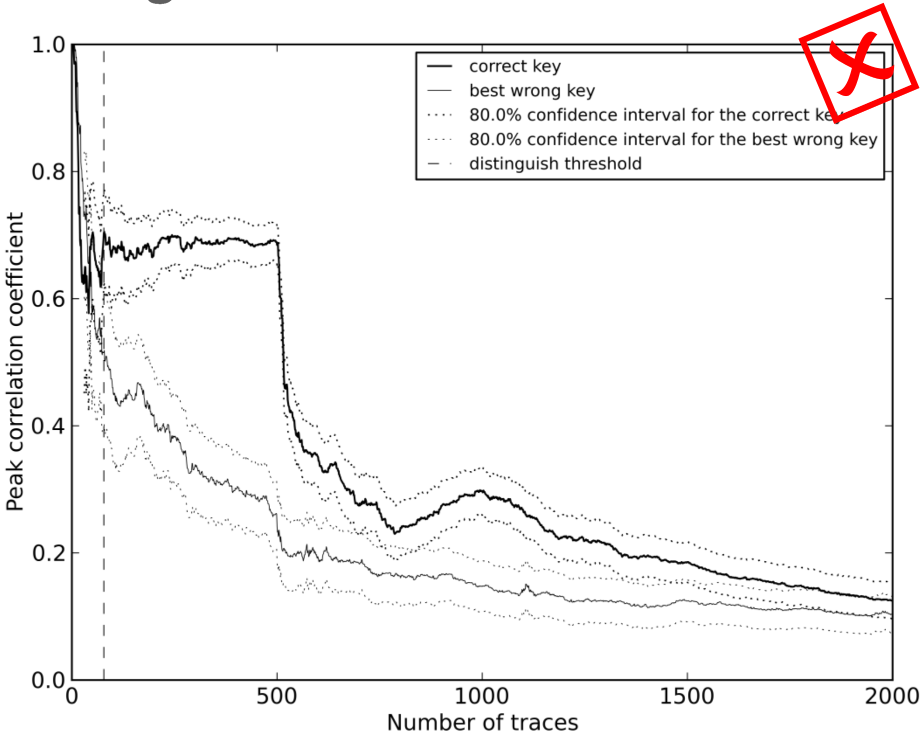


Distinguish threshold > 10000 traces

Key byte 02

Polymorphic version,

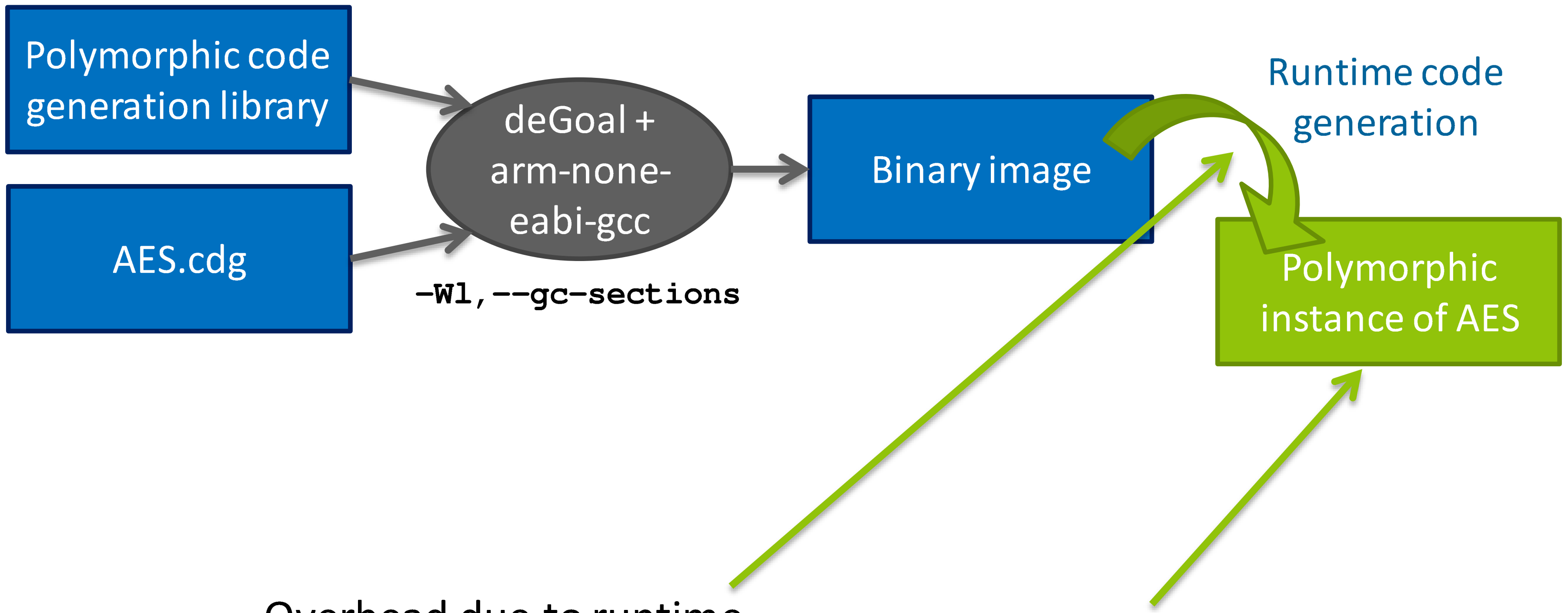
code generation intervall: **500**



Distinguish threshold = 89 traces

Key byte 02

PERFORMANCE OVERHEAD


Polymorphic code generation library
AES.cdg
deGoal + arm-none-eabi-gcc
-Wl,--gc-sections
Binary image
Runtime code generation
Polymorphic instance of AES

Overhead due to runtime code **generation**

Overhead due to the **generated** code

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			Agosta et al. (2012)
	k Min.	k Avg.	k Max.	k Min.	k Avg.	k Max.	k Min.	k Avg.	k Max.	k
ω=1	3.16	4.91	6.37	5.81	7.27	8.94	20.10	22.94	26.16	398*
ω=10	1.32	1.50	1.66	1.59	1.76	1.92	3.86	4.36	4.85	40*
ω=100	1.09	1.16	1.22	1.16	1.21	1.25	2.17	2.50	2.78	5.00
ω=1000	1.09	1.13	1.18	1.16	1.15	1.20	2.17	2.32	2.59	1.27
ω=10000	1.05	1.12	1.18	1.11	1.15	1.19	1.99	2.30	2.58	1.10

*Extrapolated values

- **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

WHAT'S NEXT?

Experimental evaluation

- State-of-the-art side channel attacks
  - Synchronisation
  - Filtering
- Faults
  - Topic to be opened
- Vulnerability of the code generator?

Open questions

- Certification of polymorphic code? Common Criteria
- Correctness of the generated code, $\forall$ alea

Merci !

Génération de code au runtime pour la sécurité des systèmes embarqués

COGITO

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

APPLICATION TO AES: ADDROUNDKEY()

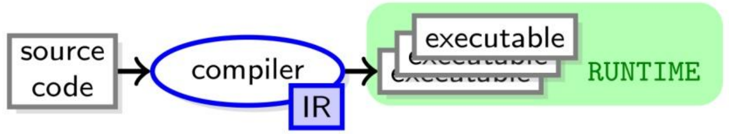
```
void addRoundKey_compilette( cdg_insn_t* code
                            , uint8_t* key_addr, uint8_t *state_addr)
{
  #[
    Begin code Prelude

    Type reg32 int 32
    Alloc reg32 state, key, i

    mv i, #(16)
    loop:
        sub i, i, #(1)
        lb state, @(#(state_addr) + i) // state = state_addr[i]
        lb key, @(#(key_addr) + i)     // key= key_addr[i]
        xor state, key
        sb @(#(state_addr) + i), state
        bneq loop, i, #(0)
    rtn
    End
  ]#;
}
```

Same code for performance purposes and for polymorphism.

The security protections are added in the backend.

APPROACHES FOR CODE SPECIALISATION

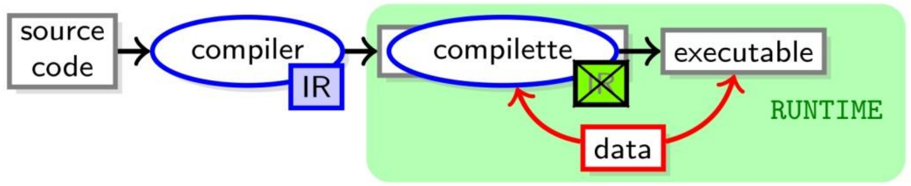
Static code versionning (e.g. C++ Templates)



- static compilation
- runtime: select executable
- memory footprint ++
- limited genericity
- runtime blindness

Runtime code generation, with deGoal

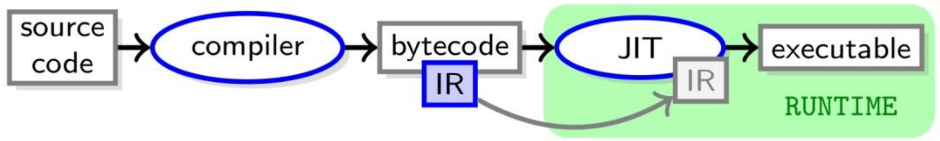
A *compilette* is an ad hoc code generator, targeting one executable



- fast code generation
- memory footprint −−
- **data-driven code generation**

Dynamic compilation

(JITs, e.g. Java Hotspot)



IR Intermediate Representation

- overhead ++
- memory footprint ++
- not designed for data dependant code-optimisations

DEGOAL SUPPORTED ARCHITECTURES

ARCHITECTURE	STATUS	FEATURES
ARM32	✓	
ARM Cortex-A, Cortex-M [Thumb-2, VFP, NEON]	✓	SIMD, [IO/OoO]
STxP70 [including FPx] (STHORM / P2012)	✓	SIMD, VLIW (2-way)
K1 (Kalray MPPA)	✓	SIMD, VLIW (5-way)
PTX (Nvidia GPUs)	✓	
MIPS	↺	32-bits
MSP430 (TI microcontroler)	✓	Up to < 1kB RAM
CROSS CODE GENERATION supported (e.g. generate code for STxP70 from an ARM Cortex-A)		

[IO/OoO]: Instruction scheduling for in-order and out-of-order cores

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

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
```

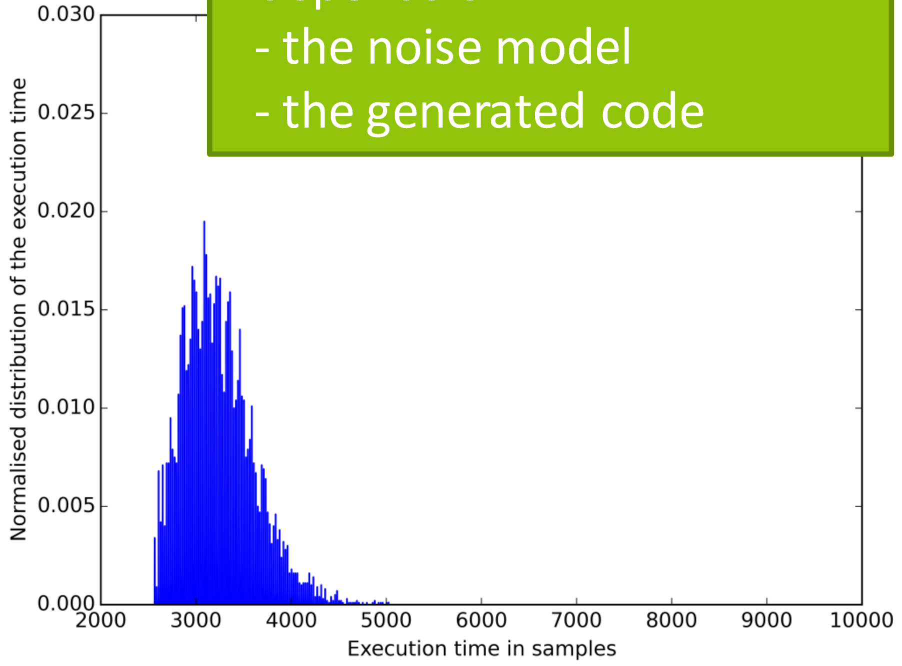
INSTRUCTION SHUFFLING

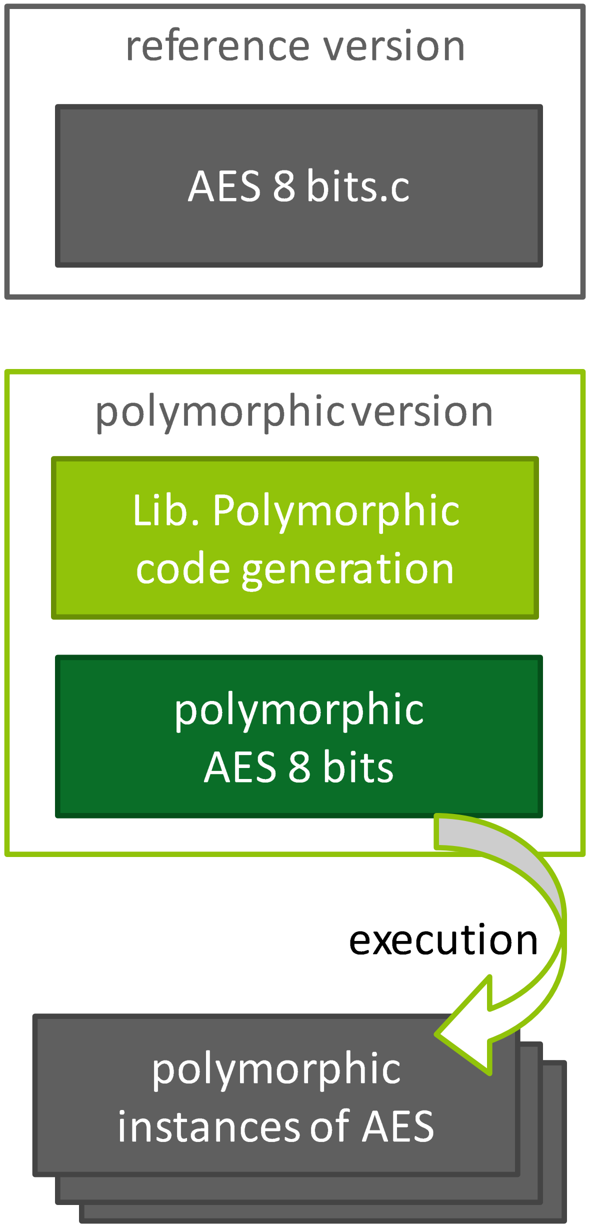
- Reorder instructions
- … but do not break the semantics of the code!
  - Defs – read registers
  - Uses – modified registers
  - *Do not* take into account result latency and issue latency
  - Special treatments for… special instructions. E.g. branch instructions

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
    Maximum standard deviation **σ**
    Minimum mean **E**
- Can insert any instruction:
  - nop
  - Arithmetic (add, xor…)
  - *Memory accesses* (lw, lb, …)
  - Power hungry instructions
    (mul, mac…)

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





Demo

leti & list