

CEA - DRT/DPACA Secure Architectures and Systems laboratory



brice.colombier@cea.fr

Pierre-Alain Moëllic

pierre-alain.moellic@cea.fr

PROSECCO Workshop

6 novembre 2018



Goal of the presentation

Goal: Present the first results of the security characterisation we performed at the SAS lab (joint team CEA Tech, Mines Saint-Etienne).

This evaluation helps to design **efficient countermeasures** by prodiving a **feedback to the designer**.

Characterisation carried out for different:

- Physical threats:
 - Side-channel analysis
 - Fault-attacks
- Hardware targets:
 - 8-bit microcontrollers
 - 32-bit microcontroller ARM Cortex M/A
- Practical use-cases:
 - VerifyPIN
 - AES encryption



Evaluation method

Two main axes:

- Leakage assessment using statistical tools
 - Attack-independent
- Attack-based methodology:

Complexity / Cost	Side-channel attacks	Fault attacks
+/\$	Correlation power analysis	Clock glitches
+++ / \$\$\$	Profiled attacks (templates, Machine learning)	Laser







Fault attacks

- VerifyPIN
- AES-128 encryption





Conclusion

Side-channel leakage assessment

Leakage assessment

Aim: conduct a statistical study to evaluate the leakages.

Statistical tests: reject or not a *null hypothesis* (i.e. the means of the target populations are equal)

Two common tools in SCA context:

- **t-test** [1]: split the traces in two sets w.r.t an intermediate value, see if they differ statistically.
 - The t statistic follows a Student law. For sufficient number of traces, |t| > 4.5 give a confidence of 99.999 % to reject the NH.
 - In our experiments: target at **bit level**.
- **F-test** [2], **SNR**: generalization of *t*-test for multiple sets. Takes the variance into consideration.
 - Ratio of inter-class VS intra-class variance.
 - In our experiments: target at **byte level**.

^[1] Tobias Schneider and Amir Moradi. "Leakage Assessment Methodology - a clear roadmap for sidechannel evaluations". IACR ePrint 2015.

^[2] Omar Choudary and Markus G. Kuhn. "Efficient template attacks." International Conference on Smart Card Research and Advanced Applications. 2013.

Comparison of unmasked and masked S-boxes

Splitting according to the value of the 8 bits at the 1st S-box output. 20000 traces of 128-bit AES encryption.

Ceatech



 \Rightarrow No more 1st order leakage with this masking scheme.

Identification of new leakage points

Ceatech

The **masks generation process** leaks information as well (*F*-test). Generation of the 6 random masks (4 for MixColumn, 2 for SubBytes):



In the **worst case scenario** (profiled attacks), these can be **combined** with other leakage points later to perform a **second order attack**. (M; SBOX(P \oplus K) \oplus M) \Rightarrow SBOX(P \oplus K)

Identification of new leakage points

Interestingly, we can **see the masks manipulation** during the encryption process. The initial (masked) key schedule can also leak information or be profiled for efficient differential fault attack (DFA):



Identification of new leakage points

Interestingly, we can **see the masks manipulation** during the encryption process. The initial (masked) key schedule can also leak information or be profiled for efficient differential fault attack (DFA):



Identification of new leakage points

Interestingly, we can **see the masks manipulation** during the encryption process. The initial (masked) key schedule can also leak information or be profiled for efficient differential fault attack (DFA):



F-test on desynchronised traces

A second order CPA can target – jointly – the **two shares**. Desynchronization-based protections can **reduce this exploitability**.

Leakage evaluation when **simulating desynchronisation** by randomly inserting *n* blocks of *w* NOPs during the execution:



→ Leakage shrinks and becomes **unexploitable** (20000 traces here).

 \rightarrow Provide hints for protecting the design.



Ongoing works

On protected AES (masking, hiding), powerful template attacks need:

- Strong information compression (PCA, LDA) or
- Detection of points of interest
- Resynchronization techniques
- → can become rapidly difficult in practice.

Machine Learning-based analysis can be helpful here [3] [4]

- Deep learning-based attacks against masking
- Denoising and resynchronization with autoencoder

^[3] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, François-Xavier Standaert Template Attacks vs. Machine Learning Revisited (and the Curse of Dimensionality in Side-Channel Analysis). COSADE 2015

^[4] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, Cécile Dumas Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database. IACR ePrint 2018

Fault attacks



Low cost: Clock glitches on a VerifyPIN

Different hardened VerifyPIN have been successfully bypassed:

- Constant-time
- Constant-time and inlined functions
- Constant-time and inlined functions and loop counter
- X Constant-time and inlined functions and double call

Limitations

The ChipWhisperer platform cannot glitch at two different times.

Plan to overcome

We shall shoot with the laser!



Preparatory work (almost 4-5 months)

- Design a custom ChipWhisperer target board:
 - Front-side access
 - Back-side access
- Prepare the target: decapsulate the chip to access the die
- ✓ Mechanical setup of the target on the bench
- ... Mapping out the faults:
 - x-y position,
 - o power,
 - o duration,
 - o delay,
 - type of fault (skip, set, reset, flip, ...)



Laser setup

Characteristics

- IR (1064nm)
- >30ps
- o 0-3W
- 3 objective lenses:
 - o x5 (20μm)
 - o x20 (5μm)
 - o x100 (1μm)





Laser setup

Characteristics

- IR (1064nm)
- >30ps
- o 0-3W
- 3 objective lenses:
 - o x5 (20μm)
 - o x20 (5μm)
 - o x100 (1μm)





Laser setup

Characteristics

- IR (1064nm)
- >30ps
- o 0-3W
- 3 objective lenses:
 - o x5 (20μm)
 - o x20 (5μm)
 - o x100 (1μm)





8-bit microcontroller results

Instruction skip fault model previously validated experimentally [5]





[5] Practical results on laser-induced instruction-skip attacks into microcontrollers. T. Riom, J.-M. Dutertre, O. Potin, J.-B. Rigaud, TRUDEVICE Workshop 2016, Barcelona



This time, all implementations are vulnerable.

- Constant-time
- Constant-time and inlined functions
- Constant-time and inlined functions and loop counter
- Constant-time and inlined functions and double call
- Constant-time and inlined functions and control-flow integrity

Paradox

Constant-time implementation makes laser attacks much easier

32-bit microcontroller

A more complex target (32 bits) implies:

- Larger area to cover for cartography (2.5x2.5cm),
- Complex micro-architecture,
- More time variability



32-bit microcontroller

A more complex target (32 bits) implies:

- Larger area to cover for cartography (2.5x2.5cm),
- Complex micro-architecture,
- More time variability



32-bit microcontroller

A more complex target (32 bits) implies:

- Larger area to cover for cartography (2.5x2.5cm),
- Complex micro-architecture,
- More time variability



Instruction corruption in flash memory

A laser shot in flash memory alters the fetched data on-the-fly.

Ceatech



Bit-set on data (and instructions) fetched from flash memory



Examples of instruction corruption

Modifying a MOVW instruction(32 bits).

bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Referer	nce in	str	ucti	ons																														
Generi	: MOV	W	1	1	1	1	0	i	1	0	0	1	0	0		im	m4		0	i	nm	3		R	d					imr	m8			
MOVW,	RO,	0	1	1	1	1	0	i	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Data co	orrupt	tion	ı																													≁		
MOVW,	RO,	4	1	1	1	1	0	i	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0 0	0
																																		_
Destina	tion	reg	iste	r co	rrup	tior	ı																			≁								
MOVW,	R1,	0	1	1	1	1	0	i	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
																																		_
Opcode	e corr	upt	ion								≁																							
MOVT,	RO,	0	1	1	1	1	0	i	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Constant-time implementation with hardened booleans.

```
1: trials = 3
2: ref_PIN[4] = {1, 2, 3, 4}
3: procedure VerifyPIN(user PIN[4])
     authenticated = FALSE
                                                     C code:
4.
    diff = FALSE
5.
                                                     if (trials > 0)
     if trials > 0 then
6.
       for i \leftarrow 0 to 3 do
7.
         if user PIN[i] != ref PIN[i] then
                                                    Assembly code:
8.
            diff = TRUE
9٠
                                                     CMP R3, 0
       if diff == TRUE then
10.
                                                     BLE address
         trials = trials - 1
11.
       else
12.
         authenticated = TRUE
13.
     return authenticated
14.
```



Performing a bit-set at **index 10**.

Instead of comparing R3, we compare **R7**.

By design, R7 stores the frame pointer, always positive.

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reference in	stru	ctio	ns													
Generic CMP	0	0	1	0	1		Rd					im	m8			
CMP R3, 0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0
Register corruption																
CMP R7, 0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	0	0

Outcome

The *trials* value is **never compared → unlimited** number of trials.

The PIN value can be brute-forced.



Laser fault injection on AES-128

Fault attack on	AES-128	encryption
-----------------	---------	------------

- 1: procedure Add_round_key
- 2: **for i** ← **0 to 3 do**
- 3: **for j** ← 0 to 3 do
- 4: state[i][j] ^= round_key[round][i][j]

```
C code:
for (int i=0; i<4; i++){
  for (int j=0; j<4; j++){
     . . .
  }}
Assembly code:
MOV RO, 0
addr_i:
MOV R1, 0
addr_j:
. . .
ADD R1, 1
CMP R1, 3
BLE addr_j
ADD RO, 1
CMP RO, 3
BLE addr_i
```



Add **5 instead of 1** after executing the body of the loop.

Reference instructions

Generic ADD	0	0	1	1	0		Rd					im	m8			
ADD RO, 1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1
Data corruption																
ADD BO E	\cap	\cap	1	1		\cap	1	\cap	1							

Outcome

The for loop exits after one execution only.



Faulting the for loops

Faulting the **inner** for loop on its first execution

C _{0,0}	<i>C</i> _{1,0}	C _{2,0}	<i>C</i> _{3,0}
$C_{0,1} \oplus K_{0,1}^{10}$	C _{1,1}	C _{2,1}	C _{3,1}
$C_{0,2} \oplus K_{0,2}^{10}$	C _{1,2}	C _{2,2}	C _{3,2}
$C_{0,3} \oplus K_{0,3}^{10}$	<i>C</i> _{1,3}	<i>C</i> _{2,3}	<i>C</i> _{3,3}

Faulting the **outer** for loop

C _{0,0}	$C_{1,0} \oplus K_{1,0}^{10}$	$C_{2,0} \oplus K_{2,0}^{10}$	$C_{3,0} \oplus K_{3,0}^{10}$
C _{0,1}	$C_{1,1} \oplus K_{1,1}^{10}$	$C_{2,1} \oplus K_{2,1}^{10}$	$C_{3,1} \oplus K_{3,1}^{10}$
<i>C</i> _{0,2}	$C_{1,2} \oplus K_{1,2}^{10}$	$C_{2,2} \oplus K_{2,2}^{10}$	$C_{3,2} \oplus K_{3,2}^{10}$
C _{0,3}	$C_{1,3} \oplus K_{1,3}^{10}$	$C_{2,3} \oplus K_{2,3}^{10}$	$C_{3,3} \oplus K_{3,3}^{10}$

What is left?

Holding **one correct** and **two faulty** ciphertexts, the attacker only needs to brute-force the first tenth round-key byte $K_{0.0}^{10}$.

Capabilities

- Temporarily alter instruction/data from flash memory,
- Corrupt the control flow of a program,
- Weaken security of embedded programs.

Limitations

- Bit-set only (so far),
- Adjacent bits only (monospot laser),
- Control-flow corruption mostly.

Future possibilities

Multispot laser

Article available at: https://ia.cr/2018/1042

Combination of protections

For the best: 2nd order CPA made harder

Principle of 2nd order CPA: attack two S-box output bytes. Traditionally, target the two shares (mask + masked value) but two consecutive bytes work well:

- $|\text{Leak}(\text{Sbox}(P_i \oplus K_i) \oplus M') \text{Leak}(\text{Sbox}(P_j \oplus K_j) \oplus M')|$
- $HW(Sbox(P_i \oplus K_i) \oplus M' \oplus Sbox(P_j \oplus K_j) \oplus M')$
- = $HW(Sbox(P_i \oplus K_i) \oplus Sbox(P_j \oplus K_j)) \rightarrow no more mask !$



For the best: 2nd order CPA made harder

Combining leakages is easy when traces are perfectly synchronised.

Ceatech



800 traces required to break 1st-order masked AES on a 32-bit microcontroller embedding an ARM Cortex M3.

Desynchronising + masking is very relevant here!



For the worst

Countermeasure against FA or SCA are usually compatible.

Countermeasure against FA and SCA can be incompatible.

Example

Redundancy-based protection against Fault Injection Analysis can **enhance** side-channel leakages...

SCA is not only for a key recovery purpose, it also helps in temporaly **profiling** fault injection (e.g. bypassing secure boot [6])

Each case must be evaluated separately.

^[6] Niek Timmers, Albert Spruyt, Bypassing Secure Boot using Fault Injection, Black Hat Europe 2016

Conclusion



- Inserting protections at software level provides a high level of flexibility (tuning, combining protections).
- Statistical leakage assessment is an interesting tool to design protections
 - Provides metrics of leakage reduction efficiency
- Advanced profiled techniques (ML) are a promising approach to model a worst case scenario
- New attack on **flash memory** of a 32-bit microcontroller
- Combinations of protections is a double-edged sword



- Inserting protections at software level provides a high level of flexibility (tuning, combining protections).
- Statistical leakage assessment is an interesting tool to design protections
 - Provides metrics of leakage reduction efficiency
- Advanced profiled techniques (ML) are a promising approach to model a worst case scenario
- New attack on flash memory of a 32-bit microcontroller
- Combinations of protections is a double-edged sword

— Questions ? —

Contacts: Brice Colombier brice.colombier@cea.fr

Pierre-Alain Moëllic pierre-alain.moellic@cea.fr

Commissariat à l'énergie atomique et aux énergies alternatives 17 rue des Martyrs | 38054 Grenoble Cedex www.cea-tech.fr

Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019