SR2I301

Sécurité des Systèmes Embarqués

Supports de cours

2020–2021

Jean-Luc Danger
Guillaume Duc
Ulrich Kühne
Laurent Sauvage

Télécom Paris

TELECOM ParisTech

Une école de l'IMT

Attaques par injection de fautes

Laurent Sauvage

Presentation Outline

Introduction to Fault Injection Attack

Countermeasures Against Fault Injection Attack

Fault Analysis on Data Encryption Standard (DES)

Presentation Outline

Introduction to Fault Injection Attack

Countermeasures Against Fault Injection Attack

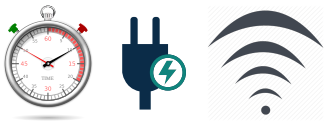
Fault Analysis on Data Encryption Standard (DES)

Cryptography is Everywhere, Mathematically Robust, but...

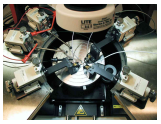
...Threatened by Attacks on Cyber-Physical Systems!

Code injection attack

Fault injection attack



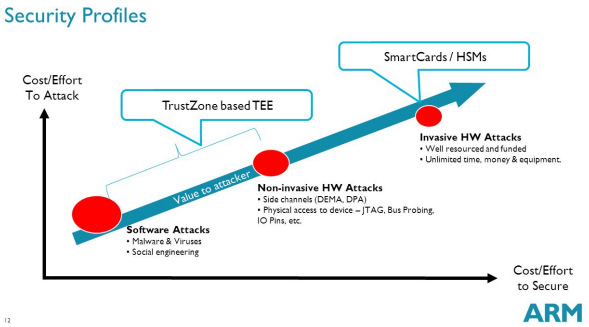
Side-channel attack



Physical attack

Local & Remote

ARM Technology Symposium 2013



Hack of Sony Playstation 3 (2010)

Sony

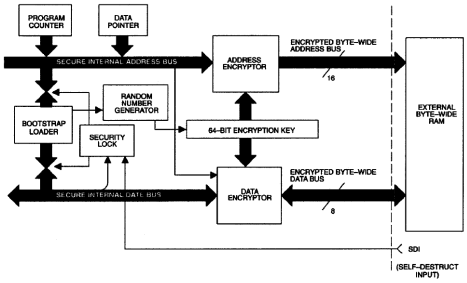
- High resources (human, funding)
- Expert in protecting game console (>20 years)

Hackers

- Have few resources
- Didn't find vulnerability for 3 years
- Hack in 5 weeks: exploit assisted by fault injection attack
- Use low-cost (<$300) equipment

A Secure Microcontroller in 1996

Dallas Semiconductor DS5002FPM



- DES-like encryption of address/data of NV memory
- 64-bit key from on-chip TRNG (unknown by everyone)
- Top-layer coating against microprobing (FIPS 140-1)
- Erasure of secrets w/o V_{CC} (10-year lithium battery)

Breaking Microcontrollers in 1996

Ross J. Anderson and Markus G. Kuhn, Tamper Resistance – a Cautionary Note, In Proceedings of the Second USENIX Workshop ON Electronic Commerce, November 18-21 1996, Oakland, California. ISBN 1-880446-83-9, pp. 1–11

A typical subroutine found in security processors

```
1 b = answer_address
2 a = answer_length
3 if (a == 0) goto 8
4 transmit(*b)
5 b = b + 1
6 a = a - 1
7 goto 3
8 ...
```

- Write a first bytes at memory address @b to the serial port
- Lines 4–7 are executed a times (e.g., $a = 128$ for a RSA-1024 signature)

Breaking Microcontrollers in 1996

Ross J. Anderson and Markus G. Kuhn, Tamper Resistance – a Cautionary Note, In Proceedings of the Second USENIX Workshop ON Electronic Commerce, November 18-21 1996, Oakland, California. ISBN 1-880446-83-9, pp. 1–11

Fault injection attack (non-invasive)

```
1 b = answer_address
2 a = answer_length
3 if (a == 0) goto 8
4 transmit(*b)
5 b = b + 1
6 a = a - 1
7 goto 3
8 ...
```

"(...) a clock (...) or a power glitch (...) transforms either the conditional jump in line 3 or the loop variable decrement in line 6 (dumping) the remaining memory, which if we are lucky will include the keys we are looking for."

Works on any (**single**) check of passwords, access rights, protocol responses, etc.

Differential Fault Analyses Are Powerful!

Exploiting couples of right/wrong computations

Number of faulted ciphertexts (C') to disclose the key (best attacker)

Algorithm	Key space	# C'
RSA (CRT) [BDL97]	2^{1024}	1
RSA-1024 [BDH+97]		3083
DES [BS97]	2^{56}	11
AES-128 [PQ03]	2^{128}	2
AES-256 [TMA11]	2^{256}	2
ECDSA/P-192 [BBB+11]	2^{192}	36
Midori-128 [**?**]	2^{128}	2
R-LWE-based PQC [BBK16]	2^{512}	2304

What about other lighweight & PQC algorithms ?

Presentation Outline

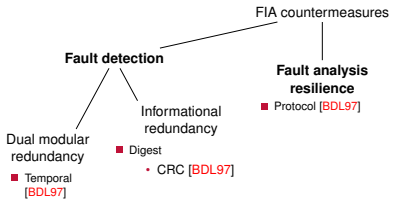
The Bellcore's...

Dan Boneh, Richard A. DeMillo, and Richard J. Lipton, On the importance of checking cryptographic protocols for faults (extended abstract),), Lecture Notes in Computer Science, vol. 1233, Springer, 1997, pp. 37–51

...Countermeasures

1. Check the output of computation using:
   - the same function (Dual modular redundancy)
   - or, the inverse function
2. Add error detection bits (e.g. CRC) to protect critical security parameters (p, q, etc.)
3. Random padding so that the signer never signs the same message (1st DFA requirement)

Protections Against FIA: a Classification



Bitflip on Secret Exponent

Differential Fault Analysis

Feng Bao, Robert H. Deng, Yongfei Han, Albert B. Jeng, A. Desai Narasimhalu, and Teow-Hin Ngair,
Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient
), Lecture Notes in Computer Science, vol. 1361, Springer, 1997, pp. 115–124

$$\begin{aligned} S &= m^d \pmod n \\ &= m_{t-1}^{d_{t-1}} \cdots m_i^{d_i} \cdots m_1^{d_1} m_0^{d_0} \pmod n \quad \text{with } m_i = m^{2^i} \pmod n \\ S' &= m_{t-1}^{d_{t-1}} \cdots m_i^{d'_i} \cdots m_1^{d_1} m_0^{d_0} \pmod n \quad \text{with } d'_i = 1 - d_i \end{aligned}$$

$$\Rightarrow \frac{S'}{S} = \frac{m_i^{1-d_i}}{m_i^{d_i}} \pmod n = \begin{cases} m_i \pmod n & \text{if } d_i = 0, \\ \dfrac{1}{m_i} \pmod n & \text{if } d_i = 1. \end{cases}$$

Bitflip on Secret Exponent

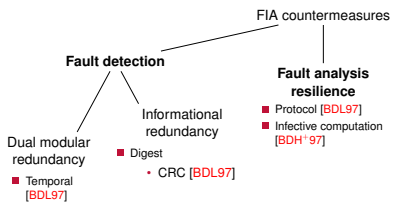
A Countermeasure: Infective Computation

Feng Bao, Robert H. Deng, Yongfei Han, Albert B. Jeng, A. Desai Narasimhalu, and Teow-Hin Ngair,
Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient
), Lecture Notes in Computer Science, vol. 1361, Springer, 1997, pp. 115–124

$$\begin{aligned} m^* &\leftarrow mr \pmod n \qquad \text{with } r \text{ a random number} \\ S^* &\leftarrow m^{*d} \pmod n \\ S &\leftarrow \frac{S^*}{r^d} \pmod n \end{aligned}$$

$$\Rightarrow \frac{S'}{S} = \frac{m_i^{1-d_i} \frac{r_i^{1-d_i}}{r_i^{d_i}}}{m_i^{d_i}} \pmod n \begin{cases} m_i r_i \pmod n & \text{if } d_i = 0, \\ \dfrac{1}{m_i r_i} \pmod n & \text{if } d_i = 1. \end{cases}$$

Protections Against FIA: a Classification



Shamir's Trick (Ring Embedding)

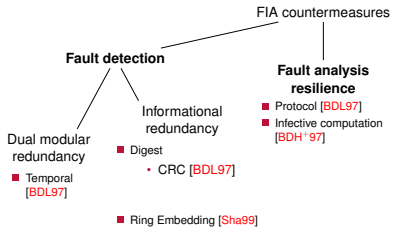
A. Shamir, Method and apparatus for protecting public key schemes from timing and fault attacks, November 23 1999, US Patent 5,991,415

Protecting the computation of $S = \mathrm{CRT}(S_p, S_q)$ with

$$\begin{cases} S_p = m^{d_p} \pmod p, \\ S_q = m^{d_q} \pmod q. \end{cases}$$

1. Select a k-bit random integer r (typically, $k = 32$)
2. Compute
$$\begin{cases} s_p^* = m^d \pmod{rp} \\ s_q^* = m^d \pmod{rq} \end{cases}$$
3. If $s_p^* \equiv s_q^* \pmod r$, return $S = \mathrm{CRT}(s_p^*, s_q^*)$

Protections Against FIA: a Classification



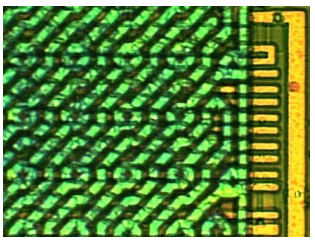
Low-cost Protections

Oliver Kömmerling and Markus G. Kuhn, Design principles for tamper-resistant smartcard processors,), USENIX Association, 1999

3.1 Randomized clock signal
- random insertion of dummy cycles + dummy activity
- jitter on clock cycle

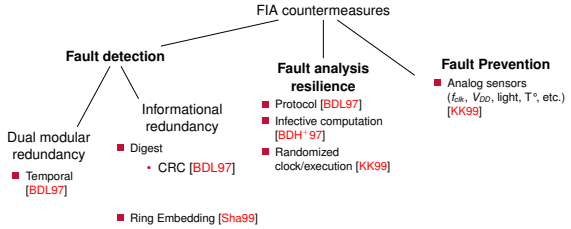
3.2 Randomized multithreading
- Analog sensors (f_{clk}, V_{DD}, light, T°, etc.)

3.6 Top-layer Sensor Meshes



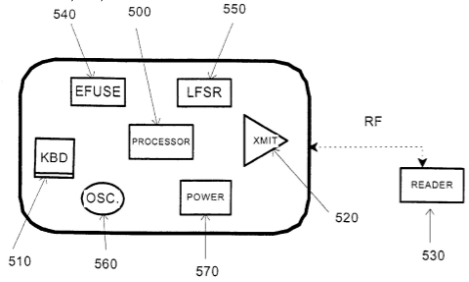
ST16SF48A high-security chip imaged by a confocal microscope

Protections Against FIA: a Classification



A Secure Credit Card in 2001

E.E. Kelley, F. Motika, P.V. Motika, and E.M. Motika, Secure credit card, November 4 2003, "US Patent 6,641,050"

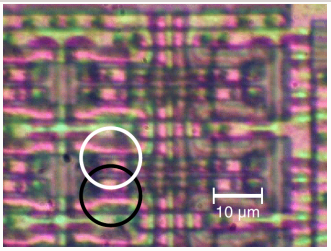


Clock (560) and power (570) generators: Welcome aboard!
Removing of attack paths!

The Dark Side of the Force

Sergei P. Skorobogatov and Ross J. Anderson, Optical Fault Induction Attacks, Cryptographic Hardware and Embedded Systems - CHES 2002, Lecture Notes in Computer Science, vol. 2523, Springer Berlin Heidelberg, 2003, pp. 2–12 (English)

Optical fault induction attack



Illumination of the white (or black) circle causes the SRAM cell (PIC 16F84, 1996, $\geq$350 nm) to change its state from '1' to '0' (or '0' to '1')

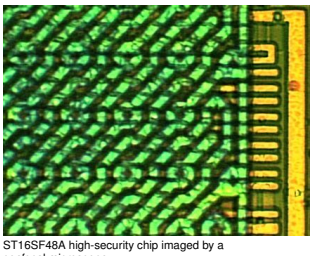
Low-cost Protections

Oliver Kömmerling and Markus G. Kuhn, Design principles for tamper-resistant smartcard processors,), USENIX Association, 1999

3.1 Randomized clock signal
- random insertion of dummy cycles + dummy activity
- jitter on clock cycle

3.2 Randomized multithreading
- Analog sensors (f_{clk}, V_{DD}, light, T°, etc.)

3.6 Top-layer Sensor Meshes



ST16SF48A high-security chip imaged by a confocal microscope

Protections Against FIA: a Classification

FIA countermeasures

Fault detection

Dual modular redundancy
- Temporal [BDL97]

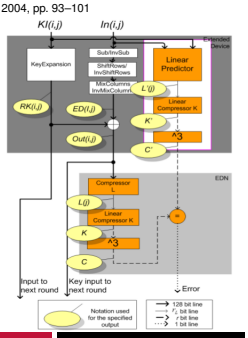
Informational redundancy
- Digest
  - CRC [BDL97]
  - Parity bits [BKK95, BBK+03]
- Ring Embedding [Sha99]

Fault analysis resilience
- Protocol [BDL97]
- Infective computation [BDH+97]
- Randomized clock/execution [KK99]

Fault Prevention
- Analog sensors (f_{clk}, V_{DD}, light, T°, etc.) [KK99]
- Embedded clock oscillator and power source [KMMM03]
- Top-layer coating (shield) [KK99]

Robust Non-linear Codes

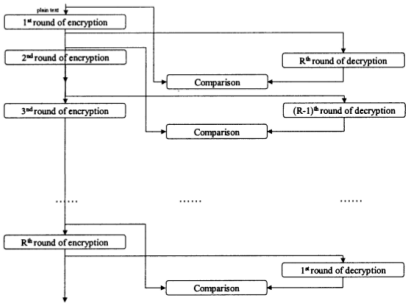
Mark G. Karpovsky, Konrad J. Kulikowski, and Alexander Taubin, Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard, 2004 International Conference on Dependable Systems and Networks (DSN 2004), 28 June - 1 July 2004, Florence, Italy, Proceedings, IEEE Computer Society, 2004, pp. 93–101



- Error coverage
  - linear: 2^{-r}
  - non-linear: 2^{-2r}
- Two cubic networks
- $r = 28 \rightarrow 77\,\%$ area overhead

Concurrent Error Detection at Round Level

Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim, Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers, IEEE Trans. on CAD of Integrated Circuits and Systems **21** (2002), no. 12, 1509–1517



Protections Against FIA: a Classification

FIA countermeasures

Fault detection

Dual modular redundancy
- Temporal [BDL97]
- Spatial/Round [KWMK02]

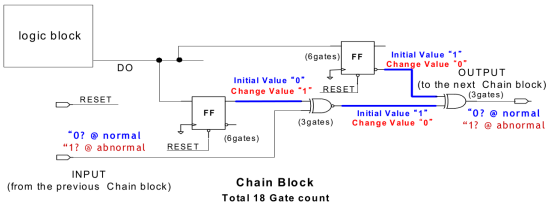
Informational redundancy
- Digest
  - CRC [BDL97]
  - Parity bits [BKK95, BBK+03]
- Ring Embedding [Sha99]
- Robust non-linear codes [KKT04]

Fault analysis resilience
- Protocol [BDL97]
- Infective computation [BDH+97]
- Randomized clock/execution [KK99]
- m-of-n encoding with precharge

Fault Prevention
- Analog sensors (f_{clk}, V_{DD}, light, T°, etc.) [KK99]
- Embedded clock oscillator and power source [KMMM03]
- Top-layer coating (shield) [KK99]
- Virtual cell detector

Virtual Cell detector

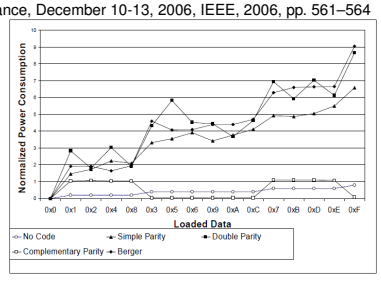
SAMSUNG DIGITall – FDTC 2007



- After reset, each cell output is '0 or '1
- Chain of dual-rail FF, with alarm propagation

Redundancy Makes SCA Easier

Vincent Maingot and Régis Leveugle, Error detection code efficiency for secure chips, 13th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2006, Nice, France, December 10-13, 2006, IEEE, 2006, pp. 561–564

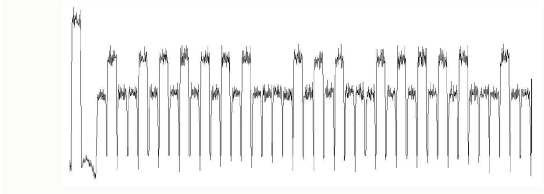


"It is indeed useless to protect a circuit against only one type of attack since the hacker will use the easiest possible attack leading to the expected secret disclosure."

Redundancy vs PACA

Frédéric Amiel, Karine Villegas, Benoît Feix, and Louis Marcel, Passive and Active Combined Attacks: Combining Fault Attacks and Side Channel Analysis, FDTC, IEEE Computer Society, 10 September 2007, Vienna, Austria, pp. 92–102

PACA on atomic S&M ($R_0 = R_0 \times R_k$) w/ redundancy

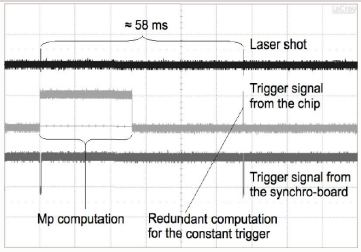


"The fault is detected at the end of the command. Unfortunately, this is too late, even if the result is not returned."

How Does Redundancy Resist HO-FIA ?

Elena Trichina and Roman Korkikyan, Multi fault laser attacks on protected CRT-RSA,), IEEE Computer Society, 2010, pp. 75–86

20-FIA on check and infective countermeasures



- 32-bit ARM Cortex M3
- Laser reload: 200 ms

Pros & Cons of Laser Fault Injection (1/2)

Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria, How to flip a bit?, 16th IEEE International On-Line Testing Symposium (IOLTS) 2010), 5-7 July, 2010, Corfu, Greece, IEEE Computer Society, 2010, pp. 235–239

✔ Effect highly local. . .

5 µm laser beam on ATmega128 chip

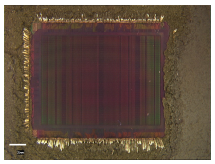
✖ . . .for all technology nodes ?

	Transistor	SRAM Cell
350nm		
130nm		
90nm		
65nm		

Pros & Cons of Laser Fault Injection (2/2)

✖ Sample preparation (depackaging, backside thinning, etc.) quite delicate

✖ High cost



Stratix Altera FPGA

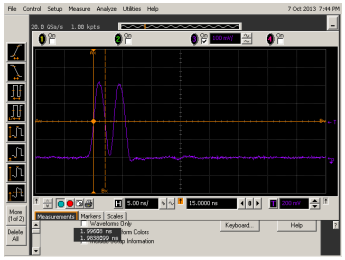
✖ Hard to use against FPGA
- no standing-out area (regular matrix)
- permanent faults (modification of the FPGA configuration) [CLC+09]

Fault Injection Using Radiated EM Pulses

Assumed Benefits

- no sample preparation
- medium cost
- efficient on FPGA
- effect rather local

EM Multi-glitch Injection @ 250 MHz



How do the defense strategies resist HO-FIA ?

- \- Fault detection (bis)
- o Fault Prevention
- \+ Fault analysis resilience

Presentation Outline

À vous de jouer !

Documents supports

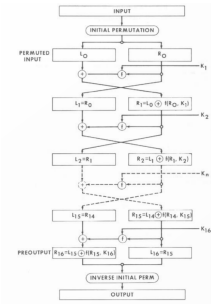
Contenu de l'archive `DFA_DES.zip` :

`fips46-3.pdf`	La publication du FIPS décrivant DES
`des_block.pyc`	Classe python DES
`des_block_demo.py`	Démo utilisation de `des_block`
`DFA_DES_challenges.pyc`	Les paramètres des challenges
`DFA_DES_elev.py`	Code minimal à compléter

Data Encryption Standard (DES) [Nat77]

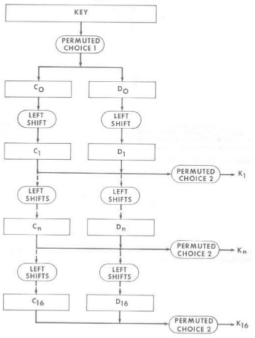
Iterative Feistel Scheme

$$C = \mathrm{IP}^{-1} \circ \left(\bigcirc_{r=1}^{16} \mathrm{F}_{k_r} \right) \circ \mathrm{IP}(P)$$



Data Encryption Standard (DES) [Nat77]

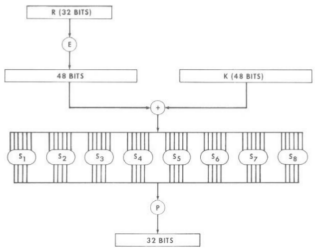
Key Scheduling



Data Encryption Standard (DES) [Nat77]

The Feistel function F

- Expansion (E) from 32 bits to 48 bits
- Key mixing
- Substitution (S_{1-8})
- Permutation (P)



À vous de jouer !

Simple Fault Analysis

Défi 1 : SFA sur chiffrement avorté

- Faute : une seule itération du calcul de la fonction de Feistel (puis IP^{-1})
- $P =$ `0x0000000000000000` et $C' =$ `0x0000000000000000`

Q1 Donner les sous-clés candidates pour $K_{1,1}$.

Q2 En supposant que le nombre de sous-clés candidates est le même pour les autres $K_{1,i}$, donner le nombre de clés candidates pour K_1.

Q3 En déduire le nombre de bits restant à trouver.

À vous de jouer !

Simple Fault Analysis

Défi 2 : SFA sur chiffrement avorté

- La clé de chiffrement est la même que précédemment.
- La faute est également la même, mais l'attaquant :
  - envoie $P = \texttt{0xffffffffffffffff}$;
  - récupère $C' = \texttt{0xffffffffffffffff}$.

Q1 Exprimer l'équation générale en sortie d'une boîte S_i en fonction de R_r et L_r, avec :
- $i \in \{1, \ldots, 8\}$ le numéro de la boîte S ;
- $r \in \{1, \ldots, 16\}$ le numéro du tour.

Q2 En déduire l'équation en sortie de la boîte S_1.

Q3 Donner les nouvelles sous-clés candidates pour $K_{1,1}$.

Q4 À l'aide des résultats du défi précédent, en déduire $K_{1,1}$.

À vous de jouer !

Simple Fault Analysis

Défi 3 : SFA sur calcul des sous-clés avorté

- Faute : une seule itération du calcul des sous-clés (K_2 à K_{16} sont nulles)
- $P = \texttt{0x0000000000000000}$ et $C' = \texttt{0xffffffffffffffff}$

Q1 Donner les sous-clés candidates pour chaque boîte S.

À vous de jouer !

Differential Fault Analysis

TD : théorie de la DFA sur DES

Q1 Donner l'équation de R_{16} en fonction de R_{15} et L_{15}.

Q2 Même question lorsque l'entrée du 16e tour est fautée ($R_{15} \to R'_{15}$, $L_{15} \to L'_{15}$ et $R_{16} \to R'_{16}$).

Q3 En déduire et donner l'équation différentielle de $\Delta R_{16} = R_{16} \oplus R'_{16}$.

Q4 Donner la liste des inconnues, expliquer pourquoi l'attaque n'est pas possible, et proposer une solution.

Q5 Donner la nouvelle équation simplifiée.

À vous de jouer !

Differential Fault Analysis

Défi 4 : DFA au 16e tour sur une boîte S

- $C = \texttt{0x2462db7fdc0060da}$ et $C' = \texttt{0x2467db7fdc00e0ca}$

Q1 Donner l'équation différentielle en sortie de boîte $S1$.

Q2 Chercher puis donner les candidates pour la sous-clé $K_{16,1}$.

À vous de jouer !

Differential Fault Analysis

Défi 5 : DFA au 16e tour sur une boîte S

- La clé de chiffrement et la boîte S fautée sont les mêmes que précédemment
- $C = \texttt{0x1ef932fcde59e25e}$ et $C' = \texttt{0x1efd32fcded9625a}$

Q1 Chercher puis donner la valeur de la sous-clé $K_{16,1}$.

À vous de jouer !

Differential Fault Analysis

Défi 6 : DFA entière

- Liste de 12 couples (C, C')
- Fautes injectées dans R_{16}

Q1 Réaliser une DFA pour extraire la valeur de K_{16}

Q2 En déduire et donner la valeur de la clé maître *KEY*

References I

[ADM+10] Michel Agoyan, Jean-Max Dutertre, Amir-Pasha Mirbaha, David Naccache, Anne-Lise Ribotta, and Assia Tria, How to flip a bit ?, 16th IEEE International On-Line Testing Symposium (IOLTS) 2010), 5-7 July, 2010, Corfu, Greece, IEEE Computer Society, 2010, pp. 235–239.

[AK96] Ross J. Anderson and Markus G. Kuhn, Tamper Resistance – a Cautionary Note, In Proceedings of the Second USENIX Workshop ON Electronic Commerce, November 18-21 1996, Oakland, California. ISBN 1-880446-83-9, pp. 1–11.

[AVFM07] Frédéric Amiel, Karine Villegas, Benoît Feix, and Louis Marcel, Passive and Active Combined Attacks: Combining Fault Attacks and Side Channel FDTC, IEEE Computer Society, 10 September 2007, Vienna, Austria, pp. 92–102.

[BBB+11] Alessandro Barenghi, Guido Marco Bertoni, Luca Breveglieri, Gerardo Pelosi, and Andrea Palomba, Fault attack to the elliptic curve digital signature algorithm wit multiple bit faults,), ACM, 2011, pp. 63–72.

References II

[BBK+03] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri, Error Analysis and Detection Procedures for a Hardware Implementation of the Ad IEEE Trans. Computers **52** (2003), no. 4, 492–505.

[BBK16] Nina Bindel, Johannes A. Buchmann, and Juliane Krämer, Lattice-based signature schemes and their sensitivity to fault attacks, 2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016, IEEE Computer Society, 2016, pp. 63–77.

[BDH+97] Feng Bao, Robert H. Deng, Yongfei Han, Albert B. Jeng, A. Desai Narasimhalu, and Teow-Hin Ngair, Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence), Lecture Notes in Computer Science, vol. 1361, Springer, 1997, pp. 115–124.

[BDL97] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton, On the importance of checking cryptographic protocols for faults (extended abstract),), Lecture Notes in Computer Science, vol. 1233, Springer, 1997, pp. 37–51.

References III

[BKK95] Adrian S. Butter, Chang Y. Kao, and James P. Kuruts, Des encryption and decryption unit with error checking, July 11 1995, US Patent 5,432,848A.

[BS97] Eli Biham and Adi Shamir, Differential Fault Analysis of Secret Key Cryptosystems, CRYPTO, LNCS, vol. 1294, Springer, August 1997, Santa Barbara, California, USA. DOI: 10.1007/BFb0052259, pp. 513–525.

[CLC+09] G. Canivet, R. Leveugle, J. Clediere, F. Valette, and M. Renaudin, Characterization of Effective Laser Spots during Attacks in the Configuration of a V VLSI Test Symposium, 2009. VTS '09. 27th IEEE, 2009, pp. 327–332.

[KK99] Oliver Kömmerling and Markus G. Kuhn, Design principles for tamper-resistant smartcard processors,), USENIX Association, 1999.

[KKT04] Mark G. Karpovsky, Konrad J. Kulikowski, and Alexander Taubin, Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard, 2004 International Conference on Dependable Systems and Networks (DSN 2004), 28 June - 1 July 2004, Florence, Italy, Proceedings, IEEE Computer Society, 2004, pp. 93–101.

References IV

[KMMM03] E.E. Kelley, F. Motika, P.V. Motika, and E.M. Motika, Secure credit card, November 4 2003, "US Patent 6,641,050".

[KWMK02] Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim, Concurrent error detection schemes for fault-based side-channel cryptanalysis of symmetric block ciphers, IEEE Trans. on CAD of Integrated Circuits and Systems **21** (2002), no. 12, 1509–1517.

[ML06] Vincent Maingot and Régis Leveugle, Error detection code efficiency for secure chips, 13th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2006, Nice, France, December 10-13, 2006, IEEE, 2006, pp. 561–564.

[Nat77] National Institute of Standards and Technology, FIPS PUB 46: Data encryption standard (des), Jan 1977.

[PQ03] Gilles Piret and Jean-Jacques Quisquater, A differential fault attack technique against SPN structures, with application to the AES and KHAZAD,), Lecture Notes in Computer Science, vol. 2779, Springer, 2003, pp. 77–88.

References V

[SA03] Sergei P. Skorobogatov and Ross J. Anderson, Optical Fault Induction Attacks, Cryptographic Hardware and Embedded Systems - CHES 2002, Lecture Notes in Computer Science, vol. 2523, Springer Berlin Heidelberg, 2003, pp. 2–12 (English).

[Sha99] A. Shamir, Method and apparatus for protecting public key schemes from timing and fault attacks, November 23 1999, US Patent 5,991,415.

[TK10] Elena Trichina and Roman Korkikyan, Multi fault laser attacks on protected CRT-RSA,), IEEE Computer Society, 2010, pp. 75–86.

[TMA11] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali, Differential fault analysis of the advanced encryption standard using a single fault,), Lecture Notes in Computer Science, vol. 6633, Springer, 2011, pp. 224–233.

SR2I301

Overview of Protections against IC Counterfeiting and Hardware Trojan Horses

Jean-Luc Danger

Outline

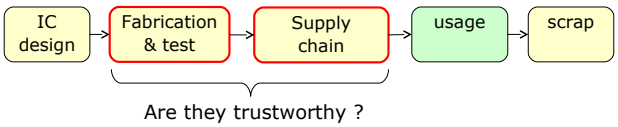
- IC Counterfeiting
  - Overview of the threat
  - Detection methods
  - Prevention methods
- Hardware Trojan Horses
  - Types
  - Detection methods
  - Prevention methods
- Conclusions

IC Counterfeiting is a reality

- The IC **Supply chain** (distributors, brokers,…) is an open door to counterfeit components [1] (SIA report)
- All sectors are impacted, including military[2] 5DoD report)
- Economic Harm
  - Reduction of the Original Component Manufacturer (OCM): market share: $169 billion in 2012 [3]
- Damage due to lack of reliability
  - The counterfeit circuit may be defective
  - Negative image of the OCM

IC counterfeiting

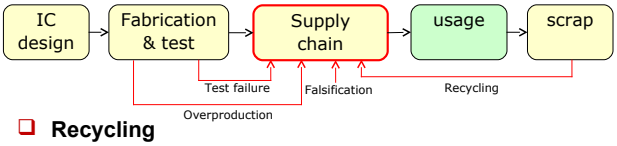
IC life cycle:



- **Counterfeiting types** [4]
  - The circuit is the **original** one but has been illegally manipulated
  - The circuit is **fake**

Counterfeiting with original circuit

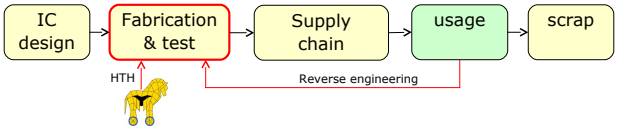
IC life cycle:



- **Recycling**
  - The circuit has been taken from old PCBs and remarked
- **Falsification**
  - The labeling, specification and certification are forged
- **Overproduction**
  - There is no legal contract for fabrication
- **With defects**
  - The component did not pass the tests

Counterfeiting with fake circuit

IC life cycle:



- **Cloning**
  - The circuit has been pirated by *reverse engineering* and redesigned identically
- **Hardware Trojan Horse (HTH)**
  - The circuit has been tampered at the fab stage and some extra logic called Hardware Trojan Horse has been added to spy or destroy it

How to protect from counterfeiting ?

- To work with **trusted partners**
  - Design House : to make ICs in trusted fabs
  - User : to buy ICs to trusted distributors
- To use **detection** techniques
  - For existing devices
  - For new devices with dedicated hardware
- To use **prevention** techniques
  - Only for new devices

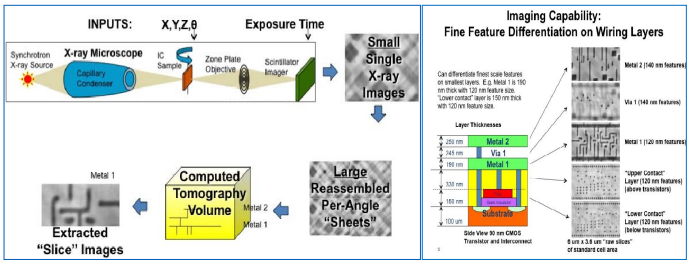
Detection Methods [4]

- 3 main types:
  - **Physical** analysis
    - Can be destructive
    - Only used to detect recycling and forged circuits
  - **Electrical** tests
  - **Aging** tests

Detection: Physical analysis

- Imaging
  - Visual inspection
  - XRAY imaging
  - Scanning Acoustic Microscopy (ultrawave)
  - Scanning Electron Microscopy
- Material Analysis
  - XRAY fluorescence spectroscopy
  - IR spectroscopy (IR absorption)
  - THz spectroscopy (absorption in metal)

Example: X-ray Nanotomography[5]



Detection: Electrical tests

- Integrity tests
  - Scan chain to detect failures
- Parametric tests
  - DC and AC parameters in
  - In various environment
  - To detect abnormal offset
- Functionnal tests
  - To detect out of range ICs
- Burn-in test
  - Accelerate the **aging** and the failure occurence

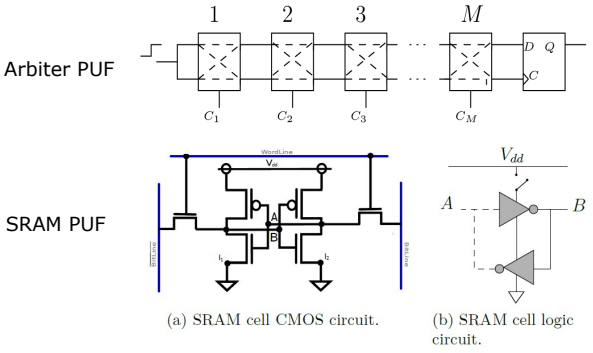
Detection: Aging tests

- Data analysis
  - Delay measurement which is very sensitive to aging
  - Machine Learning algorithms used to classify two sets of data:
    - New trusted devices and
    - unknown devices, presumably new
  - This method is impacted by process variation
- With internal sensors
  - CDIR (Combating Die and IC recycling)
  - Differential structure
    - Ring oscillator reference vs stressed
  - Usage time measurement
    - A clock counter is stored in NVM or OTP (antifuse)

Prevention: Hardware metering

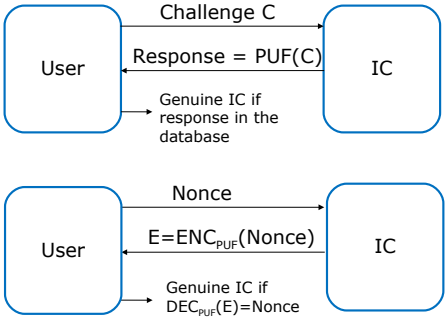
- **Passive** Hardware metering
  - Every circuit has its own ID, either in
  - Non Volatile memory : can be read or tampered
  - Physically Unclonable Function (PUF), which cannot be reverse engineered
  - An authentication protocol is build with the ID
- **Active** Hardware metering
  - The circuit is initially locked. It is unlocked only if the circuit is authenticated.

Prevention: PUF examples



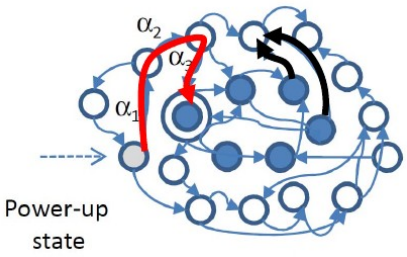
(a) SRAM cell CMOS circuit. (b) SRAM cell logic circuit.

Prevention: PUF-based authentication



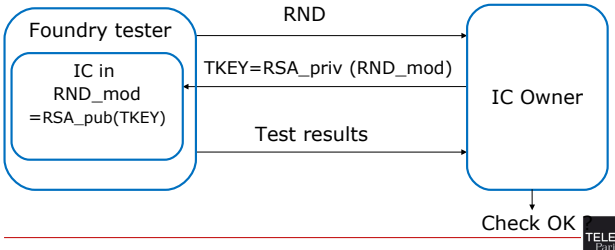
Prevention: locking mechanism [6]

- The state graph is locked at power-up, and unlocked with the correct sequence



Prevention: Secure Split Test [7]

- To secure the manufacturing test, hence counterfeiting with defects
- The test is done by using asymmetric crypto
- The IC owner unlocked the good IC with a master key



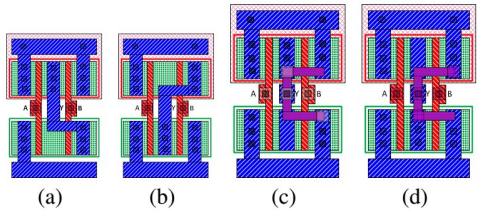
Prevention: Split manufacturing [8]

- The chip manufacturing is split into two steps.
- First step:
  - Done by any foundry, not necessarily trusted
  - In charge of the "**front end of line**": gates and first metal layers
- Second step:
  - Done by a trusted foundry
  - In charge of finishing the connections "**back end of line**"
- IARPA established a new program in 2011 based on split manufacturing: "Trusted Integrated Chips" [8]

Prevention: Watermarking [10]

- To secure IP (Intellectual Property) block inside an IC
- Many ways to insert the mark:
  - GDSII
    - Pattern specific to the design
  - FPGA
    - unused LUTs in the bitstream
  - HDL
    - Unused part of memory, or truth table combinations
  - Synthesis

Prevention: Camouflaging[11]



- Hiding of the cell layout to prevent reverse engineering by optical inspection
  - a: NAND, b: NOR
  - c: camouflaged NAND, d: camouflaged NOR

IC Counterfeiting protection efficiency

		Recycled	Forged	Over produced	Defective	Cloned	HTH
Detection	Physical tests	**	**				
	Electrical tests	**	**		*	*	*
	Aging tests	*	*				
Prevention	HW metering			*		*	
	Secure split test			**	**	*	
	Split manufacturing			*		*	*
	Watermarking					*	
	Camouflaging					*	

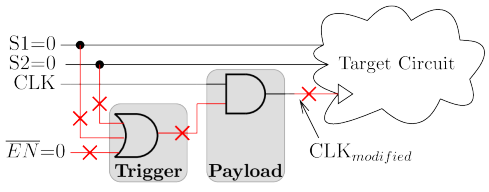
Outline

- IC Counterfeiting
  - Overview of the threat
  - Detection methods
  - Prevention methods
- Hardware Trojan Horses
  - Types
  - Detection methods
  - Prevention methods
- Conclusions

HTH: A powerful and pernicious threat

- HTH:
  - Insertion in an IC of Hardware unknown to the designer
  - Goal: spying, disturbing, destroying
  - Can be inserted at all the levels of the IC design chain
- It is not only an economic threat, it is also strategic
  - 2007 DARPA program "Trust in Ics"
  - 2011 IARPA program" Trusted Integrated Chips"[8] exloiting split manufacturing
- But it is also a weapon for the designer:
  - Backdoors

HTH Principle



- Two components:
  - TRIGGER: Reads and decode internal and rare state
  - PAYLOAD: Writes internal data
- HTH acts as a probing station, both passive (trigger) and active (payload), and is stealthy

HTH Payload examples[12]

- Kill switch
  - Simple payload, desastrous effect as Denial of Services
- Deteriorate the performances
  - Accelerate the aging, add extra delays
- Create leakages
  - Create an access to secret data, either by a functionnal channel or a side-channel
- Assist malwares
  - Exploits a hidden function. The HTH is called backdoor if the designer is the creator.

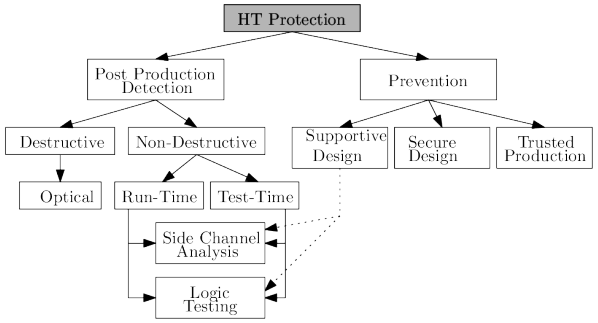
HTH Triggering examples

- Combinatorial
  - Decoding of rare event from multiple nodes
    - trigger = f(nodes)
  - Use significant number of gates
- Sequential
  - Decoding of a rare event from sequential variables
    - Trigger = f(nodes, time)
    - Less nodes but a few flip-flops
- Analog
  - Use internal sensors and external parameters
    - Example: Trigger temperature > threshold
  - Need few gates

HTH Taxonomy [13]

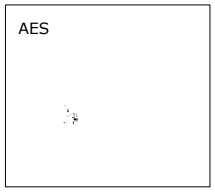
- **Insertion** stage
  - Specification, design, fabrication, test, assembly
- **Abstraction** level
  - RTL, gate, layout, physical,
- **Trigger** type
  - Combinatorial, sequential, analog
- **Payload** type
  - New behavior, less performance, leakages, DoS
- **Physical** characteristics
  - Size, distribution, parametric, functionnal, same layout

HTH protection overview [14]

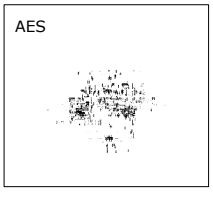


HTH detection by optical method[15]

- Needs a GDSII golden model



Trojan size = 1 AND gate



Trojan size = 128 AND gate

- Comparison between an original GDSII and a trojaned IC with a ×150 lens confocal microscope

HTH detection by optical method

- Cross correlation between the original AES layout and an affected AES layout

		Hardware Trojan size (Nb of AND gates)							
		1	2	4	8	16	32	64	128
Core utilization rate	50%	0.9991	0.9972	0.9981	0.9950	0.9933	0.9918	0.9815	0.9668
	60%	0.9987	0.9968	0.9959	0.9955	0.9944	0.9893	0.9788	0.9670
	70%	0.9989	0.9981	0.9918	0.9941	0.9881	0.9850	0.9594	0.9067
	80%	0.9999	0.9965	0.9898	0.9957	0.9780	0.9711	0.8970	0.8509
	90%	0.9988	0.9990	0.9983	0.9962	0.9832	0.9572	0.8858	0.4010
	95%	0.9997	0.9984	0.9980	0.9889	0.9589	0.9115	0.8824	0.8202
	99%	0.9917	0.938	0.9714	0.9527	0.3798	NC	NC	NC

In black: ECO routing

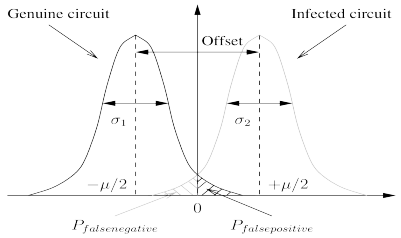
- Trojan almost impossible to insert without changing the layout, if occupancy rate > 80%

HTH detection at test time

- Logic Testing
  - To search the triggering of the Trojan
  - Need exhaustive search of a rare event => impossible
  - Rather use statistical approaches as MERO[15]
  - Or add HW to avoid rare event
- Side Channel
  - To detect the resources of the Trojan
  - By measuring:
    - the Current (IDDQ, IDDT)
    - the EM field
    - the propagation delays
  - Very sensitive to noise and process variance

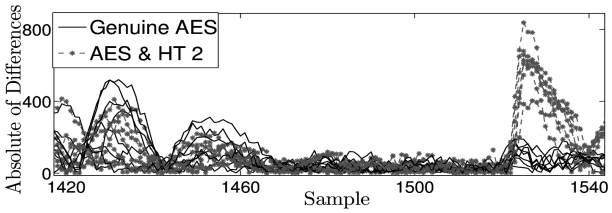
HTH detection with side-channel[17]

- Needs a golden model of the "activity"
- Measurement of local EM field with RF probes
- Impact of noise => Probability of detection



HTH detection with side-channel

- Impact of process variance



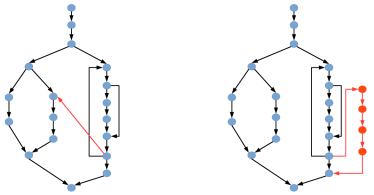
- HTH of different sizes: HTH greater than 1% can be detected with a false negative rate of 0.017%.

HTH detection at run time

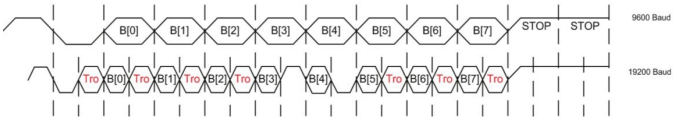
- Techniques to check the integrity in real time
- Can take advantage of SEU and attack detection techniques :
  - Error correction codes
  - Control Flow Integrity (processors)
  - Hardware Assertions checkers
  - Real time security monitor

HTH detection by flow integrity check[18]

- The processor control flow can be tampered by HTH and/or malwares
- Prevention can check the integrity of basic blocks and unexpected jumps
  - Example: Use golden tables of basic blocks CRC and jump tables



HTH detection at run time: assertions[19]



- Example: The HTH outputs a secret key with on the UART channel by doubling the Baudrate
- Property to check by Hardware:
  - The serial bits have to be stable during a fixed period.
  - If the baudrate changes, the assertion fails

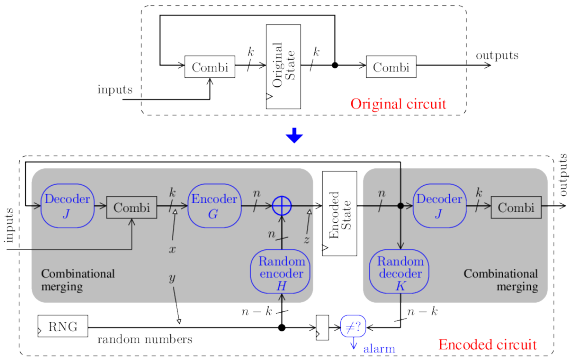
HTH prevention

- **Split manufacturing**
  - Use a "root of trust" with two steps: Front-end of line, and Back end of line
- To use the **layout-filler**
  - No more places to insert HTH on GDSII
- To **avoid rare events** during test time
- **Obfuscation**
  - To obfuscate the state transitions by keys
    - Active **Hardware metering**
  - To obfuscate by error correcting codes (**ECC**)
    - Mask the signals with random variables

Encoding the circuit [20]

- Principle:
  - The HTH has two parts:
    - **probing** (trigger) and **fault** injection (payload)
  - The **registers** are the most easy cells to detect, thus the most easy to probe for Trojan insertion
  - The **sequential variables** in registers are encoded by Linear Complementary Dual Codes (**LCD**)
  - The Dual code allows the designer to use random variables to mask the real computation
- Protection also effective against:
  - Probing attacks
  - Fault attacks
  - Side-Channel Attacks

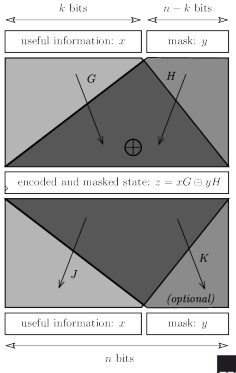
Encoding the circuit: Architecture



Encoding the circuit: Methods

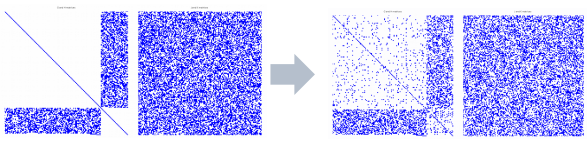
- G encodes k bits
- H is the dual of G
  - $(GH^T=0)$
- Encoding:
  - Z=xG xor yH
  - X=information
  - Y=random variable
- Decoding*
  - $J=G^T(GG^T)^{-1}$
  - $K=H^T(HH^T)^{-1}$

* Moore-Penrose pseudo-inverse



Encoding the circuit: Security Proof

- The code [n,k,d] has a proven security of d:
  - The HTH **trigger** is inefficient with less than d probes
  - The HTH **payload** is inefficient if it modifies less than d nets
- Complexity
  - Choose low density codes to encode and decode



Encoding complexity

Table 8.6 – Synthesis results of encoded circuit method, and security parameters for the SIMON co-processor.

IC (Code)	Gates	Area (μm^2)	n	k	$d_{Trigger}$	$d_{Payload}$
Original ([109,109,1])	300	1919	109	109	1	1
Encoded ([110,109,2,1])	560	3567	110	109	2	1
Encoded ([140,109,10,6])	3107	20239	140	109	10	6
Encoded ([123,109,5,3])	2348	15249	123	109	5	3

Conclusions

- Methods for counterfeiting and HTH insertions are sophisticated and increasing.
- Many protections:
  - But need resources:
    - Tools and methods for detection
    - Extra Silicon and methods for prevention
    - Split foundries
  - The optimal solution is still a challenge
    - Combination of techniques
    - With reduced complexity to get higher detection or avoidance rate
  - But very few inputs from the industrials

Key References

1. http://www.semiconductors.org/document_library_sia/anti_counterfeiting/sia_whitepaper_winning_the_battle_against_counterfeit_semiconductor_products/
2. http://www.armed-services.senate.gov/download/inquiry-into-counterfeit-electronic-parts-in-the-department-of-defense-supply-chain
3. http://www.zdnet.com/article/counterfeit-chips-a-169-billion-tech-supply-chain-headache/
4. Guin, U., Huang, K., DiMase, D., Carulli, J. M., Tehranipoor, M., & Makris, Y. (2014). Counterfeit integrated circuits: a rising threat in the global semiconductor supply chain. Proceedings of the IEEE, 102(8), 1207-1228.
5. Bajura, M., Boverman, G., Tan, J., Wagenbreth, G., Rogers, C. M., Feser, M., ... & Reynolds, P. (2011, March). Imaging Integrated Circuits with X-ray Microscopy. In Proceedings of the 36th GOMACTech Conference.
6. Alkabani, Y., & Koushanfar, F. (2007, August). Active Hardware Metering for Intellectual Property Protection and Security. In USENIX Security (pp. 291-306).
7. Contreras, G. K., Rahman, M. T., & Tehranipoor, M. (2013, October). Secure split-test for preventing IC piracy by untrusted foundry and assembly. In Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium on (pp. 196-203). IEEE.
8. Jarvis, R. W., & McIntyre, M. G. (2007). U.S. Patent No. 7,195,931. Washington, DC: U.S. Patent and Trademark Office.
9. http://www.iarpa.gov/index.php/research-programs/tic
10. Charben, E. (1998, May). Hierarchical watermarking in IC design. In Proceedings of the IEEE Custom Integrated Circuits Conference (pp. 295-298). IEEE.
11. Rajendran, J., Sam, M., Sinanoglu, O., & Karri, R. (2013, November). Security analysis of integrated circuit camouflaging. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (pp. 709-720). ACM.
12. https://www.trust-hub.org
13. Rajendran, J., Gavas, E., Jimenez, J., Padman, V., & Karri, R. (2010, May). Towards a comprehensive and systematic classification of hardware trojans. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on* (pp. 1871-1874). IEEE.
14. Francq, J., & Frick, F. (2015, March). Introduction to hardware trojan detection methods. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition* (pp. 770-775). EDA Consortium.
15. Bhasin, S., Danger, J. L., Guilley, S., Ngo, X. T., & Sauvage, L. (2013, August). Hardware trojan horses in cryptographic ip cores. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on* (pp. 15-29). IEEE.
16. Chakraborty, R. S., Wolff, F., Paul, S., Papachristou, C., & Bhunia, S. (2009). MERO: A statistical approach for hardware Trojan detection. In *Cryptographic Hardware and Embedded Systems-CHES 2009* (pp. 396-410). Springer Berlin Heidelberg.
17. Ngo, X. T., Najm, Z., Guilley, S., Bhasin, S., & Danger, J. L. (2014). Method Taking into Account Process Dispersion to Detect Hardware Trojan Horse by Side-Channel. *Proc. Security Proofs for Embedded Systems–PROOFS*.
18. Danger, J. L., Guilley, S., Porteboeuf, T., Praden, F., & Timbert, M. (2014, December). HCODE: Hardware-Enhanced Real-Time CFI. In *Proceedings of the 4th Program Protection and Reverse Engineering Workshop* (p. 6). ACM.
19. Ngo, X. T., Danger, J. L., Guilley, S., Najm, Z., & Emery, O. (2015, August). Hardware property checker for run-time Hardware Trojan detection. In *Circuit Theory and Design (ECCTD), 2015 European Conference on* (pp. 1-4). IEEE.
20. Ngo, X. T., Guilley, S., Bhasin, S., Danger, J. L., & Najm, Z. (2014, October). Encoding the state of integrated circuits: a proactive and reactive protection against hardware Trojans horses. In *Proceedings of the 9th Workshop on Embedded Systems Security* (p. 7). ACM.

SR2I301 : Sécurité des Systèmes Embarqués

Introduction

Guillaume Duc
guillaume.duc@telecom-paris.fr
P4 2020–2021

Plan

Déroulement du module & Consignes

- Site pédagogique sur eCampus
  - Supports des cours
- Fonctionnement particulier cette année
  - Distanciel synchrone (Zoom) pour la plupart des cours
  - TP/TD en présentiel si possible en fonction de la situation sanitaire
  - Référez vous à l'emploi du temps sur Synapses
  - Pas de copie papier des supports de cours

Notation

- Examen (prévu le jeudi 25 juin 13h30–15h00)
  - QCM sur les points importants des différentes parties du module
  - En présentiel, sauf impossibilité
  - Tous les documents sont autorisés
  - Durée : 1h30

Plan

Sécurité vs. sûreté de fonctionnement

- *Sûreté de fonctionnement* (*Dependability*) : propriété qui permet aux utilisateurs du système de placer une confiance justifiée dans le service qu'il leur délivre
- Plusieurs facettes
  - *Disponibilité* (*Availability*) : aptitude du système à être prêt à délivrer un service correct
  - *Fiabilité* (*Reliability*) : aptitude du système à assurer la continuité du service correct durant un certain laps de temps
  - *Sécurité-innocuité* (*Safety*) : aptitude du système à éviter des conséquences catastrophiques sur son environnement
  - ***Sécurité* (*Security*) : aptitude du système à préserver la confidentialité et l'intégrité des informations**
  - *Maintenabilité* (*Maintainability*) : aptitude du système à être maintenu ou remis en état de fonctionnement

Sécurité des SE == Sécurité Info ?

- *An embedded system is a computer system designed to do one or a few dedicated and/or specific functions often with real-time computing constraints.* (source Wikipedia)
- ⇝ À la base, suivant cette définition, un système embarqué est un ordinateur
- ⇝ Pourquoi la sécurité des systèmes embarqués nécessite un module particulier alors qu'un cours de sécurité informatique pourrait/devrait suffire ?
- ⇝ L'objectif de ce cours d'introduction est de présenter les caractéristiques qui font que les systèmes embarqués se distinguent des systèmes informatiques classiques du point de vue de la sécurité

Objectifs de sécurité génériques

Confidentialité Garantir que seuls les personnes/programmes autorisés peuvent accéder à une information

Intégrité Garantir qu'une information ne peut pas être modifiée, involontairement ou volontairement, ou garantir la détection de cette altération

Authentification Prouver l'identité d'une personne, d'un programme, d'une machine, etc.

Disponibilité Garantir qu'un système reste en permanence utilisable par les personnes autorisées

Traçabilité Permettre de savoir qui a fait quoi sur un système informatique

Particularités des Systèmes Embarqués

- Fréquemment, le propriétaire des informations (au sens large) manipulées par un système embarqué n'est pas le seul à avoir un accès (physique ou à distance) à ce système embarqué
  - Carte bancaire qui contient des secrets de la banque émettrice
  - Carte SIM qui contient des secrets de l'opérateur
  - Set-top box qui contient des secrets liés à la gestion des droits (DRM)
- Le système est souvent physiquement dans les mains de l'adversaire
- L'utilisateur légitime peut même avoir un intérêt à obtenir ces informations (et donc devenir l'adversaire)...
- ⇝ L'objectif de confidentialité peut ainsi être important mais aussi ciblé par l'adversaire

Particularités des Systèmes Embarqués

- Certaines catégories d'attaques sont souvent peu ou pas traitées dans le cadre de la sécurité informatique mais sont à prendre en compte pour les systèmes embarqués
  - Exemple : Attaques matérielles
  - Le système étant dans les mains de l'adversaire (légitimement ou non), celui-ci peut l'espionner, voire même le modifier
  - Mécanismes de déverminage (JTAG par exemple)

Particularités des Systèmes Embarqués

- Contraintes de coût, de taille, de performance (y compris l'aspect temps réel), de délai de mise sur le marché, qui peuvent entraîner une négligence de la sécurité de la partie logicielle
  - Développement trop rapide (pas assez de temps pour déverminer le programme)
  - Implémentation partielle de certains protocoles
  - Utilisation d'algorithmes moins robustes à cause des contraintes

Plan

Introduction

- Système embarqué
  - Matériel
    - Composants de calcul : micro-contrôleur, micro-processeur, SoC...
    - Composants de stockage : ROM, RAM, stockage de masse
    - Composants de communication : bus, périphériques d'E/S (UART, I2C, CAN, WiFi, ZigBee, Ethernet...)
    - Déverminage : JTAG
  - Logiciel
    - Bootloader
    - OS (temps réel, micro-noyau, hyperviseur, générique...)
    - Applications
- Attaques possibles sur chaque élément

Attaques matérielles

- Visent potentiellement tous les composants matériels qui constituent le système embarqué
- Deux grandes catégories

Attaques non intrusives Ne laissent pas de traces sur le circuit qui reste fonctionnel après l'attaque

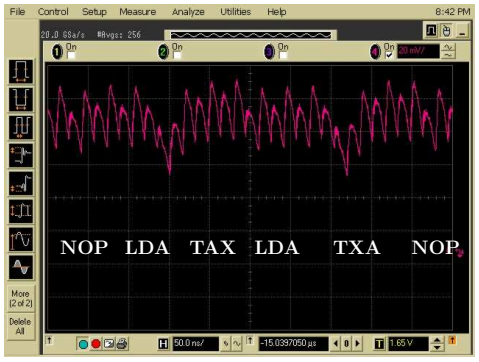
Attaques intrusives Laissent des traces évidentes sur le circuit qui peut ne plus fonctionner après l'attaque

- La distinction entre ces deux catégories est parfois floue

Side-Channel Attacks

- Attaque non intrusive
- Objectif : Obtenir des informations sur les données manipulées par un circuit (vise la propriété de confidentialité)
- Exemple : Trouver une clé secrète embarquée dans un composant cryptographique (carte à puce, ASIC, FPGA...)
- Méthode : Observer les canaux *auxiliaires* (ex. consommation d'un circuit, temps nécessaire pour effectuer le calcul, rayonnement électromagnétique émis, etc.) du composant dont l'activité peut être corrélée aux données sensibles manipulées
- Domaine de recherche très actif, nombreuses attaques réalisées en pratique : Timing Attack, SPA, DPA, CPA, EMA...

Power trace of a 6502 (8-bit CISC) microcontroler



Espionnage du bus mémoire

- Attaque non intrusive à moyennement intrusive
- Objectif : Obtenir des informations sur les données (au sens large : code et données) manipulées par le système (vise la propriété de confidentialité), voire les modifier (propriété d'intégrité)
- Exemple : Récupérer du code ou des données sensibles, voire les modifier pour obtenir un avantage

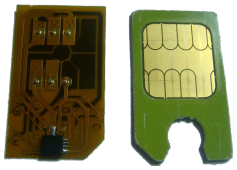
Exemple : Microsoft XBox

- Objectif de sécurité : Empêcher la modification du bootloader ou du noyau
  - Bloc de démarrage secret codé en dur dans l'ASIC du southbridge
  - Bootloader chiffré stocké dans une flash, déchiffré et vérifié par le bloc de démarrage
- Vulnérabilité : le bloc de démarrage doit être exécuté par le CPU et donc doit être transféré sur le bus reliant le southbridge au CPU
  - Même si ce bus est rapide, il peut être espionné ⇝ analyse du code de démarrage et récupération de l'algorithme de vérification du bootloader [3]

Espionnage des bus de communication

- Attaque non intrusive
- Attaque MITM (Man-In-The-Middle) sur les entrées/sorties d'un système embarqué

Exemple : X-SIM



- La X-SIM intercepte et modifie les APDU échangés entre le téléphone et la carte SIM
- Utilisé par exemple pour desimlocker un téléphone en lui faisant croire qu'il s'agit d'une SIM d'un autre opérateur

Injection de fautes

- Attaque non intrusive ou intrusive en fonction des cas
- Objectif : Perturber le bon fonctionnement du système embarqué pour lui faire commettre des erreurs
- Conséquences
  - Désactivation de protection (ex. vérification d'un code PIN)
  - Obtention d'informations secrètes (ex. récupération d'une clé secrète RSA en injectant une seule faute [1], idem pour AES en injectant quelques fautes)
- Méthodes très variées
  - Variations de la tension d'alimentation, de l'horloge, de la température de fonctionnement du circuit (pour obtenir une violation d'un chemin critique du circuit ou une erreur d'écriture en mémoire)
  - Bombardement du circuit avec des particules énergétiques (flash de lumière, laser, particules radioactives α ou β)

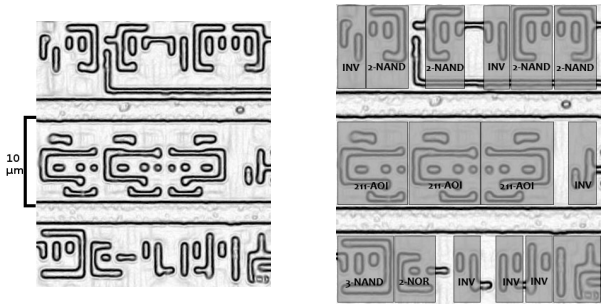
Rétro-conception de composant

- Attaque intrusive
- Objectif : Obtenir le schéma du composant pour, en général, comprendre un algorithme secret implémenté en matériel ou pour lire le contenu d'une ROM interne
- Méthode
  - Découpage du circuit en fines tranches
  - Photographie de chacune des tranches
  - Reconstruction du schéma

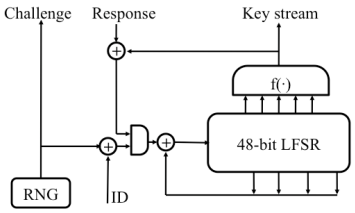
Exemple : MIFARE Classic

- Carte sans-contact (RFID) contenant de la mémoire non volatile et un algorithme d'authentification secret (Crypto-1) pour en protéger l'accès
- Utilisé principalement pour du contrôle d'accès et du ticketing (métro de Londres)
- Henryk Plötz and Karsten Nohl [4] ont réussi à retrouver l'algorithme
  - Ils ont découpé et photographié les différentes couches du circuit puis reconstitué le schéma électronique du circuit
- En analysant l'algorithme, ils ont découvert plusieurs faiblesses cryptographiques dans cet algorithme
- `http://events.ccc.de/congress/2007/Fahrplan/events/2378.en.html`

Exemple : MIFARE Classic [4]



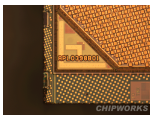
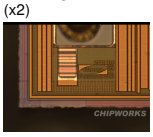
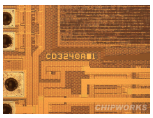
Exemple : MIFARE's CRYPTO1 exposed [4]



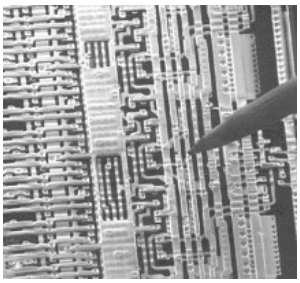
Images of MIFARE by Karsten Nohl, David Evans, Starbug and Henryk Plötz

Exemple : Analyse de composants

Exemple de déshabillage de l'iPad par `http://www.chipworks.com/`

Apple / Microprocesseur (avec DRAM)	Samsung / 1 Gb mobile DDR SDRAM (x2)
	
Broadcom / Microcontrolleur avec NVM (écran tactile)	**TI / Driver écran tactile**
	

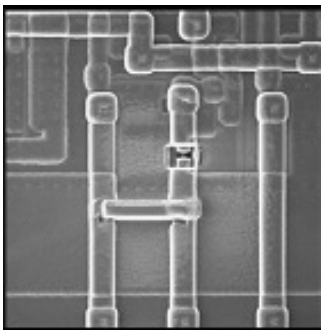
Espionnage par micro-sonde



- Attaque intrusive
- Permet par exemple de lire les données circulant sur un bus interne (clés de chiffrement, données en clair, etc.)
- Limitations : Permet de lire que quelques fils en même temps et seulement sur le niveaux de métallisation supérieurs

Image issue de la thèse de Christophe Giraud [2]

Modification d'un circuit à l'aide d'un FIB (Focused Ion Beam)



- Attaque (très) intrusive
- Désactivation de capteurs de sécurité
- Restauration de fusibles (test mode...)
- Technique ultime

Image issue de la thèse de Christophe Giraud [2]

Attaques logicielles classiques

- La partie logicielle (OS + applications) est vulnérable aux attaques logicielles classiques
- Source : 2019 CWE Top 25 Most Dangerous Software Errors, `http://cwe.mitre.org/top25/`
- Page intéressante à lire car apporte de nombreux détails sur chacune des vulnérabilités et sur les solutions possibles
- Toutes ces erreurs ne s'appliquent pas forcement à tous les systèmes embarqués mais elle constitue néanmoins une bonne base de départ

Top 25 Most Dangerous Software Errors I

1. Improper Restriction of Operations within the Bounds of a Memory Buffer
2. Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
3. Improper Input Validation
4. Information Exposure
5. Out-of-bounds Read
6. Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
7. Use After Free
8. Integer Overflow or Wraparound
9. Cross-Site Request Forgery (CSRF)

Top 25 Most Dangerous Software Errors II

10. Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
11. Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
12. Out-of-bounds Write
13. Improper Authentication
14. NULL Pointer Dereference
15. Incorrect Permission Assignment for Critical Resource
16. Unrestricted Upload of File with Dangerous Type
17. Improper Restriction of XML External Entity Reference
18. Improper Control of Generation of Code ('Code Injection')
19. Use of Hard-coded Credentials

Top 25 Most Dangerous Software Errors III

20. Uncontrolled Resource Consumption
21. Missing Release of Resource after Effective Lifetime
22. Untrusted Search Path
23. Deserialization of Untrusted Data
24. Improper Privilege Management
25. Improper Certificate Validation

Vulnérabilités logicielles spécifiques

- Tendance de certains développeurs à vouloir ré-inventer la roue pour des prétextes plus ou moins valables
  - OS
  - Pile réseau
  - Mini serveur web embarqué
  - Algorithmes de crypto légers
  - ...
- En général, cette re-implémentation introduit des failles en tout genre

Vulnérabilités logicielles spécifiques

- Absence de mécanismes de sécurité au prétexte que le système est inaccessible
  - Pas de mot de passe administrateur alors qu'un port RS232 est disponible sur le circuit (mais sans le connecteur)
  - Pacemaker configurable à distance (NFC) sans mécanisme de sécurité...

Plan

Systèmes embarqués dédiés à la sécurité

- Certains systèmes embarqués visent des objectifs de sécurité importants
- Le plus emblématique est la carte à puce
- Certaines cartes à puce (ex. cartes bancaires ou PayTV) intègrent de nombreuses contre-mesures contre les attaques matérielles et logicielles présentées précédemment

Exemple de contre-mesures sur carte à puce

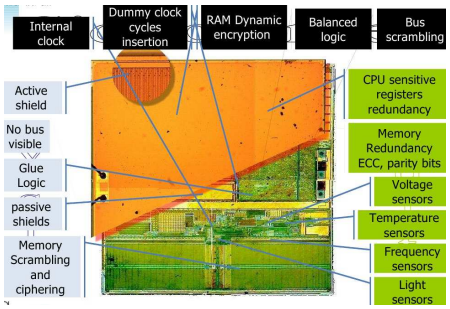


Image Assia Tria

Plan

Contenu de l'UE

- Préalables
  - Rappels d'électronique
  - Implémentation matérielle des algorithmes de cryptographie
- Attaques et contre-mesures
  - Canaux auxiliaires
  - Injection de fautes
  - Rétro-conception
  - Espionnage de bus
- Technologies
  - Génération d'aléa (TRNG, PUF)
  - Support matériel pour la sécurité logicielle

Plan

References I

[1] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton.
On the Importance of Checking Cryptographic Protocols for Faults.
In *Proceedings of Eurocrypt'97*, volume 1233 of *LNCS*, pages 37–51. Springer, May 11-15 1997. Konstanz, Germany.

[2] Christophe Giraud.
Attaques de cryptosystèmes embarqués et contre-mesures associées.
PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, 26 octobre 2007.

[3] A. Huang.
Keeping secrets in hardware : the Microsoft XBox (TM) case study.
Technical Report AI Memo 2002-008, Massachusetts Institute of Technology, May 2002.

[4] Karsten Nohl, David Evans Starbug, and Henryk Plötz.
Reverse-Engineering a Cryptographic RFID Tag.
In *USENIX Security Symposium*, pages 185–193, July 31 2008.
San Jose, CA, USA (http://www.usenix.org/event/sec08/tech/full_papers/nohl/nohl_html/Online HTML).

Side-Channel Analysis

An introduction

Ulrich Kühne
ulrich.kuhne@telecom-paris.fr
2020–2021

Plan

What it's all about...

- Understanding the notion of side-channel analysis (SCA)
- Understanding classic side-channel attacks
- Understanding counter-measures against side-channel attacks

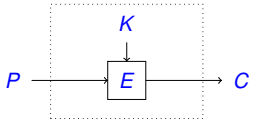
General Context

- Algorithm
- Implementation
  - Hardware (ASIC, FPGA...)
  - Software running on a processor (soft-core on an FPGA, micro-controller in an embedded system, general purpose CPU, specialized processor)
- With a specific security objective
  - Confidentiality (example: cipher algorithm)
  - Authentification (example: PIN code verification)
  - ...
- Handling a secret (can be the algorithm itself) that must not be accessible to the adversary

Example

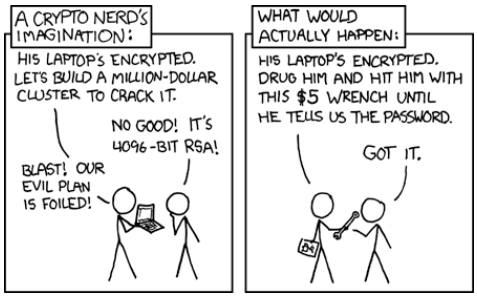
- Example: Cryptographic algorithm implemented on a smart card
- Input: plain text message
- Output: encrypted message
- By construction, the cryptographic key, which is embedded within the smart card, is not accessible via any operation on the input/output interface of the card.

Mathematical View



- KERCKHOFFS principle: P, C et E are public, security depends on K, which is unknown to the adversary
- There are numerous robust algorithms following this model

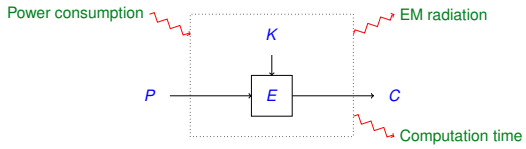
Cryptanalysis vs Reality...



[Source: https://www.xkcd.com/538/]

In real life...

...there's hardware



- Additional input/output channels: Side-channels
  - Electromagnetic radiation (EM)
  - Power consumption
  - Computation time
  - ...

Side-channel Attacks

- Side-channels depend on the implementation of an algorithm:
  - In software
  - In hardware
- Side-channels cannot be observed on the algorithmic (mathematical, cryptanalytic) level.
- The implementation may leak sensitive information (secrets) via side-channels, even if those secrets never appear on the input/output interface.
- As a consequence, a passive observation can allow an attacker to get hold of the secret!

Concrete Example

Function verifying a PIN code

```
boolean verifyPIN(byte[] inputPIN)
{
  for (int i = 0; i < correctPIN.length; i++)
    if (inputPIN[i] != correctPIN[i])
      return false;

  return true;
}
```

- Suppose that the arrays `inputPIN` and `correctPIN` have size 4 and contain digits only (0–9)
- What is the complexity of an exhaustive search (try all the PINs)?
- Can the attacker be smarter than that?

Concrete Example

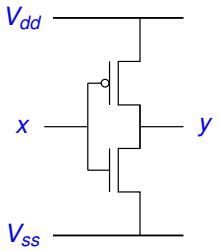
Function verifying a PIN code

- The attacker can measure the function's execution time
- Note that the function returns once it finds a wrong digit
- The attacker can try 0xxx, 1xxx, ..., 9xxx
- One of those digits will result in a slightly longer execution, indicating the first correct digit
- Using this result, she can repeat the same test for the second (third, fourth) digit
- Complexity: We need a maximum of 40 tests (vs 9999 tests for an exhaustive search)
- The side-channel exploited by the attacker is the execution time ⇒ timing attack

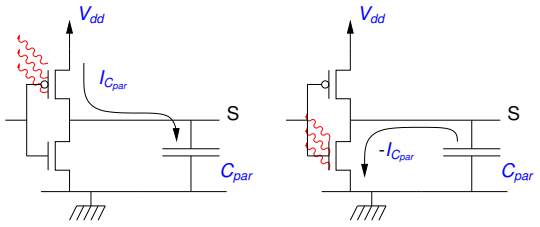
Plan

Power Consumption of a CMOS Circuit
The inverter

- Given input $x = 0$
  - → $V_x = 0$
    - → nMOS is blocking
    - → pMOS is open
  - → $V_y = V_{dd}$
  - → Logic output is $y = 1$
- Given input $x = 1$
  - → $V_x = V_{dd}$
    - → nMOS is open
    - → pMOS is blocking
  - → $V_y = 0$
  - → Logic output is $y = 0$



Power Consumption of a CMOS Circuit
Energy dissipation



Rising edge Falling edge

Power Consumption of a CMOS Circuit
Information leakage

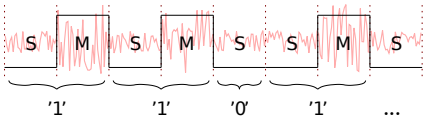
- Except for static leakage current, a CMOS circuit only consumes power during state changes of its gates (dynamic power consumption)
- By observing the power consumption of a circuit, we can deduce its activity
- Note that the number of gates changing their output depends on both the operations and the manipulated data
- Thus, the power consumption can reveal information on the executed operations and the involved data, including secrets

Simple Power Analysis (SPA)
Example: RSA

- Modular exponentiation algorithm

1: **Inputs** : M , K
2: $R = 1$;
3: **for** $i = |K| - 1; i \geq 0$; $i - -$ **do**
4: $R = R^2$;
5: **if** $K_i == 1$ **then**
6: $R = R \times M$;
7: **end if**
8: **end for**
9: **Return** $R = M^K$;

- Power consumption profile



Simple Power Analysis (SPA)
Example: RSA

- Recovery of the full secret (i.e. the key in case of RSA) with a single measurement
- Information is leaked due to different operations depending on the secret (multiply vs square) with a different power consumption profile.
- This type of attack using a single measure is called *Simple Power Analysis*
- Note that the computation time also leaks some information (difficult to exploit in this case)

Plan

Differential Power Analysis

- Often, the leakage is not as obvious
- Need to use a large number of measures
- Need to use statistical tools
- This type of attack is called DPA (*Differential Power Analysis*)
- There are several variants (CPA, ...)

DPA: The Ingredients

Leakage Model $\mathcal{M}$ A model (function) predicting the behavior of the observed side-channel of the system, depending on a hypothesis on the system state

Distinguisher $\mathcal{D}$ Statistical tool that allows to detect a correlation between the real system's behavior and our prediction

- Since the internal state of the system – in particular the secret – is unknown to the attacker, we need to make a hypothesis
- This hypothesis can be correct or wrong
- The distinguisher allows us to tell the good hypothesis (correct key) from the wrong ones (wrong keys)

DPA Manual 1/2

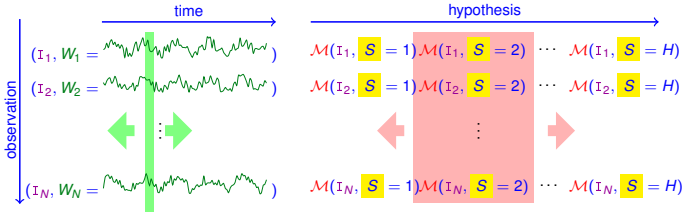
1. Determine a sensitive variable S depending on a part of the secret and on known inputs or outputs.
2. Establish a leakage model $\mathcal{M}(S)$ depending on S.
3. Perform observations (measurements) of the circuit's behavior on the considered side-channel, varying the known inputs or outputs.

DPA Manual 2/2

4. Analyze the data: For each possible value of S
   - For each known input/output P used during the observations, calculate $\mathcal{M}(S, P)$
   - Use the distinguisher $\mathcal{D}$ to check if there is a correlation between the behavior predicted by the leakage model (depending on the hypothesis) and the real world observations

- For the correct value of S, the leakage model predicts correctly the circuit's behavior. As a consequence, the observations will be correlated to the model, and the distinguisher will detect this correlation.
- For all other (wrong) values of S, the model does not predict correctly the behavior, and there will be no correlation between the model and the observations.

DPA Overview



- $\mathtt{I}_i$: Plain text message (or other known inputs/outputs)
- W_i: Measured power consumption (power trace)
- $\mathcal{M}$: Leakage model, depending on secret S (and possibly known inputs/outputs)

$\Rightarrow$ Find a correlation between and

Performing a DPA Attack

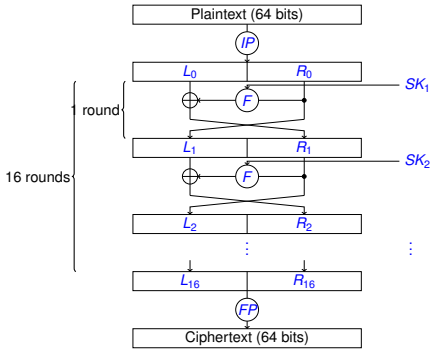
1. Which leakage model to choose?
2. Which distinguisher to choose?
3. How to perform the measurements?

Example

- Context: Hardware implementation of DES *(Data Encryption Standard)* in ECB mode
- What we are looking for: key (56 bits)
- The adversary can send plain text messages to the circuit
- She can read the cipher text and measure the power consumption during the encryption
- Used attack: DPA (*Differential Power Analysis*)

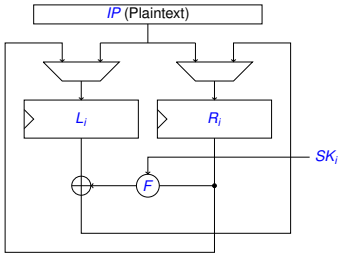
Example: DPA vs DES

DES: algorithmic view



Exemple: DPA vs DES

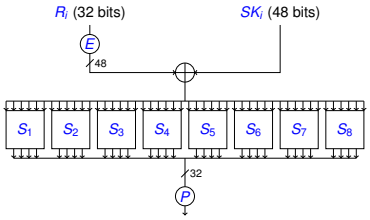
DES: iterative hardware implementation



IP Initial permutation
F Feistel function
SK_i Sub-key (round key)

Exemple: DPA vs DES

DES: Feistel function



E Extension (32 to 48 bits)
P Permutation (bit shuffling)
S_i Substitution

Example: DPA vs DES

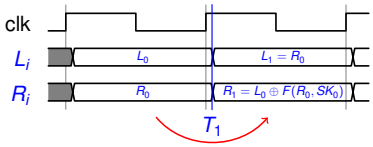
Power consumption model

- How to construct $\mathcal{M}$?
- Power consumption during encryption operation
- Problems
  - DES is not alone on the chip (I/O. . .)
  - Power consumption of DES heavily depends on the key (56 bits), but we cannot test all 2^{56} hypotheses (that's just brute force. . .)
- We need to concentrate on the power consumption of a part of the circuit, depending on a part of the key
- We consider the power consumption of the remaining circuit elements as noise

Example: DPA vs DES

State register on DES data path

- Value change of the state registers (L_i et R_i) during an encryption operation (first round)



Example: DPA vs DES

Hamming distance

Definition

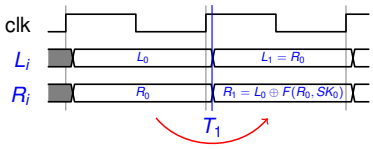
Given two bit vectors of equal length $A = \langle a_0\, a_1 \ldots a_{n-1} \rangle$ and $B = \langle b_0\, b_1 \ldots b_{n-1} \rangle$, their Hamming Distance is defined as the number of positions where they differ:

$$HD(A, B) = \sum_{i=0}^{n-1} a_i \oplus b_i$$

- HD is a good approximate model for the power consumption of a register update in CMOS logic

Example: DPA vs DES

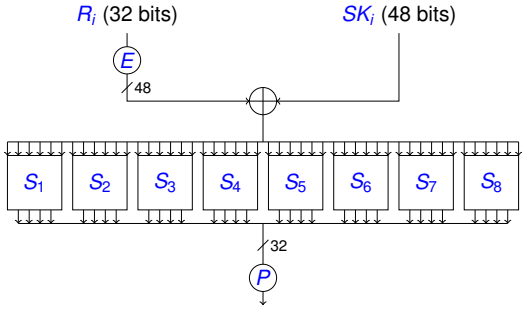
Power consumption of the state registers



- Power consumption of register R_i at time T_1: $P_{R_i}(T_1) = \delta \times \mathrm{HD}(R_0, L_0 \oplus F(R_0, SK_0))$
- Known variables: R_0 et L_0 (depending directly on plain text)
- Unknown variables: SK_0 (48 bits of the key K), T_1, and δ
- Still too many hypotheses: 2^{48}

Example: DPA vs DES

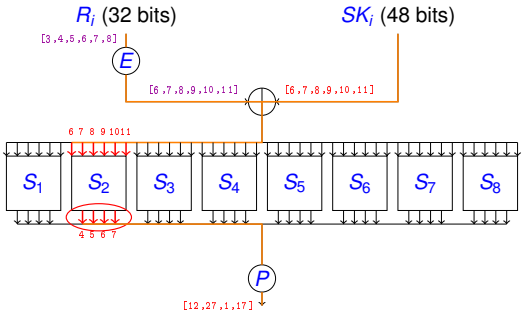
Zoom on the Feistel function



- How to construct a power consumption model depending on fewer bits of the secret key?

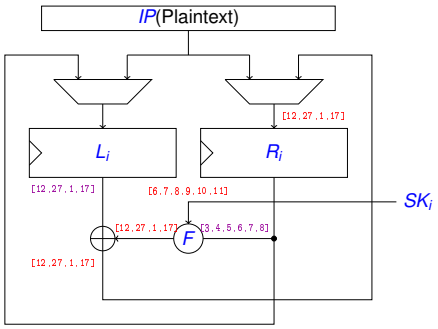
Example: DPA vs DES

Zoom on the Feistel function



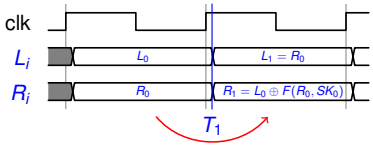
Example: DPA vs DES

Impact of the SBox 2 (first round)



Example: DPA vs DES

Power consumption of state registers (impact SBox 2)



- Considering bits `[12,27,1,17]` of register R_i
- Before T_1, their value depends on R_0 and thus directly on the (known) plain text
- After T_1, their value depends on
  - Bits `[12,27,1,17]` of L_0 (known)
  - Bits `[3,4,5,6,7,8]` of R_0 (known)
  - Bits `[6,7,8,9,10,11]` of SK_0 (unknown)

Example: DPA vs DES

Power consumption model HD on 4 bits

- Power consumption model: $P_{R_i[12,27,1,17]}(T_1) = \delta \times \mathrm{HD}(R_0[12,27,1,17], L_0[12,27,1,17] \oplus F(R_0[3,4,5,6,7,8], SK_0[6,7,8,9,10,11])$
- Depends on a hypothesis on 6 bits of the first round key ($2^6 = 64$ possible hypotheses)
- This model is only valid at instant T_1
- 5 possible output values (Hamming distance on 4 bits): $\{0, \delta, 2\delta, 3\delta, 4\delta\}$
- In the following, we suppose $\delta = 1$
- Finally: $P_4(\mathtt{I}, S) = P_{R_i[12,27,1,17]}(T_1)$, where
  - $\mathtt{I}$ is the plain text
  - S is the hypothesis on $SK_0[6,7,8,9,10,11]$

Example: DPA vs DES

Power consumption model vs actual power consumption

- Our model only predicts the power consumption of a small part of the circuit (4 flip flops) and only at one precise moment (T_1)
- Actual power consumption at T_1:

$$P_{real}(\mathtt{I}, K, T_1) = P_4(\mathtt{I}, S_{good}) + P_{rest}(\mathtt{I}, K, T_1),$$

  where S_{good} corresponds to the good hypothesis (correct value of $SK_0[6,7,8,9,10,11]$ depending on K)
- We suppose that $P_{rest}(\mathtt{I}, K, T_1)$ is statistically independent of $P_4(\mathtt{I}, S_{good})$

Example: DPA vs DES

Measurements

- For the good hypothesis on S (S_{good}), at instant T_1, the actual power consumption depends partially on our model $P_4(\mathtt{I}, S)$
- This dependency is weak, so we need a lot of measurements in order to detect it using the distinguisher
- Perform N measurements (with constant key) for varying plain text messages $\mathtt{I}_1, \ldots, \mathtt{I}_N$

Example: DPA vs DES

Measurements

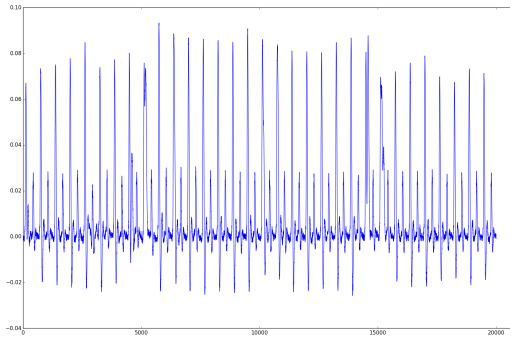
- Power measurement during one encryption operation = power trace
- Trace = vector of samples: $W(\mathtt{I}_i, K, t)$ for $t = 0, \ldots, T-1$ (with T the number of samples per trace)

$$W(\mathtt{I}_i, K, t) = P_{real}(\mathtt{I}_i, K, t) + Noise_{measure}$$

- In the following, we assume that the traces are aligned, i.e. that the index of the sample corresponding to instant T_1 is the same for all traces

Example: DPA vs DES

Example power trace



- Arbitrary units (x: time, y: power consumption)

Example: DPA vs DES

Analysis algorithm

1. Make a hypothesis on $S = S_H$ (64 possible values, including the good one: S_{good})
2. Partition the set of traces depending on the prediction of the power consumption model: for each trace $W(\mathtt{I}_i, K, t)$ ($i = 1, \ldots, N$)
   - Compute the power consumption model: $P_4(\mathtt{I}_i, S_H)$ (5 possible values)
   - Classify the trace in one of 5 sets $E_{P_4=0}, \ldots, E_{P_4=4}$:

$$E_{P_4=j} = \{W(\mathtt{I}_i, K, t) \mid P_4(\mathtt{I}_i, S_H) = j\}$$

Example: DPA vs DES
Analysis algorithm

3. For each of the 5 sets, compute a mean trace (each sample i of the mean trace is the arithmetic mean of the i-th sample of all the traces in this set):

$$\overline{W}_{P_4=j}(t) = \frac{1}{n} \sum_{W \in E_{P_4=j}} W(\mathtt{I}_i, K, t)$$

for $t = 0, \ldots, T-1$ and with $n = |E_{P_4=j}|$ the number of traces in $E_{P_4=j}$

Example: DPA vs DES
Analysis algorithm

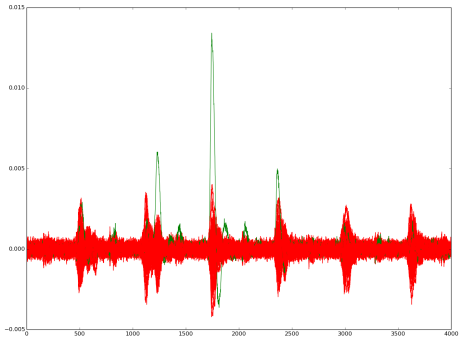
4. Compute a differential trace (for each hypothesis):

$$W_\Delta(t) = -2 \times \overline{W}_{P_4=0}(t) - \overline{W}_{P_4=1}(t) + \overline{W}_{P_4=3}(t) + 2 \times \overline{W}_{P_4=4}(t)$$

for $t = 0, \ldots, T-1$

5. Then find the maximum sample in the differential trace: $\mathcal{D}(S_H) = \max_t W_\Delta(t)$
6. Finally, we need to find out for which hypothesis on S, $\mathcal{D}(S_H)$ is maximal. This should be the good hypothesis: $S_{good} = \arg\max \mathcal{D}$

Example: DPA vs DES
Example of a differential trace



- 64 differential traces superposed for SBox 2

Example: DPA vs DES
Why does it work?

- We have:

$$W(\mathtt{I}_i, K, t) = P_{real}(\mathtt{I}_i, K, t) + Noise_{measure}$$

- At time instant T_1:

$$P_{real}(\mathtt{I}, K, T_1) = P_4(\mathtt{I}, S_{good}) + P_{rest}(\mathtt{I}, K, T_1)$$

- It follows:

$$W(\mathtt{I}_i, K, T_1) = P_4(\mathtt{I}_i, S_{good}) + P_{rest}(\mathtt{I}_i, K, T_1) + Noise_{measure}$$

- We consider the measurement noise and the power consumption of the rest of the circuit globally as noise:

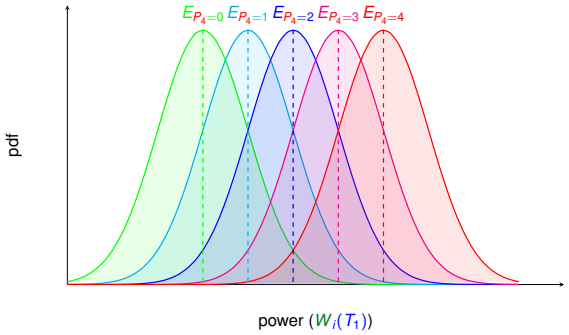
$$W(\mathtt{I}_i, K, T_1) = P_4(\mathtt{I}_i, S_{good}) + Noise$$

Example: DPA vs DES
Why does it work? (good hypothesis)

- Let's suppose we make the correct hypothesis on S (i.e. $S_H = S_{good}$)
- If we apply the power consumption model, it correctly predicts, for each observation, the behavior of 4 bits of the state register
- Therefore, the partitioning of the whole set of traces is consistent with the real behavior of these 4 bits:
  For $j \in \{0, \ldots, 4\}$, $\forall W \in E_{P_4=j}$, we have:

$$W(\mathtt{I}_i, K, T_1) = j + Noise$$

Example: DPA vs DES
Why does it work? (good hypothesis)



Example: DPA vs DES

Why does it work? (good hypothesis)

- When we compute the mean traces, this consistency is preserved:

$$\overline{W}_{P_4=j}(T_1) = j + \overline{Noise}$$

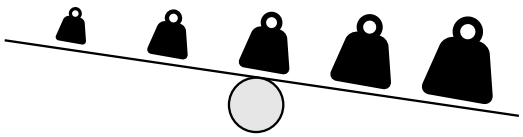
- The equation of the differential trace distinguishes this coherence for the sample correspoding to T_1:

$$\begin{aligned} W_\Delta(T_1) &= -2 \times \overline{W}_{P_4=0}(T_1) - \overline{W}_{P_4=1}(T_1) + \overline{W}_{P_4=3}(T_1) + 2 \times \overline{W}_{P_4=4}(T_1) \\ &= -2 \times (0 + \overline{Noise}) - (1 + \overline{Noise}) + (3 + \overline{Noise}) + 2 \times (4 + \overline{Noise}) \\ &\approx 10 \end{aligned}$$

Example: DPA vs DES

Why does it work? (good hypothesis)

$$\overline{W}_{P_4=0}(T_1)\ \overline{W}_{P_4=1}(T_1)\ \overline{W}_{P_4=2}(T_1)\ \overline{W}_{P_4=3}(T_1)\ \overline{W}_{P_4=4}(T_1)$$



$$W_\Delta(t) = -2 \times \overline{W}_{P_4=0}(t) - \overline{W}_{P_4=1}(t) + \overline{W}_{P_4=3}(t) + 2 \times \overline{W}_{P_4=4}(t)$$

Example: DPA vs DES

Why does it work? (bad hypothesis)

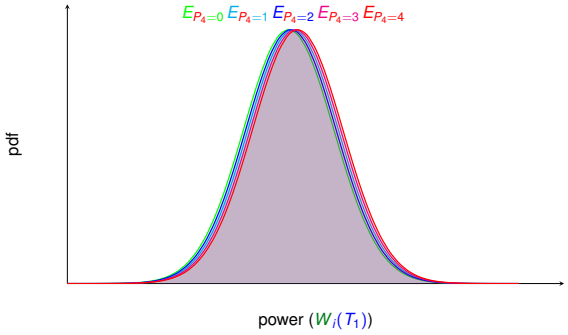
- Now suppose we have made a wrong hypothesis on S ($S_H \neq S_{good}$)
- When applying the power consumption model, it does not predict correctly the power consumption of the state register
- Therefore, the partitioning of the traces is inconsistent with the real behavior of the state register:
  For $j \in \{0, \ldots, 4\}$, $\forall W(\mathtt{I}_i, K, t) \in E_{P_4=j}$, we have:

$$W(\mathtt{I}_i, K, T_1) = k_i + \overline{Noise}$$

for some $k_i \in \{0, \ldots, 4\}$

Example: DPA vs DES

Why does it work? (bad hypothesis)



Example: DPA vs DES

Why does it work? (bad hypothesis)

- As a consequence of the inconsistent (more or less random) partitioning, the mean traces of the different partitions are identical:

$$\overline{W}_{P_4=j}(T_1) = 2 + \overline{Noise}$$

- The equation for the differential trace results in a value around 0:

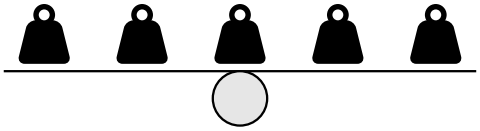
$$\begin{aligned} W_\Delta(T_1) &= -2 \times \overline{W}_{P_4=0}(T_1) - \overline{W}_{P_4=1}(T_1) + \overline{W}_{P_4=3}(T_1) + 2 \times \overline{W}_{P_4=4}(T_1) \\ &= -2 \times (2 + \overline{Noise}) - (2 + \overline{Noise}) + (2 + \overline{Noise}) + 2 \times (2 + \overline{Noise}) \\ &\approx 0 \end{aligned}$$

- This is also the case for all other samples which do not correspond to T_1, for good and bad hypotheses

Example: DPA vs DES

Why does it work? (bad hypothesis)

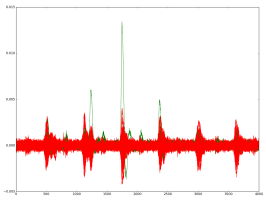
$$\overline{W}_{P_4=0}(T_1)\ \overline{W}_{P_4=1}(T_1)\ \overline{W}_{P_4=2}(T_1)\ \overline{W}_{P_4=3}(T_1)\ \overline{W}_{P_4=4}(T_1)$$



Example: DPA vs DES

Why does it work?

- As a conclusion, all samples of all differential traces are approximately zero except for the one corresponding to time instant T_1 for the good hypothesis on S



DPA in a Nutshell

1: **Inputs**: Model $\mathcal{M}$, traces W_i, inputs $\mathtt{I}_i$ for $1 \leq i \leq N$
2: **for** each hypothesis S_H on secret S **do**
3: **for** $i \in \{1, \ldots, N\}$ **do**
4: $j \leftarrow \mathcal{M}(\mathtt{I}_i, S_H)$
5: $E_{\mathcal{M}=j} \leftarrow E_{\mathcal{M}=j} \cup \{W_i\}$
6: **end for**
7: **for** $j \in \operatorname{range} \mathcal{M}$ **do**
8: compute mean trace $\overline{W}_{\mathcal{M}=j}$
9: **end for**
10: compute differential trace W_Δ
11: $\mathcal{D}(S_H) \leftarrow \max_t W_\Delta(t)$
12: **end for**
13: $S_{good} \leftarrow \arg\max \mathcal{D}$
14: **Return** S_{good}

Example: DPA vs DES

Final observations

- We have recovered 6 bits of SK_0, which gives us directly 6 bits of K
- By repeating the attack on the other S-boxes, we can recover all 48 bits of SK_0, and therefore 48 bits of K
- For the remaining 8 bits, we can attack the second round (the first round is now entirely known), or just do an exhaustive search
- Total complexity of the attack: 64 hypotheses for each of the 8 S-boxes plus exhaustive search: $64 \times 8 + 256$ operations[1]

[1] What is the complexity of one operation?

Plan

Leakage Models

- Hamming weight: $\mathcal{M}(S) = \mathrm{HW}(S)$
  - Suitable for buses which are reset to zero (or high impedance) after transmission
- Hamming distance [2]: $\mathcal{M}(S) = \mathrm{HD}(S, S_{-1}) = \mathrm{HW}(S \oplus S_{-1})$
  - Suitable for hardware implementations (CMOS power consumption)
- Switching distance [7]: $\mathcal{M}(S) = 1$ for transition $0 \to 1$, and $(1 - \delta)$ for transition $1 \to 0$, else 0
  - Suitable for near field EM

Statistical Distinguishers

Classification by [8]

- Partitioning
  - Difference of means [6]: DPA
  - Covariance [1]
  - Mutual information [4]: MIA
- Comparison
  - Correlation [2]: CPA

Correlation Power Analysis (CPA)

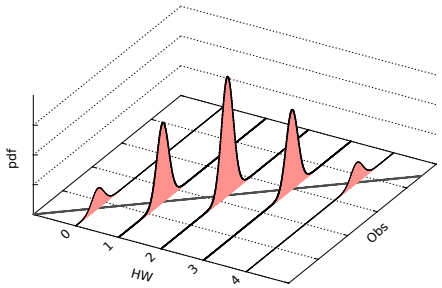
PEARSON correlation coefficient

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y},$$

where $\mathrm{cov}(X,Y) = E[(X - E[X])(Y - E[Y])]$.

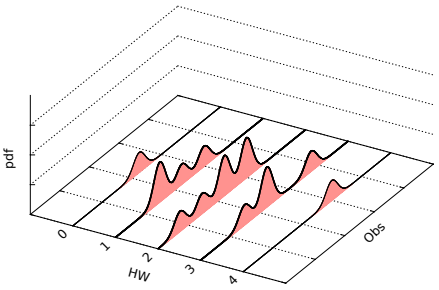
- If there is a linear dependence between the prediction of the leakage model and the real behavior of the circuit, the linear correlation coefficient can be used to test the hypothesis

Correlation Power Analysis (CPA)



Good key hypothesis $\Rightarrow$ correlation $\neq 0$

Correlation Power Analysis (CPA)



Bad key hypothesis $\Rightarrow$ correlation ≈ 0

Timing Attacks

- Attacks based on power consumption or EM radiation require physical access to the target device
- In contrast, timing attacks can be performed remotely, including over a network
- Examples:
  - Remote key recovery over the network [3]
  - Key recovery from another virtual machine running on the same host [5]
- Possible sources of timing variations:
  - Algorithmic
  - Hardware optimizations of the host processor: cache, pipeline, ...

Timing Attacks

Example: Attacking RSA over the network [3]

- RSA in OpenSSL (version 0.9.7)
- Due to some optimizations (Chinese remainder theorem, Montgomery reduction, sliding window expoentiation, Karatsuba multiplication) the execution time slightly depends on the secret key
- The attack has been demonstrated locally and remotely over a network
- Taking the mean of many tries, the latency and jitter introduced by the network are not sufficient to mask the small timing variations
- More attacks in the μ-architecture chapter

Plan

Conclusion

- Physical implementations leak information on various side-channels
  - Power
  - EM radiation
  - Timing
  - ...
- If the leakage depends on sensitive data (such as a cryptographic key), it can be exploited by a side-channel attack
- These attack mostly require physical access to the target system
- Statistical side-channel attacks can be very effective

Bibliography I

[1] Régis Bevan and Erik Knudsen.
Ways to Enhance Differential Power Analysis.
In *ICISC*, volume 2587 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2002.

[2] Éric Brier, Christophe Clavier, and Francis Olivier.
Correlation Power Analysis with a Leakage Model.
In *CHES*, volume 3156 of *LNCS*, pages 16–29. Springer, August 11–13 2004.
Cambridge, MA, USA.

[3] David Brumley and Dan Boneh.
Remote timing attacks are practical.
In *Proceedings of the 12th Conference on USENIX Security Symposium*, pages 1–14, 2003.

[4] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel.
Mutual information analysis.
In *CHES, 10th International Workshop*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442.
Springer, August 10-13 2008.
Washington, D.C., USA.

[5] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar.
S$a: A shared cache attack that works across cores and defies vm sandboxing — and its application to aes.
In *2015 IEEE Symposium on Security and Privacy (SP)*, pages 591–604, May 2015.

[6] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun.
Differential Power Analysis.
In *Proceedings of CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer-Verlag, 1999.
(PDF).

Bibliography II

[7] Éric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater.
Power and electromagnetic analysis: Improved model, consequences and comparisons.
Integration, The VLSI Journal, special issue on "Embedded Cryptographic Hardware", 40:52–60, January 2007.
DOI: 10.1016/j.vlsi.2005.12.013.

[8] François-Xavier Standaert, Benedikt Gierlichs, and Ingrid Verbauwhede.
Partition vs. Comparison Side-Channel Distinguishers: An Empirical Evaluation of Statistical Tests for Univariate Side-Channel Attacks against Two Unprotected CMOS Devices.
In *ICISC*, volume 5461 of *LNCS*, pages 253–267. Springer, December 3-5 2008.
Seoul, Korea.

Physically Unclonable Functions
PUFs
Principle, Advantages, Limitations

Jean-Luc DANGER

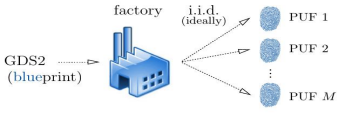
December 2019

Outline

- What and Why a PUF ?
- PUF Architectures
- PUF Reliability
- PUF Security
- Conclusions

Physically Unclonable Function: PUF

- **Function returning the fingerprint of a device**
  - Physical function,
  - which exploits material randomness, during fabrication (mismatch)
  - and is unclonable: same structure for each device

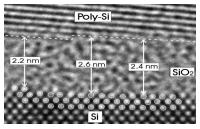


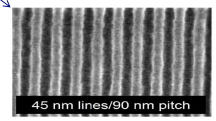
PUFs are instanciations of blueprints by a fab plant

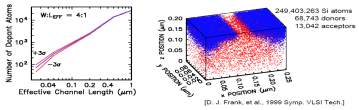
a PUF ID is unique to each device

Mismatch: CMOS process variation

- **Examples**
  - Oxide thickness
  - Metal line edge roughness
  - Random dopant fluctuation







PUF Fingerprint: 2 types

- **List of pairs challenges / responses**

Many challenges =>The PUF is "strong"

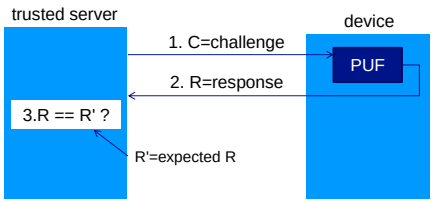
- **Unique identifier**

No challenges =>The PUF is "weak"

can be used as a cryptographic key !

Main function: Authentication

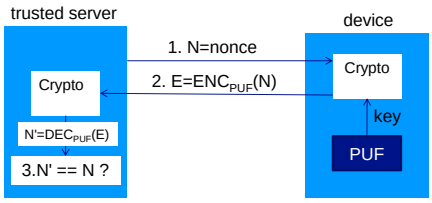
- **Use of Challenge-Response => CRP protocol**



The challenge is never sent twice to avoid replay attacks

Main function: Authentication

- **Use of the ID as a key => cryptographic protocol**



The nonce is never sent twice to avoid replay attacks

Advantages of PUF vs Non Volatile Memory "NVM"

- **PUF is self contained**
  - NVM has to be programmed with an ID, and can be tampered
- **Not clonable**
  - PUF has the same structure, NVM can be reverse engineered
- **Feasible in standard CMOS process**
  - NVM requires a specific process

Many advantages compared to an identifier stored in a NVM memory !

Important Properties to meet

related to **entropy**

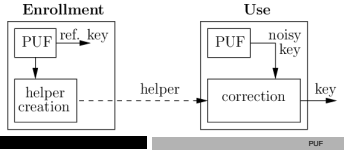
- **Reliability**
  - The PUF responses are unreliable : 1 to 15% of Bit Error Rate
- **Randomness**
  - The PUF responses can be biased: Pr(1) ≠ Pr(0)
- **Uniqueness**
  - Two devices should not have the same ID.
- **Security against attacks**
  - 2 main types: Modeling and Physical attacks
- **Latency**
- **Complexity**

PUF Application examples

- **IP block protection**
  - The IP can run only on the authorized device
- **Secure Boot**
  - The OS is loaded and deciphered only on the authorized device
- **Safe guard**
  - The data are ciphered before being stored in an untrusted device
- **RFID / NFC tag**
  - A product can be authenticated and traced (anti-counterfeiting)

PUF: Two phases of use

- **1. Enrollment**
  - To do only once after manufacturing
  - To get a "reference PUF" and a "helper data" to get it reliable
- **2. Usage or Reconstruction**
  - To obtain the PUF ID
  - The "helper data" is used to correct errors



Outline

- **What and Why a PUF ?**
- **PUF Architectures**
- **PUF Reliability**
- **PUF Security**
- **Conclusions**

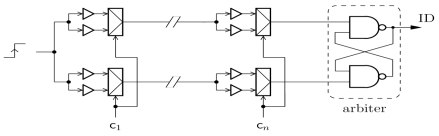
Main Classes of PUF in silicon

Two main types

- **Delay-PUF**
  - Exploits the delay difference between 2 identical delay lines.
- **Memory-PUF**
  - Exploits the difference between two inverters in an SRAM cell
- **Many other types in the literature:**
  - GLITCH PUF
  - MECCA PUF
  - VIA PUF
  - RRAM PUF
  - TERO-PUF
  - ...

Delay-PUF: Arbiter-PUF

- **Delay difference between two identical pathes:**

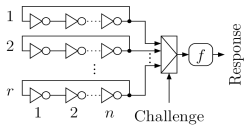


- "Strong" PUF: many challenges for CRP protocol
- Sensitive to Mathematical attacks: Modeling Attacks

Delay-PUF: RO-PUF

- **Frequency difference between two identical Ring Oscillators:**

Ring-oscillator PUF (RO-PUF):
(r rings of n inverters)



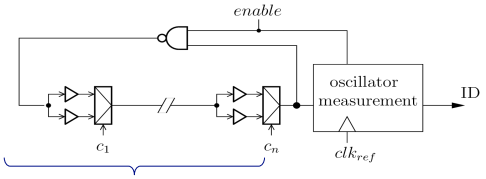
Rationale:

Challenge selects a pair i, j, $1 \leq i \neq j \leq r$.

Response is 1 if RO_i rotates $f$aster than RO_j, and 0 otherwise.

Delay-PUF: Loop-PUF

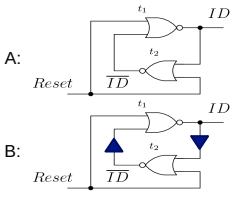
- **Frequency difference between a controlled Ring Oscillator driven by two complementary challenges**



challenge C then challenge $\overline{C}$

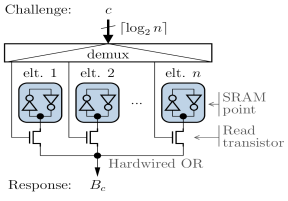
Memory-PUF: Latch-PUF

- **Imbalance between two elements of a latch**



Memory-PUF: SRAM-PUF

- **Imbalance between two inverters of an SRAM cell**



Outline

- What and Why a PUF ?
- PUF Architectures
- PUF Reliability
- PUF Security
- Conclusions

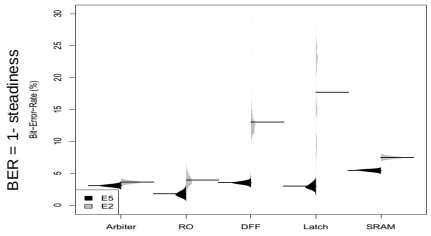
Main Properties to meet

- **Reliability**
  - The PUF response is sensitive to:
    - Noise
    - Environmental change T°C, Vdd
    - Aging

Relatively less problematical

- **Entropy**
  - Inter device: Uniqueness: Each device must have a unique fingerprint
  - Intra device: Randomness: as many bits at 0 and 1
- **Security**
  - Robustness against physical attacks: SCA, FIA
  - Robustness against modeling attacks

reliability estimate

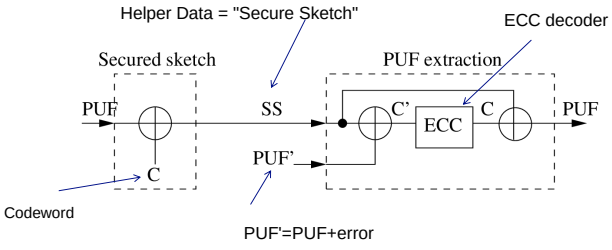


Big Lack of Reliability !

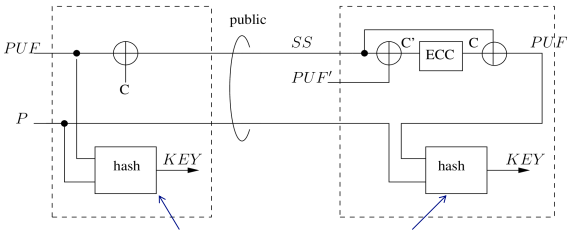
Black: 25°C
Grey: -40C

Results from the european "UNIQUE" project

Secure sketch to correct PUF

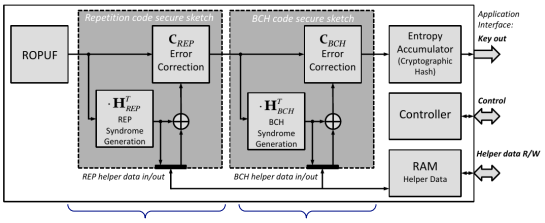


Fuzzy extraction for Key generation



The Key changes with Hash(P||PUF)

Example: PUFKY



Concatenated code = Repetition code + BCH code

BER: 10^{-1} => 10^{-4} => 10^{-9}

Reliability enhancement by filtering

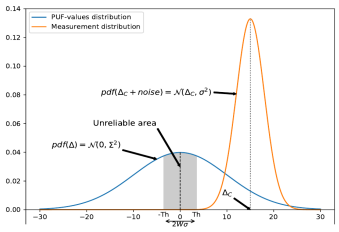


Figure 7: Unreliable area vs Distributions of Δ and noise

Applies for delay PUF having the precise delay information

Bit **unreliable** ⇔ |delay| < Th

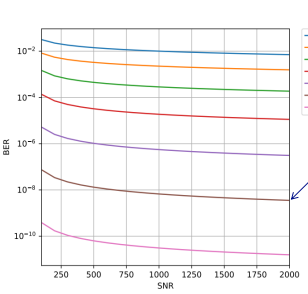
Th= Ws

The bits in the unreliable area are **discarded**

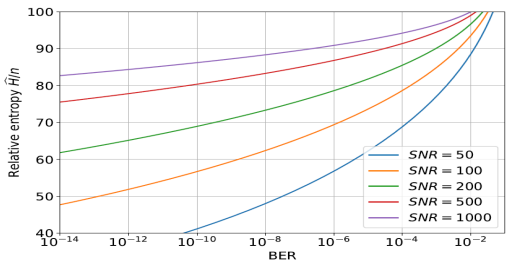
The helper data indicates the unreliable bits, but gives **no information** on the bit value

Delay PUF: filtering out unreliable challenges

RO- PUF and Loop-PUF gives the reliability level => Unreliable challenges can be filtered out



Entropy loss after bit filtering



Need for standard tests and/or stochastic model

Active discussion at ISO sub-committee 27: (**ISO 20897**)

ISO/IEC JTC 1/SC 27/WG 3 N1233

REPLACES:

ISO/IEC JTC 1/SC 27/WG 3

Information technology - Security techniques - Security evaluation, testing and specification

Convenorship: AENOR, Spain, Vice-convenorship: JISC, Japan

DOC TYPE: working draft

TITLE: **Text for ISO/IEC 1st WD 20897 — Information technology — Security requirements and test methods for physically unclonable functions for generating non-stored security parameters**

Outline

- **What and Why a PUF ?**
- **PUF Architectures**
- **PUF Reliability**
- **PUF Security**
- **Conclusions**

PUF Attacks

- **Reverse Engineering Attack**
  - Virtually impossible: same bueprint
- **Brute force**
  - Virtually impossible to store all challenge/responses (CRP)
- **Replay**
  - Sniffing CRPs and play them back
  - Can be countered at protocol level
- **Mathematical**
  - Reconstruct the PUF model: Modeling Attack (CRP only)
- **Physical attack**
  - Side-channel
  - Faults

Main threats

Modeling Attacks

- **Based on Machine Learning algorithms**
  - Take advantage of equations defining the Response from the Challenge
  - Very powerful to attack delay-PUFs
  - Applies only to CRP protocol
- **Countermeasures**
  - Combination of delay-PUFs
  - Do not used PUF in CRP protocol but for key generation

Side-Channel Attack

- **Observation of raw oscillating frequency**
  - Applies to RO-PUF and Loop PUF
  - Countermeasures:
    - RO-PUF: interleave the placement of the RO banks
    - RO and Loop PUF: Use random sequential measurement
- **Attack on the Fuzzy extractor**
  - Simple Power Analysis has been carried out on a FE
  - Template attacks have been implemented on ECC
  - Countermeasures: masking, as cryptographic blocks

Enhanced SCA

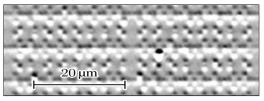
- **Combination with Machine Learning algorithms**
  - Use of noise distribution of the arbiter PUF
  - Use unsupervised ML- techniques [2]
    - SCA is performed first
    - The ML technique proposes a model for classification (like for instance the "k-means" algorithm).

Fault Injection Attack

- **Applies on Delay PUF**
  - Pulse attack (laser, EMI,…)
    - The PUF output is forced
  - Harmonics attack
    - RO PUF: The PUF frequency can be locked on external EM carrier injection
- **Countermeasures**
  - Detection
    - Use embedded sensors to detect disturbances
    - Measure online the entropy of the PUF response

PUF invasive attack

- **Applies on SRAM PUF**
  - Laser stimulation techniques exploiting the Seebeck effect
    - the off-transistor becomes to conduct under laser shot
    - Provides a current increase
  - Attack performed on AVR microcontrollers



SRAM content read out

Outline

- **What and Why a PUF ?**
- **PUF Architectures**
- **PUF Reliability**
- **PUF Security**
- **Conclusions**

Conclusions

- **A specific fingerprint for each IC**
- **Used for authentication and key generation**
- **Use two phases: enrollment (with helper data) + reconstruction**
- **Main advantages**
  - Self-generated by the device
  - No reverse engineering and limited tampering
- **Main limitations**
  - Lack of reliability
    - Necessary post-processing
  - Can be attacked physically and mathematically
    - Protections required
- **ISO Standard for PUF validation**

A few PUF providers and users

INTRINSIC ID

SECURE-IC
THE SECURITY SCIENCE COMPANY

Microsemi

NXP

SYNOPSYS®

XILINX

ALTERA

Reverse-Engineering of Hardware Circuits

Jean-Luc Danger, Sylvain Guilley

Institut Mines TELECOM / TELECOM-ParisTech

SR2I301

Presentation Outline

Presentation Outline

Definition of reverse-engineering

Definition

- When the algorithm to attack is unknown, the system is a "black-box".
- This means that the only way to attack is to analyze the input/output relationships ⇒ hard in general (but for cube attacks [DS09, DS10])
- Therefore, the first step is to recover the algorithm.

Hardware versus Software reverse-engineering

- **Software**: The compiled code can be read-out from memories, and disassembled. Typically A5/1, A5/2, Hitag2 and Keeloq have been retrieved like that.
- **Hardware**: The functionality is buried into an ASIC.

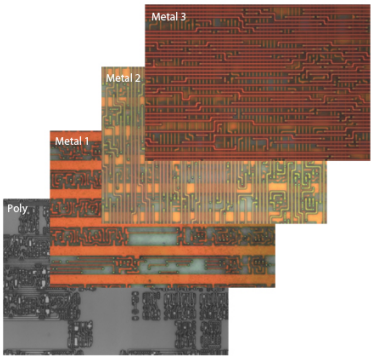
Goal of reverse-engineering

Goals

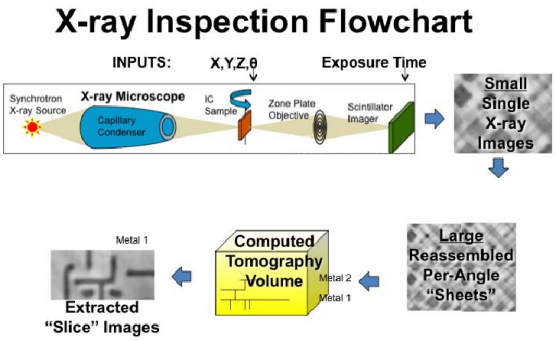
1. Retrieving an algorithm and then cryptanalyse it:
   - This was the case of CRYPTO1 (MyFare) or DSC (DECT).
   - Indeed, the confidential algorithm is most often weak.
2. Breaking a protection:
   - Understand how memories are encrypted by a secure microcontroller [Mah97].
   - Afterwards, all the code is exposed.
   - It thus becomes easy to identify bugs, that can be exploited at the software-level.
   - Thanks to buffer overflows, take the control of the application.
3. Intellectual property matters:
   - Accessing the design of a competitor, so as to steal it.
   - Checking that the competitor does not infringe my patents.

Presentation Outline

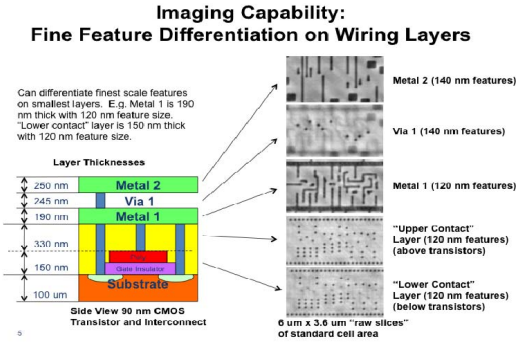
Principle of Circuit Reverse-Engineering: Delayering



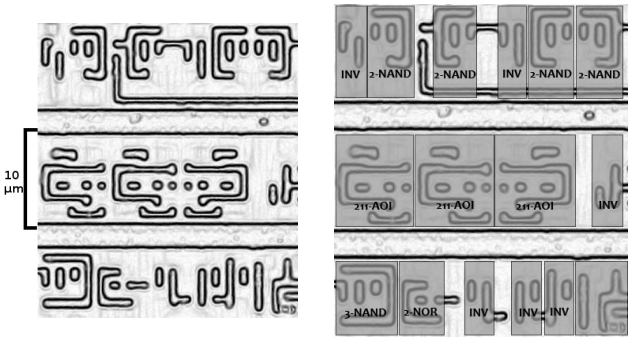
Principle of Circuit Reverse-Engineering: Tomography [BBT+11] (see also [Zei13])



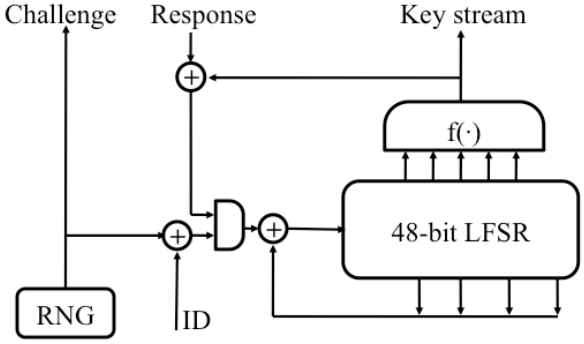
Principle of Circuit Reverse-Engineering: Tomography [BBT+11] (see also [Zei13])



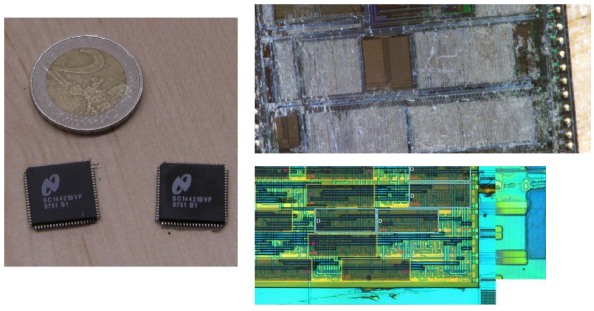
Example of MyFare (NXP) [NSP08]



Example of MyFare (NXP) – CRYPTO1 exposed [NSP08]

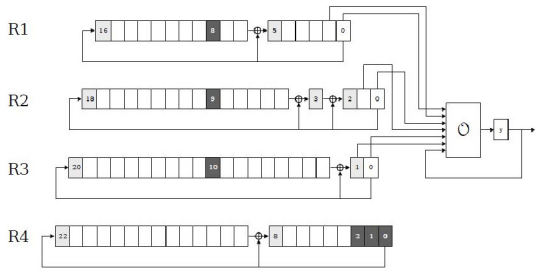


Example of DECT (Alcatel) [NTW10]



See also: https://deDECTed.org/.

Example of DECT (Alcatel) – DSC exposed [NTW10]

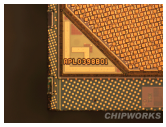


DSC = DECT Standard Cipher.

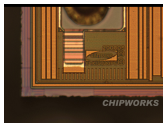
Professional techniques

Example of http://www.chipworks.com/ [TJ09] iPAD tear-down (see also http://www.ltecusa.com/ and many others).

Apple / Microprocessor (with DRAM)

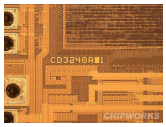


Samsung / 1 Gb mobile DDR SDRAM (x2)

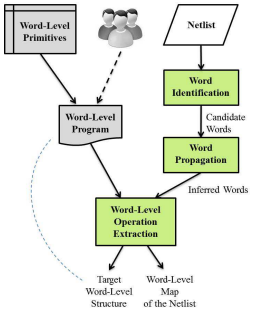


Broadcom / Microcontroller with NVM (used for touch-screen)

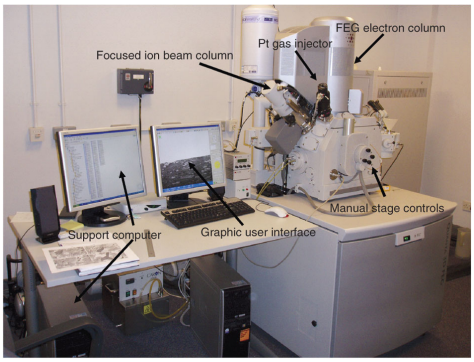
TI / Touchscreen line driver



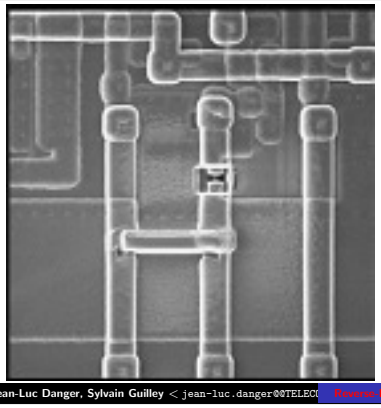
WordRev: Finding word-level structures in a sea of bit-level gates [LGS+13]



Focused Ion Beam (FIB)

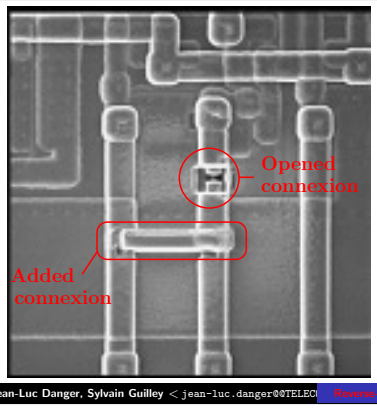


Edition of a circuit thanks to a FIB (Focused Ion Beam)



Can be used for instance to unlock memories [Gir07].

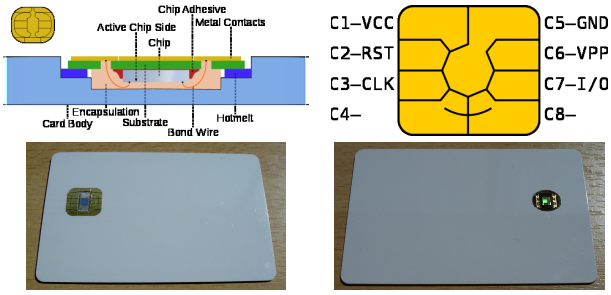
Edition of a circuit thanks to a FIB (Focused Ion Beam)



Can be used for instance to unlock memories [Gir07].

Presentation Outline

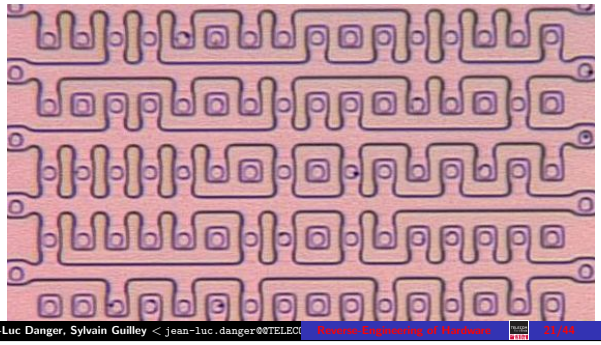
Preparation of a smartcard front- and rear-side



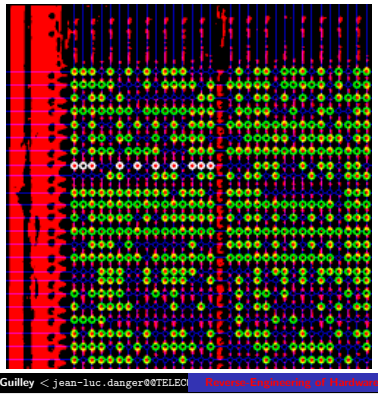
This chemical preparation enables probing and fault injection attacks.

Reading ROMs [KK99]

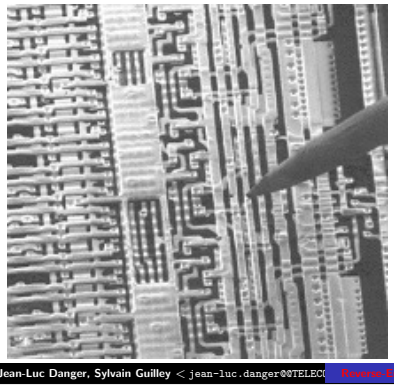
The image shows 16×10 bits in an ST16xyz. Every bit is represented by either a present or missing diffusion layer connection.



Semi-automatic extraction of ROM data (`rompar`)



Needles can be used to actively / passively probe signals



- Can be used for instance to read keys as they circulate into the circuit.
- This technique gives the attacker a considerable power.
- However, probing stations allow to simultaneously read only a couple of bits [HPS99]... and only those that are on top of the circuit.

Faults Injection Reverse-Engineering

FIRE on DES / AES

- Already illustrated by Biham & Shamir (reconstructing unknown ciphers [BS97]) on Feistel schemes.
- Works on DES, because the attacker sees *inputs and outputs* differentials (improvement in [LBGRT13]).
- With secret encodings, it still works on DES, because the sbox is non-injective [Cla07].
- FIRE on SPNs (such as AES) is an emerging topic [PMG11].

Presentation Outline

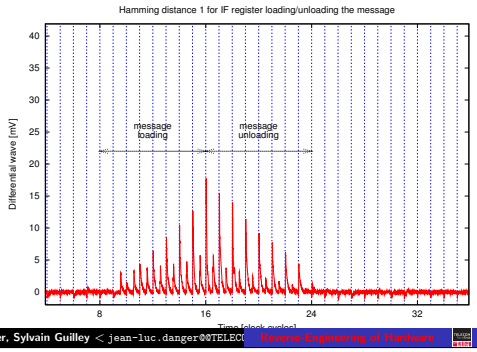
Principle of temporal localization

Use **specific inputs** and **monitor the differences** via one side-channel. For instance, imagine that the key is known.

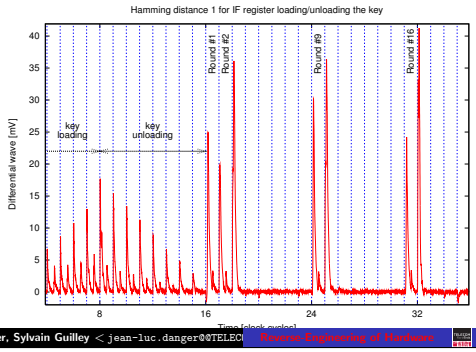
In the next three slides, we suggest to locate:

1. the plaintext loading,
2. the key loading,
3. the key schedule (diversification of the root key for each round).

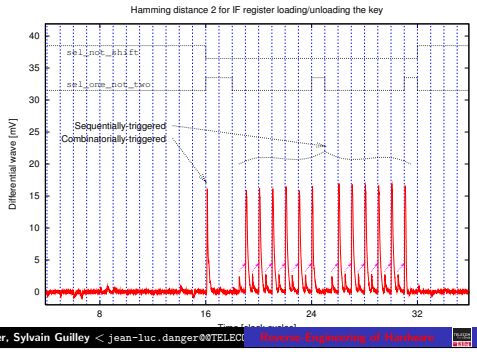
Signature of message loading / unloading in IF (**Reminder**: IF is the InterFace register)



Signature of key loading / unloading in IF, plus the CD activity

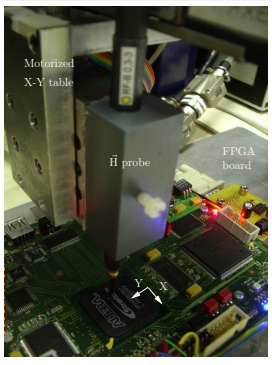


Signature of distance-2 key loading / unloading in IF and of LS^2 in CD



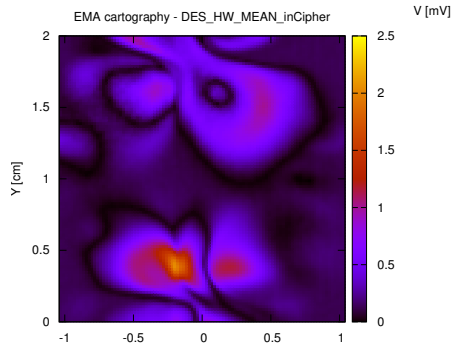
Principle of spacial localization

Using an (X, Y) cartography tool



- Extension: (X, Y, Z, θ) table.
- Helps identify zones that leak a lot; they do not necessarily correspond to the location of the modules! *Id est* they can maybe guide an attack by EMI but not by laser.

Cartography results



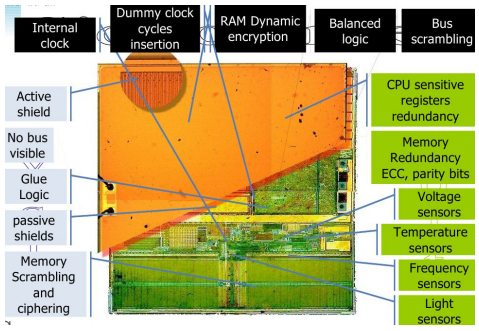
Presentation Outline

Example of the white-box cryptography

White-box cryptography [BCD09, CFD+10]

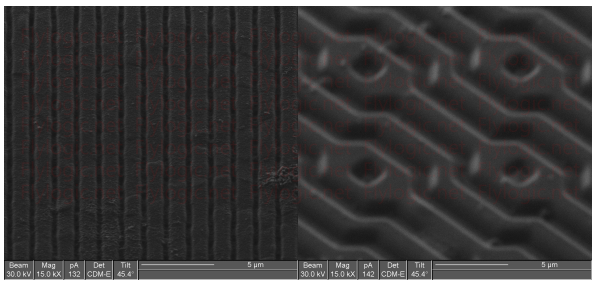
- **Principle**: provide everything (*i.e.* the compiled or the source code of the software) of an algorithm embedding a secret key.
- **Goal**: deny to the attacker the power of retrieving the couple (algorithm + key).

A summary of the countermeasures embedded in a smartcard

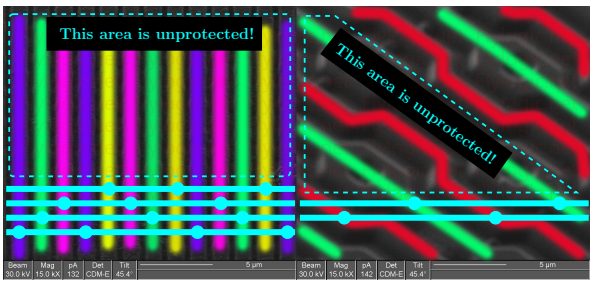


Courtesy of Assia Tria, CEA/LETI.

Active shield 1/2

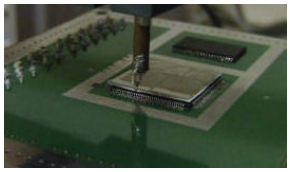


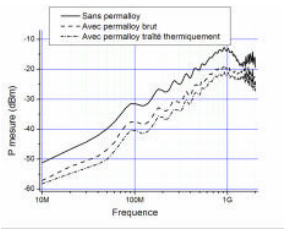
Active shield 2/2



Radiation suppression with a ferromagnetic film

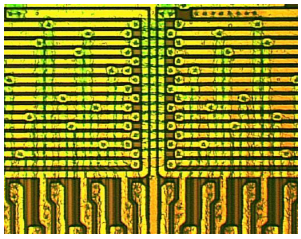
Ferromagnetic material $Ni_{80}Fe_{20}$, *aka* Permalloy.
Experiment with a layer of 20 μm depth [BBD+07].





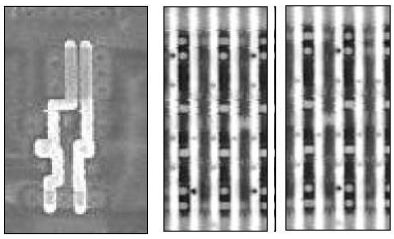
Bus scrambling

- Scrambling can be static or dynamic.
- Static scrambling can also be a feature
  - The mixing comes from a problem of routability of the data/address bus to the memory.
  - Possible because permutations affect equally the `write` and `read` operations.



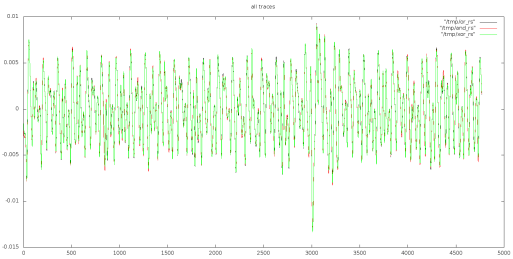
Beware of attacks [FLM10] on scrambled EEPROMs!

Hardware Camo



Hardware-level camouflage of gates. Left: an unprotected gate, whose function is easy to identify. Center, right: *almost* indistinguishable AND/OR camouflaged gates. *[courtesy of SMI / SypherMedia Library]*

Software Camo — ARM7 Native Machine Code



Traces of the electromagnetic leakage of the AND, XOR and OR

[BBD+07] L. Bouhouch, A. Boyer, S. Ben Dhia, É. Sicard, and M. Fadel.
Amélioration des performances CEM d'un microcontrôleur à l'aide d'un film ferromagnétique.
In *TELECOM 2007, 5th JFMMA*, March 2007.
Fes, Morocco. (Online PDF).

[BBT+11] M. Bajura, G. Boverman, J. Tan, G. Wagenbreth, C. M. Rogers, M. Feser, J. Rudati, A. Tkachuk, S. Aylward, and P. Reynolds.
Imaging Integrated Circuits with X-ray Microscopy.
In *Proceedings of the 36th GOMACTech Conference*, March 2011.
Orlando, FL, USA.
http://www-ssrl.slac.stanford.edu/research/highlights_archive/circuitintegrity.pdf.

[BCD09] Julien Bringer, Hervé Chabanne, and Jean-Luc Danger.
Protecting the NOEKEON Cipher against SCARE Attacks in FPGAs by Using Dynamic Implementations.
In *ReConFig*, pages 183–188. IEEE Computer Society, December 9–11 2009.
Cancún, Quintana Roo, México. DOI: 10.1109/ReConFig.2009.19,
http://eprint.iacr.org/2009/239.pdf.

[BS97] Eli Biham and Adi Shamir.
Differential Fault Analysis of Secret Key Cryptosystems.
In *CRYPTO*, volume 1294 of *LNCS*, pages 513–525. Springer, August 1997.
Santa Barbara, California, USA. DOI: 10.1007/BFb0052259.

[CFD+10] Zouha Cherif, Florent Flament, Jean-Luc Danger, Shivam Bhasin, Sylvain Guilley, and Hervé Chabanne.
Evaluation of White-Box and Grey-Box Noekeon Implementations in FPGA.
In Viktor K. Prasanna, Jürgen Becker, and René Cumplido, editors, *ReConFig*, pages 310–315. IEEE Computer Society, 2010.

[Cla07] Christophe Clavier.
Secret External Encodings Do Not Prevent Transient Fault Analysis.
In *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 181–194. Springer, 2007.
Vienna, Austria.

[DS09] Itai Dinur and Adi Shamir.
Side Channel Cube Attacks on Block Ciphers.
Cryptology ePrint Archive, Report 2009/127, March 2009.
http://eprint.iacr.org/2009/127/.

[DS10] Itai Dinur and Adi Shamir.
Generic Analysis of Small Cryptographic Leaks.
In *FDTC*, pages 39–48. IEEE Computer Society, August 21 2010.
Santa Barbara, CA, USA. DOI: 10.1109/FDTC.2010.11.

[FLM10] Jacques J. A. Fournier and Philippe Loubet-Moundi.
Memory Address Scrambling Revealed Using Fault Attacks.
In *FDTC*, pages 30–36. IEEE Computer Society, August 21 2010.
Santa Barbara, CA, USA. DOI: 10.1109/FDTC.2010.13.

[Gir07] Christophe Giraud.
Attaques de cryptosystèmes embarqués et contre-mesures associées.
PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, 26 octobre 2007.
http://www.prism.uvsq.fr/fileadmin/CRYPTO/TheseCG-new.pdf.

[GMN+13] Sylvain Guilley, Damien Marion, Zakaria Najm, Youssef Souissi, and Antoine Wurcker.
Software Camouflage.
In *FPS*, volume 8352 of *LNCS*. Springer, October, 21–22 2013.
La Rochelle, France.

[HPS99] Helena Handschuh, Pascal Paillier, and Jacques Stern.
Probing Attacks on Tamper-Resistant Devices.
In *CHES*, volume 1717 of *LNCS*, pages 303–315. Springer, August 12-13 1999.
Worcester, MA, USA.

[KK99] Oliver Kömmerling and Markus G. Kuhn.
Design Principles for Tamper-Resistant Smartcard Processors.
In *WOST '99 (USENIX Workshop on Smartcard Technology)*, pages 9–20, Berkeley, CA, USA, May 10-11 1999. USENIX Association.
Chicago, Illinois, USA (On-line paper). ISBN: 1-880446-34-0.

[LBGRT13] Hélène Le Bouder, Sylvain Guilley, Bruno Robisson, and Assia Tria.
Fault Injection to Reverse Engineer DES-like Cryptosystems.
In *FPS*, volume 8352 of *LNCS*. Springer, October, 21–22 2013.
La Rochelle, France.

[LGS+13] Wenchao Li, Adria Gascón, Pramod Subramanyan, Wei Yang Tan, Ashish Tiwari, Sharad Malik, Natarajan Shankar, and Sanjit A. Seshia.
WordRev: Finding word-level structures in a sea of bit-level gates.
In *HOST*, pages 67–74. IEEE, 2013.

[Mah97] David Paul Maher.
Fault Induction Attacks, Tamper Resistance, and Hostile Reverse Engineering in Perspective.
In *Financial Cryptography*, volume 1318 of *Lecture Notes in Computer Science*, pages 109–122. Springer, February 24-28 1997.

[NSP08] Karsten Nohl, David Evans Starbug, and Henryk Plötz.
Reverse-Engineering a Cryptographic RFID Tag.
In *USENIX Security Symposium*, pages 185–193, July 31 2008.
San Jose, CA, USA (Online HTML).

[NTW10] Karsten Nohl, Erik Tews, and Ralf-Philipp Weinmann.
Cryptanalysis of the DECT Standard Cipher.
In *FSE*, volume 6147 of *Lecture Notes in Computer Science*, pages 1–18. Springer, February 7-10 2010.
Seoul, South Korea.

[PMG11] Manuel San Pedro, Soos Mate, and Sylvain Guilley.
FIRE: Fault Injection for Reverse Engineering.
In LNCS, editor, *WISTP: Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing*, volume 6633 of *LNCS*, pages 280–293. Springer, June 1-3 2011.
Heraklion, Greece. DOI: 10.1007/978-3-642-21040-2_20.

[TJ09] Randy Torrance and Dick James.
The State-of-the-Art in IC Reverse Engineering.
In *CHES*, volume 5747 of *LNCS*, pages 363–381. Springer, September 6-9 2009.
Lausanne, Switzerland.

[Zei13] Carl Zeiss.
Package Optimization and Failure Analysis with 3D X-ray Microscopy, November 2013.
https://zeiss-microscopy.uberflip.com/i/547299-package-optimization-and-failure-analysis-with-3d-x-ray-microscopy.

Processeurs et sécurité

Guillaume Duc
guillaume.duc@telecom-paris.fr
2020–2021

Plan

Objectifs du cours

- Connaître les mécanismes de sécurité offerts par les processeurs
  - Mécanismes classiques (anneaux de protection, mémoire virtuelle)
  - Mécanismes récents (authentification de pointeurs, Intel SGX, etc.)
- Comprendre comment ces mécanismes sont utilisés pour garantir la sécurité de la partie logicielle
- Explorer quelques problèmes de sécurité posés par les processeurs récents

Contexte

- Un ordinateur « classique »
- Certains des mécanismes présentés peuvent être disponibles dans des architectures embarquées

Objectifs de sécurité et modèles d'attaque

- Très variés en fonction des mécanismes
- Objectifs
  - Confidentialité et/ou intégrité du code et/ou des données
  - Intégrité du flot d'exécution
- Attaquant
  - Accès logiciel majoritairement
    - Exécution d'un processus malicieux
    - Exploitation de vulnérabilités dans une application (débordement de tampon, etc.)
    - Contrôle du système d'exploitation
  - Accès matériel dans certains cas
    - Lecture et/ou modification du contenu de la mémoire

Plan

Plan

Anneaux de protection

Protection rings

- Concept introduit dans le matériel développé pour le système *Multics* (8 anneaux utilisés)
- Mécanisme matériel offrant une certaine protection des données contre des fautes ou des comportements malicieux du logiciel
- Différentes parties du logiciel (système d'exploitation, pilotes de périphériques, bibliothèques, applications, etc.) vont s'exécuter dans des anneaux différents avec des privilèges différents

Anneaux de protection

Protection rings

- Hiérarchie entre les anneaux
  - L'anneau le plus privilégié (généralement numéroté 0) a accès à toutes les fonctionnalités et les données du système
  - Chaque anneau à accès aux fonctionnalités et aux données des anneaux moins privilégiés
  - Par contre, dans un anneau donné, l'accès aux fonctionnalités et aux données de niveaux plus privilégiés doit passer par des interfaces bien spécifiées (appels systèmes...)

Anneaux de protection

Exemple architecture `x86`

- Présents dans l'architecture `x86` depuis le processeur 80386 (1985)
- 4 niveaux de protections offerts lorsque le processeur est en mode *protégé* (par défaut le processeur démarre en mode *réel* par compatibilité)
  - Du *ring 0* le plus privilégié
  - au *ring 3* le moins privilégié
- Le niveau courant est stocké dans le registre interne `CPL` (*Current Privilege Level*)
  - Généralement, le `CPL` est égal au `DPL` (*Descriptor Privilege Level*) indiqué dans le descripteur du segment de code courant (registre `CS`)

Anneaux de protection

Exemple architecture `x86`

- Le `CPL` est utilisé pour effectuer des contrôles
  - Le chargement d'un descripteur de segment de données (`DS`, `ES`, `FS`, `GS`, `SS`) n'est possible que si le niveau de privilège indiqué dans le descripteur (`DPL`) est numériquement supérieur ou égal au `CPL` (moins privilégié)
  - Lors d'un saut vers un autre segment de code (*far* `JMP` ou `CALL`)
    - le `DPL` du segment cible doit être égal au `CPL`,
    - ou, si le bit *conforming* est positionné dans le descripteur de segment cible, le `DPL` du segment cible peut être inférieur ou égal au `CPL` (dans ce cas, ce dernier ne change pas lors du saut). Ce cas est utilisé pour des segments contenant du code devant être partagé entre plusieurs niveaux (bibliothèques)

Anneaux de protection

Exemple architecture `x86`

- Le `CPL` est utilisé pour effectuer des contrôles (suite)
  - Lors de l'exécution de certaines instructions affectant les mécanismes de protection (instructions dites *privilégiées*) pour lesquelles le `CPL` doit être à 0
    - Exemples : Chargement du pointeur vers les tables de descripteurs de segments (`LDT` et `GDT`), table des vecteurs d'interruption (`IDT`), modification des registres de contrôle, arrêt du processeur...
  - Lors de l'exécution d'instructions concernant les I/O (instructions dites *sensibles*) pour lesquelles le `CPL` doit être inférieur ou égal au *I/O Privilege Level* (`IOPL`) défini dans le registre de drapeaux (*flags*)

Anneaux de protection

Exemple architecture `x86`

- Changement de niveau `CPL`
  - Principalement lors d'une interruption ou d'une exception (en provenance du matériel, problème d'exécution, debug, ou interruption logicielle)
  - D'autres mécanismes existent (notamment lors d'un `CALL` sur une *call gate*) mais ne sont plus couramment utilisés

Anneaux de protection

Exemple architecture `x86`

- Ces dernières années, le modèle d'anneaux du `x86` s'est complexifié avec l'introduction
  - De la virtualisation (considéré comme un *ring -1*)
  - Du *System Management Mode* (considéré comme un *ring -2*)

Anneaux de protection

Exemple architecture `ARMv7-A`

- Dans l'architecture `ARMv7-A`, trois niveaux de privilèges existent (deux dans les architectures antérieures sans virtualisation) dans la zone non sécurisée (hors *TrustZone*) liés aux modes de fonctionnement du processeur
  - PL0 : Niveau le moins privilégié, typiquement pour les applications (mode `User`)
  - PL1 : Niveau plus privilégié, typiquement pour le système d'exploitation (modes `FIQ`, `IRQ`, `Supervisor`, `Abort`, `Undefined` et `System`)
  - PL2 : Niveau le plus privilégié (mode `Hyp`)

Anneaux de protection

Exemple architecture `ARMv7-A`

- Passage de PL0 vers PL1
  - Via une interruption logicielle (instructions `SWI` (*Software Interrupt*) ou `SVC` (*Supervisor Call*)) : basculement en mode `Supervisor`
  - Via une interruption matérielle : basculement en mode `FIQ` ou `IRQ`
  - Via une exception : basculement en mode `Abort` ou `Undefined`
- Passage de PL1 vers PL0
  - Explicitement en écrivant dans le registre `CPSR` (*Current Processor Status Register*) pour changer de mode (et donc de niveau de privilège)
  - Implicitement via la restauration du `CPSR` lors d'un retour d'interruption vers le mode `User`

Plan

Introduction

- Mécanisme introduit dans les années 1960 (monde des *mainframes*) mais disponible sur les PC depuis le 80386 d'Intel
- Avantages
  - Isolation entre les espaces mémoires de différentes applications
  - Partage explicite de certaines zones mémoires (bibliothèques de code, données partagées...)
  - Pagination (permet aux applications de disposer de plus de mémoire qu'il en existe physiquement)

Principes

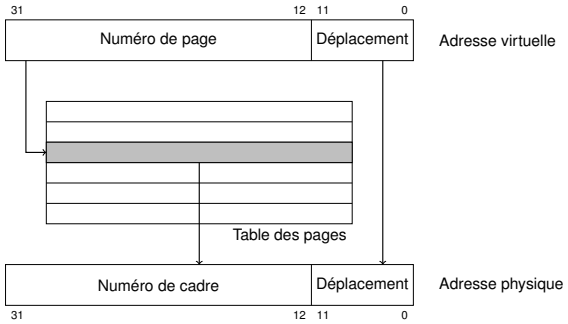
- Les applications ne manipulent pas directement des adresses *physiques* (adresses présentées aux puces de RAM) mais des adresses *virtuelles*
- Un mécanisme matériel, la MMU (*Memory Management Unit*), le plus souvent intégrée aux processeurs, se charge de faire la conversion entre les adresses virtuelles et les adresses physiques
- Cette conversion est effectuée en se basant sur une table de conversion, nommée la table des pages (*page table*), le plus souvent mise en place par le système d'exploitation

Fonctionnement

- Pour des raisons pratiques, la traduction n'est pas réalisée avec la granularité d'une adresse virtuelle (il y a 2^{32} adresses virtuelles dans le cas d'un processeur 32 bits, donc une table des pages nécessiterait de stocker les 2^{32} adresses physiques correspondantes ; chaque adresse étant codée sur 32 bits, la table ferait 16 Gio...)
- L'espace d'adressage virtuel ainsi que l'espace d'adressage physique sont découpés en *pages* (au minimum 4 kio sur `x86`)
- La conversion fait correspondre un numéro de cadre physique à un numéro de page virtuelle, le déplacement au sein de la page physique étant égal au déplacement au sein de la page virtuelle

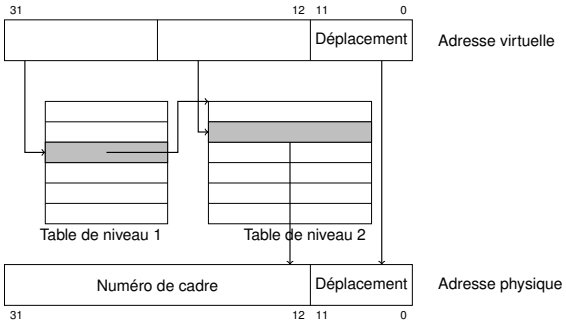
Fonctionnement

Exemple : adresses 32 bits, pages de 4 kio, 1 niveau de traduction



Fonctionnement

Exemple : adresses 32 bits, pages de 4 kio, 2 niveaux de traduction



Fonctionnement

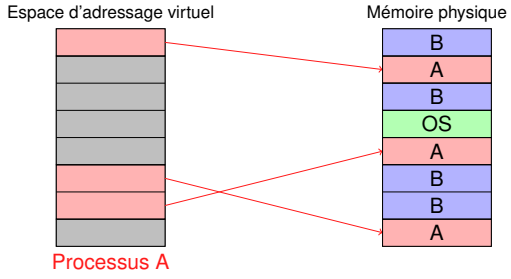
- Utilisation de plusieurs niveaux de tables de pages pour gérer plus efficacement les larges zones vides dans l'espace d'adressage virtuel d'un processus
- La MMU dispose d'un cache spécialisé (TLB, *Translation Lookaside Buffer*) destiné à conserver les traductions (page virtuelle, cadre physique) récemment utilisées
- En cas de défaut de TLB, le parcours de la table de pages (ou des différents niveaux de table) pour rechercher la traduction peut être effectué
  - Soit matériellement : la MMU connaît l'adresse de la table de premier niveau et va la parcourir elle-même (exemple architecture `x86`)
  - Soit logiciellement : la MMU déclenche une exception et c'est au système d'exploitation de charger explicitement une entrée du TLB avec la traduction demandée (exemple architecture MIPS)

Fonctionnement

- Dans une entrée de la table de pages (entrée de page, *page entry*) sont également stockées les informations suivantes
  - Validité (l'entrée est-elle valide ?)
  - Informations d'accès (les données de la page ont-elles été accédées et/ou modifiées ?)
  - Informations de protection
    - Page en lecture seule ou lecture/écriture
    - Accès en mode utilisateur ou en mode superviseur uniquement
    - Page contenant du code exécutable ou non (bit NX)

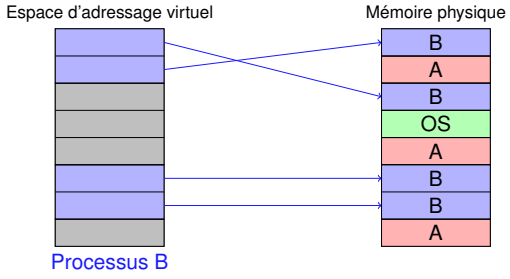
Utilisation

Flat memory model



Utilisation

Flat memory model



Utilisation

- Le système d'exploitation maintient une table de pages par processus
- Lorsque le SE bascule d'un processus à un autre, il configure la MMU pour utiliser la table de pages du nouveau processus
- La modification du registre de la MMU contenant le pointeur vers le premier niveau de la table de pages est une action qui est privilégiée (qui ne peut donc être réalisée que par le SE)
  - Lien avec les anneaux de protection
- Un processus ne peut donc pas modifier directement sa table de pages

Utilisation

- Lorsqu'un processus accède à une adresse virtuelle pour laquelle l'entrée correspondante dans la table des pages n'est pas valide (drapeau valide non positionné) ou si les informations de protection ne sont pas correctes (lecture seule pour un accès en écriture par exemple), la MMU déclenche une exception
- Le système d'exploitation reprend alors la main et peut
  - Tuer le processus en cas d'accès illégal (accès à une page mémoire non allouée, violation des attributs de protection...)
  - Dans certain cas, l'exécution peut reprendre après une action du système d'exploitation (restauration de la page demandée qui avait été évincée de la mémoire faute de place, mécanisme de *copy-on-write*...)

Utilisation

Isolation des processus

- Un processus ne peut donc pas accéder à une adresse physique si une traduction n'est pas explicitement mise en place par le système d'exploitation
- Donc le SE peut s'assurer qu'un processus ne peut pas accéder à des zones mémoires physiques appartenant à d'autres processus ou au SE lui-même

Utilisation

Pagination / *Swap*

- Si la mémoire vient à manquer, le système d'exploitation peut déplacer le contenu de certains cadres de mémoire physique vers un support de stockage de masse (disque dur) et marquer les pages de mémoire virtuelle correspondantes comme invalides
- Si un des processus dont la mémoire a été évincée essaie d'y accéder, la MMU déclenchera une exception
- Le SE reprend la main et voit que l'adresse demandée existe bien mais a été déplacée vers le disque
- Il peut remettre les données dans un cadre de mémoire physique et reconfigurer l'entrée de page correspondante
- Il relance alors l'instruction fautive du processus qui peut maintenant réussir

Exemple architecture x86

Depuis le 80386, schéma général

- *Adresse logique* (selecteur de segment + déplacement) transformée en *adresse linéaire* (mécanisme de segmentation)
  - En pratique, adresse linéaire = adresse logique (*Flat memory model*) dans la plupart des cas actuellement
- Adresse linéaire traduite en adresse physique par la MMU via une traduction à deux niveaux
  - Registre `CR3` (accès privilégié) pointe vers un répertoire de pages (*page directory*) : table de 1024 entrées de 32 bits
  - Bits 31–22 de l'adresse linéaire : index dans le répertoire de pages qui donne l'adresse de la table de pages à utiliser (table de 1024 entrées de page, *page entries*)
  - Bits 21–12 de l'adresse linéaire : index de l'entrée de page dans la table de pages
  - Bits 11–0 : Déplacement dans la page de 4 kio

Exemple architecture x86

Depuis le 80386, structure d'une entrée de page

- Bits 31–12 : bits 31 à 12 de l'adresse du cadre de page
- Bit 6 : page modifiée (*dirty*)
- Bit 5 : page accédée
- Bit 2 : accès réservé au mode superviseur (`CPL` < 3) ou autorisé en mode utilisateur
- Bit 1 : lecture seule ou lecture/écriture
- Bit 0 : validité de l'entrée

Exemple architecture x86

Extensions

- *Page Size Extension* (Pentium)
  - Une entrée dans le répertoire de pages peut directement correspondre à une entrée pour une page de 4 Mio (bit 7 (`PS`) à 1)
  - Avantage : n'occupe qu'une seule entrée du TLB (au lieu de 1024)
- *Physical Address Extension* (Pentium Pro)
  - Volonté d'utiliser des adresses physiques plus longues que 32 bits (pour pouvoir gérer plus de 4 Gio de RAM)
  - Passage d'entrées de pages de 32 à 64 bits pour pouvoir y stocker plus de bits pour l'adresse physique
  - Ajout d'un nouveau niveau de hiérarchie (pour conserver la taille d'une table de pages ou d'un répertoire de pages)

Exemple architecture x86

Extensions

- NX bit (*No eXecute*), Intel=XD bit, AMD=*Enhanced Virus Protection*
  - Depuis le 80286, possibilité d'interdire l'exécution seulement au niveau d'un segment, donc impossible dans le modèle *Flat Segment*
  - Ajout d'un bit au niveau de l'entrée de page pour indiquer que la page ne contient pas de code exécutable
  - Introduit tout d'abord par AMD dans les processeurs AMD64
  - Puis repris par Intel à partir du Pentium 4
  - Utile par exemple pour rendre la pile non exécutable (plus possibilité de faire de l'injection de code en pile)

Anneaux et mémoire virtuelle

Conclusion

- Si utilisés correctement, protègent
  - Confidentialité et intégrité des données des processus ou du système d'exploitation
  - Flot d'exécution des processus ou du système d'exploitation
- Contre
  - Un processus malicieux contrôlé par un attaquant
  - Un processus défaillant
- Fondations de la sécurité logicielle, notamment pour les systèmes multi-tâches et multi-utilisateurs (c'est-à-dire la très vaste majorité des systèmes d'exploitation actuels)

Plan

Unité de protection mémoire

***Memory Protection Unit* (MPU)**

- Version « simplifiée » d'une MMU pour les microcontrôleurs et petits microprocesseurs
- Permet uniquement du contrôle d'accès (lecture, écriture, exécution en mode utilisateur ou superviseur, et éventuellement des informations liées à la gestion du cache) sur des zones mémoires en nombre limité
- Pas de traduction d'adresses (en général)

Unité de protection mémoire

Exemple : Cortex-M

- La MPU peut protéger un nombre fixe de régions (8 en général) de taille variable (32 octets à 4 Gio)
- Quelques contraintes sur les régions (leur taille est une puissance de 2 et elles doivent être alignées sur un multiple de leur taille)
- Pour chaque région
  - Actif/Inactif
  - *Execute Never*
  - Permissions d'accès (aucun, lecture seule, lecture/écriture, pour le mode privilégié et non privilégié)
  - Type de la mémoire (cachable, partageable...)

Plan

Introduction

- La virtualisation permet de faire fonctionner plusieurs machines virtuelles sur une même machine physique
- Chaque machine virtuelle peut accueillir un système d'exploitation (invité / *guest*) et ses applications
- Partage des ressources de la machine physique (hôte / *host*) entre les machines virtuelles assuré par un hyperviseur
  - Mémoire : niveau supplémentaire de traduction d'adresse sous contrôle de l'hyperviseur et anneau de protection dédié
  - Temps processeur : partage des cœurs ou partage du temps à l'aide de timers
  - Périphériques

Introduction

- Les systèmes d'exploitation invités ne doivent plus communiquer directement avec les périphériques et le matériel mais passer par l'intermédiaire de l'hyperviseur
- Toutes les instructions privilégiées (accès au matériel, configuration du processeur, etc.) sont interceptées par l'hyperviseur
- Périphériques émulés par l'hyperviseur (pour garantir l'isolation entre les machines virtuelles)
- Problème potentiel de performance sur l'accès à certains périphériques rapides (GPU, carte réseau, etc.)
  - Accès direct parfois autorisé, mais problème de sécurité, notamment en cas de transferts DMA

Plan

Problématique
Périphériques & DMA

- La plupart des périphériques peuvent accéder directement à la mémoire centrale en utilisant des adresses physiques via le mécanisme de DMA (*Direct Memory Access*)
- Cela permet notamment aux périphériques de transférer des données vers la mémoire, sans avoir à monopoliser le processeur

Problématique
Attaque DMA

- Néanmoins, aucun contrôle n'est effectué et donc un périphérique malicieux (*firmware* corrompu) ou endommagé peut écrire ou lire n'importe où en mémoire, y compris dans les zones réservées au système d'exploitation, etc.
- Problème exacerbé dans le cas de la virtualisation puisqu'il empêche de fait l'utilisation directe d'un périphérique par une machine virtuelle (qui pourrait alors via ce dernier accéder aux données des autres machines virtuelles en RAM)

IO-MMU

- Dispositif, similaire à la MMU, permettant de traduire les adresses issues des périphériques (capables de faire du DMA) en adresses physiques pour la mémoire
- Un périphérique ne peut donc plus qu'accéder à des zones mémoires pour lesquelles le système d'exploitation ou l'hyperviseur à mis en place des traductions d'adresse
- Exemples de mise en œuvre
  - *Graphics Address Remapping Table* (GART) utilisée pour les cartes graphiques AGP et PCIe
  - Intel *Virtualization Technology for Directed I/O* (VT-d)
  - AMD *I/O Virtualization Technology*

IO-MMU
Avantages

- Protection mémoire contre les attaques DMA ou les périphériques défectueux
  - Notamment dans le cadre de la virtualisation pour laisser une machine virtuelle utiliser directement un périphérique
- Possibilité pour les périphériques d'utiliser plus de mémoire physique (PAE)
- Allocations de larges zones mémoires virtuellement contiguës

Plan

Plan

ARM TrustZone

- Ensemble de technologies permettant d'implémenter un environnement d'exécution de confiance (*Trusted Execution Environment*, TEE) sur un SoC à base d'ARM
- Elles concernent
  - Le cœur (ARM *Security Extension*)
  - Les bus internes
  - Les couches logicielles
  - Les mécanismes de débogage
- Les données et les applications s'exécutant dans le TEE sont protégées contre
  - Attaques logicielles en provenance de l'extérieur du TEE (applications, OS ou hyperviseur corrompu)
  - Quelques attaques matérielles comme des transferts DMA en provenance de périphériques non dignes de confiance
- Disponible depuis l'architecture ARMv6KZ (Cortex-A) et ARMv8M (Cortex-M)

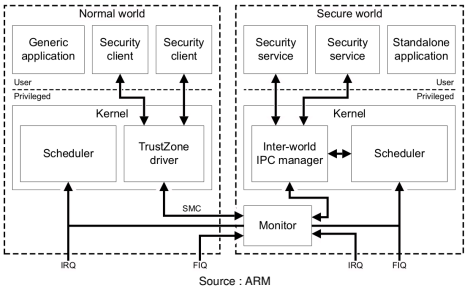
Fonctionnement de base

- Création de deux mondes : sécurisé et non sécurisé
- Cette séparation concerne
  - Le processeur qui, à un instant donné, exécute du code soit dans l'état sécurisé, soit dans l'état non sécurisé (ces états sont orthogonaux aux modes traditionnels du processeur : handler/thread/irq...)
  - Les maîtres et les esclaves sur le bus interne du SoC (un périphérique peut être configuré comme un esclave sécurisé ou non sécurisé)
  - La mémoire (l'espace d'adressage est divisé en régions sécurisées et non sécurisées)

Fonctionnement de base

- Les communications et les transitions entre ces deux mondes sont bien spécifiées
  - Un maître sécurisé (comme par exemple le processeur dans l'état sécurisé) a accès aux esclaves sécurisés et non sécurisés, à toutes les régions mémoires, etc.
  - Un maître non sécurisé n'a accès qu'aux esclaves non sécurisés, aux régions mémoires marquées comme non sécurisées, etc.
  - Un instruction spéciale (*Secure Monitor Call* / *Secure Gateway*) permet au processeur de passer de l'état non sécurisé à l'état sécurisé
  - Les interruptions peuvent être configurées pour être traitées dans l'état sécurisé ou non sécurisé (Cortex-M)

Exemple d'implémentation logicielle



Source : ARM

- Exemple de moniteur fourni par ARM : `https://github.com/ARM-software/arm-trusted-firmware`

Deux variantes

- Deux variantes de Trustzone
  - TrustZone Cortex-A (processeurs applicatifs)
  - TrustZone Cortex-M (microcontrôleurs)
- La variante pour Cortex-M est plus récente (disponible sur les Cortex-M23, M33, M35P et M55) et est une version réduite de TrustZone pour Cortex-A, adaptée pour les systèmes embarqués
  - Plusieurs points d'entrée vers le monde sécurisé (pour le Cortex-A, tous les appels vers le monde sécurisé passent par le moniteur sécurisé)
  - Des interruptions non sécurisées peuvent être traitées pendant l'exécution de code dans l'état sécurisé
  - Basculement entre les états sécurisé et non sécurisé à faible coût
- La variante pour Cortex-M dispose de plus de documentation publique (beaucoup de documents techniques pour Cortex-A sont sous NDA)

Trustzone pour Cortex-M

État du processeur

- Le processeur est soit dans l'état sécurisé, soit dans l'état non sécurisé
- Si l'extension sécurisée (*Security Extension*) est disponible, le processeur démarre dans l'état sécurisé après le reset
- Ces états sont orthogonaux par rapport au mode du processeur (*Handler*/*Thread*)
  - Il est donc possible de diviser la partie logicielle sécurisée entre un système d'exploitation sécurisé (*Handler*) et des applications sécurisées (*Thread*)

Trustzone pour Cortex-M

Partitionnement de la mémoire

- L'espace d'adressage est divisé en plusieurs zones
  - *Secure (S)* : mémoire (ou périphérique) accessible uniquement depuis un maître sécurisé (le processeur tournant en état sécurisé ou un périphérique configuré comme sécurisé)
  - *Non-secure (NS)* : mémoire (ou périphérique) accessible par n'importe quel maître
  - *Non-secure callable (NSC)* : seule zone mémoire où l'instruction `SG` (*Secure Gateway*, qui autorise le basculement de l'état non sécurisé à l'état sécurisé) est autorisée. Typiquement ne contiendra que des points d'entrées de fonctions sécurisées
- Configuration statique ou dynamique

Trustzone pour Cortex-M

Partitionnement de la mémoire

- Partitionnement contrôlé par
  - *Secure Attribution Unit* (SAU) interne
  - *Implementation Defined Attribution Unit* (IDAU) externe optionnelle

Trustzone pour Cortex-M

Secure Attribution Unit

- Permet de configurer un nombre fixe de régions mémoires (généralement 8)
- Désactivée au reset
- Configurable uniquement depuis l'état sécurisé
- Si aucune région n'est configurée ou si la SAU est désactivée (et qu'il n'y a pas d'IDAU), l'intégralité de l'espace d'adressage mémoire est considérée comme sécurisé (S)
  - Le processeur ne peut pas passer dans l'état non sécurisé

Trustzone pour Cortex-M

Secure Attribution Unit, configuration

- `SAU_RNR`
  - Bits [7 :0] : numéro de la région à configurer
- `SAU_RBAR`
  - Bits [31 :5] : Bits [31 :5] de l'adresse de base de la région (les bits [4 :0] de cette adresse valent 0)
- `SAU_RLAR`
  - Bits [31 :5] : Bits [31 :5] de la dernière adresse de la région (les bits [4 :0] de cette adresse sont fixés à 0x1F)
  - Bit 1 : 1 = la région est non-secure callable (NSC), 0 = la région est non-secure (NS)
  - Bit 0 : 1 = région active, 0 = région désactivée

Trustzone pour Cortex-M

Secure Attribution Unit, configuration

- `SAU_CTRL`
  - Bit 1 : Tout non-secure (si 1 et que la SAU est désactivée, toute la mémoire est considérée comme non-secure)
  - Bit 0 : 1 = SAU active, 0 = SAU désactivée
- `SAU_TYPE`
  - Bits [7 :0] : Nombre de régions configurables (0, 4 ou 8)

Trustzone pour Cortex-M

Implementation Defined Attribution Unit

- 8 régions configurables via la SAU peuvent être insuffisantes
- Le fabriquant du SoC peut ajouter une *Implementation Defined Attribution Unit* (IDAU)
- Plusieurs implémentations courantes utilisent le bit 28 de l'adresse pour indiquer si une région est sécurisée ou non
- Pour une adresse données, les résultats retournés par la SAU (NS, NSC, S) et par l'IDAU sont fusionnés pour donner la configuration finale pour cette adresse

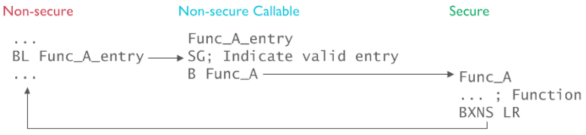
Trustzone pour Cortex-M

Transitions entre l'état sécurisé et non sécurisé

- Nouvelles instructions
  - `SG` (*Secure Gateway*) : passage de l'état non sécurisé à l'état sécurisé
  - `BXNS` : branchement vers le monde non sécurisé (utilisé par un programme sécurisé pour sauter ou revenir vers un programme non sécurisé)
  - `BLXNS` : appel d'une fonction dans le monde non sécurisé (utilisé par un programme sécurisé pour appeler une fonction non sécurisée)

Trustzone pour Cortex-M

Appel d'une fonction sécurisée depuis le monde non sécurisé

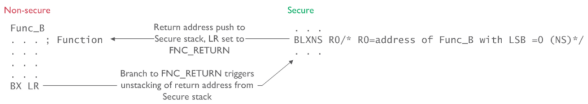


Source : TrustZone technology for the ARMv8-M architecture
(`https://static.docs.arm.com/100690/0200/armv8m_trustzone_technology_100690_0200.pdf`)

- L'instruction `SG` doit être dans une zone *non-secure callable* (NSC)

Trustzone pour Cortex-M

Appel d'une fonction non sécurisée depuis le monde sécurisé



Source : TrustZone technology for the ARMv8-M architecture
(`https://static.docs.arm.com/100690/0200/armv8m_trustzone_technology_100690_0200.pdf`)

- `BLXNS`
  - Pousse l'adresse de retour dans la pile sécurisée
  - Pousse une partie du registre PSR vers la pile sécurisée
  - Stocke la valeur `FNC_RETURN` dans le registre `r14`
  - Le bit de poids faible de l'adresse de destination doit être 0

Trustzone pour Cortex-M

Interruptions

- Chaque interruption peut être configurée pour être traitée dans l'état sécurisé ou non sécurisé (registre `NVIC_ITNS`)
- Si une interruption non sécurisée survient pendant l'exécution dans l'état sécurisé, tous les registres sont automatiquement sauvegardés dans la pile sécurisée puis effacés avant que le gestionnaire d'interruption puisse s'exécuter dans l'état non sécurisé (ce qui augmente la latence)

Trustzone

Conclusion

- Le modèle d'attaque considère principalement les attaques logicielles en provenance du monde non sécurisé (application ou système d'exploitation) ou les périphériques malicieux
- Pas de prise en compte des autres attaques matérielles (injection de fautes, canaux auxiliaires, espionnage de bus, lecture de la mémoire, etc.)
- Permet d'exécuter du code sensible (exemple : application bancaire) à côté d'applications non dignes de confiance (OS et applications non contrôlées)

Trustzone

Conclusion

- TrustZone doit être couplé avec d'autres contre-mesures et en particulier un mécanisme permettant de s'assurer de l'intégrité du code lors du démarrage et de la configuration des mécanismes de protection
- Si le contenu de la flash peut être modifié par exemple, un adversaire pourrait modifier le logiciel s'exécutant dans l'espace sécurisé, ruinant ainsi la sécurité fournie par TrustZone

Plan

Pointer Authentication [11]

Présentation

- Mécanisme introduit par l'architecture ARMv8.3-A (octobre 2016)
- Permet de rendre difficile la modification d'un pointeur en mémoire (par exemple dans la pile) par un adversaire
- L'espace d'adressage dans l'architecture AArch64 est inférieur à 64 bits
  - Donc dans un pointeur 64 bits, un certain nombre de bits sont inutilisés
  - Ces bits peuvent être utilisés pour y stocker un MAC (appelé PAC, *Pointer Authentication Code*) permettant ainsi d'authentifier le pointeur et d'empêcher sa modification

Pointer Authentication [11]

Instructions

- Deux séries d'instructions principales
  - `PAC*` : permet de calculer un PAC et de l'intégrer à un pointeur
  - `AUT*` : permet de vérifier un PAC et de restaurer la valeur du pointeur (ou de le remplacer par un pointeur volontairement invalide en cas d'échec de la vérification)
- Quelques instructions supplémentaires (variantes *verify-then-return*, *verify-and-branch*...)
- La plupart de ces instructions sont codées de manière à être considérées comme des `NOP` dans les versions précédentes de l'architecture
  - Elles peuvent donc être ajoutées dans du code sans casser la rétro-compatibilité

Pointer Authentication [11]

PAC

- Le PAC est calculé à l'aide de l'algorithme QUARMA (développé *ad-hoc*) à partir :
  - De la valeur du pointeur
  - D'une clé secrète de 128 bits
  - D'un « contexte » de 64 bits
- 5 registres internes pour stocker des clés sont disponibles
  - Instruction/Données, clés A/B, plus une clé à usage de l'instruction générique `PACGA`
  - Ces registres ne peuvent être lus ou écrits dans le niveau EL0 (*user mode*), seulement utilisés par les instructions `PAC*` et `AUT*`
  - Ces registres peuvent être modifiés dans les niveaux de privilèges supérieurs (EL1, 2 et 3)
  - La génération et la gestion des clés est à la charge de la partie logicielle

Pointer Authentication [11]

Exemple

- Prologue de fonction (`x30` contient l'adresse de retour de la fonction)

```
PACIASP             # Eq. PACIA x30, sp
SUB sp, sp, #0x40
STP x29, x30, [sp, #0x30]
ADD x29, sp, #0x30
```

- Épilogue de fonction

```
LDP x29, x30, [sp, #0x30]
ADD sp, sp, #0x40
AUTIASP             # Eq. AUTIA x30, sp
RET
```


Pointer Authentication [11]

Sécurité

- Taille du PAC : entre 3 et 31 bits en fonction de la configuration (taille de l'espace d'adressage virtuel du processeur et utilisation des *tagged addresses*)
- Résistant même si l'attaquant peut lire la mémoire (MAC et clés stockées dans des registres inaccessibles)
- L'attaquant ne peut modifier un pointeur protégé sans être détecté seulement s'il peut deviner la bonne valeur du PAC correspondant (dépend de la taille du PAC)
- Substitution de pointeurs authentifiés : les différentes clés et l'usage du contexte permettent de limiter ce problème
- Gestion et réutilisation de clés : nécessité d'un bon aléa, problème avec *fork* (nécessité de partager les mêmes clés entre le processus père et le fils)

Plan

Intel Memory Protection Extensions (MPX)

Présentation

- Extensions [7] de l'ISA développées par Intel pour améliorer la sécurité mémoire, disponibles depuis l'architecture *Skylake* (2015)
- Permet de vérifier dynamiquement qu'une adresse appartient bien à une certaine plage (début-fin) définie préalablement
- Imposent des changements au niveau
  - Du matériel (nouvelles instructions et registres)
  - Du système d'exploitation (gestion mémoire et gestion des exceptions)
  - Du compilateur (ajout des instructions)
  - Des applications

Exemple

Sans protection

(a) Original code

```
struct obj { char buf[100]; int len }
obj* a[10]                                  ;; Array of pointers to objs
for (i=0; i<M; i++):                        ;; M may be greater than 10
  ai = a + i                                ;; Pointer arithmetic on a
  objptr = load ai                          ;; Pointer to obj at a[i]
  lenptr = objptr + 100                     ;; Pointer to obj.len
  len = load lenptr
```

Source [10]

- Garantir que *ai* ne déborde pas du tableau *a*
- Garantir que *lenptr* reste bien dans les limites de la structure *obj*

Matériel

- Nouveaux registres de 128 bits stockant la définition d'une région (une limite basse et une limite haute)
  - 4 dans l'architecture *Skylake* : `bnd0`–`bnd3`
- Nouvelles instructions
  - `bndmk` : création d'une nouvelle région
  - `bndcl`/`bndcu` : compare une adresse avec la limite basse/haute d'une région
  - `bndmov` : copie le contenu d'un registre vers/depuis la mémoire (en général la pile)
  - `bndldx`/`bndstx` : charge/stocke le contenu d'un registre vers la *Bounds Table*
- Des registres de configuration et d'état

Exemple

Avec MPX

(b) Intel MPX

```
obj* a[10]
a_b = bndmk a, a+79                         ;; Make bounds [a, a+79]
for (i=0; i<M; i++):
  ai = a + i
  bndcl a_b, ai                             ;; Lower-bound check of a[i]
  bndcu a_b, ai+7                           ;; Upper-bound check of a[i]
  objptr = load ai
  objptr_b = bndldx ai                      ;; Bounds for pointer at a[i]
  lenptr = objptr + 100
  bndcl objptr_b, lenptr                    ;; Checks of obj.len ]
  bndcu objptr_b, lenptr+3                  ]
  len = load lenptr
```

Source [10]

Bounds Table

- La définition d'une région peut être associée à chaque pointeur grâce à une table stockée en mémoire (*Bounds Table*)
- Les instructions `bndldx` et `bndstx` permettent, à partir de l'adresse d'un pointeur, d'accéder à la définition d'une région
- Le mécanisme permettant de passer de l'adresse d'un pointeur à la zone mémoire où est stockée la définition de la région est similaire à celui utilisé par le mécanisme de mémoire virtuel
  - Le registre `BNDCFGx` contient l'adresse d'une `Bounds Directory`
  - Une partie des bits de l'adresse est utilisée comme index dans cette BD
  - L'entrée de la BD pointe vers une BT
  - Le reste des bits de l'adresse est utilisé comme index dans cette BT

Rôle du système d'exploitation

- Gestion des *Bounds Tables*
  - La `Bound Directory` est allouée au démarrage d'un programme
  - Les `Bound Tables` sont créées dynamiquement à la demande par le système d'exploitation lorsqu'un programme fait un `bndctx` (ce qui déclenche une exception la première fois)
- Gestion des exceptions

Conclusions

- Analyses des performances, de la sécurité et de l'utilisabilité dans [10]
  - Impact sur les performances significatif (malgré le support matériel : peu de registres, pas de mécanisme de cache pour les BT, etc.)
  - Pas de protection contre les erreurs temporelles (usage d'un objet avant son initialisation ou après sa destruction)
  - Problèmes divers (support du multithreading, support de certaines constructions du C/C++, etc.)
- Arrêt du support par *Linux* (2018) et probable par *gcc*

Plan

Présentation

- Extensions développées par Intel pour permettre de garantir l'intégrité et la confidentialité d'opérations sensibles sur un ordinateur dont le code privilégié (système d'exploitation, hyperviseur...) est potentiellement malicieux
- Nouvelle solution au problème de l'exécution sécurisée à distance
- Disponibles depuis l'architecture *SkyLake* (2015)
- Analyse détaillée dans [1] (lecture très intéressante)

Fonctionnement
Protection de la mémoire

- Un processus qui a besoin d'exécution sécurisée va s'exécuter dans une enclave
- Une région particulière de la mémoire (*Processor Reserved Memory*, PRM) est dédiée à ces enclaves
- Le processeur protège cette zone contre tout accès qui ne proviendrait pas d'une enclave (noyau, hyperviseur, SMM, accès DMA en provenance d'un périphérique...)
- Les données qui sont situées dans cette zone sont aussi chiffrées et protégées en intégrité à la volée par un module matériel dédié sur le processeur
  - Empêche les attaques physiques par espionnage des bus du processeur ou du contenu de la RAM

Fonctionnement (suite)
Protection de la mémoire

- Cette zone est divisée en page (*Enclave Page Cache*, EPC)
- Chaque EPC peut contenir du code ou des données d'une enclave
- L'affectation des EPC à une enclave est gérée par le logiciel (non digne de confiance) et contrôlée par la processeur pour garantir qu'une page n'est allouée qu'à une enclave et une seule
- Le système d'exploitation peut evincer des pages EPC vers la mémoire en utilisant des instructions dédiées
  - Le contenu de ces pages est protégé (confidentialité et intégrité) afin de ne pas pouvoir être altéré

Fonctionnement (suite)

Initialisation d'une enclave

- Le code et les données initiales de l'enclave sont chargés par un logiciel non digne de confiance (système d'exploitation par exemple)
  - Grâce à des instructions spécifiques permettant de copier des données depuis la mémoire (hors PRM) vers des pages EPC
  - Le code et les données initiales sont en clair et ne peuvent donc contenir de secrets
  - Pendant le chargement, un condensé cryptographique est calculé sur les données chargées, ce qui permettra d'attester du code exécuté
- À la fin du chargement, la procédure est désactivée ce qui empêche la modification du code et des données de l'enclave par du logiciel non digne de confiance

Fonctionnement (suite)

Exécution dans une enclave

- L'exécution du code d'une enclave ne peut être démarré que par une instruction spéciale
- Le code est exécuté en mode protégé, dans le niveau de privilèges 3 et avec les traductions d'adresse (MMU) mises en place par le système d'exploitation et éventuellement l'hyperviseur
- Lors d'une interruption du flot d'exécution d'une enclave (interruption, exception, etc.), le processeur sauvegarde l'état du processeur dans une zone dédiée au sein de l'enclave et efface les registres du processeur avant de transférer le contrôle
  - Le système d'exploitation n'a ainsi pas accès au contexte matériel de l'enclave lors de l'interruption

Fonctionnement (suite)

Attestation

- Mécanisme d'attestation à distance
- Permet de prouver à distance qu'un message a bien été émis par un code donnée, s'exécutant dans une enclave sur un processeur capable de la sécuriser

Conclusion

- Technologie très prometteuse, surtout dans un contexte de *cloud computing*
- Néanmoins, un certain nombre de problèmes subsistent :
  - Attaque contre les caches [4]
  - Problèmes de synchronisation en cas de multithread [13]
  - Possibilité de protéger du code malveillant au sein d'une enclave [12]
  - Variantes de Spectre contre SGX ?
  - Confiance en Intel ?

Plan

Intel Control-Flow Enforcement Technology (CET)

Présentation

- Spécifications [6] développées par Intel
  - Pas encore de processeurs les intégrant
- Permet de se prémunir contre les attaques *Return/Jump Oriented Programming (ROP/JOP)*

Shadow stack

- Une nouvelle pile est utilisée (*Shadow stack*)
- Lors d'une instruction d'appel de sous-routine (`CALL` par exemple), l'adresse de retour est poussée vers la pile et vers la *shadow stack*
- Lors d'un retour (`RET`), l'adresse de retour est récupérée depuis les deux piles ; si les deux version diffèrent, une exception est levée
- La *shadow stack* est stockée dans une zone mémoire qui ne peut être modifiée que par les instructions de saut (et donc pas par un adversaire) grâce à la MMU

Indirect Branch Tracking

- Nouvelle instruction `ENDBRANCH` pour marquer les destinations valides d'un saut ou d'un appel
- Lorsque la protection est activée, les saut ou les appels dans le programme doivent arriver sur l'instruction `ENDBRANCH` ; dans le cas contraire, une exception est levée

Plan

Plan

Bugs matériels

- Les processeurs actuels comportent plusieurs centaines de millions de transistors et incluent de nombreux mécanismes pour accélérer les traitement
  - Exécution out-of-order
  - Prédiction de branchement
  - Multi-threading
  - Renommage de registre...
- Tout comme les logiciels, ils ne sont pas à l'abri de bugs, pouvant remettre en cause la sécurité

Bugs processeur

- Le bug le plus connu est celui de la division fautive du Pentium découvert en 1994
- Autre bug connu du Pentium : le F0 0F
  - L'instruction `lock cmpxchg8b eax` (en code machine `f0 0f c7 c8`) qui devrait normalement lever une exception fait planter le processeur qui doit alors être reinitialisé
  - Comme cette instruction peut être exécutée sans privilèges, cela crée un déni de service (un bug similaire existe dans les processeurs Cyrix)

Bugs processeur

- Les bugs des processeurs sont souvent inconnus du grand public
- Mais, s'ils touchent des fonctionnalités liées à la protection mémoire, ils peuvent avoir un impact important sur la sécurité
- Et quid des bugs qui n'en seraient pas (portes dérobées) ?
- Il est assez facile d'introduire une porte dérobée dans un processeur... (cf. [2])

Plan

Introduction

- Du fait de leur évolution ces 30 dernières années, les processeurs x86 comportent de nombreux modes de fonctionnement
- Principalement pour garder la compatibilité ascendante avec le 8088...
- Mais certains de ces modes sont méconnus et peuvent poser des problèmes de sécurité importants

System Management Mode

- Mode spécial des processeurs x86 utilisé principalement pour gérer la carte mère, l'alimentation et le contrôle de la température (ventilateurs, etc.)
- Mode 16 bits
  - Accès libre à l'intégralité de la mémoire
  - Aucun mécanisme de protection mémoire (ni segmentation ni pagination)
  - Accès libre aux IO
- Durant l'exécution en mode SMM, l'OS est suspendu
  - L'OS n'est pas informé de cette interruption
  - Il ne peut pas faire respecter sa politique de sécurité
- $\rightsquigarrow$ Le code s'exécutant en mode SMM a le contrôle total sur la machine

System Management Mode

- Le processeur entre en mode SMM en recevant une System Management Interrupt (interruption matérielle générée par le chipset, peut être déclenchée par une instruction d'IO)
- Le code du gestionnaire d'interruption (qui donc s'exécute en mode SMM) est situé dans une zone de la RAM appelée SMRAM
- Initialement, cette zone mémoire n'était pas particulièrement protégée...
- Actuellement, le chipset protège l'accès à cette zone mais cette protection n'est pas parfaite et des failles ont été trouvées par le passé [3]

Plan

Attaque *Meltdown* [9]

- Exécution des instructions dans le désordre (*out-of-order*)
  - Les processeurs disposent de plusieurs unités de calcul et de traitement (plusieurs ALU...)
  - Afin de maximiser l'utilisation de ces ressources, les instructions sont réordonnées (en conservant les dépendances éventuelles)
- Exécution spéculative
  - Des instructions peuvent être exécutées avant d'être certain qu'elles doivent l'être (i.e. après un saut conditionnel, avant d'avoir vérifié que les instructions précédentes ne génèrent pas d'exceptions...)
  - Si elles ne devaient pas l'être, leurs résultats sont annulés et aucun effet n'est visible par le logiciel
  - *Néanmoins, l'exécution de ces instructions peut laisser des traces au niveau micro-architecture qui peuvent être détectées*

Attaque *Meltdown* [9]

Exemple simple

- Considérons le code suivant (exemple tiré de [9]) :

```
exception();
accès mémoire(tableau[donnée * 4096]);
```

- Normalement la seconde instruction n'est pas exécutée
- Mais à cause de l'exécution spéculative dans le désordre, elle peut être exécutée avant que le processeur ne s'aperçoive qu'il ne devait pas le faire
- L'effet de cette instruction sera annulé (registre de destination restauré) mais elle aura eu un effet de bord sur le cache

Attaque *Meltdown* [9]

Exemple simple, suite

- En effet, l'accès mémoire va placer la ligne située à l'adresse `tableau[donnée * 4096]` dans le cache du processeur
- Il est possible pour un autre processus, via une attaque par canal auxiliaire contre le cache, de détecter ce chargement et de retrouver l'adresse accédée (et donc ici, la valeur de `donnée`)

Attaque *Meltdown* [9]

Application à la lecture de toute la mémoire

- Pour des questions de performance, dans de nombreux systèmes d'exploitation, le code et les données du noyau sont mappés dans l'espace d'adressage de chaque processus
- De même pour l'intégralité de la mémoire physique (sur les systèmes 64 bits)
- Ces adresses sont simplement marquées comme non accessibles depuis l'espace utilisateur au niveau de la MMU
- Donc si un processus essaie d'accéder à ces adresses, la MMU va déclencher une exception et le processus va être tué par le système d'exploitation

Attaque *Meltdown* [9]

Application à la lecture de toute la mémoire

- Exemple, cartographie mémoire sous Linux sur architecture x86_64 (cf. `Documentation/x86/x86_64/mm.txt`)

```
0000000000000000 - 00007fffffffffff (=47 bits) user space, different per mm
hole caused by [47:63] sign extension
ffff800000000000 - ffff87ffffffffff (=43 bits) guard hole, reserved for hypervisor
ffff880000000000 - ffffc7ffffffffff (=64 TB) direct mapping of all phys. memory
ffffc80000000000 - ffffc8ffffffffff (=40 bits) hole
ffffc90000000000 - ffffe8ffffffffff (=45 bits) vmalloc/ioremap space
ffffe90000000000 - ffffe9ffffffffff (=40 bits) hole
ffffea0000000000 - ffffeaffffffffff (=40 bits) virtual memory map (1TB)
... unused hole ...
ffffec0000000000 - fffffbffffffffff (=44 bits) kasan shadow memory (16TB)
... unused hole ...
ffffff0000000000 - ffffff7fffffffff (=39 bits) LDT remap for PTI
ffffff8000000000 - ffffffefffffffff (=39 bits) cpu_entry_area mapping
fffffff0000000000 - fffffff7fffffffff (=39 bits) %esp fixup stacks
... unused hole ...
ffffffef00000000 - ffffffffffffffff (=64 GB) EFI region mapping space
... unused hole ...
ffffffff80000000 - ffffffff9fffffff (=512 MB) kernel text mapping, from phys 0
ffffffffa0000000 - (fixmap start) ( 1526 MB) module mapping space (variable)
(fixmap start) - ffffffffff5fffff kernel-internal fixmap range
ffffffffff600000 - ffffffffff600fff (=4 kB) legacy vsyscall ABI
ffffffffffe00000 - ffffffffffffffff (=2 MB) unused hole
```


Attaque *Meltdown* [9]

Application à la lecture de toute la mémoire

```
; rcx = kernel address
; rbx = probe array
retry:
  mov al, byte [rcx]   ; RAX[7:0] = RAM[RCX] -> Exception
  shl rax, 0xc         ; RAX = RAX * 4096
  jz retry             ; if RAX == 0 goto retry
  mov rbx, qword [rbx + rax] ; RAM[RBX+RAX] = RBX
```

Attaque *Meltdown* [9]

Application à la lecture de toute la mémoire

- Pendant que la première instruction s'exécute, les trois instructions suivantes (qui en dépendent) sont mises en attente
- Dès que le résultat de la lecture est disponible, ces trois instructions commencent à s'exécuter
- La première instruction déclenche une exception (accès invalide à une adresse noyau depuis l'espace utilisateur)
- Si la dernière instruction a le temps de commencer à s'exécuter, la ligne dont l'adresse est `RBX+RAX` est chargée dans le cache du processeur
- Or cette adresse dépend de la donnée lue par la première instruction

Attaque *Meltdown* [9]

Application à la lecture de toute la mémoire

- Certes, le processus est tué dans l'opération (il est possible sur certaines architectures d'empêcher l'exception sans pour autant permettre l'accès direct au registre impacté)
- Mais un processus complice peut récupérer l'adresse accédée via une attaque SCA contre le cache et donc en déduire la donnée lue dans la zone noyau
- Comme l'intégralité de la mémoire physique est mappée dans l'espace noyau, il est ainsi possible de lire le contenu de toute la RAM physique

Attaque *Meltdown* [9]

Solutions

- Au niveau logiciel, il est possible de limiter l'impact de cette attaque en arrêtant de mapper les structures du noyau dans l'espace utilisateur
- KAISER [5]
- Impact non négligeable sur les performances car nécessité de modifier les tables de page à lors des basculements espace utilisateur / espace noyau

Attaque *Spectre* [8]

- Dans les différentes variantes de l'attaque *Spectre*, les mécanismes de prédiction de branchement du processeur sont visés ainsi que l'exécution spéculative qui en résulte
  - Lors d'une instruction de branchement conditionnel, le processeur fait une hypothèse sur le résultat de ce branchement et commence à exécuter spéculativement les instructions suivantes
  - Lors d'une instruction de saut, le processeur prédit également l'adresse de destination du saut
- Ces deux mécanismes fonctionnent avec apprentissage et cet apprentissage peut être utilisé par l'attaquant pour forcer l'exécution spéculative d'une portion de code qui ne devrait pas l'être
- Cette exécution spéculative va laisser des traces, notamment au niveau du cache, qui vont pouvoir être exploitées comme pour l'attaque *Meltdown*

Attaque *Spectre* [8]

Variante 1

- Dans le code du processus visé (exemple tiré de [8])

```
if (x < array1_size)
   y = array2[array1[x] * 256]
```

- Hypothèses
  - `x` est contrôlable par l'adversaire et est choisi (hors des limites du tableau) de manière à ce que `array1[x]` pointe vers une valeur secrète k
  - `array1_size` et `array2` ne sont pas présents dans le cache mais k l'est
  - Lors des exécutions précédentes du test, la valeur de `x` fournie était correcte et donc le prédicteur de branchement va considérer que le résultat du test sera probablement vrai

Attaque *Spectre* [8]

Variante 1

- Lors de l'exécution avec la valeur de `x` malicieuse, `array_size` n'étant pas dans le cache, le processeur se base sur la prédiction de branchement et se met à exécuter spéculativement l'instruction suivante en attendant la lecture de `array_size`
- La valeur de `array1[x]` (k) étant connue, le processeur peut commencer tout de suite la lecture à l'adresse `array2[array1[x] * 256]`, qui n'étant pas dans le cache, sera rapatriée depuis la mémoire
- Lorsque le résultat du test est finalement connu, l'effet de la seconde instruction est annulé
- Néanmoins, une ligne de cache dont l'adresse dépend de la valeur secrète k aura été chargée dans le cache et donc une attaque SCA contre ce dernier permettra de retrouver k

Attaque *Spectre* [8]

Variante 2

- L'adresse de destination d'un saut n'est pas toujours connue (ex. instruction `jmp [eax]`)
- Si la détermination de la destination du saut est retardée par un défaut de cache, le processeur va utiliser le prédicteur de branchement pour la deviner et exécuter spéculativement les instructions
- Si ce mécanisme peut être entraîné malicieusement par l'adversaire, ce dernier peut s'en servir pour faire exécuter spéculativement des instructions à des adresses qu'il a choisies (similaire au ROP)
- L'adversaire choisi ces instructions de telle sorte à ce qu'elles fassent fuir de l'information sensible via le cache

Plan

Conclusion

- Le support d'exécution offre un certain nombre de mécanismes pour améliorer la sécurité logicielle
- Néanmoins, il appartient au logiciel (le plus souvent au système d'exploitation ou à l'hyperviseur) de bien s'en servir
- N'empêchent pas tous les problèmes logiciels...

Références

[1] Victor Costan and Srinivas Devadas.
Intel SGX explained.
Cryptology ePrint Archive, Report 2016/086, 2016.
http://eprint.iacr.org/2016/086.

[2] Loïc Duflot.
Bogues et piégeages des processeurs, quelle conséquence sur la sécurité ?
In *SSTIC*, June 2008.
http://www.ssi.gouv.fr/archive/fr/sciences/fichiers/lti/sstic08-duflot.pdf.

[3] Loïc Duflot, Olivier Levillain, Benjamin Morin, and Olivier Grumelard.
System management mode design and security issues.
In *IT Defense*, February 2010.
http://www.ssi.gouv.fr/IMG/pdf/IT_Defense_2010_final.pdf.

[4] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller.
Cache Attacks on Intel SGX.
In *Proceedings of the 10th European Workshop on Systems Security*, EuroSec'17, pages 2 :1–2 :6, New York, NY, USA, 2017. ACM.

[5] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice, and Stefan Mangard.
KASLR is Dead : Long Live KASLR, pages 161–176.
Springer International Publishing, Cham, 2017.

[6] Intel.
Control-flow enforcement technology specification.
https://software.intel.com/sites/default/files/managed/4d/2a/control-flow-enforcement-technology-preview.pdf, May 2019.

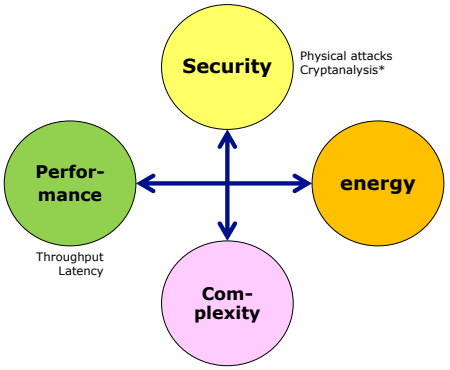
[7] Intel.
Intel® Memory Protection Extensions (Intel® MPX).
https://software.intel.com/en-us/isa-extensions/intel-mpx, July 2019.

[8] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom.
Spectre Attacks, January 2018.

Implementation Trade-offs for Symmetric Cryptography

Jean-Luc Danger
Télécom Paris

Implementation Trade-offs

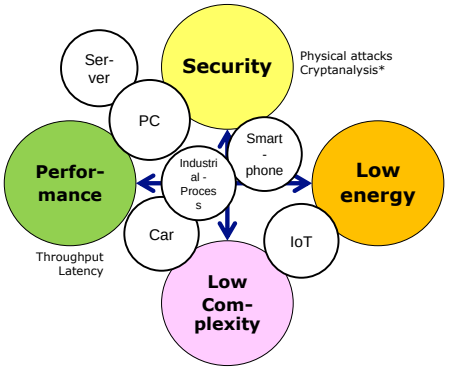


* Depends on algorithm, not implementation

Cryptography type vs applications

- **Classical Cryptography**
  - Servers, PCs, Smartphones
  - Main constraints = security, performance
- **Lightweight cryptography for IoT**
  - Always connected to a network
    - Sensor networks, cars, industrial process...
  - Main constraints = security, complexity
- **Ultra-lightweight cryptography**
  - Temporary connected
    - RFIDs, sensors
  - Main constraints = complexity, energy

Implementation Trade-offs



* Depends on algorithm, not implementation

Software metrics

- **Complexity**
  - Code size
  - Memory
- **Performance**
  - Throughput
  - Latency
- **Energy**
  - nJoule/bit
- **Security**
  - Physical attack resistance : SCA and Fault
    - No metrics, just assesment by succes rate, guessing entropy, fault models…
- **Testbench for lightweight crypto in software : FELICS**

Dinu, D., Biryukov, A., Großschädl, J., Khovratovich, D., Corre, Y. L., & Perrin, L. (2015, July). FELICS–Fair Evaluation of Lightweight Cryptographic Systems. In *NIST Workshop on Lightweight Cryptography*.

Hardware metrics

- **Security**
  - Physical attack resistance : SCA and Fault
    - No metrics, just assesment by succes rate, guessing entropy, fault models
- **Complexity**
  - Gate Equivalent (NAND 4 transistors)
    - e.g. AES 128-bit key => ~2500GE
  - Memory
    - RAM, LUTs
    - Registers
- **Performance**
  - Throughput
    - Block ciphers = clock frequency*nb_bits / nb_rounds
    - Stream ciphers = clock frequency
  - Latency
- **Energy**
  - pJoule/bit

Hardware vs Software

- **Hardware is always better**
  - Parallelism => more throughput and less latency
  - Energy : at least 20 times lower in HW
  - Security : more side-channel attacks in SW
    - Many leakage points for DPA
    - Great SNR
    - Cache attacks

	SW-XTEA			HW-AES		
Payload [B]	Time [ms]	Throughput [kB/s]	Energy [mJ]	Time [ms]	Throughput [kB/s]	Energy [mJ]
1	2,48	0,39	0,23	0,24	4,03	0,01
15	2,51	5,83	0,23	0,28	53,26	0,01
16	4,99	3,13	0,46	0,52	30,21	0,03
31	5,03	6,02	0,47	0,55	54,82	0,03
32	7,51	4,16	0,69	0,79	39,33	0,04
47	7,55	6,08	0,70	0,83	55,35	0,04
48	10,03	4,68	0,93	1,07	43,75	0,06
63	10,06	6,11	0,94	1,11	55,59	0,06
64	12,54	4,98	1,17	1,35	46,34	0,07
79	12,58	6,13	1,17	1,38	55,74	0,07
80	15,05	5,19	1,40	1,63	48,04	0,08
95	15,09	6,15	1,40	1,66	55,86	0,09
96	17,57	5,34	1,63	1,90	49,26	0,10
104	17,59	5,77	1,64	1,92	52,85	0,10

Botta, M., Simek, M., & Mitton, N. (2013, July). Comparison of hardware and software based encryption for secure communication in wireless sensor networks. In *Telecommunications and Signal Processing (TSP), 2013 36th International Conference on* (pp. 6-10). IEEE.

Lightweight Crypto type SPN

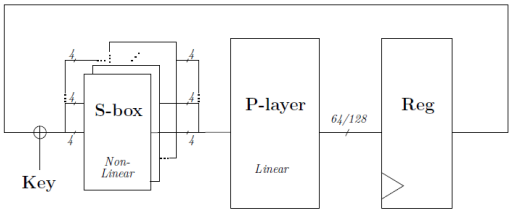
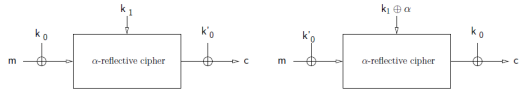


Figure 1: Generic iterated round function (SPN structure)

SPN with α reflexion



(a) Encryption (b) Decryption

Figure 2: α-reflective ciphers for encryption and decryption

Lightweight Crypto type Feistel

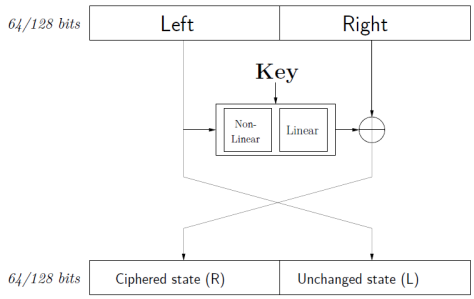


Figure 3: Generic Feistel Round

Summary of common Lightweight Crypto

Cipher	Block size	Type	Rounds	E/ED	Key Process	Focus
PRINCE	64	SPNα	12	SHW	KExp	Latency
NOEKEON	128	SPN	16	SHW	KS	Area
Piccolo	64	GFN	33	SHW	KS	Energy
Midori	64	SPN	16	MDO	None	Energy
	128	SPN	20	MDO	None	Energy
SIMON	64	Feistel	44	SHW	KS	Area
	128	Feistel	68	SHW	KS	Area
Speck	64	Feistel	27	SHW	KS	Area
	128	Feistel	32	SHW	KS	Area
SKINNY	64	SPN	32	MDO	Tweak	Area
	128	SPN	40	MDO	Tweak	Area
LED	64	SPN	48	-	None	Area
Mantis	64	SPNα	14	SHW	Tweak	Latency
PRESENT	64	SPN	32	MOp	KS	Area
GIFT	64	SPN	28	MOp	KS	Area
	128	SPN	40	MOp	KS	Area

Table 1: Summary of properties of Lightweight block ciphers

Stage TPT Etienne Tehrani

Complexity comparison 1

Table 1. Hardware implementation results of selected symmetric encryption algorithms.

Algorithm		key size	block size	cycles/ block	Tech. [μm]	Area [GE]
Stream Ciphers						
Trivium	[13]	80	1	1	0.13	2,599
Grain	[13]	80	1	1	0.13	1,294
Block Ciphers						
PRESENT	[36]	80	64	547	0.18	1,075
SEA	[23]	96	96	93	0.13	3,758
mCrypton	[6]	96	64	13	0.13	2,681
ICEBERG	[23]	128	64	16	0.13	7,732
HIGHT	[16]	128	64	34	0.25	3,048
AES	[10]	128	128	1,032	0.35	3,400
AES	[14]	128	128	160	0.13	3,100
DESXL	[22]	184	64	144	0.18	2,168

Poschmann, A., Moradi, A., Khoo, K., Lim, C. W., Wang, H., & Ling, S. (2011). Side-channel resistant crypto for less than 2,300 GE. *Journal of Cryptology*, *24*(2), 322-345.

Complexity Example 2 with 64-bit blocks

Cipher	Ref.	Tech (nm)	Architecture (cycle/round)	Area (GE)	Latency (ns)	TPmax (Gbps)	TP@100kHz (kbps)
Block size : 64-bit							
PRINCE	[7]	90	1/32	7996	13.9	4.56	-
PRESENT	[3]	90	32/32	1560	52.16	1.23	-
LED	[8]	180	48/48	1265	-	-	3.4
Midori64	[2]	90	16/16	2450	33.92	1.89	-
Piccolo	[10]	130	33/33	1362	-	-	12.12
	[10]	130	2112/33	818	-	-	193.94
SIMON 64	[4]	130	44/44	1417	-	-	133.3
	[3]	90	44/44	1458	-	80.52	0.79
Speck-64	[4]	130	27/27	1658	-	-	206.5
SKINNY 64	[5]	180	32/32	1696	59.84	0.95	177.78
	[5]	180	1/32	17454	51.59	1.24	6400
	[5]	180	128/32	1399	121.60	0.09	8.12
	[5]	180	2048/32	1172	2170.88	0.02	2.03
Mantis7	[3]	90	32/32	1477	58.88	0.96	-
	[5]	180	1/14	11209	20.5	3.12	-
	[5]	180	1/14	23926	11.0	5.82	-
GIFT-64	[3]	90	28/28	1345	51.24	1.25	-
	[3]	90	112/28	1113	239.68	0.06	-
	[3]	90	2048/32	930	4784.64	0.01	-

Table 2: Comparison of Area and Latency for 64-bit block size unrolled, round and serialized implementations of Lightweight block ciphers

Stage TPT Etienne Tehrani

Complexity Example 2 with 128-bit blocks

Cipher	Ref.	Tech (nm)	Architecture (cycle/round)	Area (GE)	Latency (ns)	TPmax Gbps	TP@100kHz kbps
Block size : 128-bit							
Midori128	[2]	90	20/20	3661	48.80	2.62	-
Speck-128	[4]	130	32/32	2727	-	-	376.5
SIMON 128	[4]	130	68/68	2090	-	-	182.9
	[3]	90	68/68	2064	127.16	1.01	-
SKINNY 128	[5]	180	40/40	2391	115.60	1.11	320.00
	[5]	180	1/40	32415	97.93	1.31	12800
	[5]	180	320/40	1840	329.60	0.14	14.68
	[5]	180	5120/40	1481	5376.00	0.02	1.83
GIFT-128	[3]	90	40/40	2104	74.00	1.73	-
	[3]	90	40/40	1997	74.00	1.73	-
	[3]	90	160/40	1455	360.00	0.08	-
	[3]	90	5120/40	1213	12595.20	0.01	-

Table 3: Comparison of Area and Latency for 128-bit block size unrolled, round and serialized implementations of Lightweight block ciphers

Stage TPT Etienne Tehrani

Complexity vs throughput vs latency

- **Complexity ~ a . throughput**
  - Example with AES-128
    - Complexity = 10 rounds (unrolled with pipeline) => rate = F
    - Complexity = 1 round => rate = F/10
    - Complexity =¼ round (with 32 bits) => rate = F/40
- **Complexity ¹ latency**
  - Example with AES-128
    - Complexity = 10 rounds (unrolled with pipeline) => latency = 10/F
    - Complexity = 1 round => latency = 10/F
    - Complexity =¼ round (with 32 bits) => latency = 40/F

Energy

- **Very Important for lightweight cryptography**
- **Highly depends on complexity and glitches**

Table 2: Energy consumption per byte for software implementations.

Software Implementations			
Primitive	**Type**	**Platform**	E_n **(nJ/byte)**
Chaskey fast [13]	MAC	ARM M0/STM32F030R8	21.4
Chaskey compact [13]	MAC	ARM M0/STM32F030R8	19.8
Speck 64 bit block [14]	block cipher	ATtiny45	214
Speck 128 bit block [14]	block cipher	ATtiny45	252
Simon 64 bit block [14]	block cipher	ATtiny45	394
Simon 128 bit block [14]	block cipher	ATtiny45	604
AES-128 fast [15]	block cipher	AT90USB162	1,031
AES-128 compact [15]	block cipher	AT90USB646	1,114
DESXL [16]	block cipher	ATmega128	8,830
PRESENT-80 [16]	block cipher	ATmega128	11,099
Lesamnta-LW [17]	hash	8 bit Renesas H8	14,948
D-QUARK [18]	hash	ATtiny45	39,919
PHOTON-160 [18]	hash	ATtiny45	43,560
SPONGENT-160 [18]	hash	ATtiny45	75,050

P; Conr, P. Schaumont :"the role of energy in the lightweight cryptographic profile", NIST document

Energy in HW implementation

Table 4: Absolute energy consumption per byte for block cipher hardware implementations from [27, 28].

Block Cipher Hardware Implementations	
STM 90 nm, 10 MHz [27]	
Block Cipher	E_n **(pJ/byte)**
Midori-128	11.7
PRINCE	18.1
NOEKEON	21.1
PRESENT	21.5
AES	21.9
SIMON 128/128	41.5
UMC 0.130 µm, 100 KHz [28]	
Block Cipher	E_n **(pJ/byte)**
KLEIN-parallel	105.9
PRINCE	170.4
PRESENT	189.5
LED	477.6
CLEFIA	566.2
KATAN-64	793.7
AES	389.0 - 2315.8

More than 1000 times lower than SW

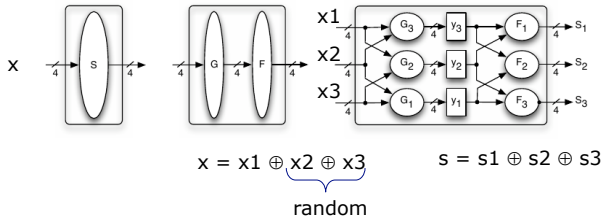
P; Conr, P. Schaumont :"the role of energy in the lightweight cryptographic profile", NIST document

Security

- **Cryptanalysis**
  - Depends only on the algorithm
  - More key bits => more rounds => less throughput
- **Side Channel attack**
  - Protections by hiding or masking => extra complexity, at least x2 in HW, much more in SW
- **Fault Injection attack**
  - Protection by redundancy
    - Spatial => complexity at least x2
    - Temporal => performance at least /2

Example of masking protection: Threshold Implementation*

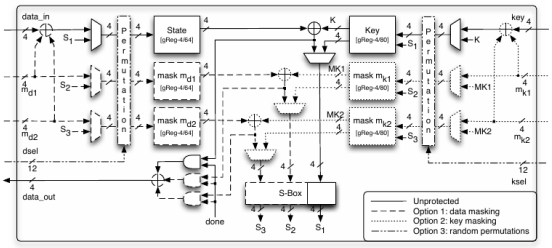
- **Proven protection against 1st order SCA**
  - Use of 3 shares



*Nikova, S., Rechberger, C., & Rijmen, V. (2006, December). Threshold implementations against side-channel attacks and glitches. In *International Conference on Information and Communications Security* (pp. 529-545). Springer Berlin Heidelberg.

Example of masking protection: Threshold Implementation*

- **Overall architecture**



TI results

Profile	Sharing Data [Y/N]	Sharing Key [Y/N]	Rand. Perm. [Y/N]	Cycles total [clk]	Cycles rel. [%]	Current total [μA]	Current rel. [%]	Area total [GE]	Area rel. [%]
1	N	N	N	547	100	1.34	100	1,111	100
2	Y	N	N	547	100	2.86	213	2,282	205
3	Y	N	Y	547	100	3.10	231	2,417	218
4	Y	Y	N	578	106	4.23	316	3,322	299
5	Y	Y	Y	578	106	5.02	375	3,582	322

Table 4. Breakdown of the post-synthesis implementation results of different architectures of a serialized PRESENT-80. P stands for Profile.

P	m_{d1} / m_{d2} %	m_{d1} / m_{d2} GE	m_{k1} / m_{k2} %	m_{k1} / m_{k2} GE	Rand. Perm. %	Rand. Perm. GE	S-box %	S-box GE	FSM %	FSM GE	State %	State GE	Key %	Key GE	other %	other GE	Sum GE	rel. %
1	0	0	0	0	0	0	3	32	13	145	35	389	45	498	4	47	1,111	100
2	34	778	0	0	0	0	17	387	6	146	17	389	22	498	3	75	2,282	205
3	32	778	0	0	5	121	16	387	6	146	16	389	21	498	4	98	2,417	218
4	22	778	20	970	0	0	11	355	5	156	12	389	15	498	5	176	3,322	299
5	22	778	27	970	7	243	10	355	4	155	11	389	14	498	5	194	3,582	322

Complexity x3

Poschmann, A., Moradi, A., Khoo, K., Lim, C. W., Wang, H., & Ling, S. (2011). Side-channel resistant crypto for less than 2,300 GE. *Journal of Cryptology*, *24*(2), 322-345.

Conclusion

- **The implementation is more important as the algorithm itself, to meet all the properties of:**
  - Security
  - Performance
  - low energy
  - complexity
- **Especially for lightweight cryptography**
- **HW is always better than SW**

True Random Number Generation
TRNG

Jean-Luc Danger
December 2019

Outline

RNG Applications

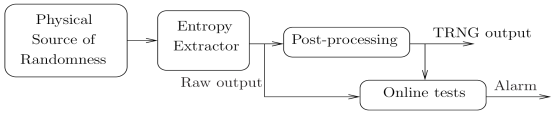
- **Cryptography**
  - Key generation
    - Symmetric/asymetric crypto
    - Stream ciphering (if deterministic RNG)
  - Initialization vector
    - For cryptographic operating modes
  - Authentication
    - Challenges
    - Nonce
  - Protection
    - Masks to thwart Side-Channel Attacks
- **System validation**
  - Monte-Carlo simulation
  - Statistical analysis
  - Channel emulation
- **Games, gambling**

RNG types

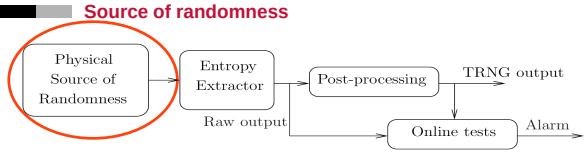
- **Pseudo Random number (PRNG)**
  - Deterministic
  - Fast
  - Periodic (based generally on LFSRs)
  - Adapted for stream ciphering
- **True Random Number generator (TRNG)**
  - Unpredictable
  - Use physical sources
- **Hybrid Random Number generator (HRNG)**
  - TRNG used as a seed of PRNG

TRNG Architecture

- **2 main blocks**
  - Entropy Source
  - Entropy extractor
- **2 optional blocks**
  - Postprocessing
  - Embedded Tests



Source of randomness



Two types of randomness source:

- Non Physical noise
  - Keyboard and mouse activity, Processor load, network data rate,…
- Physical noise
  - Noise in electronics circuits
  - Others: desintegration of radioactive atomic kernel,…

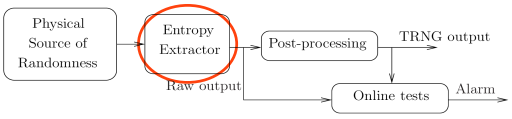
Noise in electronics circuit

- **Noise = Sum of different phenomenon:**
  - Thermal noise
  - 1/F noise
  - Shot noise
  - Popcorn noise
  - Crosstalk
  - Interference

Can be source of attacks !

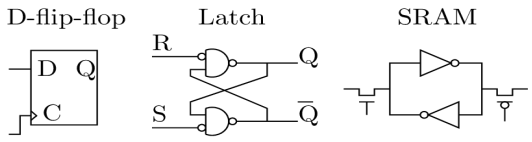
Randomness extraction

- **To sample the noise at source level**
  - Use of a sample/hold circuit, or memory element



Randomness extraction

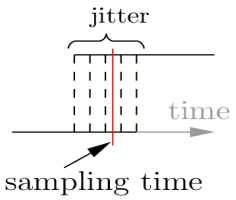
- **Memory elements**



- **Noise Axis**
  - Horizontal: **phase** noise (frequency) or **jitter** (time)
  - Vertical: **amplitude** noise

Phase noise

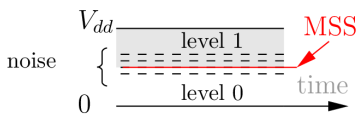
- **Combination of random jitter and deterministic jitter (unwanted)**



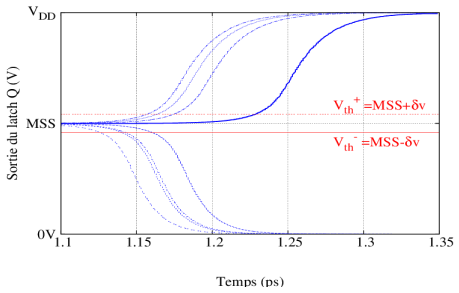
Amplitude noise

Also called vertical noise

The signal is at the boundary of 0 and 1 level, around the "**metastable** state"
MSS (~Vdd/2)



Metastability phenomenon



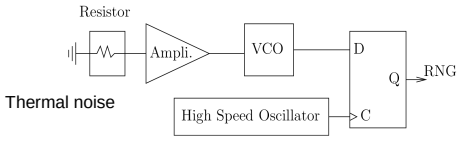
TRNG types

- **Jitter-based TRNG**
  - One jittery clock samples another jittery clock
- **Metastability-based TRNG**
  - The bi-stable is placed in the MSS state, its output depends on the vertical noise
- **Mixed TRNG**
  - Exploits both phenomemon

Outline

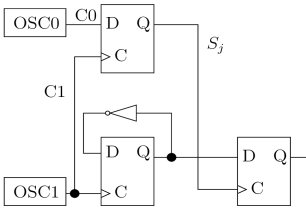
- **Introduction to TRNG**
- **Architectures**
- **Randomness**
- **Security**
- **Conclusion**

Two ring oscillators
Intel first generation



Benjamin Jun and Paul Kocher. The Intel Random Number Generator, 1999. http://www.cryptography.com/intelRNG.pdf.

Two-Ring oscillator TRNG

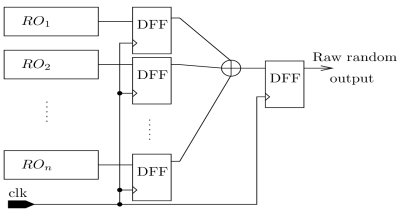


•There is a race (which depends on the jitter) between the rising edges:
  ✓Sj = 1 if rising_edge(C0) is ahead of rising_edge(C1), else 0

•RNG = LSB of the number of rising_edge(C1) between two rising_edge(C0)

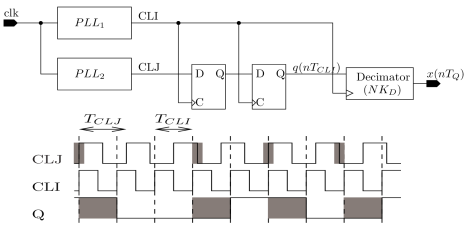
Paul Kohlbrenner and Kris Gaj. An embedded true random number generator for FPGAs. In Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays, 2004.

Multiple ROs TRNG



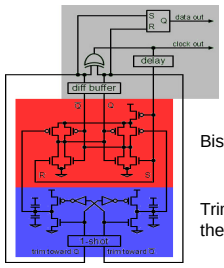
Knut Wold and Chik How Tan. Analysis and enhancement of random number generator in fpga based on oscillator rings. In ReConFig, pages 385–390, 2008.

PLL-based TRNG



Viktor Fischer, Milos Drutarovský, Martin Simka, and Nathalie Bochard. High performance true random number generator in altera stratix fplds. In Field Programmable Logic and Application, volume 3203 of Lecture Notes in Computer Science, pages 555–564. Springer 2004.

Intel Ivy bridge TRNG



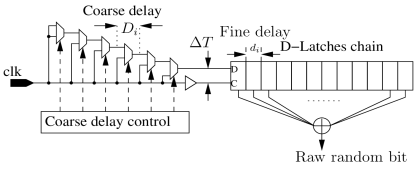
Bistable cell

Trimming circuit to keep the metastable state

Mike Hamburg, Paul Kocher, and Mark E. Marson. Analysis of intel's ivy bridge digital random number generator, March, 12 2012.

Open Loop TRNG

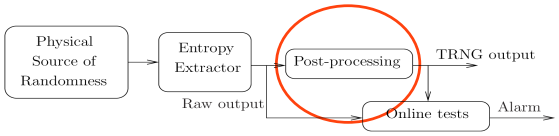
- **A chain of latches with D = C**



Danger, J. L., Guilley, S., & Hoogvorst, P. (2009). High speed true random number generator based on open loop structures in FPGAs. *Microelectronics journal*, *40*(11), 1650-1656.

Postprocessing

- **Mandatory process:**
  - The source of entropy is biased by the environment
  - The extraction block is not efficient enough
  - The samples are correlated



Post processing for better entropy

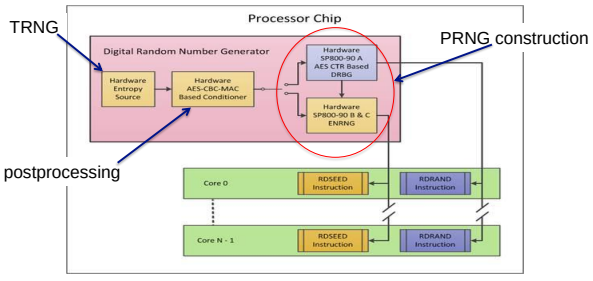
- **Goal: To increase the entropy**
  - by reducing the Bit rate
  - Or increasing data compression
- **Classical methods:**
  - XOR corrector
    - N bits are XORed to get one output bit
    - Constant Bit rate reduction (N)
    - XOR construction proposed by M. Dichtl*
  - Von Neumann corrector
    - 2 identical bits: freeze
    - 2 different bits: 1 if "10" , 0 if "01"
    - Variable Bit rate reduction (at least 4)

* Dichtl, M. (2007, January). Bad and good ways of post-processing biased physical random numbers. In Fast Software Encryption (pp. 137-152). Springer Berlin Heidelberg.

Cryptographic Postprocessing

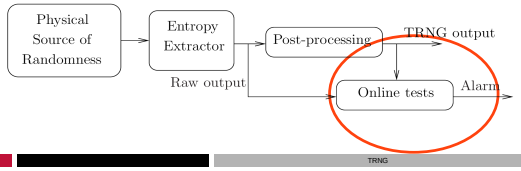
- **Goal: To increase unpredictibility**
  - Exploits non-linearity properties
  - The entropy is the same
- **Hash function**
  - Unpredictability ensures by One-way function
- **Symmetric cryptography cipher**
  - Could be the same as the crypto block.

Intel TRNG postprocessing



On-Line tests

- **Embedded tests = health tests**
  - To ensure dynamically that the output bits have good randomness property :
    - Under environmental variations
    - Under attacks



Example of on-line tests

- **Custom sanity check**

Intel's Ivy bridge

Bit pattern	Allowable number of occurrences per 256-bit sample
1	109 < n < 165
01	46 < n < 84
010	8 < n < 58
0110	2 < n < 35
101	8 < n < 58
1001	2 < n < 35

- **Use off-line statistical tests (described in next section)**
  - Can be costly => partial tests
- **2nd order effect:**
  - the noise generated by the test impacts the TRNG which passes the test more easily

Outline

- **Introduction to TRNG**
- **Architectures**
- **Randomness**
- **Security**
- **Conclusion**

Security evaluation requirements

- **In theory:**
  - Randomness: Entropy
    - Shannon $H(x) = -\sum p(x) \log_2(p(x))$
    - Min-Entropy $H(x) = -\log_2 \max(p_i(x))$

x is a bit vector

  - Unpredictability: Conditionnal Entropy
    - Stochastic process
      - H(state_{i} | state_{i-1}) ?
- **In practice**
  - Assesment Methods
    - Statistical tests
    - Build a stochastic model

Statistical tests

- **Off-line tests**
  - FIPS140-2
  - NIST SP800-22
  - DIEHARD
  - AIS31
- **On-line tests**
  - To detect in real time
    - failures abnormal behaviour
    - Health tests
  - partial statistical tests

FIPS140-2 tests

- **4 tests applied on 20000-bit sequences**
- **Can be embedded in the circuit:**
  - Monobit (pr(bit=0))
  - Poker : uniform distribution of 4-bit groups
  - Runs : check the number of run sequences of '0' and '1' of length between 1 and 6
  - Long runs : no run of '0' or '1' with a length equal or greater than 26 should occur

NIST FIPS (Federal Information Processing Standards). Security Requirements for Cryptographic Modules publication 140-2, May 25 2001. http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

NIST test suite SP800-22

- **List of 15 demanding tests, some requiring 1Mbit of data or up to 1Gbit**
- **Applies to random number generators for cryptographic applications**
  - Usually unfit for testing raw physical entropy sources
  - Fit for testing post-processed or "ready-to-use" outputs

Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, and San Vo. A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications, april 2010.

NIST test suite SP800-22

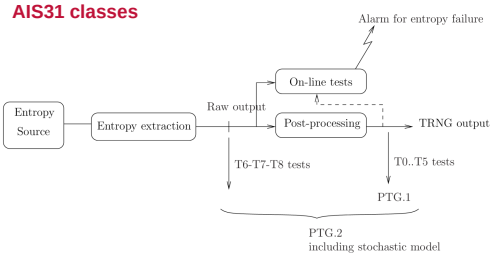
- **NIST test suite (SP800-22)**
  - Among them:
    - Similar to what was first included in FIPS140-2 +
    - Random Binary Matrix Rank : check for linear dependency among fixed substrings
    - Check for periodicity (DFT, template matching tests (2 variants))
    - Compression test (Maurer's "universal statistical" test, similar to a Lempel-Ziv compression, LFSR)
    - Random walk (Cumulative sum, random excursion tests (2 variants))

BSI AIS31

Bundesamt für Sicherheit in der Informationstechnik

- **Strongly common criteria oriented**
- **New approach: testing TRNG output as well as the randomness source**
  - 9 statistical tests
  - Recommends using on-line tests,
    - Entropy source failure tests
    - Statistical defect tests on the raw random sequence
    - Statistical defect tests on the post-processed sequence
  - Defines TRNG classes (PTG.x)

AIS31 classes



•PTG.1 represents the lower-end class and PTG.2 and PTG.3 the higher-end classes
•PTG.2 adds minimum entropy requirements for post-processed bits and introduces raw data online tests and entropy source stochastic model
•PTG.3 requires cryptographic post-processing (class DRG.3 or DRG.4)

AIS31 tests

- **Test T0**
  - disjointness test
    - 2^{16} 48-bit blocks must be different
    - Can be performed on line
    - Rejection probability 10^{-17}
- **Tests T1-T4**
  - Idem FIPS140-2 with Rejection probability 10^{-6}
- **Test T5:autocorrelation test**
  - Performed on 10000 bits Between the sequence of 5000 bits and the 5000 bits shifted sequence
- **Tests T6-8**
  - Dedicated to raw sequences:
    - Multinomial distribution test
    - Homogeneity test
    - Coron's entropy test

AIS31 Randomness Model

- **AIS-31 Stochastic process**
  - The conditionnal entropy define the dependence between the last state Rn and the previous states
    $$H(R_n | R_1, \ldots R_{n-1}) = -\sum p(R_n | R_1, \ldots R_{n-1}) \log_2 p(R_n | R_1, \ldots R_{n-1})$$
  - The process should be ideally stationnary

NIST tests SP-800 90A, 90B and 90C

- **SP-800 90A considers DRNG**
- **SP-800 90B considers entropy source design and validation**
  - SP-800 90B defines requirements similar to BSI's AIS31
  - Two categories defined:
    - Entropy source
    - Full entropy source
  - Minimal online health tests defined
    - Lightweight runs detection test (repetition count test)
    - Lightweight uniformity test (adaptive proportion test)
- **SP-800 90C considers design of hybrid RNGs using SP-800 90B + SP-800 90A**

ISO standard

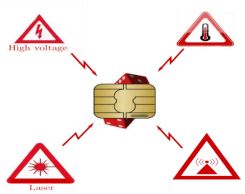
- **ISO /IEC 20543**
  - Test and analysis methods for random bit generators
    - Define RNG classes (example: deterministic, non-deterministic, hybrid) within ISO/IEC 19790 and 15408
    - Define evaluation methods for the various RNG classes

Outline

- **Introduction to TRNG**
- **Architectures**
- **Randomness**
- **Security**
- **Conclusion**

TRNG threats

- **Malevolent Actions**
  - To decrease the entropy by introducing biases
  - To bypass on line tests
  - To lock the TRNG
  - To locate the TRNG
    - To attack it afterward

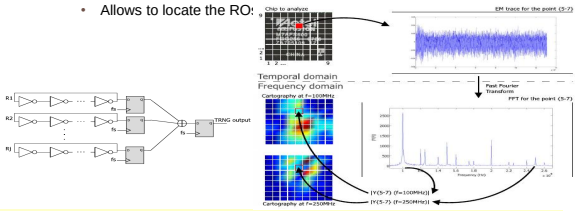


TRNG attack types

- **Passive attacks**
  - Observation of the TRNG activity
    - Frequency of RO TRNG
    - Register loads
  - Allows the attacker **to locate** the TRNG
- **Reverse Engineering**
  - Allows the attacker **to locate** the TRNG
- **Active attacks**
  - **Bias** generated by fault injection
  - **Force** the TRNG output or the on-line test alarm
- **Trojans**
  - Bias generated at design or manufacturing stage

Passive attacks

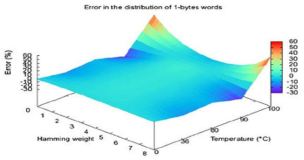
- **Observing the EM field in RO TRNG**
  - Allows to locate the RO



Bayon, P., Bossuet, L., Aubert, A., & Fischer, V. (2013, May). Electromagnetic analysis on ring oscillator-based true random number generators. In Circuits and Systems (ISCAS),

Active attacks on RO TRNG

- **Different sources:**
  - T°C and laser: very sensitive
  - Xray and Radioactivity: not sensitive



Soucarros, M., Clédière, J., Dumas, C., & Elbaz-Vincent, P. (2013). Fault analysis and evaluation of a true random number generator embedded in a processor. Journal of Electronic Testing, 29(3), 367-381.

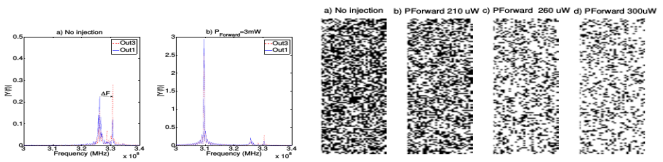
Active attacks on RO TRNG

- **Frequency coupling by the Power Supply**
  - Performed successfully on logic gates, microcontroller and Smartcard (EMV)

Markettos, A. T., & Moore, S. W. (2009). The frequency injection attack on ring-oscillator-based true random number generators. In Cryptographic Hardware and Embedded Systems-CHES 2009 (pp. 317-331). Springer Berlin Heidelberg.

Active attacks on RO TRNG

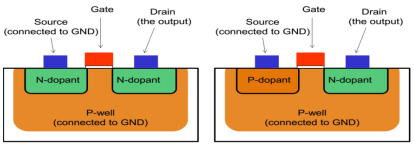
- **Frequency coupling by EM field**



Bayon, P., Bossuet, L., Aubert, A., Fischer, V., Poucheret, F., Robisson, B., & Maurine, P. (2012). Contactless electromagnetic active attack on ring oscillator based true random number generator. In Constructive Side-Channel Analysis and Secure Design (pp. 151-166). Springer Berlin Heidelberg.

Attack by Hardware Trojan Horse

- **The Dopant of transitors is changed**
  - Extremely difficult to detect optically
  - Can drastically reduced the TRNG entropy



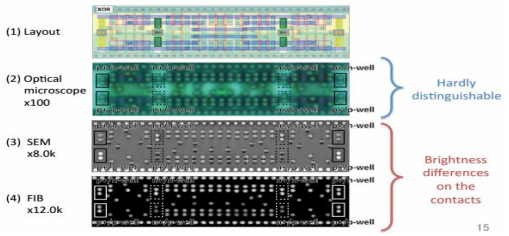
NMOS transistor

Dopant Change => NMOS transistor with floating output

Becker, G. T., Regazzoni, F., Paar, C., & Burleson, W. P. (2013). Stealthy dopant-level hardware trojans. In *Cryptographic Hardware and Embedded Systems-CHES 2013* (pp. 197-214). Springer Berlin Heidelberg.

Detection of Stealthy Trojan

- Stealthy dopant-level circuits are measurable



Sugawara, T., Suzuki, D., Fujii, R., Tawa, S., Hori, R., Shiozaki, M., & Fujino, T. (2014). Reversing Stealthy Dopant-Level Circuits. In *Cryptographic Hardware and Embedded Systems–CHES 2014* (pp. 112-126). Springer Berlin Heidelberg.

Outline

- **Introduction to TRNG**
- **Architectures**
- **Randomness**
- **Security**
- **Conclusion**

Conclusion 1/2

- **Necessity to assess Entropy and Unpredictability**
  - Statistical tests
  - Model
- **Necessity to embed Online test**
  - To detect abnormal behaviour
    - Environment
    - Attacks
  - Future ISO standards in preparation
- **Designing a TRNG is still a challenge**
  - Security
    - To pass statistical tests
    - To build a model
    - To be robust against attacks
  - Design challenges

Conclusion 2/2

- **Design challenges**
  - Type of randomness extraction
  - Type of Postprocessing
  - Complexity
  - Design automation
  - Bit rate
  - Testability (statistical and integrity tests)
  - Power consumption

=> A test chip is recommended !