FPGA-specific Physical Attacks and Efficient Countermeasures

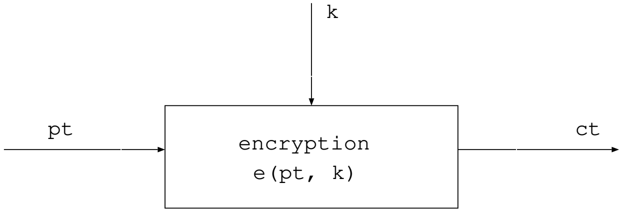
Francesco Regazzoni

Contents

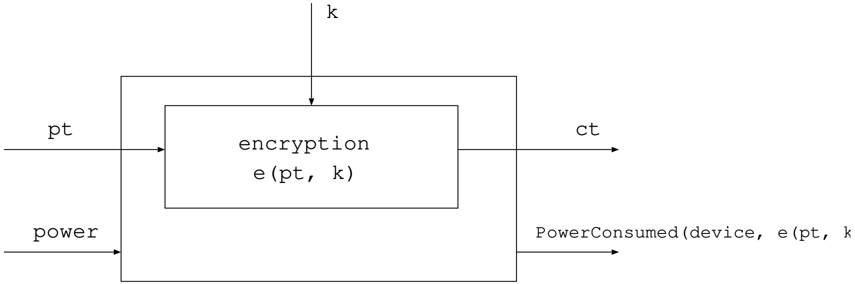
What are Physical Attacks

- Physical attacks recover secrets by exploiting the implementation

Why Physical Attacks Exist?



Why Physical Attacks Exist?



Why Physical Security is so Important Today?

Long Time Ago | Past | Present

Why Physical Security is so Important Today?

Long Time Ago | Past | Present

Mainframes | Personal Computer | Pervasive

Physical Attacks: the Weakest Point

Active	Passive
Fault Injection	Power Analysis
	Timing Analysis
	Electromagnetic attacks

Contents

Outline

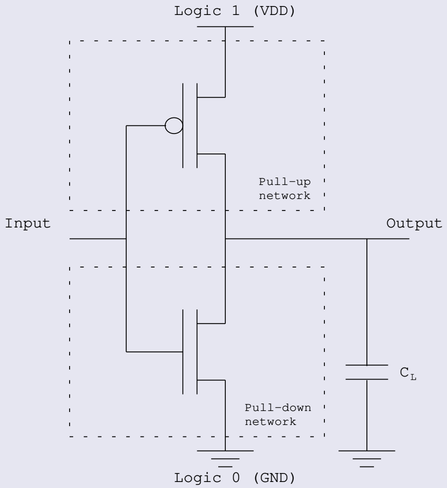
Power Analysis Attacks

Power Analysis Attacks exploit the relation between the power consumed and the processed data.

Paul Kocher, Joshua Jaffe, and Benjamin Jun, “**Differential Power Analysis**”, in Proceedings of *Advances in Cryptology-CRYPTO’99*, Santa Barbara, California, USA, August 15-19, 1999. (Cited by 9848)

- Cheap
- Powerful

Why Power Analysis Attacks Exist?



Most Common Power Analysis Attacks

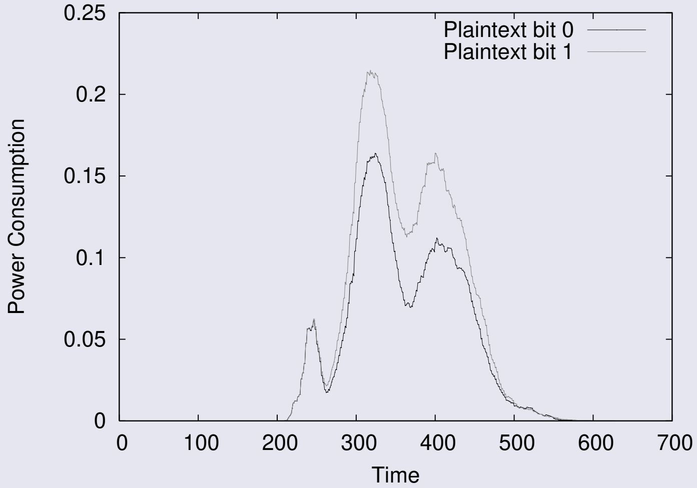
- Simple Power Analysis
- Differential Power Analysis

Simple Power Analysis (SPA)

- **Goals**: The adversary attempt to recovery the secret key using a small set of power traces
- **Requirements**: Knowledge about the implementation

- Visual Inspection
- Template Attacks
- Collision Attacks

Visual Inspection



Differential Power Analysis (DPA)

- **Goals**: The adversary make hypotheses on smaller portion of the keys and verify it on the power traces
- **Requirements**: Knowledge about the implemented algorithm

Distinguishers

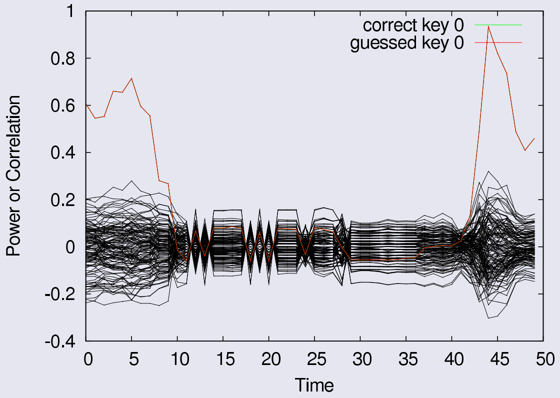
- Difference of means
- Correlation
- Multivariate statistic

Differential Power Analysis (DPA)

1. Select the target attack point
2. Encrypts (decrypts) a number of known plain-texts (cipher-texts) and measures the consumed power
3. Compute the hypothetical intermediate based on a key guess the known plain-text
4. Verify the guess over the power traces

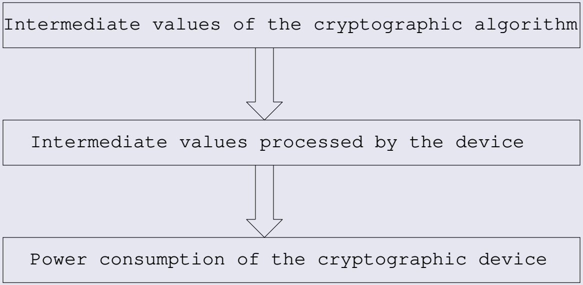
Example of Differential Power Attacks

Simulate whole embedded processor at SPICE



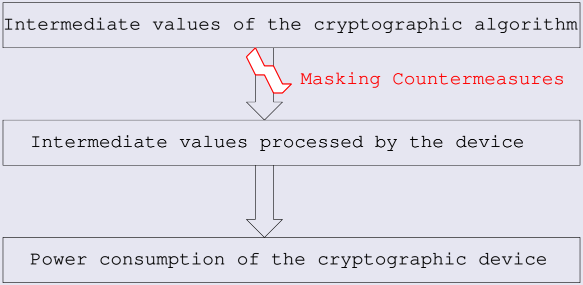
Countermeasures

Power consumption **independent** from processed key dependent data


Intermediate values of the cryptographic algorithm
Intermediate values processed by the device
Power consumption of the cryptographic device

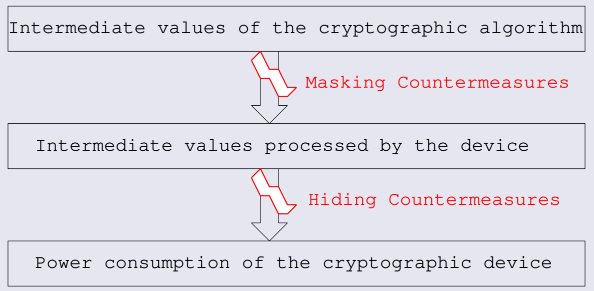
Countermeasures

Power consumption **independent** from processed key dependent data


Intermediate values of the cryptographic algorithm
Masking Countermeasures
Intermediate values processed by the device
Power consumption of the cryptographic device

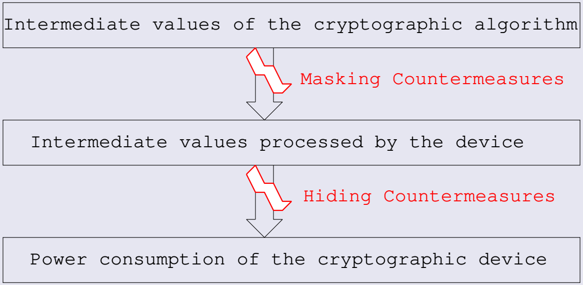
Countermeasures

Power consumption **independent** from processed key dependent data


Intermediate values of the cryptographic algorithm
Masking Countermeasures
Intermediate values processed by the device
Hiding Countermeasures
Power consumption of the cryptographic device

Countermeasures

Power consumption **independent** from processed key dependent data



Intermediate values of the cryptographic algorithm
Masking Countermeasures
Intermediate values processed by the device
Hiding Countermeasures
Power consumption of the cryptographic device

They can be implemented in **Software** or in **Hardware**

Masking

- Decreases the correlation applying a random mask to the intermediate values
- $x_m = x \oplus m$ ($\oplus$ mask operation, m mask, x the secret key value, or the input data value, or both of them)
- The algorithm is executed using x_m and m
- At the end the mask is removed

Hiding

- Can be mitigated by proper adding random instructions
- Dedicated Logic Styles (WDDL, ..)

EM Attacks

Outline

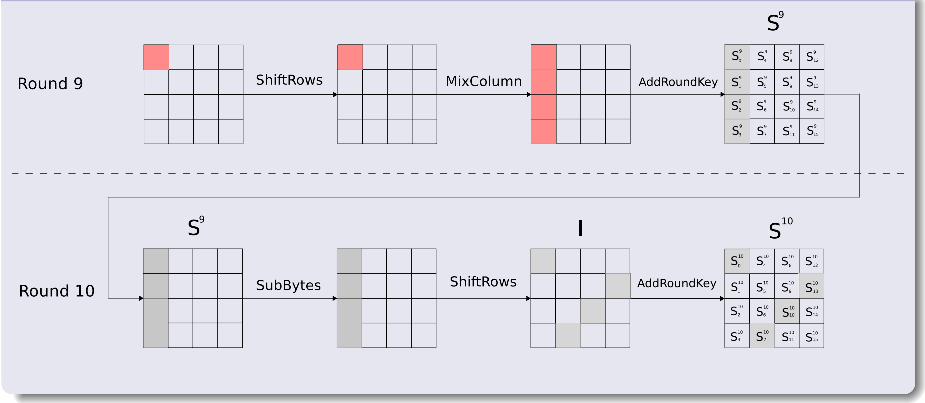
Contents

Fault Attacks

- **Goals**: The adversary attempt to recovery the secret key exploiting the relation between a faulty output and the correct one
- **Requirements**: Fault in the right position

- Laser or equivalent
- Control of the power supply

Key Recovery



- Single byte fault per column before the last MixColumn
- Single byte fault in the earlier round

Skip Check

- Inject a fault to generate a number not random
- Inject a fault to skip a security check
- ...

Countermeasures

- Add space redundancy
- Add time redundancy

Contents

Hardware Trojans

A **deliberate** and **malicious** change to an IC that adds or removes functionality or reduces reliability

Effects of Trojans

- Denial of Service
- Modify Data
- Leak Information

Trojan Activation

- Input Activated
- Time Bombs

Input Activated

- Sequential
- Single shot

Time Bombs

- Counter Trigger
- Random Trigger

Who can place a Trojan?

Detection Techniques

- Functional Testing
- Formal Verification
- Trojan Detection Circuit
- Side Channel
- Optical Inspection

Contents

SASEBO SAKURA

Reconfigurable Devices and Physical Attacks

- Measure directly

- Implement directly the countermeasure

- Less Freedom
- Tools more “closed”

Specific Block

- LUT size 6 bits
- 4 6-bit LUTs into a Slice

Target FPGA: Xilinx Virtex-5

- Larger and more complex devices
- Embed multipliers, RAM memories, full processors
- Slice:
  - 4 flip-flops
  - 4 6-input LUTs
  - 2 multiplexers (F7MUX and F8MUX)
- Slices can be configured as distributed RAMs
- Very suitable for mapping 8-bit input Look-up-tables

Sbox of Oswald and Schramm

- S-box: inversion over $GF(2^8)$ and affine mapping (easy to mask):
  - Transform the masked input to the composite field $GF(2^4) \times GF(2^4)$
  - invert it there efficiently
  - transform it back to the $GF(2^8)$

Sbox of Oswald and Schramm

- S-box: inversion over $GF(2^8)$ and affine mapping (easy to mask):
  - Transform the masked input to the composite field $GF(2^4) \times GF(2^4)$
  - invert it there efficiently
  - transform it back to the $GF(2^8)$

- Oswald and Schramm approach for software:
  - perform the inversion in $GF(2^4)$ combining XOR operations with four pre-computed tables: T_{d_1}, T_{d_2}, T_m and T'_{inv}.
  - Transform the result back to $GF(2^8)$ with two additional tables: T'_{map} (from $GF(2^8)$ to $GF(2^4) \times GF(2^4)$) and $T'_{map^{-1}}$ (from $GF(2^4) \times GF(2^4)$ to $GF(2^8)$)
  - The affine transformation is integrated with the isomorphic mapping

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m: input two elements of $GF(2^4)$, output an element of $GF(2^4)$

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ $\checkmark$
- T_{d_2}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ $\checkmark$
- T_m: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ $\checkmark$
- T'_{inv}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{map}: input an element of $GF(2^8)$, output an element of $GF(2^4)$

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{map}: input an element of $GF(2^8)$, output an element of $GF(2^4)$ ✓

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{map}: input an element of $GF(2^8)$, output an element of $GF(2^4)$ ✓
- $T'_{map^{-1}}$: input two elements of $GF(2^4)$, output an element of $GF(2^4)$

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{map}: input an element of $GF(2^8)$, output an element of $GF(2^4)$ ✓
- $T'_{map^{-1}}$: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓

Why it is suitable?

- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{map}: input an element of $GF(2^8)$, output an element of $GF(2^4)$ ✓
- $T'_{map^{-1}}$: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓

- All these tables have input size of 8 bits: fit **perfectly** our target FPGA

Protected Design Flow

Reverse Engineering the bit Stream

Hardware Trojans On FPGA

Reverse Engineering Bit Stream + Trojan

Why Physical Security is so Important Today?

Long Time Ago | Past | Present

Mainframes | Personal Computer | Pervasive

Why Physical Security is so Important Today?

Long Time Ago | Past | Present

Mainframes | Personal Computer | Pervasive

SHARING!!!

Remote Attacks on FPGAs

Countermeasures

cFA

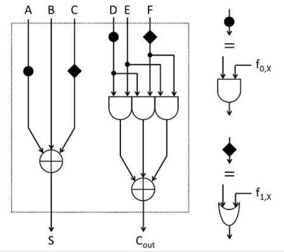
Symmetric-key cryptography

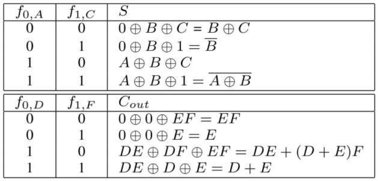
- bit permutation
- rotation
- addition modulo 2^n (in ARX-based ciphers)
- $S = A \oplus B \oplus C_{in} C_{out} = AB + (A + B)C_{in} = AB \oplus AC_{in} \oplus BC_{in}$
- addition modulo 2, i.e. exclusive OR (XOR)
- substitution box (S-box)
- quadratic functions (for threshold implementations) $f(x, y, z, w) = a_0 \oplus a_1 x \oplus a_2 y \oplus a_3 z \oplus a_4 w \oplus a_{12} xy \oplus a_{13} xz \oplus a_{14} xw \oplus a_{23} yz \oplus a_{24} yw \oplus a34zw$

N. Mentens, E. Charbon, and F. Regazzoni, "Rethinking Secure FPGAs: Towards a Cryptography-friendly Configurable Cell Architecture and its Automated Design Flow", FCCM 2018

cFA

- Fine-grained reconfigurable architecture
- Matrix of configurable Full Adder (cFA) cells
- One cFA (6 inputs and 2 outputs) can be programmed to 8 functions
- 8 functions are in standard cell libraries: re-use ASIC synthesis tools



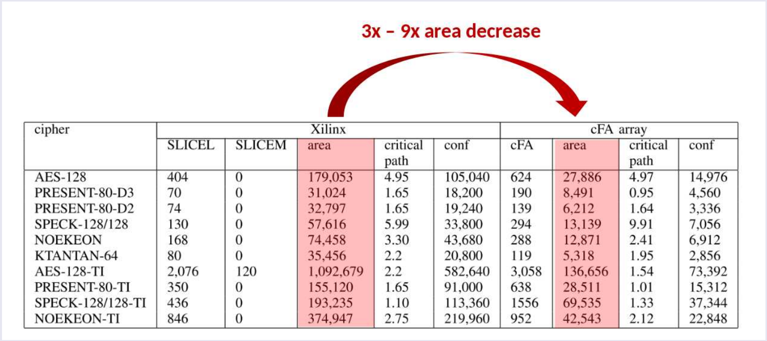


$f_{0,A}$	$f_{1,C}$	S
0	0	$0 \oplus B \oplus C = B \oplus C$
0	1	$0 \oplus B \oplus 1 = \overline{B}$
1	0	$A \oplus B \oplus C$
1	1	$A \oplus B \oplus 1 = \overline{A \oplus B}$

$f_{0,D}$	$f_{1,F}$	C_{out}
0	0	$0 \oplus 0 \oplus EF = EF$
0	1	$0 \oplus 0 \oplus E = E$
1	0	$DE \oplus DF \oplus EF = DE + (D+E)F$
1	1	$DE \oplus D \oplus E = D + E$

N. Mentens, E. Charbon, and F. Regazzoni, "Rethinking Secure FPGAs: Towards a Cryptography-friendly Configurable Cell Architecture and its Automated Design Flow", FCCM 2018

cFA


3x – 9x area decrease

cipher	Xilinx					cFA array			
	SLICEL	SLICEM	area	critical path	conf	cFA	area	critical path	conf
AES-128	404	0	179,053	4.95	105,040	624	27,886	4.97	14,976
PRESENT-80-D3	70	0	31,024	1.65	18,200	190	8,491	0.95	4,560
PRESENT-80-D2	74	0	32,797	1.65	19,240	139	6,212	1.64	3,336
SPECK-128/128	130	0	57,616	5.99	33,800	294	13,139	9.91	7,056
NOEKEON	168	0	74,458	3.30	43,680	288	12,871	2.41	6,912
KTANTAN-64	80	0	35,456	2.2	20,800	119	5,318	1.95	2,856
AES-128-TI	2,076	120	1,092,679	2.2	582,640	3,058	136,656	1.54	73,392
PRESENT-80-TI	350	0	155,120	1.65	91,000	638	28,511	1.01	15,312
SPECK-128/128-TI	436	0	193,235	1.10	113,360	1556	69,535	1.33	37,344
NOEKEON-TI	846	0	374,947	2.75	219,960	952	42,543	2.12	22,848

N. Mentens, E. Charbon, and F. Regazzoni, "Rethinking Secure FPGAs: Towards a Cryptography-friendly Configurable Cell Architecture and its Automated Design Flow", FCCM 2018

Acknowledgemnts

Partially funded by Horizon 2020 EU Research & Innovation programme under GA No 957269 (EVEREST Project)

Questions?

Thank you for your attention!

mail: f.regazzoni@uva.nl

mail: francesco.regazzoni@usi.ch