Automatic Application of Side Channel Countermeasures: History and Perspectives

Francesco Regazzoni

Contents

What Are Physical Attacks

$\|$

What Are Physical Attacks

||

Physical attacks recover secrets by exploiting the implementation

Types of Physical Attacks

Active	Passive
Fault Injection	Power Analysis
	Timing Analysis

Side Channels Are Used in Many Fields

- Pizza Delivery
- Energy Consumption
- Biology
- ...
- Cryptography

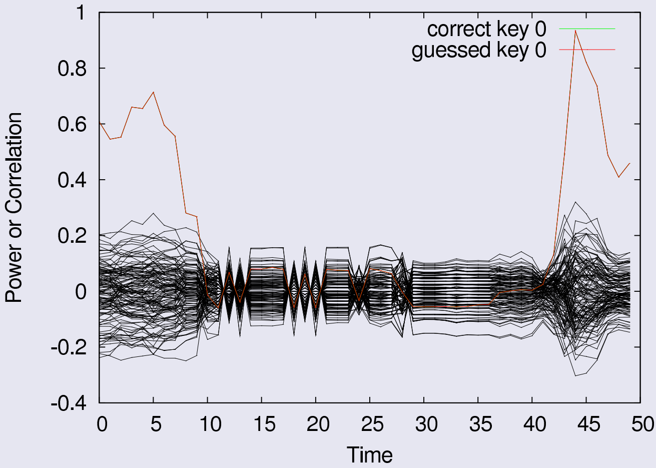
Differential Power Analysis (DPA)

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

Verify the hypotheses

- Difference of means
- Correlation
- Multivariate statistic

Example of Differential Power Attacks



Why Physical Security is so important?

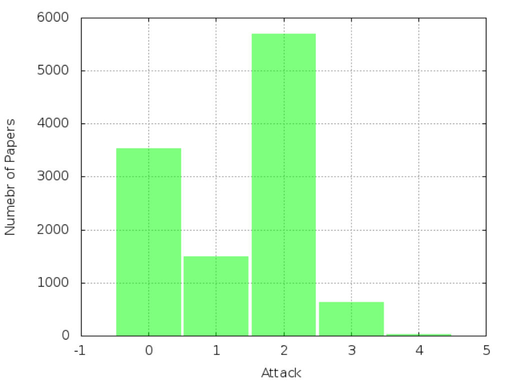
- IoT
- Cyber Physical Systems
- Implantable devices
- ...
- Shared resources on cloud!

Contents

Two Main Directions...

Countermeasures ∥ Better Attacks

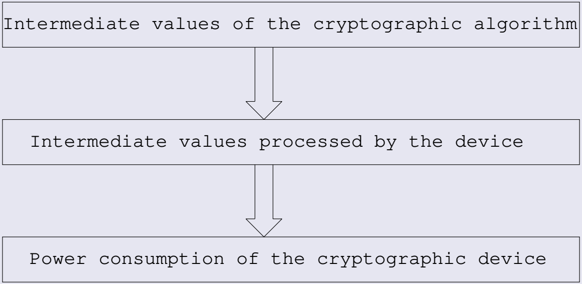
Research Activity per Attack (approx)



- 1996 Timing Attacks
- 1997 Fault Injection Attacks
- 1999 Power Analysis Attacks
- 2002 Electromagnetic Attacks
- 2012 Photon Emission

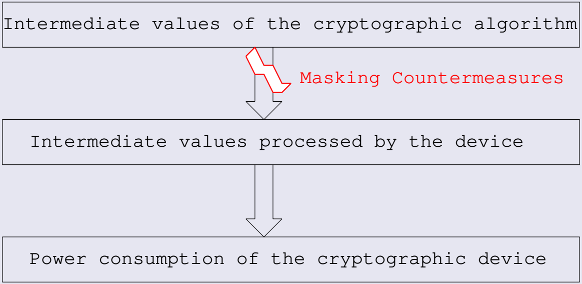
Countermeasures

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



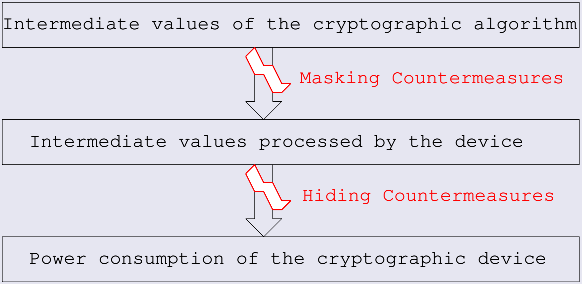
Countermeasures

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



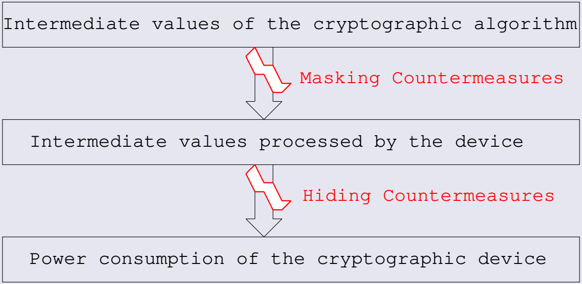
Countermeasures

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



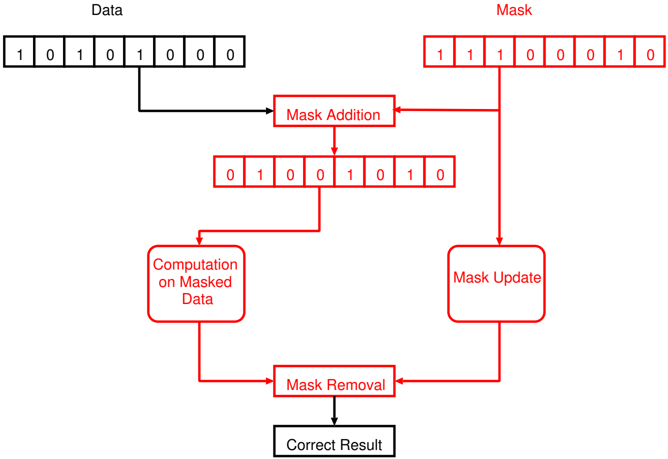
Countermeasures

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

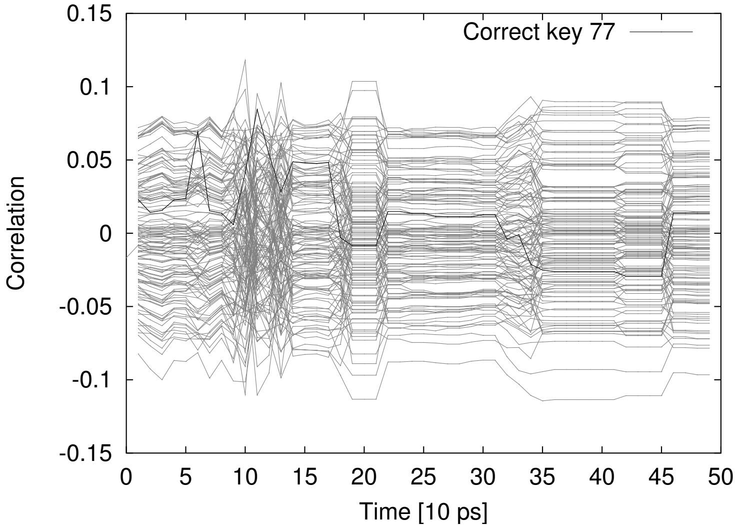


They can be implemented in **Software** or in **Hardware**

More Details on Masking



More Details on Hiding



Contents

From Idea to ASIC: the design flow....

"Surely the purpose of science is to ease human hardship"

Galileo, Bertolt Brecht

A bit of history

- 1948 Transistor
- Design done by hand
- 1970 Automated place and route
- 1980 Chip design with programming languages

A bit of history

- 1948 Transistor
- Design done by hand
- 1970 Automated place and route
- 1980 Chip design with programming languages

- Chip is most likely to function correctly
- Chip is easier to be verified
- Designer can handle more complex designs
- Birth of commercial EDA companies

First consideration....

- 1996 Timing Attacks

- 2023...

A bit late....

Why Automation....

- Security is very often considered at later stages of design
- Cost and Time to Market
- Possible Security pitfalls
- Handle the Complexity

… for security?

- Security is very often considered at later stages of design
- Cost and Time to Market
- Possible Security pitfalls
- Handle the Complexity

EXTRA CONSTRAINT

Use as much as possible “standard” design commodities!

A bit of history

- 1996 Physical attacks
- Countermeasures done by hand
- 2004 Secured synthesis and place and route
- 2009 Tool driven by a security variable

A bit of history

- 1996 Physical attacks
- Countermeasures done by hand
- 2004 Secured synthesis and place and route
- 2009 Tool driven by a security variable

Still only goals

- Chip would most likely to function securely
- Chip security would be easier to be verified
- Designer could handle more complex designs
- Birth of commercial EDA security companies (?)

Contents

Where are we?

Step One

INPUT:

- HDL Description
- Technological Library (area, timing, power)
- Synthetic Library (multipliers...)
- Constraints

OUTPUT:

- DPA resistant Gate Level Netlist
- Estimation of area, timing, power (!)
- Timing constraints

Automated Synthesis

WDDL:

- Build using standard gates
- For selected gates in the library, make the correspondent WDDL gate
- Synthesis, using existing tools (limiting the used gates)
- Replace the gates with the WDDL correspondent

CML:

- Design a new library from begin
- Characterize the library and generate all the needed files
- Synthesis using existing tools

Automated Place and Route

INPUT:

- DPA resistant Gate Level Netlist
- Technological Library
- Estimation of area, timing, power (!)
- Timing constraints
- Secure Place and Route Script

OUTPUT:

- DPA resistant fabrication file

Automated Place and Route

- Define a larger wire
- Place and route using the larger wire
- Edit the design file cutting the wires in two
- Careful for instance with T-shapes

- **Number of Samples** Easy but based on specific attack scenario
- **Success Rate** Based on specific attack scenario

$$\mathrm{Succ}^{K}_{\mathrm{attack}} = \Pr[f = 1]$$

- **Information Theory** Complex but independent from the attack scenario

$$\mathrm{H}[K|L] = -\sum_{k} \Pr[k] \cdot \sum_{x} \Pr[x] \int \Pr[l|k,x] \cdot \log_2 \Pr[k|l,x]\, dl.$$

Step Two

Towards Automatic Application of Countermeasures

Inputs:

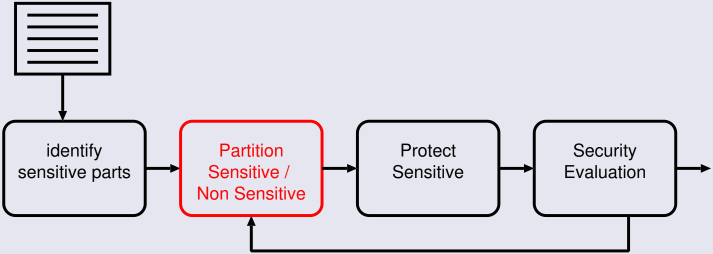
- Unprotected Algorithm
- Countermeasure

Output:

- Algorithm where the countermeasure is Applied

- Algorithm where the countermeasure is applied does NOT mean protected Algorithm

Putting all together



- Generate useful power traces?
- Measure the DPA resistance?
- Countermeasure and its design flow?
- Partition the algorithm?

Customizable Processors

```
// Calculate S-box (plaintext XOR key)
int PRESENT(int plaintext, int key) {
int result = 0; // initialize the result
plaintext = plaintext ^key; // perform the xor with the key
result = S[plaintext]; // perform the S-box
return result; }; // return the result
```

Customizable Processors

// *Calculate S-box (plaintext XOR key)*

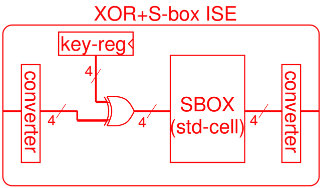
int PRESENT(int plaintext, int key) {

1 **int result = 0;** // *initialize the result*

2 plaintext = plaintext ^key; // perform the xor with the key

3 result = S[plaintext]; // perform the S-box

4 **return result; };** // *return the result*



// *Calculate S-box (plaintext XOR key)*

int PRESENT_XOR+S-box-ISE(int plaintex) {

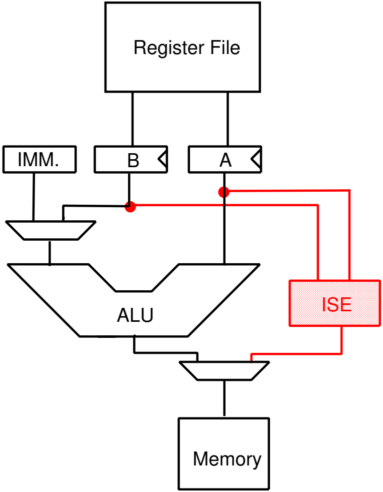
1 **int result = 0;** // *initialize the result*

// *instantiate the new instruction s-box(pt ^key)*

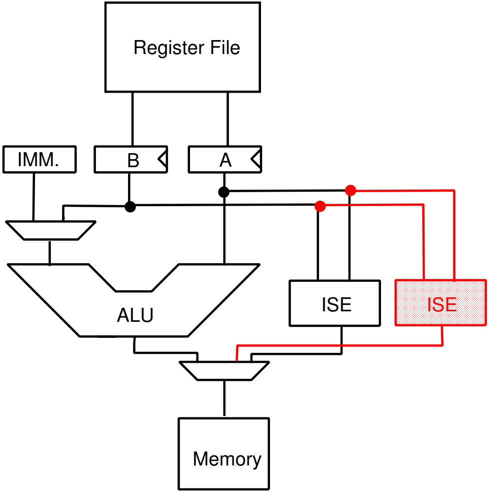
2 **Instr_1(plaintex, result);**

3 **return result; };** // *return the result*

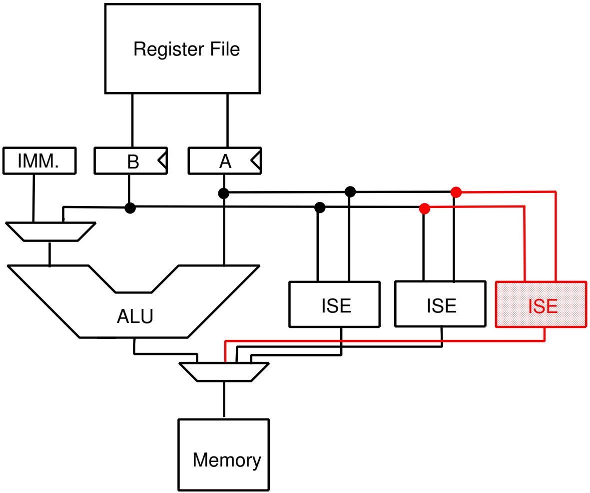
Protected / Non Protected CO-Design!



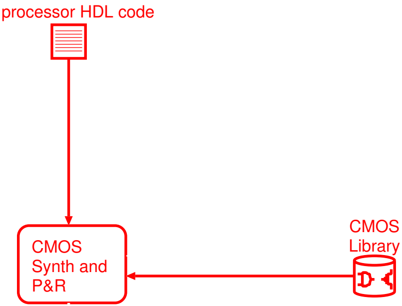
Protected / Non Protected CO-Design!



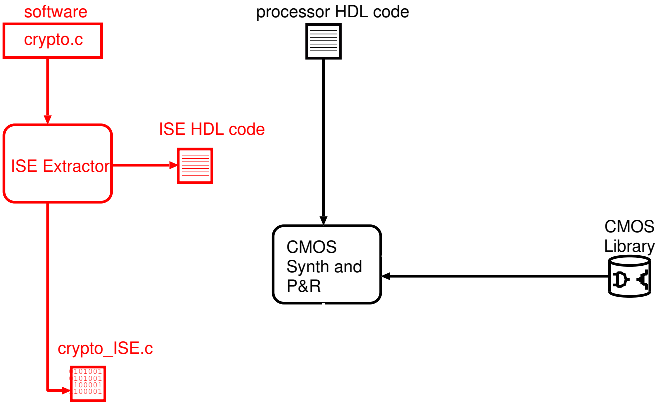
Protected / Non Protected CO-Design!

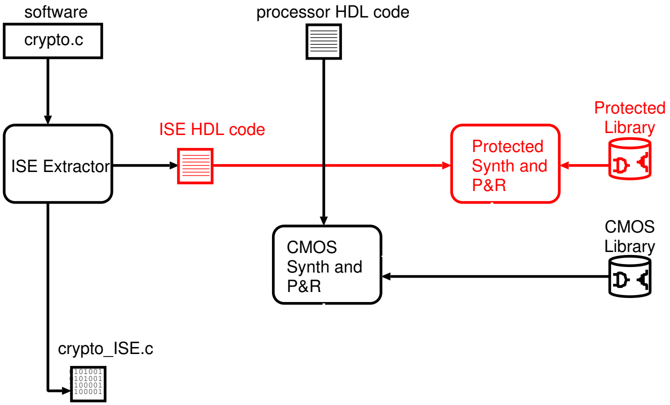


CMOS Design Flow

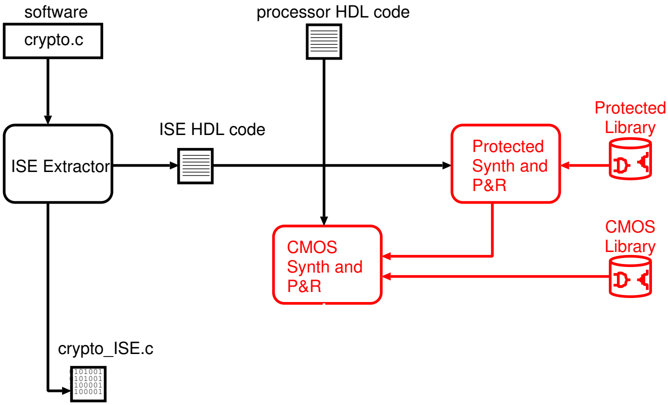


Processor Customization

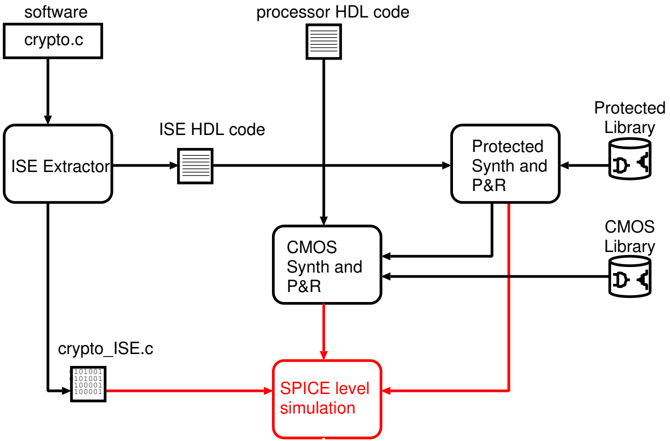



software
crypto.c
processor HDL code
ISE Extractor
ISE HDL code
Protected Synth and P&R
Protected Library
CMOS Synth and P&R
CMOS Library
crypto_ISE.c
101001
101001
100001
100001

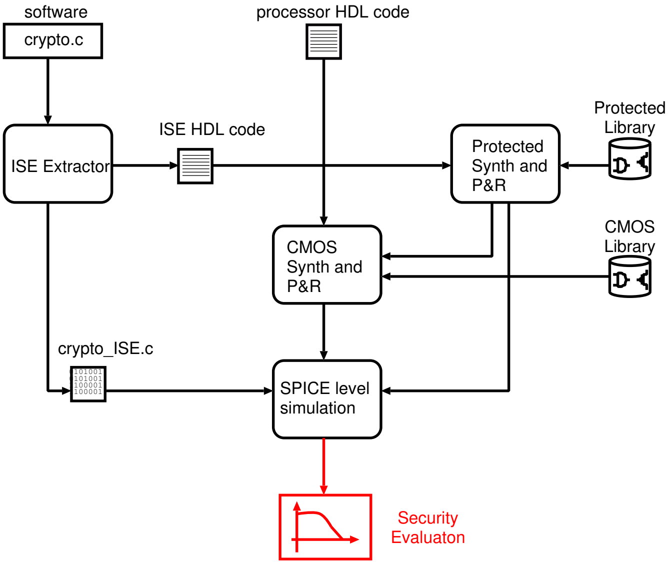
Hybrid Design Flow



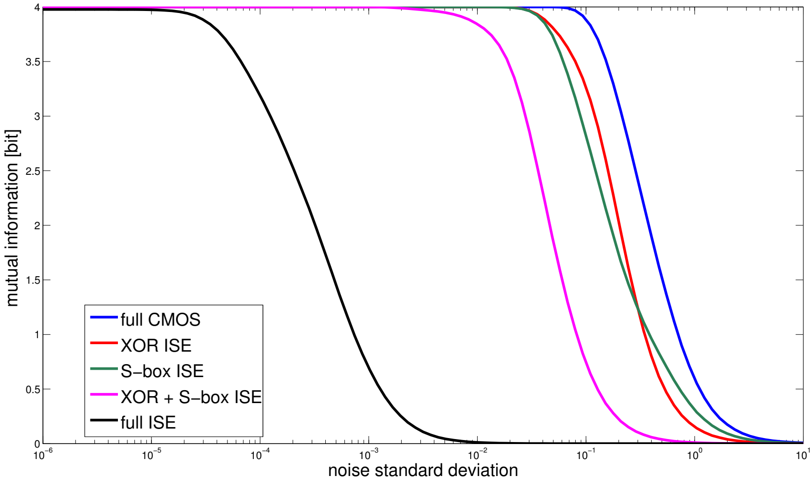
Simulation Environment



Design Evaluation



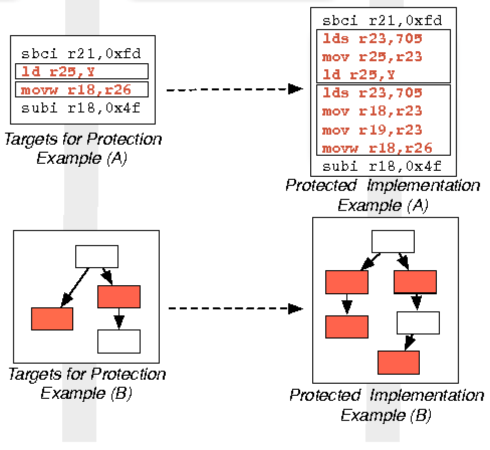
Security Evaluation



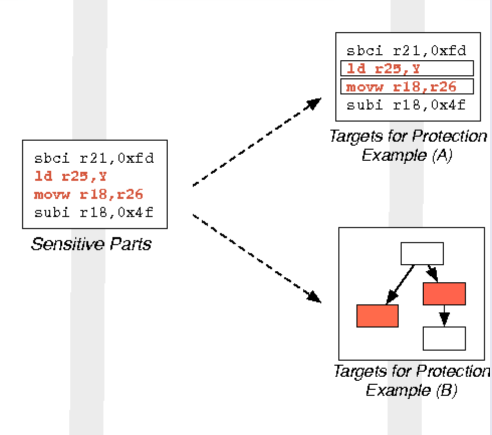
What about software?

- Power Analysis: random precharging, masking
- Timing attacks
- Domain Specific Languages
- Verification (mainly on properly applied masking)

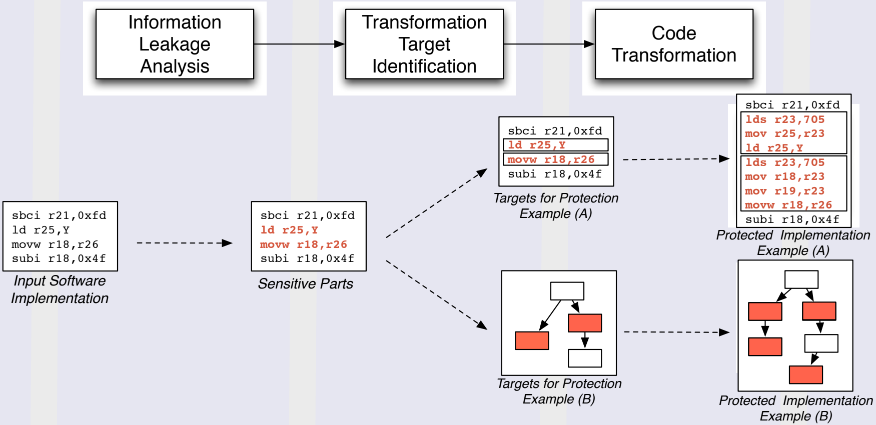
Code Transformation



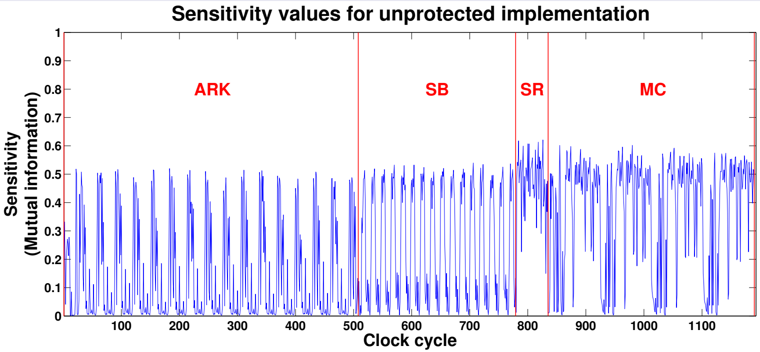
Transformation Target Identification



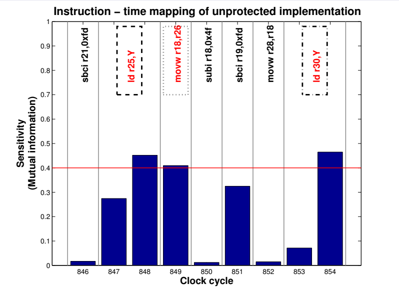
Overall Software Flow



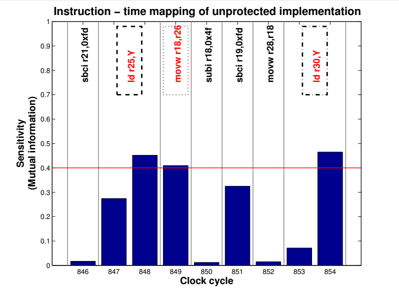
Information Leakage Analysis

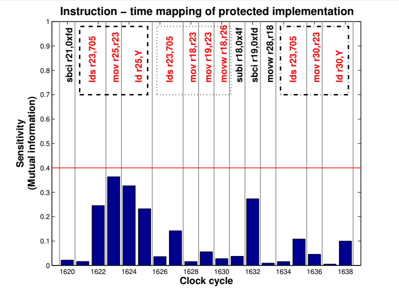


Example on Software

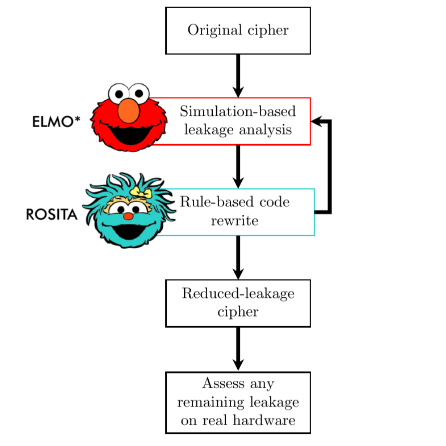


Example on Software

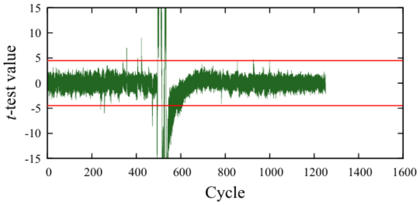




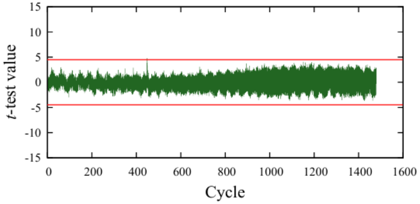
Code Re-Write Engine



Code Re-Write Engine



(a) AES original implementation.



(d) AES fixed with ROSITA.

Step Three

Towards Verification

Inputs:

- Algorithm where the countermeasure is Applied
- Countermeasure

Output:

- Assertion of the Correct Application of the Countermeasure

- Assertion of the correct application of the countermeasure does NOT mean protected Algorithm

Do We Need Verification?

```
void maskedARK() {
 unsigned char i;
 for (i=0;i<16;i++){
   st[i] = pt[i] ^
      (key[i] ^ mask[i]);
 }
}
```

avr-gcc-4.5.3-O3

```
          .text
.global ARK
          .type   ARK, @function
ARK:
/* prologue: function */
/* frame size = 0 */
/* stack size = 0 */
.L__stack_usage = 0
          lds r24,key
          lds r25,pt
          eor r24,r25
          lds r25,mask
          eor r24,r25
          sts st,r24
          lds r24,key+1
          lds r25,pt+1
          eor r24,r25
          ...
```

Sensitivity Definition

Goal

Given a **program**, find the **sensitive** operations, which **leak critical** information.

Define three types for variables:

- Secret
- Public
- Random

Methodology

- Represent the program as a graph
- Use satisfiability queries to detect the dependencies and sensitivity

Dependency Check

- Is it a **Don't care** from random point of view?
- If at least one bit is not a don't care, it is random, so ok.
- Else, check if is a **Don't care** from some secret variable?
- If at least a bit is not a don't care, then is sensitive.

- Compiler problems
- Programmer problems (shift with hamming distance leakage)
- Countermeasure problem (Goubin [2001])

Contents

One

Security Evaluation

Goals:

- Identify weaknesses in the design

Open problems:

- At which level of abstraction?
- How realistic is it?

Two

Metric

Goals:

- Measure the weaknesses in the design

Open problems:

- Which metrics do we use for other attacks?
- Can these metrics be combined?

Three

Other Attacks?

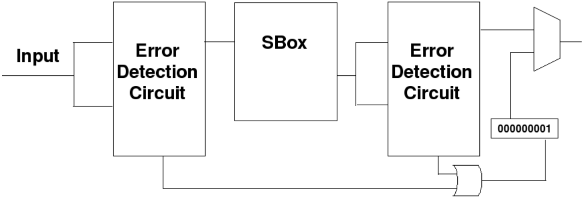
Goals:

- Global protections against physical attacks

Open problems:

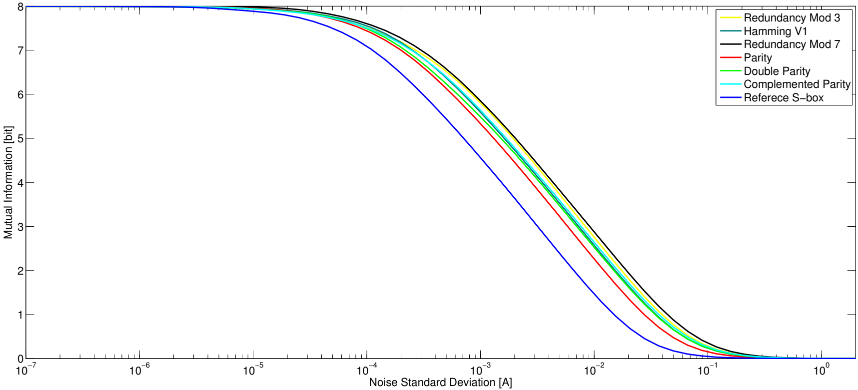
- Countermeasure for them?
- Which metric?

Effects of Error Correcting Codes on DPA

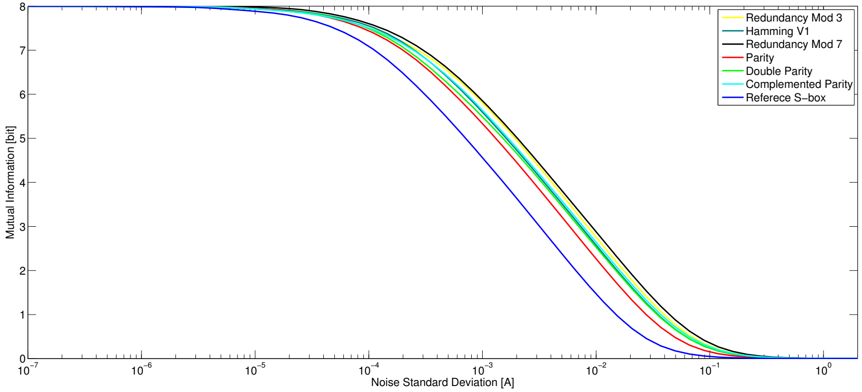


- Reference
- Parity
- Complemented Parity
- Double Parity
- Residue Modulo 3
- Residue Modulo 7
- Hamming Code

Error Correcting Code



Error Correcting Code



I am helping the DPA attacker!

Conclusions

- Automation is necessary for handling security
- Metrics are a fundamental brick for design automation
- Power analysis attack is not solved, yet is only the first one

Acknowledgments

EVEREST
DESIGN ENVIRONMENT
FOR EXTREME-SCALE BIG DATA ANALYTICS
ON HETEROGENEOUS PLATFORMS

Questions?

Thank you for your attention!

mail: f.regazzoni@uva.nl

mail: francesco.regazzoni@usi.ch