Towards the Automatic Application of Physical Attacks Countermeasures

Dr. Francesco Regazzoni

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

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
- 2018...

Now is time!

Why Automation....

Why Automation....

Design done by hands ||

Why Automation....

Design done by hands $\Vert$ Simple circuits

Why Automation....

Design done by hands EDA tools	Simple circuits

Why Automation....

Design done by hands	Simple circuits
EDA tools	Complex and large circuits

... for security?

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

- 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!

Where are we?

A bit of history

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

---

[a] Kris Tiri, Ingrid Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation" DATE 2004:

[b] Francesco Regazzoni, Alessandro Cevrero, François-Xavier Standaert, Stéphane Badel, Theo Kluter, Philip Brisk, Yusuf Leblebici, Paolo Ienne, "A Design Flow and Evaluation Framework for DPA-Resistant Instruction Set Extensions" CHES 2009

A bit of history

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

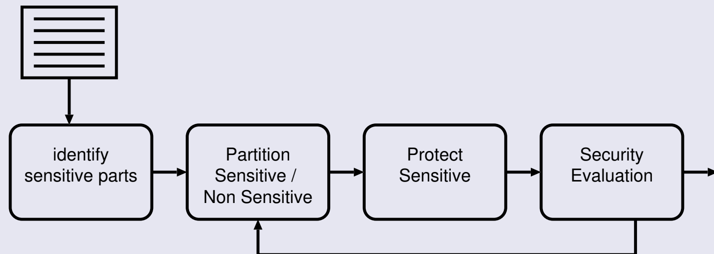
[a] Kris Tiri, Ingrid Verbauwhede, "A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation" DATE 2004:

[b] Francesco Regazzoni, Alessandro Cevrero, François-Xavier Standaert, Stéphane Badel, Theo Kluter, Philip Brisk, Yusuf Leblebici, Paolo Ienne, "A Design Flow and Evaluation Framework for DPA-Resistant Instruction Set Extensions" CHES 2009

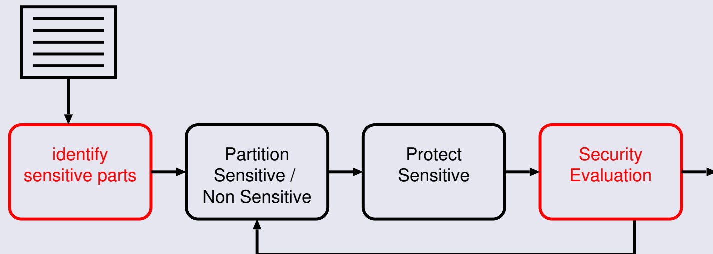
Still only goals

- Chip would most likely to function securely
- Chip security would be easier to be verified
- Designer could handle more complex designs

Enabling the automatic design for DPA resistance

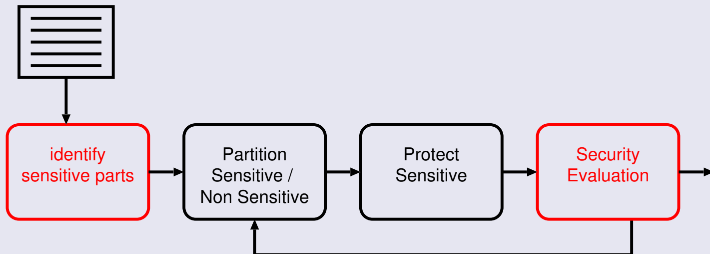


Needed "Basic Blocks"



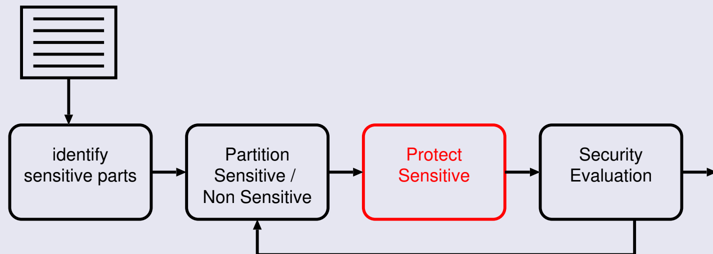
- Generate useful power traces?

Needed "Basic Blocks"



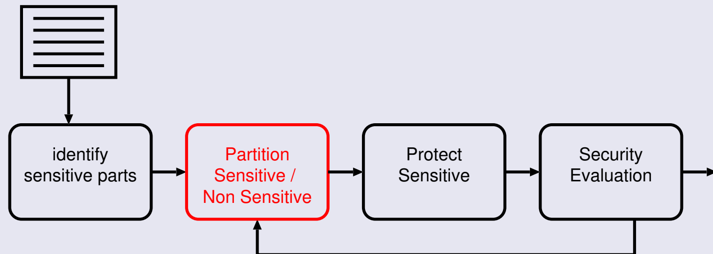
- Generate useful power traces?
- Measure the DPA resistance?

Needed "Basic Blocks"



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

Needed “Basic Blocks”



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

Where are we?

Step One

Automated Synthesis

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 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

Step Two

Towards Automatic Application of Countermeasures

Inputs:

- Unprotected Algorithm
- Countermeasure

Output:

- Algorithm where the countermeasure is Applied

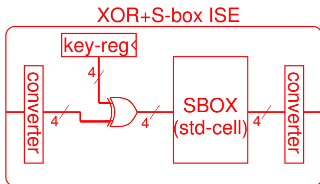
- Algorithm where the countermeasure is applied does NOT mean protected Algorithm

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
```

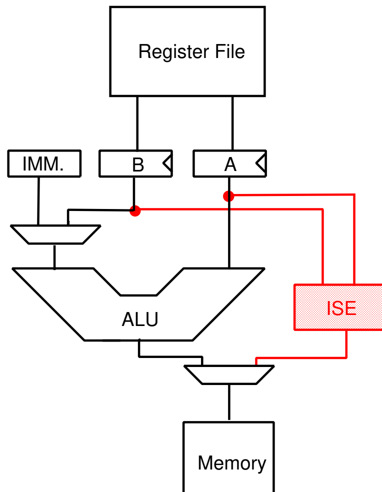
Customizable Processors

```
// 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
```

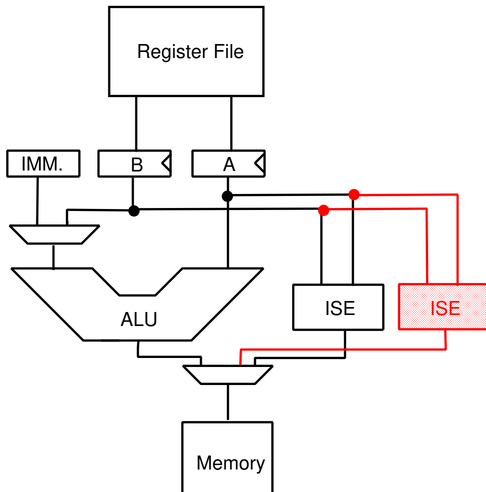



```
// Calculate S-box (plaintext XOR key)
int PRESENT_XOR+S-box-ISE(int plaintex) {
1 int result = 0; // initialize the result
 // instantiate the new instruction s-box(pt ^key)
2 Instr_1(plaintex, result);
3 return result; }; // return the result
```

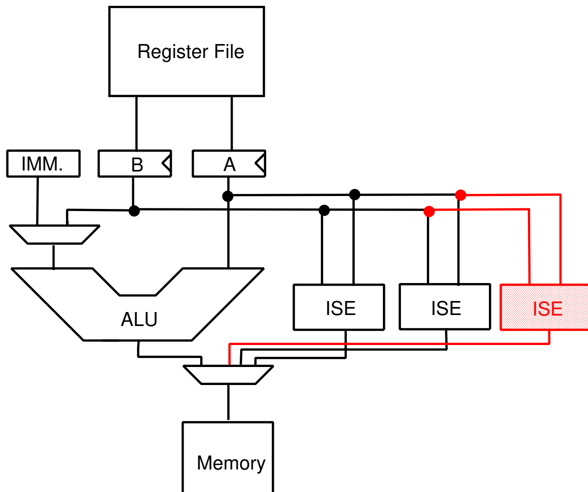
Protected / Non Protected CO-Design!



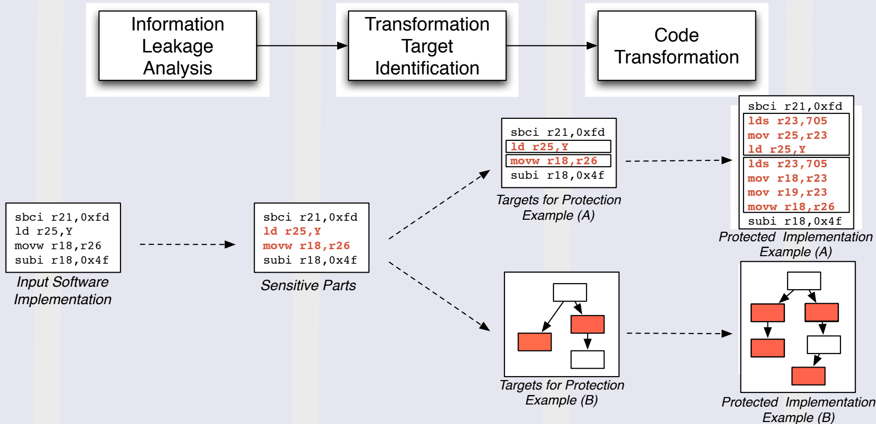
Protected / Non Protected CO-Design!



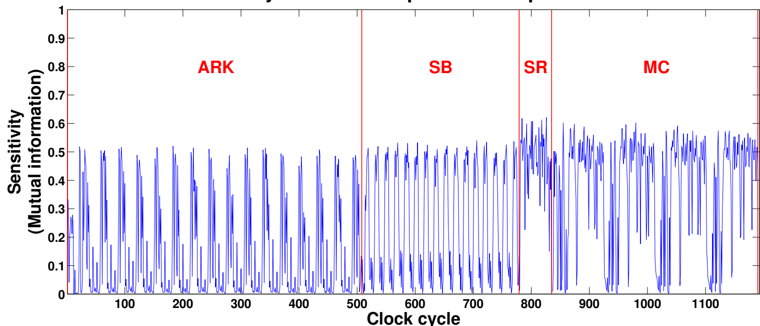
Protected / Non Protected CO-Design!



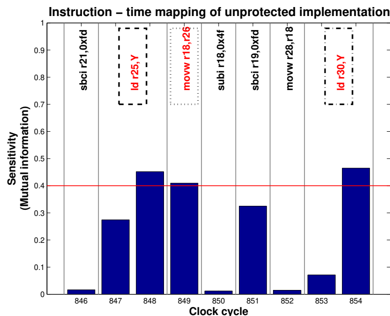
What about Software?



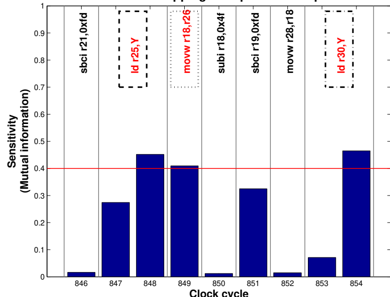
Information Leakage Analysis

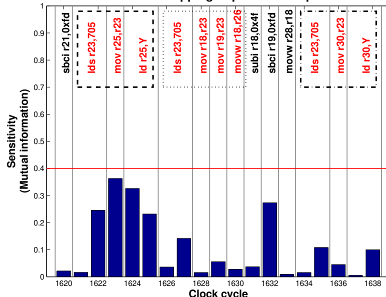


Example on Software

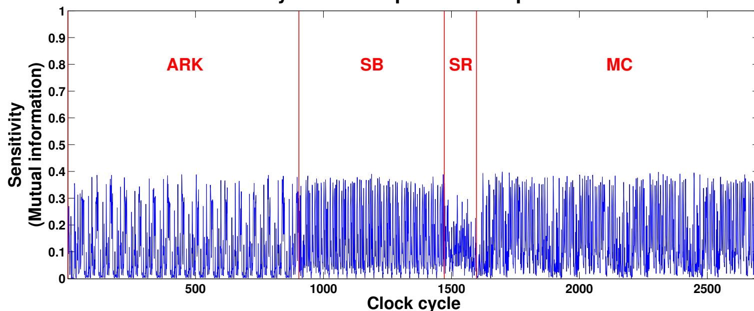


Example on Software





Security Evaluation



Step Three

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?

```c
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.

Identified Problems

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

Conclusions

- Physical security is a concern
- Design automation and Verification for physical security is crucial for Embedded systems
- Initial steps for power analysis are promising
- This is just the beginning...

Questions?

Thank you for your attention!

mail: regazzoni@alari.ch