FPGA Implementations of the AES Masked Against Power Analysis Attacks

Francesco Regazzoni[1,3], Yi Wang[2], and François-Xavier Standaert[1]

[1]UCL Crypto Group, Université catholique de Louvain, Louvain-la-Neuve, Belgium.
[2]Embedded System and Networking Laboratory, HuNan University, Changsha, China.
[3]ALaRI - University of Lugano, Lugano, Switzerland.

Motivations

- FPGAs are now **large** and **powerful** devices
- Many **algorithmic optimizations** for masking have been proposed in the past

Motivations

- FPGAs are now **large** and **powerful** devices
- Many **algorithmic optimizations** for masking have been proposed in the past

- Can we **combine efficiently** technological advances with algorithmic optimizations?

Goals

- Propose a **masked** implementation of AES **suitable for** state of the art **FPGAs**
- **Maximize** the exploitation of the FPGAs technology improvements **combining** them with algorithmic optimizations
- The device occupation should be **as limited as possible**
- The throughput should **fulfill** the needs of most applications

Outline

Why Masking?

- Power Analysis Attacks can be counteracted by altering the power characteristic of the device
- **Boolean masking** is appealing:
  - rather simple to implement
  - does not require novel hardware
  - leads to well quantifiable security level

Overview of Boolean 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

Challenges of Boolean masking

- Efficient for linear functions
- Significant penalty for non linear transformations
  - Computational overhead: $2^n * 2$ XOR operations and $2^n * 2 + 1$ memory transfers (the size of the look-up table is limited to 2^n)
  - Memory overhead: look-up table of size 2^{2n} (no computational overhead)

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

Outline

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 $GF(2^4)$ ✓
- 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 $GF(2^4)$ ✓

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?

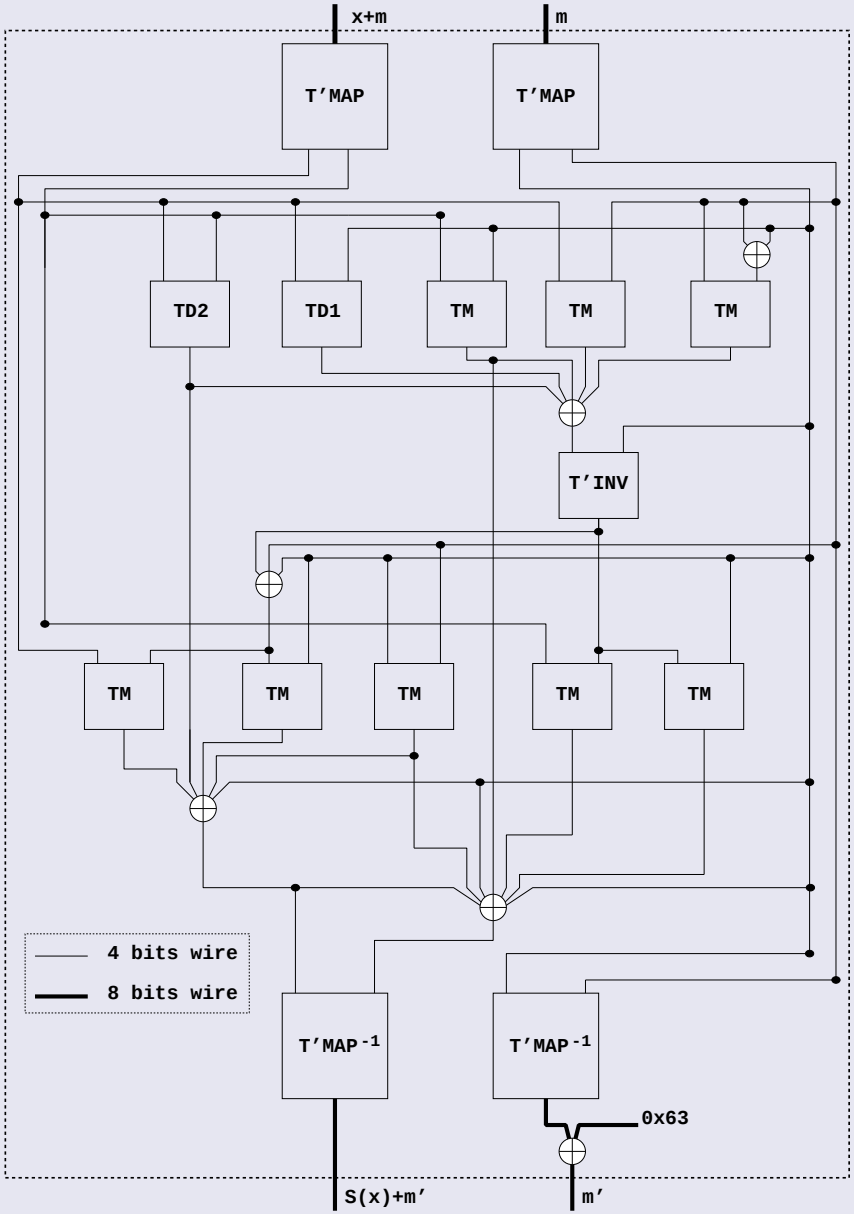
- **Virtex-5 maps well 8-bit input Look-up-tables**
- T_{d_1}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_{d_2}: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T_m: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
- T'_{inv}: input two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

S-box on Virtex-5



S-box on Virtex-5

Table: Implementation results of S-box on Virtex-5

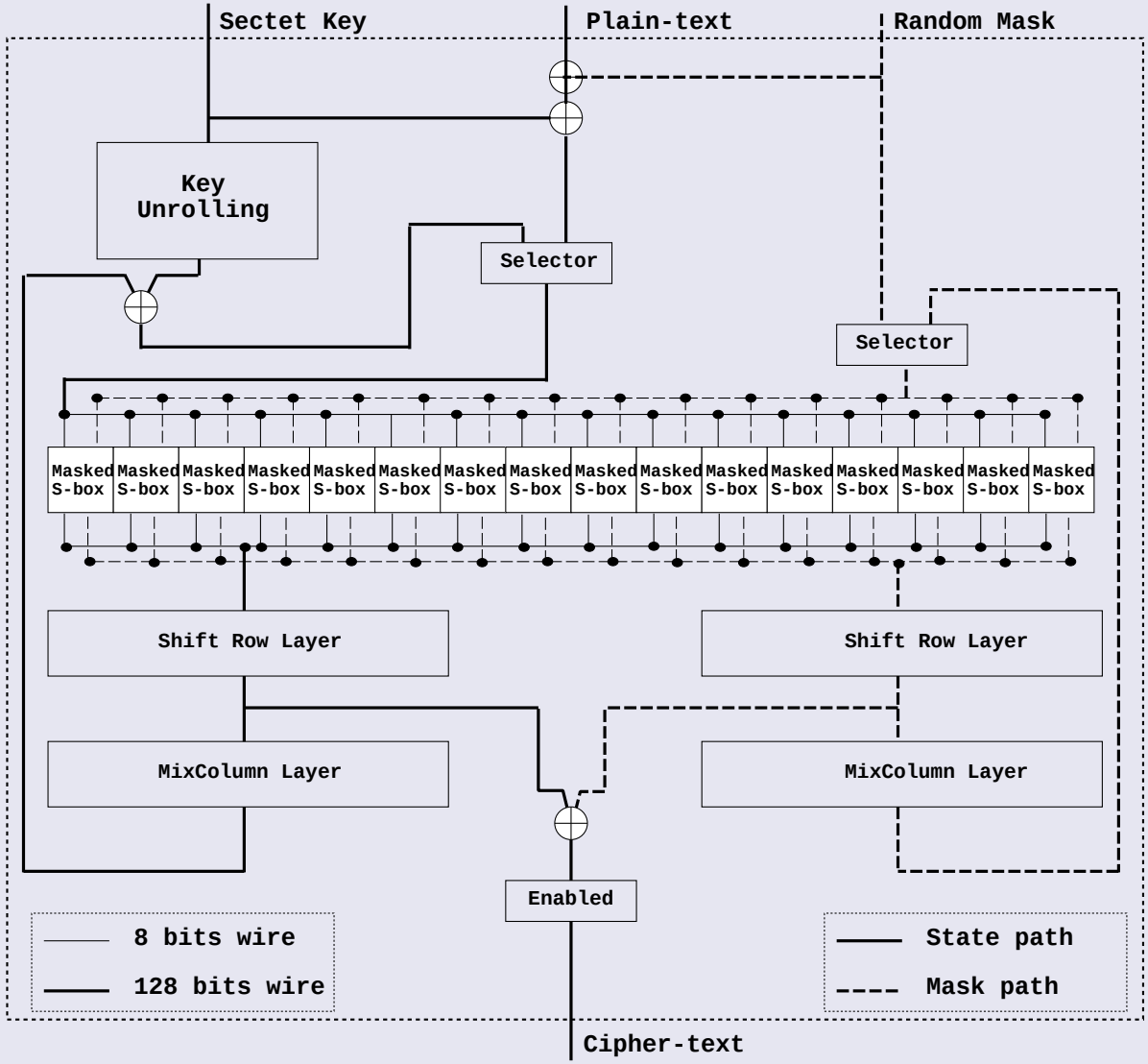
XC5vlx50	Slices	LUTs	Registers
Reference S-Box	8	32	0
Masked S-box	61	208	7

- Tool was forced to used distributed RAM
- Result: fastest which does not violate the timing constraints

Outline

1. Masking and FPGAs
2. S-box design
3. **AES designs**

The whole AES on Virtex-5



The whole AES on Virtex-5

Table: Implementation results of masked AES on Virtex-5

	Reference 32 bit	Masked 32 bit	Reference 128 bit	Masked 128 bit
Number of Slices	290	637	478	1,462
Number of LUTs	595	1,429	1,557	4,772
Number of Registers	467	643	648	904
Clock Cycles core (+ interface)	44 (+8)	44 (+8)	11 (+8)	11 (+8)
Clock (ns)	5	10	4	10
Frequency (MHz)	200	100	245	100
Throughput (Mbit/s) core	581	290	2909	1163
Throughput (Mbit/s) core + interface	492	246	1684	673

Comparison

- Comparison difficult (not only the design, also tool versions, device architecture and vendor, strategy for DPA resistance)
- Mentens et al.: combines Boolean with multiplicative masking. Area overhead of secured core 20%, speed by 30%.
- Kamoun et al.: masked AES S-box on Virtex-4. Area overhead of 44%, frequency decrease of 31%.
- Nassar et al: precharged logic, and a target device coming from a different vendor. Protected core 3 times bigger, speed decreased of one third

Comparison

- Comparison difficult (not only the design, also tool versions, device architecture and vendor, strategy for DPA resistance)
- Mentens et al.: combines Boolean with multiplicative masking. Area overhead of secured core 20%, speed by 30%.
- Kamoun et al.: masked AES S-box on Virtex-4. Area overhead of 44%, frequency decrease of 31%.
- Nassar et al: precharged logic, and a target device coming from a different vendor. Protected core 3 times bigger, speed decreased of one third

- The penalty of our protected designs is in line with the one of previous works

Conclusions

- Explored Boolean masking to protect AES on FPGA
- We exploit the slice structure of Xilinx Virtex-5 FPGA and software algorithmic optimizations
- Our masked implementations allow sufficient performances and keep the device occupation acceptable

Questions?

THANK YOU FOR YOUR ATTENTION!

mail: francesco.regazzoni@uclouvain.be