FPGA Implementations of the AES Masked Against Power Analysis Attacks

Francesco Regazzoni\textsuperscript{1,3}, Yi Wang\textsuperscript{2}, and François-Xavier Standaert\textsuperscript{1}

\textsuperscript{1}UCL Crypto Group, Université catholique de Louvain, Louvain-la-Neuve, Belgium.  
\textsuperscript{2}Embedded System and Networking Laboratory, HuNan University, Changsha, China.  
\textsuperscript{3}ALaRI - University of Lugano, Lugano, Switzerland.
FPGAs are now **large** and **powerful** devices

Many *algorithmic optimizations* for masking have been proposed in the past.
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?
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

1. Masking and FPGAs
2. S-box design
3. AES designs
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
- 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: $2^n \times 2$ XOR operations and $2^n \times 2 + 1$ 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)
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

1. Masking and FPGAs
2. S-box design
3. AES designs
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 $GF(2^8)$
S-box 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: $T_{d1}$, $T_{d2}$, $T_m$ 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
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 $GF(2^4)$
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 $GF(2^4)$
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 $GF(2^4)$ ✓
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 $GF(2^4)$ ✓
- $T_{inv}^t$: 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 $GF(2^4)$ ✓
  - $T_{inv}^\prime$: input two elements of $GF(2^4)$, output an element of $GF(2^4)$ ✓
  - $T_{map}^\prime$: 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 $GF(2^4)$
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
Table: Implementation results of S-box on Virtex-5

<table>
<thead>
<tr>
<th></th>
<th>Slices</th>
<th>LUTs</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference S-Box</td>
<td>8</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>Masked S-box</td>
<td>61</td>
<td>208</td>
<td>7</td>
</tr>
</tbody>
</table>

- Tool was forced to use distributed RAM
- Result: fastest which does not violate the timing constraints
1. Masking and FPGAs
2. S-box design
3. AES designs
The whole AES on Virtex-5

- **Key Unrolling**
- **Selector**
- **Masked S-box**
- **Shift Row Layer**
- **MixColumn Layer**

8 bits wire

128 bits wire

**Cipher-text**

**State path**

**Mask path**

**Enabled**

**Plain-text**

**Random Mask**

**Selected Key**
Table: Implementation results of masked AES on Virtex-5

<table>
<thead>
<tr>
<th></th>
<th>Reference 32 bit</th>
<th>Masked 32 bit</th>
<th>Reference 128 bit</th>
<th>Masked 128 bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slices</td>
<td>290</td>
<td>637</td>
<td>478</td>
<td>1,462</td>
</tr>
<tr>
<td>Number of LUTs</td>
<td>595</td>
<td>1,429</td>
<td>1,557</td>
<td>4,772</td>
</tr>
<tr>
<td>Number of Registers</td>
<td>467</td>
<td>643</td>
<td>648</td>
<td>904</td>
</tr>
<tr>
<td>Clock Cycles core (+ interface)</td>
<td>44 (+8)</td>
<td>44 (+8)</td>
<td>11 (+8)</td>
<td>11 (+8)</td>
</tr>
<tr>
<td>Clock (ns)</td>
<td>5</td>
<td>10</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Frequency (MHz)</td>
<td>200</td>
<td>100</td>
<td>245</td>
<td>100</td>
</tr>
<tr>
<td>Throughput (Mbit/s) core</td>
<td>581</td>
<td>290</td>
<td>2909</td>
<td>1163</td>
</tr>
<tr>
<td>Throughput (Mbit/s) core + interface</td>
<td>492</td>
<td>246</td>
<td>1684</td>
<td>673</td>
</tr>
</tbody>
</table>
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
THANK YOU FOR YOUR ATTENTION!

mail: francesco.regazzoni@uclouvain.be