Hiding Higher-Order Leakages in Hardware


Amir Moradi
Ruhr-Universität Bochum
Acknowledgement

- Pascal Sasdrich
- Tobias Schneider
- Alexander Wild
Story?

- Threshold Implementation
  - should be explained?
  - 1st-order security, but higher orders

- Solutions
  - Higher-order TI
    - might be restricted to univariate
    - area overhead might be problematic
    - finding uniform representations might be challenging
  - Stay with 1st-order TI and make the higher-order attacks hard
    - Increase the noise
    - Reduce the signal
Noise Addition

- Instead of generic ones we plan a systematic approach dedicated to TI
- Case study: PRESENT
  - Particularly PRESENT Sbox \( y = S(x) \)
  - Algebraic degree of 3, minimum 4 shares
    - Or decomposition to quadratic functions
    - Thanks to the classifications in
    - \( S : A' \circ C^4_{266} \circ A \) 7 different ways to decompose
    - \( (Q_{12} \circ Q_{12}), (Q_{293} \circ Q_{300}), (Q_{294} \circ Q_{299}), (Q_{299} \circ Q_{294}), (Q_{299} \circ Q_{299}), (Q_{300} \circ Q_{293}), \text{ and } (Q_{300} \circ Q_{300}) \)
    - It means, e.g.,
      \( S : A'' \circ Q_{12} \circ A' \circ Q_{12} \circ A \) with three affine functions \( (A, A', A'') \)
Affine Functions

- How to implement?
- Uniform TI of $Q_{12}$ is easily made by direct sharing

- How many of such 3-tuple affines exist?

- We exclude those with $Q_{300}$, as its uniform TI needs (at least) two stages
Changing the Affines on the fly

- At least 3 shares should be used (2nd-order Boolean masking)
  - but can provide only 1st-order security
  - due to the quadratic function
- If we change the affines dynamically, the hope is to make the 2nd-order attacks harder
- It should not affect the 3rd-order vulnerability
Changing the Affines on the fly (Implementation)

- Spartan-6 FPGA of SAKURA-G
- Let us focus on $S : A'' \circ Q_{12} \circ A' \circ Q_{12} \circ A$ with 147,456 affines

- Option 1: save all affines
  - Each affine: $4 \times 4$ binary matrix + 4-bit constant: 20 bits
  - $147,456 \times 20 \times 3 : 8640$ kbit
    - XC6SLX75 has 3096 kbit (BRAM)
    - So, all of them do not fit to a single LX75 FPGA
Changing the Affines on the fly (Implementation)

- Option 2: compute them on the fly
  - pretty efficient in terms of area overhead
  - Problem: not a constant rate of finding the affines
    - For example, several ones are found sequentially and for a long time no one is found
    - So, it happens that many encryptions are performed with a fixed set of affines -> contradiction with the goal
Changing the Affines on the fly (Implementation)

- Closer look at the 147,456 affines: 384 x 384
  - 384 different input affines \( A \)
  - 384 different output affines \( A'' \)
  - each 384 affine set is made of 48 linear functions and 8 different constants

- Option 3: save the linear functions of \( A \) and \( A'' \), and compute \( A' \) on the fly
  - To compute the middle affine we need inverse of \( A \) and \( A'' \)
  - 48 x 2 x 16 bits (for linear of \( A \) and inverse) the same for \( A'' \)
  - So, 48 x 2 x 2 x 16 : 3 kbits which fit into a single BRAM
  - + extra logic to compute \( A' \)
Changing the Affines on the fly (Implementation)

- Option 4: save the linear functions of A and A“, and compute A’ on the fly
  - Saving the 3 kbits might be challenging for ASIC platforms
  - Can we do better?
- Again a close look at the linear functions of A and A“
  - There is a pattern in their binary matrix representations
  - It is possible to derive Boolean functions for the bits of the matrix as well as for the constants.
  - So, given a few random bits a matrix and a constant for A (the same for A“) can be derived.
    - The middle affine A’ can be computed similar to the previous option
Implementation

- **PRESENT-80 following a round-based fashion**
  - Pipeline with two stages, due to the middle register in the decomposed Sbox
  - 33 clock cycles latency with two full encryptions
Implementation

- **PRESENT-80 following a round-based fashion**
  - Pipeline with two stages, due to the middle register in the decomposed Sbox
  - 33 clock cycles latency with two full encryptions

- Comparison between the options to realize random affine

<table>
<thead>
<tr>
<th>Method</th>
<th>Resource Utilization</th>
<th>Reconfig. Time</th>
<th>Affine Triples Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Logic (LUT)</td>
<td>Memory (FF)</td>
<td>BRAM16</td>
</tr>
<tr>
<td>Option 1</td>
<td>204</td>
<td>0</td>
<td>60</td>
</tr>
<tr>
<td>Option 2</td>
<td>562</td>
<td>250</td>
<td>20</td>
</tr>
<tr>
<td>Option 3</td>
<td>139</td>
<td>20</td>
<td>1</td>
</tr>
</tbody>
</table>
Evaluation by Welch’s $t$-test

- measure power traces with digital oscilloscope
- determine distinguisher, e.g.:
  - fix vs. random plaintext (non-specific $t$-test)
- group traces depending on distinguisher
- compute sample mean for each point in time
- compute sample variance for each point in time
- determine $t$-statistic for each point in time:

$$t = \frac{\mu(T \in G_1) - \mu(T \in G_0)}{\sqrt{\frac{\delta^2(T \in G_1)}{|G_1|} + \frac{\delta^2(T \in G_0)}{|G_0|}}}$$

where $\mu$ denotes the sample mean and $\delta$ denotes the sample variance.

**Fail/Pass Criteria:** If there is any point in time for which the $t$-statistic exceeds a threshold of ± 4.5 the device under test fails.

Results

fixed affine

25 million traces
Results

random, affine

200 million traces
Reducing the Signal

- Dual-rail logic styles, to (ideally) equalize the power/energy consumption
- They usually fail to be ideal, but can for sure reduce the signal
- Limited routing resources in FPGAs -> not much success to mount
- Slightly different approach: Glitch-Free Duplication (GliFreD)
  - Each LUT enabled once at each two clock cycles (precharge vs eval) and is followed by a two flip-flops (master-slave)
GliFreD (example)
Case Studies

- **KATAN-32**
  - 1\textsuperscript{st}-order TI
  - 2\textsuperscript{nd}-order TI
  - 1\textsuperscript{st}-order TI by GliFreD

- **PRESENT-80**
  - 1\textsuperscript{st}-order TI
  - 2\textsuperscript{nd}-order TI
  - 1\textsuperscript{st}-order TI by GliFreD
Implementations

- GliFreD circuits form a pipeline with a very short critical path
  - Frequency (hence throughput) can be higher than non-GliFreD designs
  - Achievements depend on the application and the design nature

<table>
<thead>
<tr>
<th>Profile</th>
<th>Resources</th>
<th>Frequency (MHz)</th>
<th>Latency (#clock)</th>
<th>Pipeline (stage)</th>
<th>Throughput (Mbit/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KATAN-1st</td>
<td>34 LUT, 96 FF</td>
<td>225.38</td>
<td>273</td>
<td>1</td>
<td>26.42</td>
</tr>
<tr>
<td>KATAN-2nd</td>
<td>65 LUT, 180 FF</td>
<td>321.54</td>
<td>273</td>
<td>1</td>
<td>37.69</td>
</tr>
<tr>
<td>KATAN-1st-G</td>
<td>114 LUT, 548 FF</td>
<td>438.21</td>
<td>546</td>
<td>1</td>
<td>25.68</td>
</tr>
<tr>
<td>PRESENT-1st</td>
<td>808 LUT, 384 FF</td>
<td>206.61</td>
<td>64</td>
<td>2</td>
<td>413.22</td>
</tr>
<tr>
<td>PRESENT-2nd</td>
<td>2245 LUT, 1680 FF</td>
<td>203.46</td>
<td>128</td>
<td>4</td>
<td>406.92</td>
</tr>
<tr>
<td>PRESENT-1st-G</td>
<td>5442 LUT, 12672 FF</td>
<td>458.09</td>
<td>704</td>
<td>11</td>
<td>458.09</td>
</tr>
</tbody>
</table>

- For a Spartan-6 FPGA of SAKURA-G
Evaluations

- SAKURA-G
- Running the designs @ 24MHz
- Measurements with 500MS/s
- non-specific $t$-test
  - 1$^{st}$- to 5$^{th}$-order
- several million traces
KATAN-1\textsuperscript{st}

(a) Sample Trace

(b) PRNG off (10,000 traces)

(c) first-order

(d) second-order

(e) third-order

1 million traces
KATAN-2\textsuperscript{nd}

100 million traces
KATAN-1\textsuperscript{st} GliFreD

1 billion traces

(a) Sample Trace

(b) first-order

(c) second-order

(d) third-order
PRESENT-1st

10 million traces

(a) Sample Trace
(b) first-order
(c) second-order
(d) third-order
PRESENT-2nd

300 million traces
PRESENT-1\textsuperscript{st} GliFreD

(a) Sample Trace

(b) first-order

(c) second-order

(d) third-order

1 billion traces
Sum Up

- Higher-order TI is a promising approach
- Instead of going to the direction of no-ending provable higher-order resistance,
  - we looked at the problem from another point of view
  - by staying with the 1\textsuperscript{st}-order TI with sound mathematical proofs
    - and increased the noise following a systematic approach (dedicated to decomposition in TI)
    - or reduce the signal by power-equalization approaches (dedicated to Xilinx FPGAs)
  - In short, we still have 1\textsuperscript{st}-order security
  - and at the same time made higher-order attacks harder
  - for sure, no provable resistance against higher-order attacks...
Thanks!

any questions?

amir.moradi@rub.de

Embedded Security Group, Ruhr-Universität Bochum, Germany