Threshold Implementations

Benedikt Gierlichs

Reference:
A more efficient Threshold Implementation of AES
Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen

Introduction - masking

- Masking countermeasure against side channel attacks
  - Process random shares instead of direct values
  - Often boolean masking:
    1st order masking: \( v \rightarrow (\text{mask}, v \oplus \text{mask}) \) with random mask
    - Masking linear function: \( f(v) = f(\text{mask} \oplus v \oplus \text{mask}) = f(\text{mask}) \)
      - Processing \( f() \) on either share cannot leak any information
      - Processing \( f() \) on both shares in parallel is 1st order DPA secure
  - Masking non-linear function (S-box): \( g(v) \neq g(\text{mask}) \oplus g(v \oplus \text{mask}) \)
    - Need a 2nd function: \( g(v) = g(\text{mask}) \oplus h(\text{mask}, v \oplus \text{mask}) \)
    - Processing \( g() \) on one share cannot leak any information
    - Processing \( h() \) on both shares provable 1st order DPA secure?

Introduction - glitches

- \( h(\text{mask}, v \oplus \text{mask}) \)
  - Processing \( h() \) on both shares may not be 1st order secure!
  - Function \( h() \) knows both shares
  - Depends on implementation of function

- Glitches are temporary intermediate states of combinational logic
- Glitches can be a serious security problem

ab = \( (a_0 \oplus a_1)(b_0 \oplus b_1) = a_0b_0 \oplus a_0b_1 \oplus a_1b_0 \oplus a_1b_1 \)
Share1 = \( a_0b_0 \) Share2 = \( a_0b_1 \oplus a_1b_0 \oplus a_1b_1 \)

- Suppose \( a_1 \) arrives late, \( a_0 \) not relevant

<table>
<thead>
<tr>
<th>#1</th>
<th>b0</th>
<th>b1</th>
<th>AND</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0→1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1→0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0→1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1→0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0→1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1→0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0→1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1→0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
\[ a b = (a_0 \oplus a_1)(b_0 \oplus b_1) = a_0 b_0 \oplus a_0 b_1 \oplus a_1 b_0 \oplus a_1 b_1 \]

- **Not provable 1st order DPA secure!**

<table>
<thead>
<tr>
<th>(a_1)</th>
<th>(b_0)</th>
<th>(b_1)</th>
<th>AND</th>
<th>XOR</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>0+1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1+0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0+1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>1+0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>0+1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1+0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0+1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1+0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

b = 0 average = 2
b = 1 average = 2.5

- **Engineering approach:**
  - Try to fix implementation
  - Not provable secure but hopefully secure in practice
  - Requires design iterations and testing

- **Mathematical approach:**
  - Change the masking to entirely avoid the problem

**Introduction - glitches**

- Share 1 = \(a_0 b_0\)
- Share 2 = \((a_0 b_1 \oplus a_1 b_0) \oplus a_1 b_1\)
- Suppose \(a_1\) arrives late, \(a_0\) not relevant

**Threshold Implementations**

- Based on Boolean masking and multiparty computation
- **Pros**
  - Security in a circuit with glitches, any HW technology
  - Provable secure against 1st order DPA
- **Cons**
  - Only 1st order DPA resistant
  - High-degree non-linear functions difficult to implement
Threshold Implementations

Shares

$(x_1, y_1, z_1, \ldots)

S_1

(x_2, y_2, z_2, \ldots)

S_2

\vdots

(x_n, y_n, z_n, \ldots)

S_n

3 properties

Threshold Implementations

$(x_1, y_1, z_1, \ldots)

\oplus

S_1

(x_2, y_2, z_2, \ldots)

\oplus

S_2

\vdots

\vdots

(x_n, y_n, z_n, \ldots)

\oplus

S_n

=(x, y, z, \ldots)

Correctness

Threshold Implementations

$(x_1, y_1, z_1, \ldots)

S_1

(x_2, y_2, z_2, \ldots)

S_2

\vdots

\vdots

(x_n, y_n, z_n, \ldots)

S_n

=(x, y, z, \ldots)

Correctness, non-completeness

Threshold Implementations

Non-completeness

• Example

$$S(x, y, z) = x + yz$$

$$S_1 = x_2 + y_2 z_2 + y_2 z_3 + y_3 z_2$$

$$S_2 = x_3 + y_3 z_3 + y_3 z_1 + y_1 z_3$$

$$S_3 = x_1 + y_1 z_1 + y_1 z_2 + y_2 z_1$$

• To protect a function with degree $d$, at least $d+1$ shares are required
Threshold Implementations

Correctness, non-completeness, uniformity

Security

non-complete

total power is linear function of components, e.g. sum

Security

non-complete

total power is linear function of components, e.g. sum
Security

- Non-complete
- Uniformly distributed
- No leakage
- Provable 1st order DPA secure!

Implementations

- Only nonlinear part of AES
- Based on multiplicative inverse in GF(256) and affine transformation
- Possible to implement with tower field approach (GF(2^4))
Our AES implementation

- Goal: compact and secure implementation
- Serial data path, 1 S-box
- Main idea: adjust number of shares as needed to minimize gate count
  - Linear part: only 2 shares
  - S-box: 2 to 5 shares

Our implementation of AES S-box

- 2 shares
- 4 input, 3 output shares
- 5 shares
Our implementation of AES S-box

registers after every nonlinear function

September 2014

Our implementation of AES S-box

re-masking to change the number of shares and preserve uniformity

September 2014

Implementation results

<table>
<thead>
<tr>
<th>State</th>
<th>Key</th>
<th>MixCol</th>
<th>Controll</th>
<th>Key</th>
<th>MUX</th>
<th>Other</th>
<th>Total</th>
<th>cycles</th>
<th>rand bits²</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>2529</td>
<td>2526</td>
<td>4244</td>
<td>1120</td>
<td>166</td>
<td>64</td>
<td>376</td>
<td>89</td>
<td>11114/11031</td>
</tr>
<tr>
<td>This paper</td>
<td>1608</td>
<td>1896</td>
<td>3708</td>
<td>770</td>
<td>221</td>
<td>48</td>
<td>746</td>
<td>21</td>
<td>9162</td>
</tr>
<tr>
<td>This paper²</td>
<td>1608</td>
<td>1896</td>
<td>3003</td>
<td>544</td>
<td>221</td>
<td>48</td>
<td>746</td>
<td>21</td>
<td>8171</td>
</tr>
</tbody>
</table>

But we still require $44 \times 20 \times 10 = 8800$ random bits per AES operation.
Not practical.

• 18% smaller, 7.5% faster
• 8% less randomness for re-masking

Our TI of S-box uses 3.7k GE (3k GE)
– Based on plain Canright S-box 233 GE (Moradi et al.)
• Our TI of AES uses 9k GE (8k GE)
– Based on plain AES 2.4k GE (Moradi et al.)

September 2014

Implementation results

<table>
<thead>
<tr>
<th>State</th>
<th>Key</th>
<th>MixCol</th>
<th>Controll</th>
<th>Key</th>
<th>MUX</th>
<th>Other</th>
<th>Total</th>
<th>cycles</th>
<th>rand bits²</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>2529</td>
<td>2526</td>
<td>4244</td>
<td>1120</td>
<td>166</td>
<td>64</td>
<td>376</td>
<td>89</td>
<td>11114/11031</td>
</tr>
<tr>
<td>This paper</td>
<td>1608</td>
<td>1896</td>
<td>3708</td>
<td>770</td>
<td>221</td>
<td>48</td>
<td>746</td>
<td>21</td>
<td>9162</td>
</tr>
<tr>
<td>This paper²</td>
<td>1608</td>
<td>1896</td>
<td>3003</td>
<td>544</td>
<td>221</td>
<td>48</td>
<td>746</td>
<td>21</td>
<td>8171</td>
</tr>
</tbody>
</table>

• 18% smaller, 7.5% faster
• 8% less randomness for re-masking

• Our TI of S-box uses 3.7k GE (3k GE)
– Based on plain Canright S-box 233 GE (Moradi et al.)
• Our TI of AES uses 9k GE (8k GE)
– Based on plain AES 2.4k GE (Moradi et al.)
**Practical security evaluation**

- FPGA implementation on SASEBO-G
  - Crypto FPGA: only TI-AES and PRNG
  - Use keep hierarchy constraint for TI-AES module
  - Measurements: 1ohm resistor, passive probe, 1GS/s
  - Measurements cover first 1.5 rounds

- Low noise measurements
  - Real attacks will be more difficult

- Goals
  - Verify 1\textsuperscript{st} order attack security
  - Check higher order attack security

---

**AES implementation**

---

**Practical security evaluation**

- PRNG off, 1\textsuperscript{st} order CPA, HD model at S-box output
- Highest peak 3 cycles later, input to MC

---

**Practical security evaluation**

- PRNG off, 1\textsuperscript{st} order correlation collision attack
Practical security evaluation

- PRNG on, 1st order CPA / correlation collision attack
- 10 million traces

Practical security evaluation

- PRNG on, 2nd order CPA, HD model at S-box output

Practical security evaluation

- PRNG switched on
- 1st order attack resistant using 10 million traces
- Best 2nd order attack requires 600k traces
- Noise makes 2nd order attacks more difficult
- Number of traces increases quadratically in noise standard deviation
- We had very little noise in the measurements
More recent results

• 1st order TI of AES with >2 shares: requires several million traces to break with 2nd order attack
  [not yet published]

• Extension of Threshold Implementation to higher orders
  – Theory & Proofs
  – Example: 2nd order TI of KATAN block cipher
  – No leakage in 1st and 2nd moment with 300 MILLION traces