A More Efficient Threshold Implementation of AES

Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventsislav Nikov, and Vincent Rijmen
Threshold Implementations
Threshold Implementations

Countermeasure against Differential Power Analysis
Threshold Implementations

Countermeasure against Differential Power Analysis
Threshold Implementations

Countermeasure against Differential Power Analysis

instantaneous power consumption

intermediate results of the cryptographic algorithm.
Threshold Implementations

Countermeasure against Differential Power Analysis

- **Circuit level**
  - WDDL cells

- **Algorithmic level**
  - Introducing Noise (not provably secure)
    - random delays
    - dummy operations
  - Masking (provably secure)
  - Leakage resilient crypto (limits encryptions per key)
Threshold Implementations

Countermeasure against Differential Power Analysis

- Circuit level
  - WDDL cells

- Algorithmic level
  - Introducing Noise (not provably secure)
    - random delays
    - dummy operations
  - Masking (provably secure)
  - Leakage resilient crypto (limits encryptions per key)
Threshold Implementations
Threshold Implementations

Masking Scheme based on Secret Sharing and Multiparty Computation
Threshold Implementations

Masking Scheme based on Secret Sharing and Multiparty Computation

Pros:
Threshold Implementations

Masking Scheme based on Secret Sharing and Multiparty Computation

Pros:
✓ Security in a circuit with glitches
Threshold Implementations

Masking Scheme based on Secret Sharing and Multiparty Computation

Pros:
✓ Security in a circuit with glitches
✓ Efficient in HW
Threshold Implementations

Masking Scheme based on Secret Sharing and Multiparty Computation

Pros:
✓ Security in a circuit with glitches
✓ Efficient in HW
✓ Any HW technology
Threshold Implementations

Masking Scheme based on Secret Sharing and Multiparty Computation

Pros:
✓ Security in a circuit with glitches
✓ Efficient in HW
✓ Any HW technology

Cons:

 Threshold Implementations
Threshold Implementations

Masking Scheme based on Secret Sharing and Multiparty Computation

Pros:
✓ Security in a circuit with glitches
✓ Efficient in HW
✓ Any HW technology

Cons:
✗ High order non-linear function are challenging
Threshold Implementations

Masking Scheme based on Secret Sharing and Multiparty Computation

Pros:
✓ Security in a circuit with glitches
✓ Efficient in HW
✓ Any HW technology

Cons:
× High order non-linear function are challenging

AES (11k), Present (3k), Noekeon, Keccak (30k)
roughly 3 times larger than unshared
Threshold Implementations

\[(x, y, z, \ldots) \rightarrow S \rightarrow (a, b, c, \ldots)\]
Threshold Implementations

\[(x_1, y_1, z_1, \ldots) \rightarrow S_1 \rightarrow (a_1, b_1, c_1, \ldots)\]

\[(x_2, y_2, z_2, \ldots) \rightarrow S_2 \rightarrow (a_2, b_2, c_2, \ldots)\]

\[\vdots\]

\[(x_s, y_s, z_s, \ldots) \rightarrow S_s \rightarrow (a_s, b_s, c_s, \ldots)\]
Threshold Implementations

Shares

\( (x_1, y_1, z_1, ...) \) \( \rightarrow \) \( S_1 \) \( \rightarrow \) \( (a_1, b_1, c_1, ...) \)

\( (x_2, y_2, z_2, ...) \) \( \rightarrow \) \( S_2 \) \( \rightarrow \) \( (a_2, b_2, c_2, ...) \)

\( \vdots \) \( \vdots \) \( \vdots \)

\( (x_s, y_s, z_s, ...) \) \( \rightarrow \) \( S_s \) \( \rightarrow \) \( (a_s, b_s, c_s, ...) \)
Threshold Implementations

\[ \begin{align*}
S_1 : (x_1, y_1, z_1, ...) & \rightarrow (a_1, b_1, c_1, ...) \\
S_2 : (x_2, y_2, z_2, ...) & \rightarrow (a_2, b_2, c_2, ...) \\
\vdots & \quad \quad \quad \quad \quad \vdots \\
S_s : (x_s, y_s, z_s, ...) & \rightarrow (a_s, b_s, c_s, ...) 
\end{align*} \]

 Shares

3 properties
Threshold Implementations

\[ (x_1, y_1, z_1, ...) \oplus (x_2, y_2, z_2, ...) \oplus \cdots \oplus (x_s, y_s, z_s, ...) = (x, y, z, ...) \]

\[ (a_1, b_1, c_1, ...) \oplus (a_2, b_2, c_2, ...) \oplus \cdots \oplus (a_s, b_s, c_s, ...) = (a, b, c, ...) \]

Correctness
Threshold Implementations

$$(x_1, y_1, z_1, ...)$$

$$
\oplus
$$

$$(x_2, y_2, z_2, ...)$$

$$
\oplus
$$

\vdots

$$
\oplus
$$

$$(x_s, y_s, z_s, ...)$$

$$=\quad (x, y, z, ...)$$

Correctness, Non-completeness
Threshold Implementations

\[(x_1, y_1, z_1, \ldots) \xrightarrow{S_1} (a_1, b_1, c_1, \ldots)\]

\[(x_2, y_2, z_2, \ldots) \xrightarrow{S_2} (a_2, b_2, c_2, \ldots)\]

\[(x_s, y_s, z_s, \ldots) \xrightarrow{S_s} (a_s, b_s, c_s, \ldots)\]

\[\oplus\]

\[=\]

\[(x, y, z, \ldots) = (a, b, c, \ldots)\]

Correctness, Non-completeness
Threshold Implementations

\[(x_1, y_1, z_1, ...) \oplus (x_2, y_2, z_2, ...) \oplus \ldots = (x, y, z, ...)\]

Correctness, Non-completeness
Threshold Implementations

\[ S(x, y, z) = x \oplus yz \]
\[ = (x_1 \oplus x_2 \oplus x_3) \oplus (y_1 \oplus y_2 \oplus y_3) \oplus (z_1 \oplus z_2 \oplus z_3) \]

\[ S_1(x_2, x_3, y_2, y_3, z_2, z_3) = x_2 \oplus y_2 z_2 \oplus y_2 z_3 \oplus y_3 z_2 \]
\[ S_2(x_1, x_3, y_1, y_3, z_1, z_3) = x_3 \oplus y_3 z_3 \oplus y_3 z_1 \oplus y_1 z_3 \]
\[ S_3(x_1, x_2, y_1, y_2, z_1, z_2) = x_1 \oplus y_1 z_1 \oplus y_1 z_2 \oplus y_2 z_1 \]

Correctness, Non-completeness
Threshold Implementations

\[
\begin{align*}
S_1(x_1, y_1, z_1, \ldots) & \equiv (a_1, b_1, c_1, \ldots) \\
S_2(x_2, y_2, z_2, \ldots) & \equiv (a_2, b_2, c_2, \ldots) \\
\vdots & \equiv \vdots \\
S_s(x_s, y_s, z_s, \ldots) & \equiv (a_s, b_s, c_s, \ldots)
\end{align*}
\]

\[
(x, y, z, \ldots) \equiv (a, b, c, \ldots)
\]

Correctness, Non-completeness

Need at least \(d+1\) shares for a function of degree \(d\)
Threshold Implementations

\[(x_1, y_1, z_1, \ldots) \oplus (a_1, b_1, c_1, \ldots) = (x, y, z, \ldots)
\]

\[(x_2, y_2, z_2, \ldots) \oplus (a_2, b_2, c_2, \ldots) = (x, y, z, \ldots)
\]

\[\vdots\]

\[(x_s, y_s, z_s, \ldots) \oplus (a_s, b_s, c_s, \ldots) = (x, y, z, \ldots)
\]

Correctness, Non-completeness, Uniformity
Threshold Implementations

Uniformity

A masking $X$ is uniform $\iff \exists$ a constant $p$ s.t. $\forall x$ we have:
if $X \in \text{Sh}(x)$ then $\Pr(X|x) = p$,
else $\Pr(X|x) = 0$. 
Threshold Implementations

Uniformity

A masking $X$ is uniform $\iff \exists$ a constant $p$ s.t. $\forall x$ we have:

- if $X \in \text{Sh}(x)$ then $\Pr(X|x) = p$,
- else $\Pr(X|x)=0$.

If the unshared function is a permutation, the shared function should also be a permutation.
Threshold Implementations

Uniformity

If uniformity cannot be achieved during $S_i$ calculation:
Threshold Implementations

Uniformity

If uniformity can not be achieved during $S_i$ calculation:

- Apply re-masking

\[
\begin{align*}
  s_1 &\rightarrow s_1 \oplus m_1 \\
  s_2 &\rightarrow s_2 \oplus m_2 \\
  s_3 &\rightarrow s_3 \oplus m_1 \oplus m_2
\end{align*}
\]
Threshold Implementations

Uniformity

If uniformity can not be achieved during $S_i$ calculation:

- **Apply re-masking**

  $s_1 \rightarrow s_1 \oplus m_1$

  $s_2 \rightarrow s_2 \oplus m_2$

  $s_3 \rightarrow s_3 \oplus m_1 \oplus m_2$

- **Increase the number of shares**
Threshold Implementations

Uniformity

If uniformity can not be achieved during $S_i$ calculation:

• Apply re-masking

\[
\begin{align*}
  s_1 &\rightarrow s_1 \oplus m_1 \\
  s_2 &\rightarrow s_2 \oplus m_2 \\
  s_3 &\rightarrow s_3 \oplus m_1 \oplus m_2
\end{align*}
\]

• Increase the number of shares

• Decompose the function
Threshold Implementations

Decomposition

\[ F_1(x_1, y_1, z_1, \ldots) \oplus F_2(x_2, y_2, z_2, \ldots) \oplus \cdots \oplus F_s(x_s, y_s, z_s, \ldots) = (x, y, z, \ldots) \]

\[ R_1(a_1, b_1, c_1, \ldots) \oplus R_2(a_2, b_2, c_2, \ldots) \oplus \cdots \oplus R_s(a_s, b_s, c_s, \ldots) = (a, b, c, \ldots) \]

\[ S = G \circ F \]

Separate non-linear functions with registers
TI on AES

TI on AES


- All operations on 3 shares
TI on AES


- All operations on 3 shares
- 5 pipeline stages in S-box
TI on AES


- All operations on 3 shares
- 5 pipeline stages in S-box
- Tower field GF($2^2$)
TI on AES


- All operations on 3 shares
- 5 pipeline stages in S-box
- Tower field GF(2^2)
- Requires extra randomness (48 bits per S-box)
TI on AES

TI on AES


- IDEA: Adjust the number of shares as needed
TI on AES


• IDEA: Adjust the number of shares as needed

• RESULT: Smaller area, less clock cycles, less extra randomness
TI on AES


• IDEA: Adjust the number of shares as needed

• RESULT: Smaller area, less clock cycles, less extra randomness

• Data flow as in EUROCRYPT 2011
TI on AES


• IDEA: Adjust the number of shares as needed
• RESULT: Smaller area, less clock cycles, less extra randomness
• Data flow as in EUROCRYPT 2011
• Linear part: only 2 shares
TI on AES


- IDEA: Adjust the number of shares as needed
- RESULT: Smaller area, less clock cycles, less extra randomness
- Data flow as in EUROCRYPT 2011
- Linear part: only 2 shares
- S-box: 2 to 5 shares
TI on AES


- IDEA: Adjust the number of shares as needed
- RESULT: Smaller area, less clock cycles, less extra randomness
- Data flow as in EUROCRYPT 2011
- Linear part: only 2 shares
- S-box: 2 to 5 shares
- Tower field $\text{GF}(2^4)$
TI on AES

S-box

lin. map

GF(2⁴)

square scaler

GF(2⁴)
multiplier

GF(2⁴)

inverter

GF(2⁴)
multiplier

GF(2⁴)
multiplier

inv. lin. map
TI on AES

S-box

5 shares
TI on AES

S-box

5 shares, 4 input 3 output shares
TI on AES

S-box

5 shares, 4 input 3 output shares, 2 shares
TI on AES

S-box

5 shares, 4 input 3 output shares, 2 shares, 4 shares
TI on AES
S-box

5 shares, 4 input 3 output shares, 2 shares, 4 shares, 3 shares
TI on AES

S-box

5 shares, 4 input 3 output shares, 2 shares, 4 shares, 3 shares

registers after every nonlinear function
TI on AES
S-box

5 shares, 4 input 3 output shares, 2 shares, 4 shares, 3 shares
registers after every nonlinear function
re-masking to change the number of shares
## TI on AES
### Implementation Results

<table>
<thead>
<tr>
<th></th>
<th>State Array</th>
<th>Key Array</th>
<th>S-box</th>
<th>Mix Col.</th>
<th>Cont.</th>
<th>MUXes</th>
<th>Other</th>
<th>Total</th>
<th>cycles</th>
<th>rand bits **</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amir et al.</td>
<td>2529</td>
<td>2526</td>
<td>4244</td>
<td>1120</td>
<td>166</td>
<td>376</td>
<td>153</td>
<td>11114</td>
<td>266</td>
<td>48</td>
</tr>
<tr>
<td>This paper</td>
<td>1698</td>
<td>1890</td>
<td>3708</td>
<td>770</td>
<td>221</td>
<td>746</td>
<td>69</td>
<td>9102</td>
<td>246</td>
<td>44</td>
</tr>
<tr>
<td>This paper*</td>
<td>1698</td>
<td>1890</td>
<td>3003</td>
<td>544</td>
<td>221</td>
<td>746</td>
<td>69</td>
<td>8171</td>
<td>246</td>
<td>44</td>
</tr>
</tbody>
</table>

* compile Ultra  
** per S-box  

- Based on plain Canright S-box (233 GE)  
- Based on plain Amir’s AES (2.4 GE)  
- Keeping Hierarchy
TI on AES
Practical Security Evaluation
TI on AES
Practical Security Evaluation

- Goals:
  1. Verify resistance against first order attacks
  2. Evaluate resistance against HO attacks
TI on AES

Practical Security Evaluation

• Goals:
  1. Verify resistance against first order attacks
  2. Evaluate resistance against HO attacks

• Univariate attacks $\rightarrow$ shares are processed in parallel
TI on AES
Practical Security Evaluation

- **Goals:**
  1. Verify resistance against first order attacks
  2. Evaluate resistance against HO attacks

- **Univariate attacks** → shares are processed in parallel

- **Adversary friendly conditions**
  1. PRNG not active during TI-AES → less noise
  2. Well alignment
  3. Adversary knows the implementation (masks unknown)
TI on AES
Practical Security Evaluation

- PRNG off, first order DPA, HD model at S-box output
- Highest peak 3 cycles later, input MC
TI on AES
Practical Security Evaluation

- PRNG off, first order correlation collision attack
TI on AES
Practical Security Evaluation

- PRNG on, first order DPA / correlation collision attack
- 10 million traces
TI on AES
Practical Security Evaluation

- PRNG on, second order DPA
- HD model at S-box output
TI on AES
Practical Security Evaluation

- PRNG on, second order correlation collision attack
TI on AES
Practical Security Evaluation
TI on AES
Practical Security Evaluation

• Goal 1: verify resistance against first order attacks
  – Evaluation limited by number of traces
  – 10 million traces
Goal 1: verify resistance against first order attacks
- Evaluation limited by number of traces
- 10 million traces

Goal 2: evaluate resistance against HO attacks
- Most trace-efficient second order attack requires 600k traces
- Second Order attacks: Number of traces scales quadratically in the noise standard deviation (we had little noise)
Conclusion
Conclusion

- TI of AES with 8k gates
Conclusion

- TI of AES with 8k gates
- TI can be efficient
Conclusion

- TI of AES with 8k gates
- TI can be efficient
- Adjusting the number of shares!!
Conclusion

- TI of AES with 8k gates
- TI can be efficient
- Adjusting the number of shares!!
- Room for improvement:
  - Solutions to uniformity problems
  - Security against higher order DPA
Conclusion

- TI of AES with 8k gates
- TI can be efficient
- Adjusting the number of shares!!
- Room for improvement:
  - Solutions to uniformity problems
  - Security against higher order DPA
- Consider countermeasures during design process
Thank You!