Countermeasures against DPA: Masking and Threshold Implementation

Begül Bilgin
KU Leuven (COSIC)
Adversary Models

Black-box

Gray-box

White-box

Non-invasive

- Active
  - Temperature or voltage change
  - ... (Light attacks, Laser cutters)

- Passive
  - Side channel analysis
  - ... (Photonic inspection, Probing)

- Timing
  - EM, Power Analysis
    - Simple
    - Differential
Countermeasures against DPA

- Multiple traces
  - same key
  - different plaintext

- Each signal switch consumes different power
  - $0 \rightarrow 0 : \text{low}$
  - $0 \rightarrow 1 : \text{high}$
  - $1 \rightarrow 0 : \text{high}$
  - $1 \rightarrow 1 : \text{low}$

- Leakage of $\text{Enc}(\text{pt}, \text{key})$ creates a fingerprint
  - Simplified using divide and conquer strategy
    (e.g. $\text{out}=\text{Sbox}(\text{inp} \oplus \text{key})$)

- The relation between
  - Instantaneous power consumption
  - Intermediate results of the algorithm
Countermeasures against DPA

• Limit number of encryptions per key
  - Distribution of key is difficult
  - Leakage resilient algorithms
  - Performance drop

• Decrease Signal-to-Noise Ratio (SNR)
  - Decreasing signal (~constant power imp., special cells)
  - Increasing noise (dummy operations, shuffling)

• Randomise the leakage
  - Masking
Randomise the representation and calculation of sensitive variables.

They can be formally proved!

Under certain assumptions
Masking

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

Both linear and nonlinear parts of the algorithm

Operates on sensitive (secret dependent) variable
Many different versions: Boolean, arithmetic, multiplicative, …

Always active
No unmasking!
Boolean Masking
(Representing variables)

\[(x, y, ...) \implies F \implies (a, ...)\]

\[\oplus\]

Random element \((x_1, y_1, ...)\) Mask \(\text{Share 1 (sh1)}\)

\[=\]

Randomised element \((x_2, y_2, ...)\) Masked Data \(\text{Share 2 (sh2)}\)
Boolean Masking
(Representing variables)

Random element \((x_1, y_1, \ldots)\)
Randomised element \((x_2, y_2, \ldots)\)
Mask
Share 1 (sh1)
Share 2 (sh2)

Assumptions:
HW leakage
\[ \mathcal{L}_{tot} = \mathcal{L}_{sh1} + \mathcal{L}_{sh2} \]

<table>
<thead>
<tr>
<th>unshared ((x))</th>
<th>shares ((x_1, x_2))</th>
<th>HW</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1,1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0,1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1,0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Take means
Take difference
Adversary Models

- Black-box
  - Non-invasive
    - Active
      - Temperature or voltage change
    - Passive
      - Side channel analysis
  - (Semi)-invasive
    - Active
      - Light attacks
      - Laser cutters
    - Passive
      - Photonic inspection
      - Probing

- Gray-box
  - Non-invasive
    - Timing
  - (Semi)-invasive
    - EM, Power Analysis
      - Simple
      - Differential

- White-box
  - Non-invasive
  - (Semi)-invasive

\[ L_{imp}(k, p, F, N) \]
Boolean Masking
(Representing variables)

Random element \((x_1, y_1, \ldots)\)

Randomised element \((x_2, y_2, \ldots)\)

Mask

Share 1 \((\text{sh1})\)

Share 2 \((\text{sh2})\)

Assumptions:

HW leakage

\[ \mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{sh1}} + \mathcal{L}_{\text{sh2}} \]

<table>
<thead>
<tr>
<th>unshared ((x))</th>
<th>shares ((x_1, x_2))</th>
<th>HW</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1,1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0,1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0,1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1,0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Take means

Take difference
Boolean Masking
(Calculation with shared variables)

\[
\text{Lin}(\text{in}_1 \oplus \text{in}_2) = \text{Lin}(\text{in}_1) \oplus \text{Lin}(\text{in}_2)
\]
Boolean Masking
(Calculation with shared variables)

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

Component (shared) functions

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

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

Extra mask

\[S(in_1 \oplus in_2) \neq S(in_1) \oplus S(in_2)\]
Boolean Masking
(Calculation with shared variables)

\[(x_1, y_1, z_1, ...) \rightarrow S \rightarrow (a_1, b_1, c_1, ...)\]

\[(x_2, y_2, z_2, ...) \rightarrow S' \rightarrow (a_2, b_2, c_2, ...)\]
Boolean Masking
(Calculation with shared variables)

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

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

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

What if you do NOT want to store component functions in memory?
e.g. Hardware implementation

Focus only on **AND** and **XOR** gates
**Boolean Masking**  
*(Calculation with shared variables)*

\[
(x_1, y_1) \rightarrow S \rightarrow (a_1) \\
(x_2, y_2) \rightarrow S' \rightarrow (a_2)
\]

<table>
<thead>
<tr>
<th>(x,y)</th>
<th>a=x&amp;y</th>
<th>(x1,y1)</th>
<th>(x2,y2)</th>
<th>a1=x1&amp;y1</th>
<th>a2=x1&amp;y2+y2&amp;x1+x2&amp;y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>0</td>
<td>0,0</td>
<td>0,0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,1</td>
<td>0,1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0</td>
<td>1,0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,1</td>
<td>1,1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0,1</td>
<td>0</td>
<td>0,0</td>
<td>0,1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,1</td>
<td>0,0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0</td>
<td>1,1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,1</td>
<td>1,0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1,0</td>
<td>0</td>
<td>0,0</td>
<td>1,0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,1</td>
<td>1,1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0</td>
<td>0,0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,1</td>
<td>0,1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1,1</td>
<td>1</td>
<td>0,0</td>
<td>1,1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,1</td>
<td>1,0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0</td>
<td>0,1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,1</td>
<td>0,0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
**Boolean Masking**

*(Calculation with shared variables)*

\[
\begin{align*}
S & \quad (x_1, y_1) \\
S' & \quad (x_2, y_2)
\end{align*}
\]

\[
\begin{align*}
S & \quad (a_1) \\
S' & \quad (a_2)
\end{align*}
\]

**Extra mask \((z_1)\)**

---

<table>
<thead>
<tr>
<th>((x,y))</th>
<th>(a=x&amp;y)</th>
<th>((x_1,y_1))</th>
<th>((x_2,y_2))</th>
<th>(a_1=x_1&amp;y_1)</th>
<th>(a_2=x_1&amp;y_2+x_2&amp;y_1+x_2&amp;y_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>0</td>
<td>0,0</td>
<td>0,0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,1</td>
<td>0,1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0</td>
<td>1,0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,1</td>
<td>1,1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0,1</td>
<td>0</td>
<td>0,0</td>
<td>0,1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,1</td>
<td>0,0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0</td>
<td>1,1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,1</td>
<td>1,0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1,0</td>
<td>0</td>
<td>0,0</td>
<td>1,0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,1</td>
<td>1,1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0</td>
<td>0,0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,1</td>
<td>0,1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1,1</td>
<td>1</td>
<td>0,0</td>
<td>1,1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0,1</td>
<td>1,0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,0</td>
<td>0,1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1,1</td>
<td>0,0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Boolean Masking
(Calculation with shared variables)

1) \[ S(x_1,y_1) = x_1 y_1 \]
   \[ S'(x_1,y_1,x_2,y_2) = x_1 y_2 \oplus x_2 y_1 \oplus x_2 y_2 \]

2) \[ S(x_1,y_1,z_1) = x_1 y_1 \oplus z_1 \]
   \[ S'(x_1,y_1,x_2,y_2,z_1) = x_1 y_2 \oplus x_2 y_1 \oplus x_2 y_2 \oplus z_1 \]

3) \[ S(z_1) = z_1 \]
   \[ S'(x_1,y_1,x_2,y_2,z_1) = x_1 y_1 \oplus (x_1 y_2 \oplus (x_2 y_1 \oplus (x_2 y_2 \oplus z_1))) \]

Assumptions:
HW leakage
In and out of \( F \) leaks

Calculation of \( F \), i.e. intermediates of component functions, leak

Trichina AND gate
Trichina AND gate

\[ a_1 = x_1 y_1 \oplus (x_1 y_2 \oplus (x_2 y_1 \oplus (x_2 y_2 \oplus z_1))) \]

\[ a_2 = z_2 \]

Assumptions:
- HW leakage
- In and out of \( F \) leaks
- Calculation of \( F \) leaks

Implicit assumption:
- Gates are perfect

Reality:
- CMOS gates are not perfect, i.e. they can glitch
Glitches

\[ a_1 = x_1 y_1 \oplus (x_1 y_2 \oplus (x_2 y_1 \oplus (x_2 y_2 \oplus z_1))) \]

\[ a_2 = z_2 \]
Glitches

\[ a_1 = x_1y_1 \oplus (x_1y_2 \oplus (x_2y_1 \oplus (x_2y_2 \oplus z_1))) \]

\[ a_2 = z_2 \]
Short Mid Summary

Your countermeasure is as secure as the strength of your assumptions
Threshold Implementations
Threshold Implementations

- Introduced in 2006 by Nikova et al.
- Provides provable security under the assumptions
  - HW leakage
    - $L_{tot} = L_{sh1} + \ldots + L_{sh_s}$
  - Imperfect gates
- Shown to be efficient
Threshold Implementations

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

\[S_1 \rightarrow (a_1, b_1, c_1, \ldots) \oplus S_2 \rightarrow (a_2, b_2, c_2, \ldots) \oplus \ldots \oplus S_s \rightarrow (a_s, b_s, c_s, \ldots) = (a, b, c, \ldots) \]

Correctness
Threshold Implementations

\[
S_1 (x_1, y_1, z_1, ...) \oplus S_2 (x_2, y_2, z_2, ...) \oplus \cdots \oplus S_s (x_s, y_s, z_s, ...) = (x, y, z, ...)
\]

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

Correctness, Non-completeness
Threshold Implementations

\[
\begin{align*}
(x_1, y_1, z_1, \ldots) &\rightarrow L_1 &\rightarrow (a_1, b_1, c_1, \ldots) \\
L_2 &\rightarrow (a_2, b_2, c_2, \ldots) \\
L_s &\rightarrow (a_s, b_s, c_s, \ldots) \\
\end{align*}
\]

\[
\begin{align*}
(x, y, z, \ldots) &\rightarrow L_1 \\
L_2 &\rightarrow (a, b, c, \ldots) \\
\end{align*}
\]

Correctness, Non-completeness
Threshold Implementations

Correctness, Non-completeness
Threshold Implementations

\[ S(x, y) = xy = (x_1 \oplus x_2 \oplus x_3)(y_1 \oplus y_2 \oplus y_3) \]

\[ S_1(x_2, x_3, y_2, y_3) = x_2y_2 \oplus x_2y_3 \oplus x_3y_2 \]
\[ S_2(x_1, x_3, y_1, y_3) = x_3y_3 \oplus x_3y_1 \oplus x_1y_3 \]
\[ S_3(x_1, x_2, y_1, y_2) = x_1y_1 \oplus x_1y_2 \oplus x_2y_1 \]

Correctness, Non-completeness
Threshold Implementations

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

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

**Correctness, Non-completeness**

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

S-box is typically fixed and has high degree

$S = G \circ F$

Separate non-linear functions with registers
Threshold Implementations

\[(x_1, y_1, z_1, ...) \quad + \quad (a_1, b_1, c_1, ...) \quad = \quad (x, y, z, ...)\]

\[(x_2, y_2, z_2, ...) \quad + \quad \vdots \quad + \quad (a_2, b_2, c_2, ...) \quad = \quad (a, b, c, ...)\]

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 $\Pr(X|x)=0$.

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

Uniformity

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

- **Apply re-masking**
  
  $a_1 \rightarrow a_1 \oplus m_1$
  
  $a_2 \rightarrow a_2 \oplus m_2$
  
  $a_3 \rightarrow a_3 \oplus m_1 \oplus m_2$

- **Increase the number of shares**

- ** Decompose the function**
Applications of TI

• **First-order TI**
  • **Present** [JoC’11]
  • All 3- and 4-bit perm. [CHES’12]
  • 5-bit and 6-bit special perm. [C&C’15]
  • KECCAK [Cardis’14]
  • AES [EC ’11, IEEE trans’15]

• **Higher-order TI**
  • KATAN
  • AES S-box
Conclusion
There is no UNIQUE answer!!!
**Reality**

*HD leakage, adversary is capable of DPA, limited to analysis on single moment in time,…*
Thank you!