Masking as a Side-Channel Countermeasure in Hardware


Amir Moradi
Ruhr-Universität Bochum
Agenda

- Physical Attacks and Side-Channel Analysis Attacks
- Measurement setup
- Power Analysis Attacks
- Countermeasures
  - Hiding
  - Masking
    - In hardware
Physical Attacks

are classified into groups:

- From the attacker’s behavior point of view
  - Passive Attacks
    - The device works normally; the attacker just listens
  - Active Attacks
    - The attacker makes trouble for the device

- From invasiveness point of view
  - Invasive Attacks
    - Without any limit; the strongest attacker model
  - Semi-Invasive Attacks
    - The device is depacked, but no electrical contact
      - to read/induce faults
  - Non-Invasive Attacks
    - No modification at all
### Physical Attacks (examples)

<table>
<thead>
<tr>
<th></th>
<th>Passive</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Invasive</strong></td>
<td>Probing</td>
<td>Forcing Permanent Faults</td>
</tr>
<tr>
<td><strong>Semi-Invasive</strong></td>
<td>Optical Inspection</td>
<td>Light Attack Radiation Attack</td>
</tr>
<tr>
<td><strong>Non-Invasive</strong></td>
<td>Side-Channel Attacks</td>
<td>Clock Glitch Power Glitch Temperature</td>
</tr>
</tbody>
</table>

*Side-Channel Attacks: Power Analysis/EM Analysis/Timing Analysis*
Power Analysis Attacks

- In principle, power analysis (PA) attacks are known as the most powerful side-channel attack.
- They are able to break any implementation observing its instantaneous power consumption during the computation of the cryptographic algorithm.
  – unless special/dedicated countermeasure has been considered during the design of the target device.
- The cost of the attack depends on the target device, it might be very cheap using a 1.5K oscilloscope, or it may need special measurement equipments which cost more than 30K
Measurement Setup
Examples for EM Setup
Examples for EM Setup
Characteristics of Single Points

- Usually the clock cycles are clearly distinguishable.
- If a measurement for the same operation and the same data is repeated a couple of times, the results will not be exactly similar because of noise that we call electronic noise.

![Graph showing voltage over time]
Electronic Noise

- has usually normal distribution
  - Is defined by mean $\mu$ and variance $\sigma^2$ as

$$f(x) = \frac{1}{\sqrt{2 \cdot \pi \cdot \sigma}} \cdot \exp\left(-\frac{1}{2} \cdot \left(\frac{x - \mu}{\sigma}\right)^2\right)$$

- Above example: at 362ns, $\mu=111.86$, $\sigma=1.32$
Data Dependency

- Histogram of a point of the power traces when the LSB of the processed data is “1” or “0” (separately)

\[ \mu = 138 \text{ [mV]} \]

\[ \mu = 140 \text{ [mV]} \]

- In most cases, it is valid to approximate the distribution of the data-dependent part of the power consumption of a cryptographic device by a normal distribution [if the processed data is uniformly distributed] [DPABook]
DPA

- DPA attacks mainly exploit the data-dependent power consumption.
- The basic idea of the attack is to analyze the power consumption of a device at a fixed moment of time for different plain-/ciphertexts of encryption or decryption.
- Considering a single point of power traces, the attack scenario is exactly the same as the timing attacks which we have seen.
- Then, the attack runs on each point of the power traces independently and recovers not only the correct secret but also the correct time instances that the target intermediate value is processed.
DPA (cont’d)

- To run a DPA we need to have some knowledge about the leakage model and the design architecture of the target device (similar to most of the side-channel attacks).
- Suppose that we have an AES implementation where the power values are dependent on the intermediate values (a general statement).
- Attack strategy:
  - The attacker collected a set of power values at a specific time instance \( V= (v_1, v_2, \ldots, v_n) \) for \( n \) encryptions of plaintexts \( P= (p_1, p_2, \ldots, p_n) \).
  - The attack is divided into 16 parts, each key byte separately divide-and-conquer scheme
  - We consider only one key byte, let say first key byte
- If each bit of the intermediate value, e.g., Sbox output, affects the power values, we can make an attack like this (considering the LSB of the Sbox output):
DPA Algorithm

- The attacker guesses the key byte as $K$
  - She makes two empty groups: group1 and group0
  - for $i=1$ to $n$, she computes $\text{Sbox}(\text{firstbyte}(p_i) \text{ XOR } K)$
    - if the target bit (LSB here) of the result is “1” the corresponding power value ($v_i$) goes to group1 otherwise to group0
  - In average the values of group1 should be different than that of group0 if $K$ is the correct guess
    - Averaging helps. Then, the difference between the average of two groups is assigned to the guessed key byte $K$ (the same concept as $t$-test)
- This is repeated for all key candidates ($K=0$ to 255)
  - The highest difference between the averages shows the most probably value for the guessed key byte
- This scheme is called “classical DPA” or “difference of means”
Differential Power Analysis (DPA)

- Classifying the power consumption values in two groups
- Comparing e.g., mean of the groups

\[
\begin{array}{cccccccc}
\text{Diff. of Means} & & & & & & & \\
0.001 & 0.002 & & & & 0.001 \\
\vdots & & & & & & & \\
0.020 & & & & & & & \\
\vdots & & & & & & & \\
0.001 & & & & & & & \\
\end{array}
\]

\[
\begin{array}{cccccccc}
p & 12 & 3d & 78 & \cdots & f9 & ab & 3d \\
power & 0.12 & 0.01 & 0.14 & \cdots & 0.20 & 0.06 & 0.02 \\
\text{Sbox} & \begin{array}{cccccccc}
\{k=00\} S & c9 & 27 & bc & \cdots & 99 & 62 & 27 \\
\text{LSB} & 1 & 1 & 1 & \cdots & 1 & 0 & 1 \\
\{k=01\} S & 7d & eb & b6 & \cdots & 41 & ac & eb \\
\text{LSB} & 1 & 1 & 0 & \cdots & 1 & 0 & 1 \\
\{k=ff\} S & 55 & 25 & 17 & \cdots & 6f & 20 & 25 \\
\text{LSB} & 1 & 1 & 1 & \cdots & 1 & 0 & 1 \\
\end{array}
\end{array}
\]
DPA (cont’d)

- The model we used was a bit of Sbox output, but it can be anything depending on the design architecture.
- For software implementations (µCs) because of the pre-charged buses selecting a bit of the intermediate value according to the design (Sbox or Ttable) should work
  - Not always all bits (LSB to MSB) work, each bit should be checked independently.
- But for hardware implementations, the design architecture should be carefully studied
  - Then the target bit which can be bit flip (changing/unchanging a register bit) should be selected.
DPA (cont’d)

- Other types of DPA considering more bits of the intermediate values have been introduced
- Then, instead of two groups the power values are categorized in more groups, and variance of means can be used instead of difference of means to distinguish the correct guess
- On the other hand, we can do better if the leakage model of the target device is known
- In order to do so, a tool to compare the leakage model and power values should be used — “Correlation Coefficient”
Correlation

- The correlation coefficient measures how linearly dependent two random variables (two vectors) are

\[
\rho = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}
\]

Discrete equation (*Pearson Correlation*):

\[
r = \frac{\sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n} (X_i - \bar{X})^2 \sum_{i=1}^{n} (Y_i - \bar{Y})^2}}
\]

The correlation coefficient is in the range of \([-1, 1]\), where 1 indicates perfect linear dependency, -1 perfect inverted linear dependency and 0 indicates no linear dependency.
CPA

- How to use correlation in power analysis?
- We need a leakage model
  - How the intermediate values affect the power consumption
  - HW usually for µCs
  - HD usually for hardware implementations
- If the leakage model and the assumed design architecture are correct, power values are highly correlated with the result of the leakage model given the intermediate values
CPA Algorithm

- The attacker guesses the key byte as $K$
  - for $i=1$ to $n$, she computes $\text{Sbox}(\text{firstbyte}(p_i) \text{ XOR } K)$
    - she applies the leakage model, e.g., HW, on above computed Sboxes and makes a vector $HV=(h_{v1}, h_{v2}, \ldots, h_{vn})$ of hypothetical power values
  - She uses correlation to compare the power values of one time instance $V=(v_1, v_2, \ldots, v_n)$ and the hypothetical power values $HV=(h_{v1}, h_{v2}, \ldots, h_{vn})$
- This is repeated for all key candidates ($K=0$ to $255$)
  - The highest (or lowest) correlation coefficient shows the most probable value for the guessed key byte
- This also should be repeated for all time instances (all points of power traces)
Correlation Power Analysis (CPA)

- Hypothetical model for power consumption
- Compare the model with side-channel leakage (power)

\[
P \xrightarrow{Sbox} \begin{array}{cccc}
12 & 3d & 78 & \ldots \\
0.12 & 0.01 & 0.14 & \ldots \\
\end{array}
\begin{array}{ccc}
f9 & ab & 3d \\
0.20 & 0.06 & 0.02 \\
\end{array}
\]

\[
[k=00] S \begin{array}{ccc}
c9 & 27 & bc \\
4 & 4 & 5 \\
\end{array}
\begin{array}{ccc}
99 & 62 & 27 \\
4 & 3 & 4 \\
\end{array}
\]

\[
[k=01] S \begin{array}{ccc}
7d & eb & b6 \\
6 & 6 & 5 \\
\end{array}
\begin{array}{ccc}
41 & ac & eb \\
2 & 4 & 6 \\
\end{array}
\]

\[
[k=ff] S \begin{array}{ccc}
55 & 25 & 17 \\
4 & 3 & 4 \\
\end{array}
\begin{array}{ccc}
6f & 20 & 25 \\
6 & 1 & 3 \\
\end{array}
\]

Correlation

\[
\begin{array}{c}
0.011 \\
0.060 \\
0.231 \\
0.095 \\
\end{array}
\]
Countermeasures

- **Hiding**
  - Randomization
    - Noise addition techniques: dummy instructions, clock randomization, shuffling
  - Equalization
    - Filtering
      - Dual-rail circuits, random precharge

- **Masking**
  - Boolean secret sharing
  - Multiplicative secret sharing
  - Combination of / conversion between Boolean and multiplicative masking
  - Threshold implementation
Hiding

- To hide the power consumption of key-dependent cryptographic computations it is necessary to decrease the signal to noise ratio (SNR)
  - Either by increasing the noise or by decreasing the signal

- Randomization techniques attempt to randomize the power consumption by constantly changing the execution order or by generating noise directly
  - increasing the noise

- Equalization techniques attempt to achieve an equal power consumption at each moment in time, independent of the performed operation
  - decreasing the signal

\[ SNR = \frac{\text{var}(\text{signal})}{\text{var}(\text{noise})} \]
Masking

- Masking randomizes the intermediate values of a cryptographic computation to avoid dependencies between these values and the power consumption.

- It is applied on an algorithmic level (unlike hiding):
  - does not rely on the power consumption characteristics of the device.

- Each intermediate value is concealed by a random mask that is different for every execution.

- Basically corresponds to one of the following two secret sharing schemes with two shares:
  - *Boolean secret sharing*
  - *Multiplicative sharing*
Boolean secret sharing

- First order boolean secret sharing (two shares):
  - secret: $x$
  - random: $m$
  - shares: $(x_1, x_2)$
  - $x_1 = x \oplus m$
  - $x_2 = m$
  - $x_1 \oplus x_2 = x$

  - One needs to know share $x_1$ and $x_2$ to compute secret $x$
  - neither of them alone provides enough information

- Linear Function $F$
  - Definition: $F(x \oplus z) = F(x) \oplus F(z)$
  - Boolean share before $F$:
    - $(x_1, x_2)$ with $x_1 \oplus x_2 = x$
  - Boolean shares after $F$:
    - $(F(x_1), F(x_2))$ with
      - $F(x_1) \oplus F(x_2) = F(x_1 \oplus x_2) = F(x)$
Boolean secret sharing

- All linear functions in a cipher can be masked by boolean secret sharing, since the mask is changed in an easily computable way
  - AES AddRoundKey, ShiftRows, MixColumns
- Non-linear functions are more difficult to mask
  - AES SubBytes is non-linear
    • 16 modified AES Sboxes must be computed, stored and used
      - Very inefficient (infeasible on a smartcard for example)
    • But SubBytes is based on the computation of the multiplicative inverse in $GF(2^8)$
      - Compatible to multiplicative masking:
        $$(x \times m)^{-1} = x^{-1} \times m^{-1}$$
Multiplicative secret sharing

- First order multiplicative secret sharing (two shares):
  - secret: \( x \)  
  - shares: \( (x_1, x_2) \)  
  - random: \( m \)  
  \[
  x_1 = x \cdot m \\
  x_2 = m^{-1} \\
  x_1 \cdot x_2 = x
  \]

- One needs to know share \( x_1 \) and \( x_2 \) to compute secret \( x \)

- Problem: The value zero is always masked to the value zero
  - Zero value model attacks possible

- AES Sbox is an affine mapping of the multiplicative inverse:
  \[
  S(x) = A(x^{-1}) = L(x^{-1}) \oplus c \\
  x_1^{-1} = (x \cdot m)^{-1} = x^{-1} \cdot m^{-1}
  \]

- Idea: Transform the boolean mask into a multiplicative mask, perform the inversion, transform the mask back and perform the affine mapping
Conversion between the masking schemes

- This modified inversion in $GF(2^8)$ computes $x^{-1} \oplus m$ from $x \oplus m$
- The mask does not change and the affine mapping can be applied with boolean masking like below:

$$A(x^{-1} \oplus m) = L(x^{-1} \oplus m) \oplus c = L(x^{-1}) \oplus c \oplus L(m) = S(x) \oplus L(m)$$
AES example with boolean masking only

plaintext $p, m$ and $m'$ random masks, $m'' = m \oplus MC(SR(m'))$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Share 1</th>
<th>Share 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>$p_1 = p \oplus m$</td>
<td>$p_2 = m$</td>
</tr>
<tr>
<td>AddRoundKey</td>
<td>$s_1 = p_1 \oplus k$</td>
<td>$s_2 = p_2$</td>
</tr>
<tr>
<td>SubBytes</td>
<td>for $j = 0:15$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$s_{1j} = S_{j'}(s_{1j})$</td>
<td>$s_2 = m'$</td>
</tr>
<tr>
<td></td>
<td>end</td>
<td></td>
</tr>
<tr>
<td>ShiftRows</td>
<td>$s_1 = SR(s_1)$</td>
<td>$s_2 = SR(s_2)$</td>
</tr>
<tr>
<td>MixColumns</td>
<td>$s_1 = MC(s_1)$</td>
<td>$s_2 = MC(s_2)$</td>
</tr>
<tr>
<td>END</td>
<td>$s_1 = s_1 \oplus m''$</td>
<td>$s_2 = s_2 \oplus m''$</td>
</tr>
</tbody>
</table>
AES example with boolean masking only

- Precomputation of $S'_j$:
  
  $\text{for } j = 0:15$
  
  $\text{for } i = 0:255$
  
  $S'_j(i \oplus m_j) = S(i) \oplus m_j'$
  
  $\text{end}$
  
  $\text{end}$

- $m \neq m'$ is a necessary condition! Why?
  
  – Assume $m = m'$:
    
    $s_1 = S'(s_1)$  
    $s_{1\text{new}} = S(p \oplus k) \oplus m$
    
    $\text{HD}(s_1, s_{1\text{new}}) = \text{HW}(s_1 \oplus s_{1\text{new}}) = \text{HW}(p \oplus k \oplus S(p \oplus k))$
    
  – Hamming distance would be independent of the mask
Univariate vs. Multivariate Attacks

DPA/CPA/MIA

squaring: 2\(^{nd}\) order univariate

bivariate MIA

combining: DPA/CPA

multiply: 2\(^{nd}\) order bivariate

addition: 1\(^{st}\) order bivariate
Masking in Hardware

- Pre-computing the masked tables in software
  - Sequential operations, Time consuming, Low efficiency
- High efficiency is desired in HARDWARE
  - amongst the main reasons
- ad-hoc/heuristic schemes
- Processing the mask ($m$) and masked data ($i\oplus m$) simultaneously
  - joint distribution of SC leakages mainly because of GLITCHES
  - possible attacks
- Systematic schemes
  - Threshold Implementation, Security against 1st order attacks
Former Attempts

A Side-Channel Analysis Resistant Description of the AES S-box @ FSE 2005
Former Attempst

A Side-Channel Analysis Resistant Description of the AES S-box @ FSE 2005
Former Attempts

A Side-Channel Analysis Resistant Description of the AES S-box @ FSE 2005

GF(16) to GF(256)

\( A^4 + X \)
Former Attempts

The introduction of the toggle-count model

Successful CPA attack on a masked implementation using the revealed toggle-count model
Threshold implementation

- Let us denote an intermediate value of a cipher by $x$, made of $s$ single-bit signals \(<x_1, \ldots , x_s>\).

- The underlying concept of TI is to use Boolean masking to represent $x$ in a shared form $(x^1, \ldots , x^n)$, where $x = \bigoplus x^i$ and each $x^i$ similarly denotes a vector of $s$ single-bit signals.

- A linear function $l(.)$ can be trivially applied over the shares of $x$ as $l(x) = \bigoplus l(x^i)$.

- Realizing non-linear functions (Sbox) over Boolean masked data is challenging.

- If the algebraic degree of the Sbox is denoted by $t$ and the desired security order by $d$, the minimum number of shares to realize the Sbox under the TI settings is $n = t \cdot d + 1$. 
Threshold implementation

- The shared Sbox provides the output \( y = S(x) \) in a shared form \((y^1, ..., y^m)\) with at least \( m = \binom{n}{t} \) shares.

- For sure, it should satisfy \( y = \bigoplus y^j \)
  - Known as **correctness** property

- A smallest non-linear function, \( t = 2 \).
  - So, at least \( n = 3 \) shares are required (for first-order security).

- Each output share \( y^j \) is given by a component function \( f^j(.) \) over a subset of the input shares. To achieve the \( d \)th-order security, any \( d \) selection of the component functions \( f^j(.) \) should be independent of at least one input share.
  - Known as **non-completeness** property
Threshold implementation

- To achieve non-completeness, consider a bijective Sbox:

\[ x_1 \oplus x_2 \oplus x_3 = x \]
\[ y_1 \oplus y_2 \oplus y_3 = y \]

- Example:

\[ x = (a, b, c, d) \quad y = (e, f, g, h) \]
\[ S_1(a, b, c, d) = e \]
\[ e = a \oplus bc \oplus d = a_1 \oplus a_2 \oplus a_3 \oplus (b_1 \oplus b_2 \oplus b_3)(c_1 \oplus c_2 \oplus c_3) \oplus d_1 \oplus d_2 \oplus d_3 \]
\[ e = a_1 \oplus a_2 \oplus a_3 \oplus b_1 \oplus b_2 c_1 \oplus b_2 c_2 \oplus b_2 c_3 \oplus b_3 c_1 \oplus b_3 c_2 \oplus b_3 c_3 \oplus d_1 \oplus d_2 \oplus d_3 \]

Each \( f \) should be independent of one share.
Threshold implementation

- **Example (continued)**

\[
\begin{align*}
& f_1 = b_2c_3 \oplus b_3c_2 \oplus a_2 \oplus d_2 \oplus b_2c_2 \\
& f_2 = b_3c_1 \oplus b_1c_3 \oplus a_3 \oplus d_3 \oplus b_3c_3 \\
& f_3 = b_1c_2 \oplus b_2c_1 \oplus a_1 \oplus d_1 \oplus b_1c_1
\end{align*}
\]

- It is secure against first-order attacks
  - Can be attacked by second order attacks with the *centered product* (multivariate) or with the *mean-free square* (univariate)

\[
\text{can be arbitrarily distributed among two component functions}
\]

- are clear where to go (which \( f \))
Threshold implementation

- The security of masking schemes is based on the uniform distribution of the masks.
  - As the last property, the design should be **uniform**, as it is used as input in further parts of the implementation.

- Suppose that for a certain input \( x \) all possible sharings \( \chi = \{(x^1, \ldots, x^n)|x = \oplus x^i\} \) are given to a TI Sbox.

- The set made by the output shares, \( \{(f^1(.), \ldots, f^m(.))|(x^1, \ldots, x^n) \in \chi\} \) should be drawn **uniformly** from the set \( \{(y^1, \ldots, y^m)|S(x) = y = \oplus y^i\} \)

- For a bijective Sbox \( n = m \) each \( (x^1, \ldots, x^n) \) should be mapped to a unique \( (y^1, \ldots, y^n) \).
  - It is sufficient to check whether the TI Sbox forms a bijection with \( s \cdot n \) input (and output) bit length.
Threshold implementation

- Realization of the masked Sboxes with high algebraic degree is challenging.

- If $t > 2$, we can decompose the Sbox into quadratic bijections.
  - If we can write $S : G \times F$, where both $G$ and $F$ are quadratic bijections, we are able to implement the 1st-order TI of $F$ and $G$ with the minimum number of shares $n=3$.

- Such a construction needs register between masked $F$ and $G$ to isolate the corresponding glitches.

- Achieving non-completeness and correctness is easy
  - The challenge is to uniformity.
Threshold implementation

- Remasking: heuristics to satisfy *uniformity*
  - extra fresh randomness required

Remasking scheme for a 3-share case
Threshold implementation

Pushing the Limits: A Very Compact and a Threshold Implementation of AES @ EUROCRYPT 2011
Threshold implementation

- For *uniformity* we can apply *affine* equivalence property.

- We find affine functions $A_1$ and $A_2$ in such a way that $F : A_2 \times Q \times A_1$

- If we are able to represent a *uniform* sharing of the quadratic function $Q$, applying $A_1$ on all input shares, and $A_2$ on all output shares gives us a uniform sharing of $F$.

- There are not many of such different quadratic $Q$ functions (up to affine equivalent)
  - 5 in case of 4-bit bijections
  - So, a uniform sharing of all quadratic 4-bit functions can be obtained.

Threshold Implementations of Small S-boxes @ Cryptography and Communications 7(1): 3-33 (2015)
any questions?

amir.moradi@rub.de

Embedded Security Group
Ruhr-Universität Bochum
Germany