

Secure Implementation of Cryptographic Algorithms

Chapter 5 Differential Power Analysis (DPA)

Wieland Fischer & Berndt Gammel
Infineon Technologies

5.4. DPA Countermeasures

5.4.1. Principles of DPA countermeasures

5.4.2. Algebraic Countermeasures

Secret Sharing

Boolean Masking

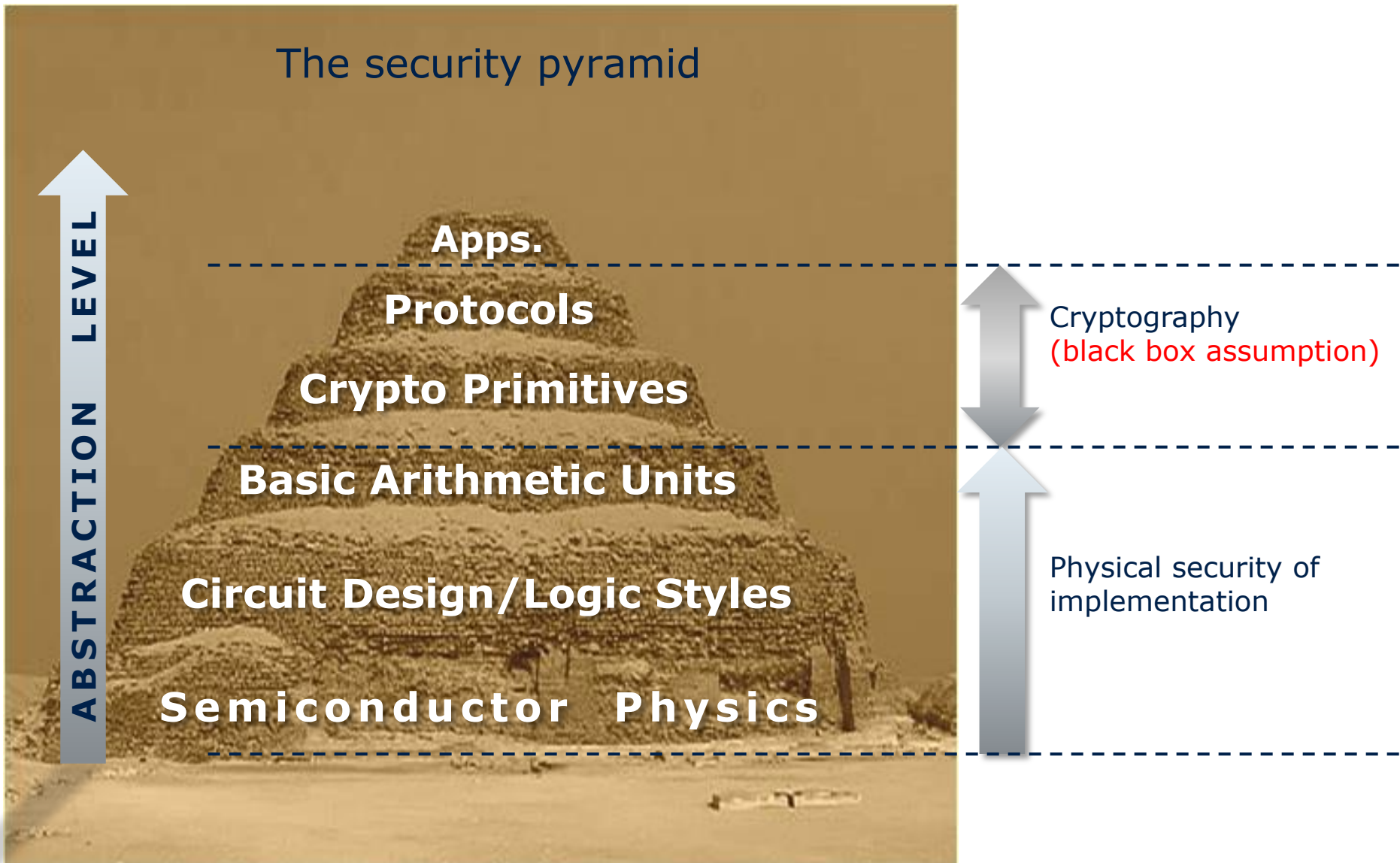
5.4.3. Boolean Masking for AES

5.4.4. Blinding Methods for RSA

5.4.5. Hiding Countermeasures

5.4.1. Principles of DPA countermeasures

The security pyramid



5.4.1. Principles of DPA countermeasures

■ Countermeasures on the level of *cryptographic protocols*

- Assumption: The leakage of the key is *bounded* (the key can be used at least once without revealing the entire key to the attacker).
- Based on this assumption it is possible to develop protocols (e.g. for authentication & secure messaging) where the key is continuously updated such that the key cannot be revealed by DPA [K99], [K05], [GFM09], [MSGR10].
 - Disadvantage: Existing crypto protocols usually cannot be modified easily, if there is a large user base.
 - Advantage: No expensive DPA countermeasures (performance, code, area) need to be implemented on the lower levels.
 - Remark 1: Countermeasures against attacks on a single encryption (TA, SPA) still must be implemented.
 - Remark 2: Just limiting the number of e.g. failing authentication steps is usually not acceptable as a countermeasure.
- A first practical system employing a continuous key update scheme is the CIPURSE™ protocol provided by the OSPT Alliance for electronic fare collection in public transport. [OSPT]

5.4.1. Principles of DPA countermeasures

■ Countermeasures on the level of *cryptographic primitives*

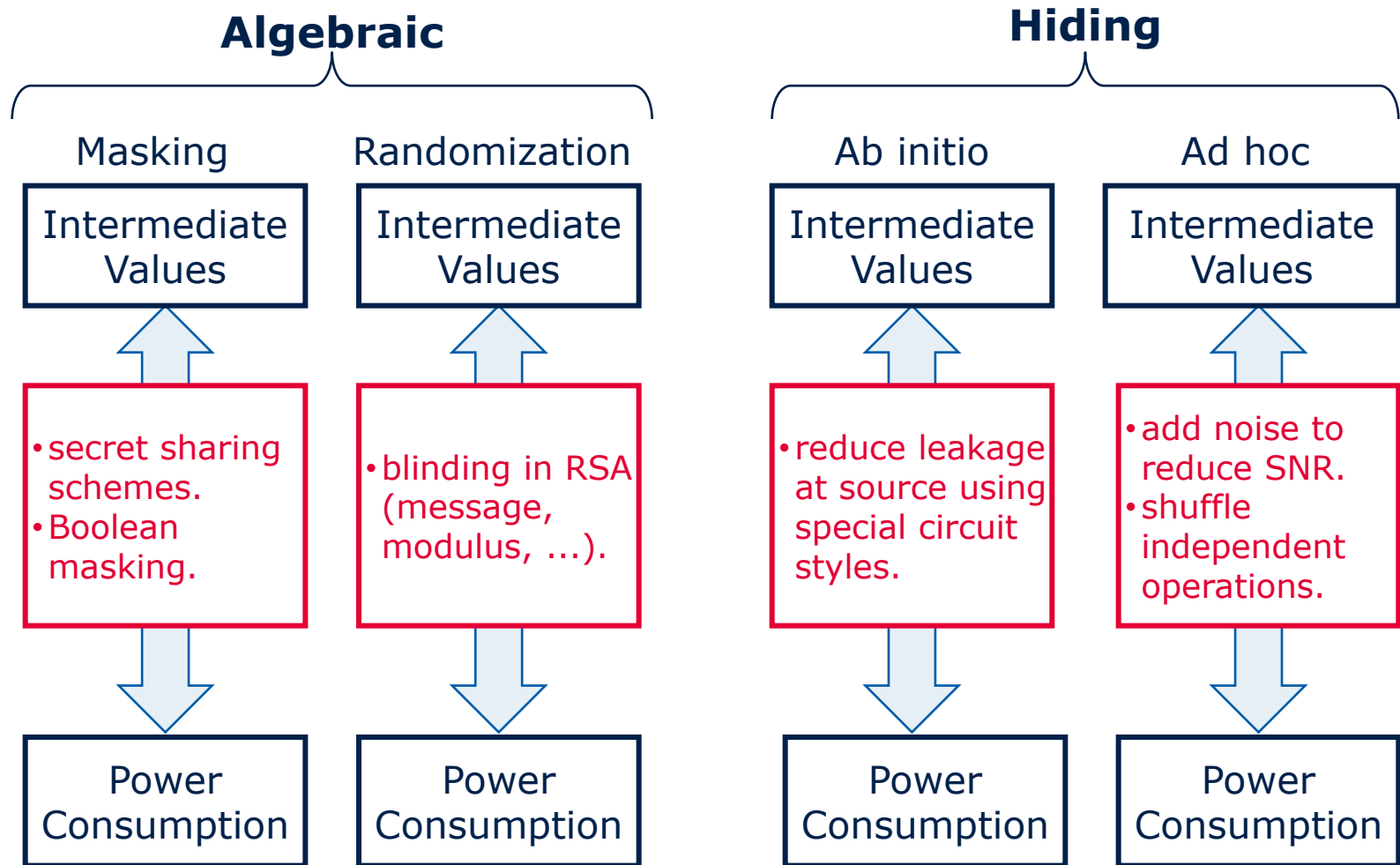
- A new research direction called “Provable Security against Physical Attacks” or “Leakage Resilient Cryptography” is evolving, which aims at developing new crypto primitives which are inherently secure against bounded leakage.
[DP08]

→ Assumption as on previous page: The leakage of the key is *bounded* (the key can be used at least once without revealing the entire key to the attacker).

→ So far no practical schemes have been developed, but the research is just getting started.

5.4.1. Principles of DPA countermeasures

- Countermeasures on the level of *implementation/circuit design*.



5.4.2. Algebraic countermeasures

Secret Sharing

- Basic idea of algebraic countermeasures:
Randomize all intermediate results of an algorithm with a random number that is unknown to the attacker. As the random number is not known, the attacker cannot exploit the leakage of the device.
- We encountered already an example!
In Chap 5.2. “DPA on RSA with CRT I”: DPA on one of the exponentiations $S_p = m^{d_p} \bmod p$ or $S_q = m^{d_q} \bmod q$ is not possible, because the attacker does not know the factors p, q .
 - CRT works with a “random” factorization of N .
The two *shares* p, q together uniquely define the secret N .
- Secret sharing schemes:
The secret (and intermediate variables) is split into two or more *shares* at random, such that the attacker cannot obtain the secret if he does not know all the shares.

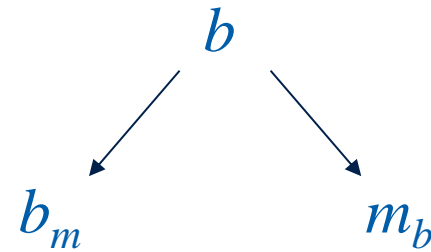
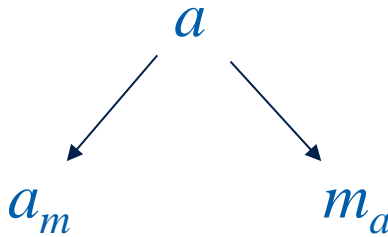
The simplest secret sharing scheme is the *Boolean Masking*.

5.4.2. Algebraic countermeasures

Boolean Masking

■ Boolean Masking:

Split all variables $a, b, \dots \in \mathbf{F}_2^n$ into two shares:



Choose $m_a, m_b \in \mathbf{F}_2^n$ independently and uniformly distributed “masks”.

$$a \rightarrow (a_m, m_a) = (a \oplus m_a, m_a)$$

$$a = a_m \oplus m_a$$

$$b \rightarrow (b_m, m_b) = (b \oplus m_b, m_b)$$

$$b = b_m \oplus m_b$$

5.4.2. Algebraic countermeasures

Boolean Masking

- Assume that all intermediate values v of a cryptographic algorithm are masked, i.e. each value v is represented by v_m and m_v , such that
$$v = v_m \oplus m_v.$$
- It holds that
 - v_m is statistically independent of v .
 - m_v is statistically independent of v .
 - The power consumption $P_{\text{exp}}(v_m)$ that is caused by the processing of v_m is statistically independent of v .
 - The power consumption $P_{\text{exp}}(m_v)$ that is caused by the processing of m_v is statistically independent of v .
- If the attacker cannot or does not exploit the $P_{\text{exp}}(v_m)$ and $P_{\text{exp}}(m_v)$ simultaneously, the implementation is secure against DPA.

5.4.2. Algebraic countermeasures

Boolean Masking

■ Remark 1:

In practice, “independently and uniformly distributed *mask*” means, that a mask m_x must never be used to mask any other value than x , and that every possible mask value must occur equally often in an unpredictable random sequence.

■ Remark 2:

Boolean Masking can be seen as a special case of the Vernam cipher (one-time pad) in cryptography: $M_i \rightarrow (M_i + R_i \bmod n, R_i)$, where M_i = message, and R_i = random key stream, $M_i, R_i \in \mathbb{Z}/n\mathbb{Z}$.

■ Remark 3:

The scheme is also called 1st-order Boolean masking.

A generalization is the splitting of a into n shares, e.g.

$$a \rightarrow (a \oplus m_1 \oplus m_2 \oplus \dots \oplus m_n, m_1, m_2, \dots, m_n).$$

This is called (n, n) -threshold secret sharing (“n-out-of-n secret sharing”) or n^{th} -order Boolean masking. n^{th} -order Boolean masking can prevent an n^{th} -order DPA attack [CJRR99] - see later lecture.

5.4.2. Algebraic countermeasures

Boolean Masking

- Boolean masking is typically used for block ciphers like DES and AES.
- Boolean masking is easy for a \mathbf{F}_2 -linear function f , because it holds:

$$f(x) = f(x_m \oplus m_x) = f(x_m) \oplus f(m_x)$$

Hence, the function can be applied independently to the masked data and the mask.

- Boolean masking on nonlinear functions like s-boxes is more difficult:

5.4.2. Algebraic countermeasures

Boolean Masking

■ Boolean masking of s-Box:

- Let S denote the s-box.
- Let S' denote the masked s-box
- m is the input mask and m' is the output mask
- x is the input data and y is the output data.

■ Then we have:

$$y = S(x) = S'(x \oplus m) \oplus m'$$

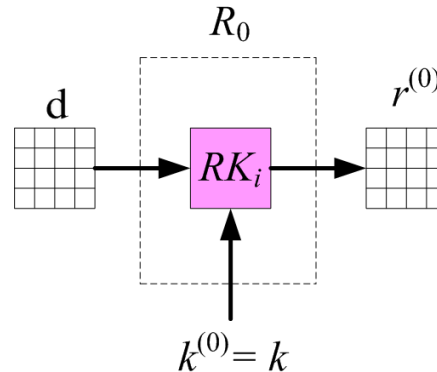
■ Computing the masked s-box is expensive:

- Typically the elements of the masked s-box S' are computed iteratively from each element of S .
- The masks and hence the masked s-box table must be updated for each encryption.

5.4.3. Boolean Masking for AES

We consider a basic masking scheme with a minimal number of masks for a software implementation of AES. We detail only the data path. The key expansion must also be masked (but this is skipped and not detailed here).

Initial round:



■ **RK** (AddRoundKey: $r_j^{(0)} = d_j \oplus k_j^{(0)}$ for bytes $j = 0, 1, 2, \dots, 15$)

Initially, we mask the data with mask

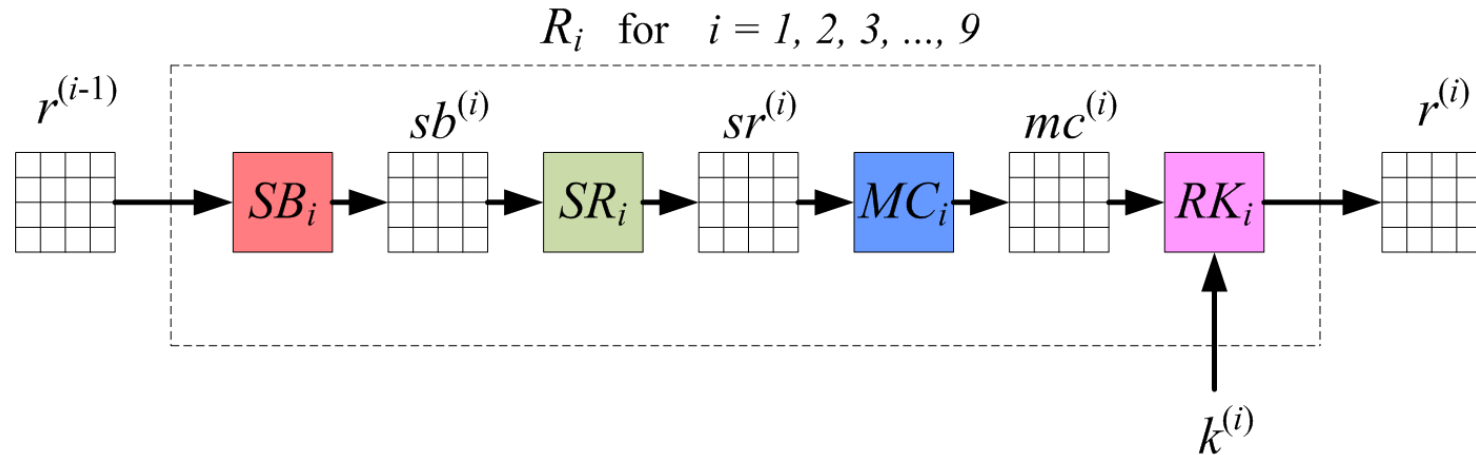
$$m: d'_j = d_j \oplus m.$$

After the RK operation the bytes of the state are masked:

$$(d_j \oplus m) \oplus k_j^{(0)} = (d_j \oplus k_j^{(0)}) \oplus m = r_j^{(0)} \oplus m = r'^{(0)}_j.$$

5.4.3. Boolean Masking for AES

Regular rounds:



- **SB** (SubBytes, $sb_j^{(i)} = S(r_j^{(i-1)})$ for $j = 0, 1, 2, \dots, 15$)
 - Use masked s-box $S'(x)$ for table lookup.
- **SR** (ShiftRows, rotates rows to the left with offsets 0, 1, 2, 3)
 - Move mask and data bytes “synchronously”. Up to this point we have the same mask for all bytes. Therefore this operation does not affect masking.

5.4.3. Boolean Masking for AES

■ **MC** (MixColumns)

In MC a linear combination of the bytes of a column are computed. E.g. the first byte is given by

$$mc_1^{(i)} = x \cdot sr_0^{(i)} \oplus (x+1) \cdot sr_1^{(i)} \oplus sr_2^{(i)} \oplus sr_3^{(i)}.$$

If the addition operations are not done in the correct order, the mask could cancel out and unmasked intermediate results would appear.

Hence additional masks are needed. It is efficient to introduce 4 independent masks m_1, m_2, m_3, m_4 , one for each row.

It is advantageous to use the same mask set in all rounds. This allows a precomputation.

■ **RK** (AddRoundKey, $r_j^{(i)} = mc_j^{(i)} \oplus k_j^{(i)}$ for $j = 0, 1, 2, \dots, 15$)

Applying AddRoundKey operation to the state preserves the masking.

5.4.3. Boolean Masking for AES

Putting the things together:

■ Precomputation for each encryption:

- Generate input and output masks

$m, m' \in \mathbb{F}_2^8$ for the s-box.

- Precompute the masked s-box

$S'(x)$ such that $S(x) = S'(x \oplus m) \oplus m'$.

- Generate four input masks $m_1, m_2, m_3, m_4 \in \mathbb{F}_2^8$ for MixColumns.

- Precompute transformed masks

$(m'_1, m'_2, m'_3, m'_4) = \text{MixColumns}(m_1, m_2, m_3, m_4)$.

- Precompute mask for remasking

$m'' = (m_1, m_2, m_3, m_4) \oplus m'$.

In total we need 6 bytes mask values.

5.4.3. Boolean Masking for AES

Execution:

■ Initialize:

- Mask plaintext with the MixColumns output row masks (m'_1, m'_2, m'_3, m'_4) .

■ For rounds $k = 0$ to 9:

□ AddRoundKey.

Note: round key $k_j^{(i)}$ is prepared with mask $(m'_1, m'_2, m'_3, m'_4) \oplus m$. Hence result is masked with m .

- SubBytes using S' . Result is masked with m' .

- ShiftRows. Result is masked with m' .

- if $(k \neq 9)$

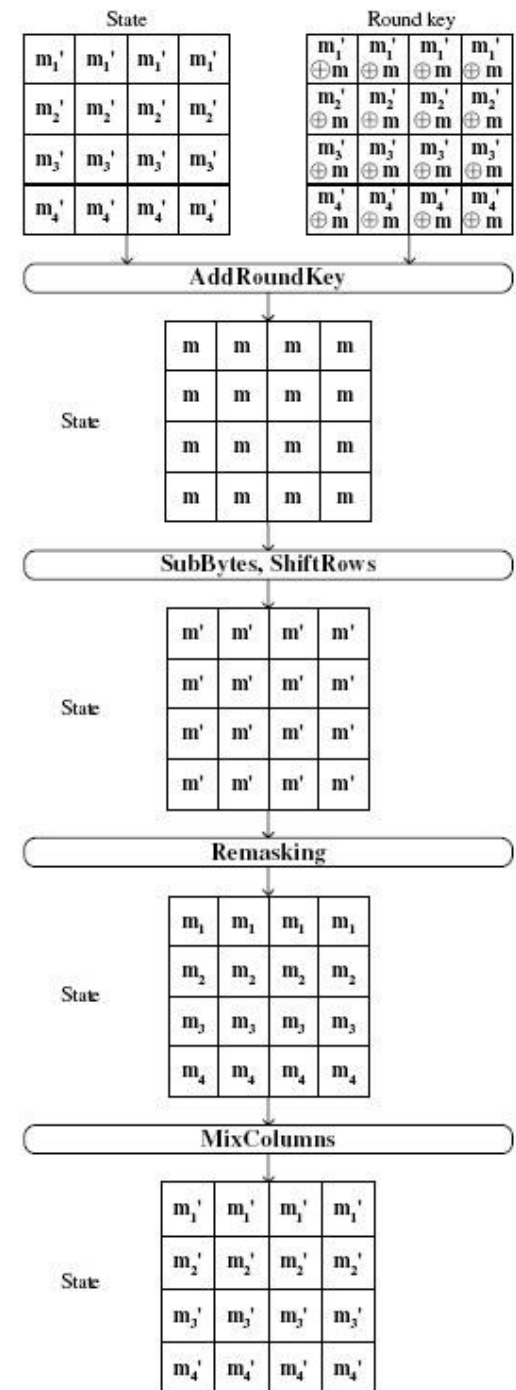
→ Remasking step with $m'' = (m_1, m_2, m_3, m_4) \oplus m'$.

→ MixColumns.

Result is masked with (m'_1, m'_2, m'_3, m'_4) .

- AddRoundkey $k_j^{(10)}$. Result is masked with m .

- Remove mask by adding m .



5.4.3. Boolean Masking for AES

Performance Figures:

An implementation of the presented masking scheme on an 8-bit microcontroller leads to the following figures.

- Execution time of an unprotected reference implementation:
4427 Clock cycles.
- Execution time of the masked implementation:
8420 clock cycles.
 - About 2800 clock cycles are spent only on mask pre-computations (for SubBytes, MixColumns, mask preparations).

→ The most expensive part is the recomputation of the s-box.

5.4.3. Boolean Masking for AES

Pitfalls leading to leaks in masked implementation

- The sequence of mask values m_1, m_2, \dots can be predicted by the attacker (e.g. is pseudorandom and the seed is known or can be guessed).
- The sequence of mask values m_1, m_2, \dots is not uniformly distributed.
- The hardware leaks Hamming distance of two values, e.g. the Hamming distance between the previous and the next value stored in a register.

□ Example:

Assume that the mask m_v and the masked value v_m are processed consecutively using the same register R , e.g.

$$R := v_m ; \dots ; R := m_v .$$

In this case the power consumption is proportional to

$$\text{dist}(v_m, m_v) = \text{hw}(v_m \oplus m_v) = \text{hw}(v) .$$

Hence, the power consumption depends on the value v – the value that we wanted to protect with the masking!


5.4.3. Boolean Masking for AES

Pitfalls leading to leaks in masked implementation

- Basic Requirement:

There must be no joint power consumption of masked values and mask.

- There are several occasions where the programmer needs to pay attention in order to fulfill this requirement:

-  Masked data and masks are subsequently stored in the same register (register file, pipeline, ...).

-  Masked data values that use the same mask are subsequently stored in the same register.

-  Masked data and masks use the same paths in combinational logic

-  Masked data values that use the same mask use the same combinational logic.

-  Some parallelized operation on masked data and masks is performed.

...

- The programmer should always have these pitfalls in mind and keep masked data and their masks apart.

- Knowledge on register allocation by the compiler and the microarchitecture of the controller is important.

5.4.4. Blinding Methods for RSA

- These specific countermeasures try to avoid the generation of the signal (on which the attacker is correlating) itself.
- The aim is to avoid the intermediary values, which the attacker is able to compute, if he has the correct hypothesis. E.g. the value of $m^{Di} \bmod N$ should not occur while the exponentiation $m^d \bmod N$ is executed.

1. Randomization of modulus N : Compute $(m^d \bmod (N \cdot r_1)) \bmod N$

- Use r_1 uniformly randomly chosen from, e.g., $[2^{31}, 2^{32}[$.
- The intermediary values are now: $m^{Di} \bmod (N \cdot r_1)$. Since the attacker does not now r_1 he can not compute these values and correlate on these.
- Disadvantage: The involved numbers grow by about 32 bits.
- Advantage: No big performance penalty.

5.4.4. Blinding Methods for RSA

2. Randomization of message m : Compute $((m + N \cdot r_2)^d \bmod (N \cdot r_1)) \bmod N$

- Use r_1, r_2 uniformly randomly chosen from e.g. $[2^{31}, 2^{32}[$.
- The principle and the pros/cons are the same as with countermeasure 1. But this countermeasure only works together with the previous one.

3. Randomization of the exponent d : Compute $m^{(d + r_3 \cdot \varphi(N))} \bmod N$

- use r_3 uniformly randomly chosen from e.g. $[2^{31}, 2^{32}[$.
- This works by Fermat's little theorem, since $m^{\varphi(N)} \bmod N = 1$, if $\gcd(m, N) = 1$.
- Here the attacker can not make any hypothesis on the exponent since the exponent changes in every run and the attacker does neither know r_3 nor $\varphi(N) = (p-1)(q-1)$.
- Disadvantage:
One has to know $\varphi(N)$ which is usually not necessarily available in some standardized cryptographic library interfaces.

5.4.4. Blinding Methods for RSA

4. Blinding of message m : Choose random r
and compute $s := (r^d)^{-1} \bmod N$,
i.e. $s \cdot r^d \bmod N = 1$.
Compute $((m \cdot r)^d \bmod N) \cdot s \bmod N$.

- Since $(m \cdot r)$ instead of m will be signed/decrypted, the attacker can not make hypotheses on intermediary values.
- Disadvantage: The random values (r, s) need to be computed which is an additional exponentiation \rightarrow operation takes twice as long.
- Solution for this problem:
 - Compute and store (r, s) once.
 - Then update and store the pair after each RSA-computation by $r := r^2 \bmod N$ and $s := s^2 \bmod N$.

5.4.5. Hiding Countermeasures

- Hiding countermeasures are generic countermeasures:
 - They keep the algorithm as it is and calculate the same intermediate values as unprotected implementations, but hide the data-dependent power consumption of the device.
 - They are not specific for the executed algorithm.
- Data-dependent power consumption can be hidden by either
 - burying the signal in (additional) noise,
 - making the power consumption as uniform as possible.
- Typically the attack can not be avoided completely.
The attacker needs more traces to ensure a certain side-channel signal-to-noise ratio, but the attack method doesn't change.
- Hiding countermeasures can be implemented on all architectural abstraction levels (logic style, hardware architecture, software).

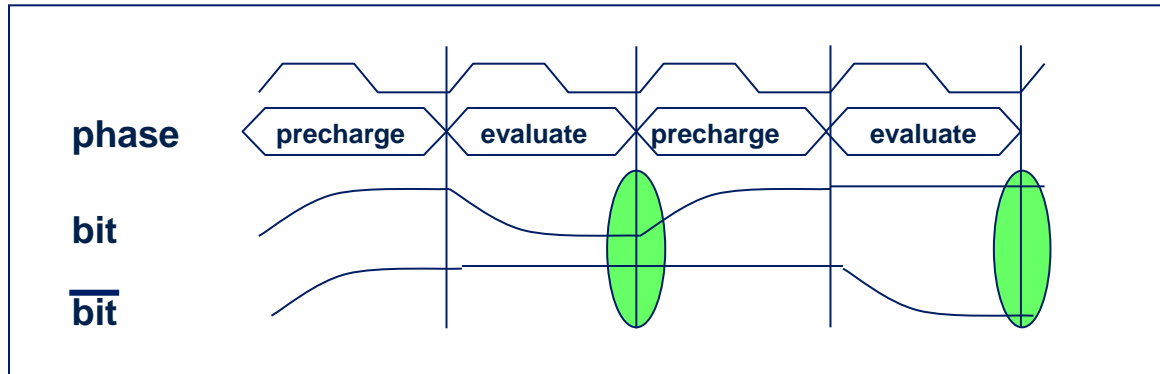
5.4.5. Hiding Countermeasures

Abstraction level	Hiding Countermeasures
Software	<ul style="list-style-type: none">○ Choice of instructions○ Shuffling○ ...
Hardware architecture	<ul style="list-style-type: none">○ Noise engines○ Shuffling○ ...
Ab initio methods: Circuit design & logic style	<ul style="list-style-type: none">○ Dual-rail precharge logic (DCVSL)○ Sense amplifier based logic (SABL)○ Masked dual-rail precharge logic (MDPL)○ ...

5.4.5. Hiding Countermeasures

Circuit design & logic style

Dual-Rail with Precharge Logic Style:



- Each data value is represented by a pair of wires.
- Switch between evaluation phase and precharge phase:
 - Precharge phase: Both wires are set to 1.
 - Evaluation phase: One wire switches to 0. The fact which wire switches to 0 determines whether the pair of wire represents a logical 0 or a logical 1.
- The precharge scheme leads to a constant number of switching events at the output of each gate independent of the data value.
- For a constant power consumption it is necessary that the wires are pairwise balanced.

5.4.5. Hiding Countermeasures

Circuit design & logic style

Dual-Rail with Precharge Logic Style:

■ Challenge:

The most difficult part of the implementation is the balancing of the complementary wires, such that the wire capacitances are equal for both rails.

■ Disadvantages:

- Switching activity is always at maximum rate, which can lead to increased power consumption compared to CMOS.
- The area is typically significantly larger than for CMOS (about 50% - 100%).
- There is no standard synthesis tool flow available.

Several other logic styles have been proposed:

- SABL (sense amplifier based logic), MDPL (masked dual-rail with precharge logic), WDDL (wave dynamic differential logic), TDPL (three-phase dual-rail precharge logic), ...
- All proposed logic styles share the same disadvantages as DCVSL and none of the proposed logic styles sustainably prevents DPA, if used as the only countermeasure.

5.4.5. Hiding Countermeasures

Abstraction level	Hiding Countermeasures
Software	<ul style="list-style-type: none">○ Choice of instructions○ Shuffling○ ...
Hardware architecture	<ul style="list-style-type: none">○ Noise engines○ Shuffling○ ...
Circuit design & logic style	<ul style="list-style-type: none">○ Dual-rail precharge logic (DCVSL)○ Sense amplifier based logic (SABL)○ Masked dual-rail precharge logic (MDPL)○ ...

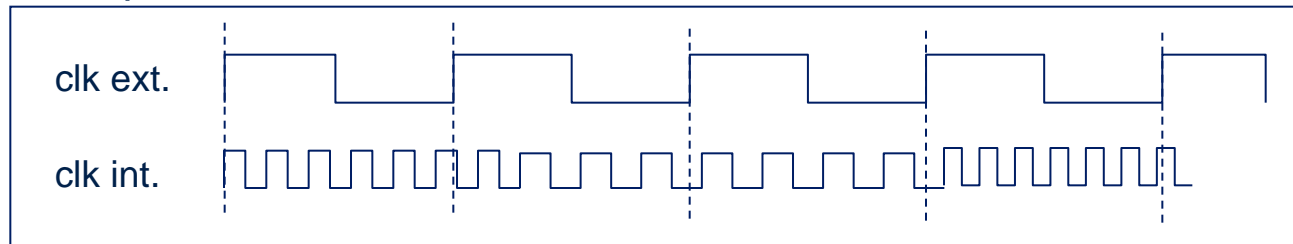
5.4.5. Hiding Countermeasures

Hardware architecture

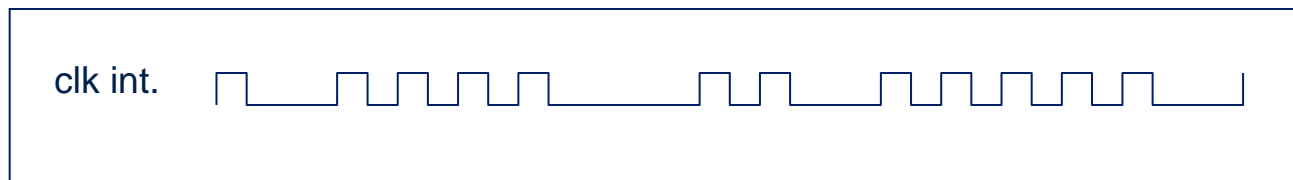
■ Hinder alignment of power traces:

- Build in elements (in SW or HW) that randomly delay the execution of (sections of) the algorithm such that the attacker can not easily align the power traces.

Examples for a hardware solution:



Randomized internal clock frequency.



Randomly delayed clock pulses.

- If the attacker cannot align the traces, he needs more power traces because the signal will be spread over several clock cycles and hence gets smaller.
- However, the attacker might be able to align the traces (using e.g. pattern matching algorithms) before computing the correlation traces.

5.4.5. Hiding Countermeasures

Hardware architecture

■ Noise Engines:

- Randomly charge and discharge capacitances.
- Randomly activate additional components of the chip that consume power (dummy logic or unused modules not used for the cryptographic computation).
- This can, e.g., be done autonomously by the underlying hardware or triggered by the software.

5.4.5. Hiding Countermeasures

Abstraction level	Hiding Countermeasures
Software	<ul style="list-style-type: none">○ Choice of instructions○ Noise○ Shuffling○ ...
Hardware architecture	<ul style="list-style-type: none">○ Noise engines○ Shuffling○ ...
Circuit design & logic style	<ul style="list-style-type: none">○ Dual-rail precharge logic (DCVSL)○ Sense amplifier based logic (SABL)○ Masked dual-rail precharge logic (MDPL)○ ...

5.4.5. Hiding Countermeasures Software

Altering the power consumption of a device is only possible to a limited degree in software:

- 1. Choice of instructions:** choose instructions whose power consumption is not strongly correlated to the processed data.
E.g. use highly parallel operations, ...
(strongly dependent on processor)
- 2. Noise:** activate additional components on the chip, which consume a significant amount of power and hence cause noise.

5.4.5. Hiding Countermeasures Software

3. Random delays and shuffling:

Change the moment of time when the attacked intermediate result is processed at random.

- Method 1: Random delays
Randomly insert dummy operations.

- Method 2: Shuffling
Randomly change the sequence of operations that can be performed in arbitrary order, i.e. exploit invariants in the algorithm.
Example: order of S-box look-ups in block ciphers.

5.4.5. Hiding Countermeasures Software

On the effectiveness of random delays and shuffling:

- When a random delay is inserted, the attacked operation is executed with probability p at time T (e.g. where it would occur with no random delay).
- Let $\text{corr}(H_{\text{ck}}, P_{\text{ct}})$ be the correlation coefficient of the correct key hypothesis with the power consumption at time T , when the critical operation is executed.
Let $\text{corr}(H_{\text{ck}}, \underline{P}_{\text{ct}})$ be the correlation coefficient of the correct key hypothesis with the power consumption at time T .
- One can show, that $\text{corr}(H_{\text{ck}}, P_{\text{ct}})$ is linearly reduced with p [MOP07]:

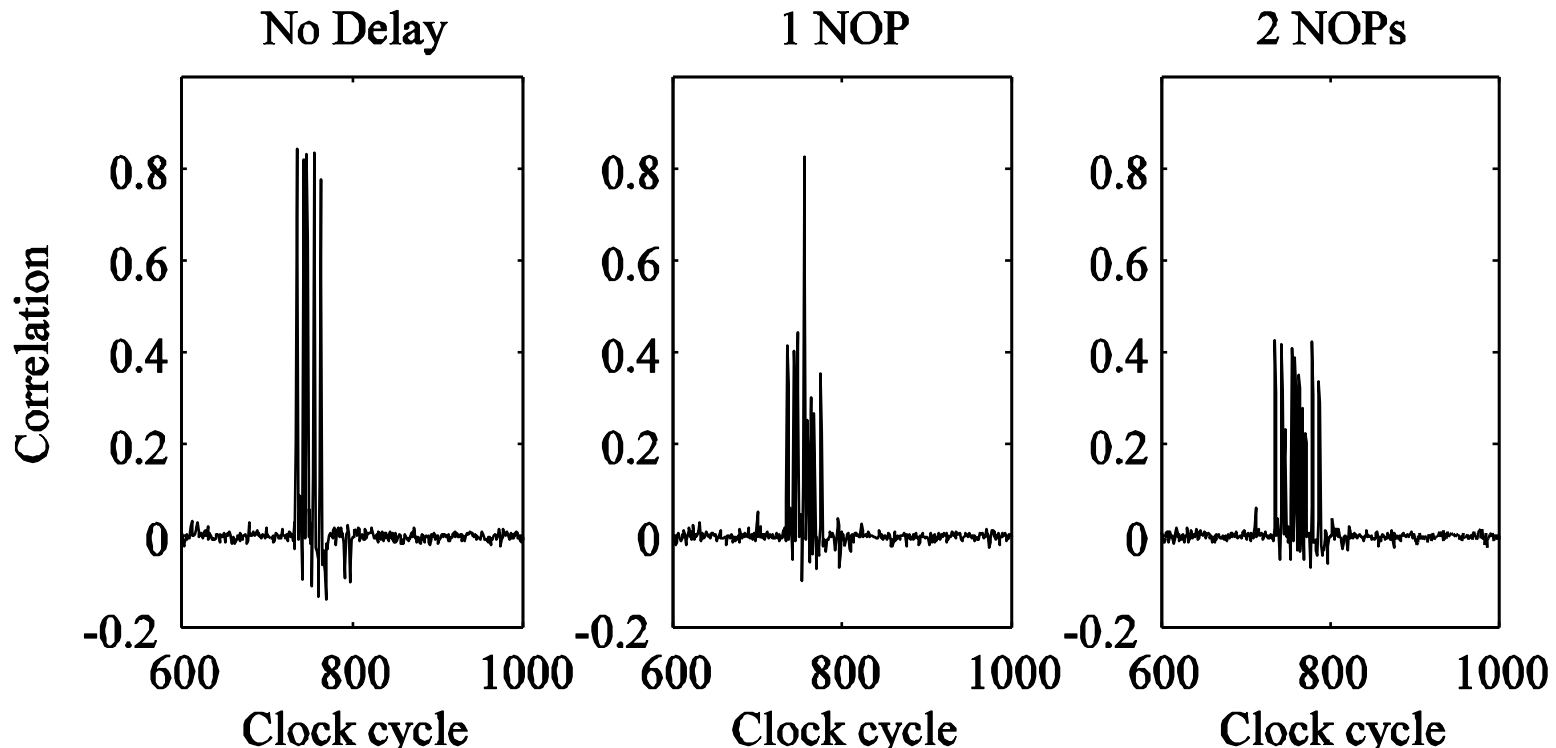
$$\text{corr}(H_{\text{ck}}, \underline{P}_{\text{ct}}) \sim p \cdot \text{corr}(H_{\text{ck}}, P_{\text{ct}})$$

- A *linear* decrease of the correlation coefficient means a *quadratic* increase of the number of needed measurements!

So one could think that this countermeasure is very effective...

5.4.5. Hiding Countermeasures Software

- Example: DPA on AES S-box of 8-bit μC implementation (10000 traces). One or two random NOP are inserted before start of the AES.



- Remark: One NOP is not enough delay, because the operation took 2 clock cycles \rightarrow no effect on the maximum correlation. Two NOPs reduce the correlation by a factor of two.

5.4.5. Hiding Countermeasures Software

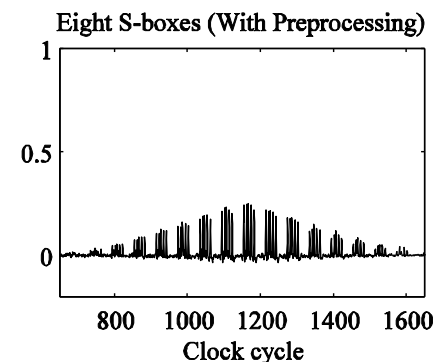
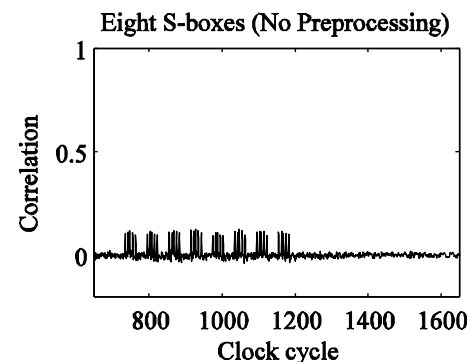
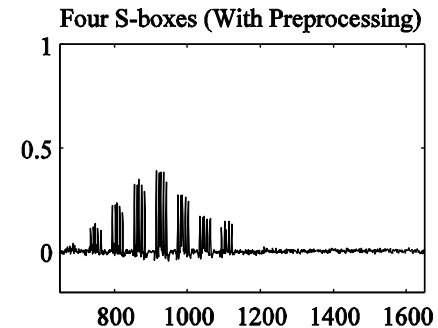
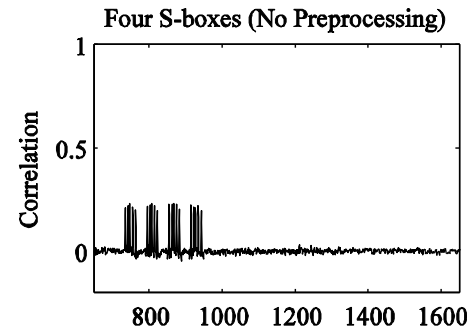
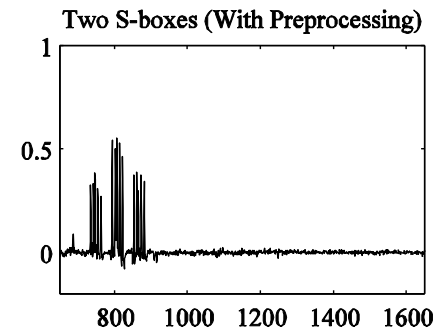
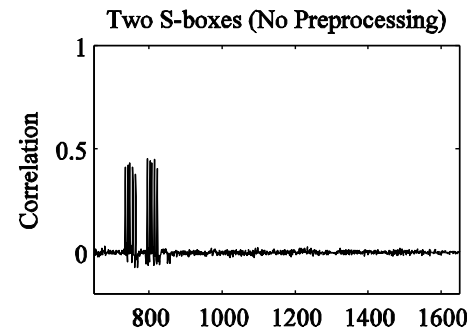
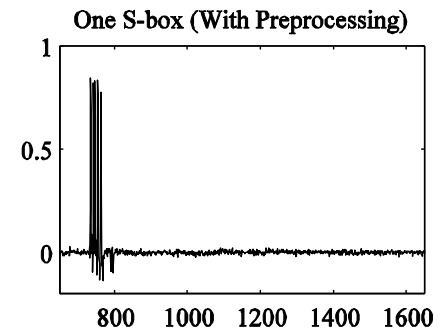
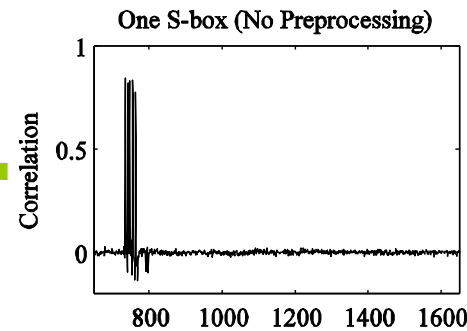
- However, an attacker might choose another attack strategy. For example the attacker could do *windowing*.
- Windowing means that the attacker sums up the power consumption in a certain range:
 - Assume that the sequence of the 16 S-box lookups in the AES is shuffled. The attacker could sum up the power consumption of the 16 S-box lookups.
- Now let us assume, that the power consumptions P_i at n clock cycles are independent and the variances of the power consumption are equal. Then one can show for the correlation coefficient [MOP07]:

$$\mathbf{corr}(H_{\text{ck}}, \sum_{i=1}^n P_i) = \mathbf{corr}(H_{\text{ck}}, P_1) / \sqrt{n}$$

- With windowing the correlation decreases only with the square root of n . Hence the number of needed measurements increases only linearly. (Note, each trace is used n times, hence n^2 traces are used in the evaluation.)

5.4.5. Hiding Counterterm. Software

- Example:
DPA on AES s-box of 8-bit μC implementation (10000 traces).
Shuffling of 1, 2, 4, and 8 s-boxes, respectively, is implemented.
- Left column of figures:
Standard DPA.
- Right column of figures:
DPA attack using windowing.



5.4.5. Hiding Countermeasures Software

On the effectiveness of random delays and shuffling:

- In the best case for the designer, a linear reduction of the correlation coefficient is achieved. This means that there is a quadratic increase of the effort for the attacker.
 - If the attacker successfully performs windowing, there is only a linear increase of the effort for the attacker.
 - In the worst case the attacker can align the power traces and completely remove the countermeasure. Alignment can often be done using pattern matching techniques.
- Hence, take care that the attacker cannot completely remove the randomization countermeasure by filtering and pattern matching techniques.
- Take care that it is hard for the attacker to find out how the randomization is done in order to make windowing harder. Dummy operations should not be distinguishable from real ones.

5.4.5. Hiding Countermeasures Software

On the effectiveness of adding amplitude noise:

- Let $P = P_n + P_{\text{exp}}$ = total power consumption of the chip.
- Let P_n = noise contribution in the power consumption .
- Let P_{exp} = exploitable contribution in the power consumption.
- Let $\text{SNR} = \text{var}[P_{\text{exp}}] / \text{var}[P_n]$ be the side-channel signal-to-noise ratio.
- Let H_{ck} = correct key hypothesis.

Then we have

$$\begin{aligned}\text{corr}(H_{\text{ck}}, P) &= \text{corr}(H_{\text{ck}}, P_{\text{exp}} + P_n) \\&= \frac{\mathbf{E}[H_{\text{ck}} (P_{\text{exp}} + P_n)] - \mathbf{E}[H_{\text{ck}}] \mathbf{E}[P_{\text{exp}} + P_n]}{\sqrt{\text{var}[H_{\text{ck}}]} \sqrt{\text{var}[P_{\text{exp}} + P_n]}} \\&= \frac{\mathbf{E}[H_{\text{ck}} P_{\text{exp}}] + \cancel{\mathbf{E}[H_{\text{ck}} P_n]} - \mathbf{E}[H_{\text{ck}}] \mathbf{E}[P_{\text{exp}}] - \cancel{\mathbf{E}[H_{\text{ck}}] \mathbf{E}[P_n]}}{\sqrt{\text{var}[H_{\text{ck}}]} \sqrt{\text{var}[P_{\text{exp}}]} \sqrt{(1 + \text{var}[P_n] / \text{var}[P_{\text{exp}}])}} \\&= \frac{\text{corr}(H_{\text{ck}}, P_{\text{exp}})}{\sigma[P] / \sigma[P_{\text{exp}}]} = \frac{\text{corr}(H_{\text{ck}}, P_{\text{exp}})}{\sqrt{(1 + 1 / \text{SNR})}}\end{aligned}$$

5.4.5. Hiding Countermeasures Software

- For low SNR ratios, the correlation coefficient increases approximately with the *square root* of the SNR.
→ doubling the amount of noise (i.e. $P_n \rightarrow P_n + P_n$) reduces $\text{corr}(H_{\text{ck}}, P)$ by $\sqrt{2}$ and hence doubles the number of needed traces.
- A coarse estimation:
Assume a perfect correlation between correct key hypothesis and power consumption, i.e. $\text{corr}(H_{\text{ck}}, P_{\text{exp}}) = 1$. Then we have

$$\text{corr}(H_{\text{ck}}, P) = \frac{1}{\sqrt{(1 + 1 / \text{SNR})}} = \frac{\sigma[P_{\text{exp}}]}{\sigma[P]}$$

- Let us attack n out of $n+m$ wires of a bus (or any n out of $n+m$ components) that have roughly the same power contribution.
Then the $\text{SNR} = n/m$.

$$\text{corr}(H_{\text{ck}}, P) = \frac{1}{\sqrt{(1 + m/n)}} = \sqrt{\frac{n}{m + n}}$$

- For $n = 1, m = 31$ we find

$$\text{corr}(H_{\text{ck}}, P) \approx 0.18 \text{ and } \# \text{traces} \approx 28 / \text{corr}(H_{\text{ck}}, P)^2 \approx 860.$$

5.4. DPA Countermeasures

References

- [CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, Pankaj Rohatgi: *Towards Sound Approaches to Counteract Power-Analysis Attacks*. In: Michael J. Wiener (Ed.), *Advances in Cryptology – CRYPTO’99*. 19th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 1999, Proceedings. Volume 1666, Lecture Notes in Computer Science, pp. 398-412. Springer, 1999.
- [DP08] S. Dziembowski, K. Pietrzak: *Leakage-Resilient Cryptography*. In: 49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA, pp. 293-302, IEEE Computer Society, 2008.
- [GFM09] Berndt M. Gammel, Wieland Fischer, Stefan Mangard: *Erzeugung eines Session-Schlüssels zur Authentisierung und sicheren Datenübertragung*. Deutsches Patent 10200924604 B4, filed June 10, 2009.
- [K99] Paul C. Kocher: *Leak-Resistant Cryptographic Indexed Key Update*. US Patent 6,539,092 B1, filed July 2, 1999.
- [K05] Paul C. Kocher: *Design and Validation Strategies for Obtaining Assurance in Countermeasures to Power Analysis and Related Attacks*. NIST Physical Security Testing Workshop, Sept. 26-29, Honolulu, USA, 2005. Available at: <http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-3/physec/physecdoc.html>

5.4. DPA Countermeasures

References

- [MOP07] Stefan Mangard, Elisabeth Oswald, Thomas Popp: Power Analysis Attacks – Revealing the Secrets of Smart Cards, Springer Verlag, 2007.
- [MSGR10] Marcel Medwed, François-Xavier Standaert, Johann Großschädl, Francesco Regazzoni: *Fresh Re-keying: Security against Side-Channel and Fault Attacks for Low-Cost Devices*. In: Daniel J. Bernstein and Tanja Lange (Eds.) Progress in Cryptology - AFRICACRYPT 2010, Third International Conference on Cryptology in Africa, Stellenbosch, South Africa, May 3-6, 2010, Proceedings. Volume 6055, Lecture Notes in Computer Science, pp. 279-296. Springer, 2010.
- [OSPT] OPST Alliance: Open Standard for Public Transportation (OSPT). The CIPURSE™ Open Standard. Available at: <http://www.osptalliance.org> (2013).