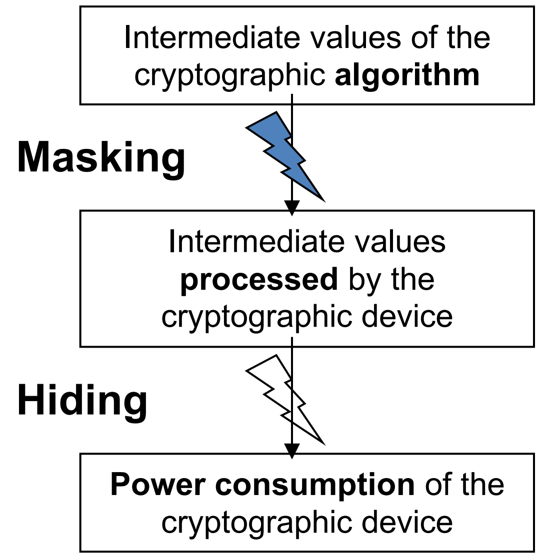
Applied Security

Countermeasures for AES

Countermeasure: Hiding



- General idea
  - Remove dependency between processed intermediate values (performed operations) and power consumption
- Hiding
  - Dummy operations: indistinguishable from rest but work on random data, results are discarded
  - Shuffling: reordering of independent operations
  - In software: No change of power consumption characteristic of the cryptographic device necessary
  - In hardware: could use less leaky logic style, but this requires to newly build devices, cannot secure existing devices

Recap SNR, correlation, NNT

- We call the part of P_{op}, P_{data} that is actually exploited in an attacks P_{exp} and the remaining part P_{noise}
  - $P_{total} = P_{exp} + P_{noise}$

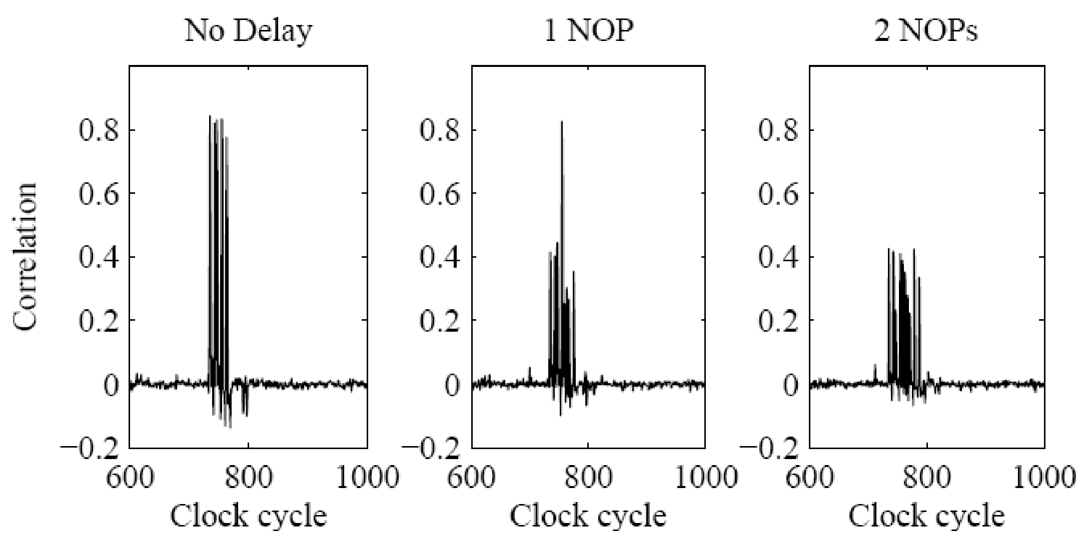
- SNR = $Var(P_{exp})/Var(P_{noise})$

$$\rho(H_i, P_{total}) = \frac{\rho(H_i, P_{exp})}{\sqrt{1 + \frac{1}{SNR}}}$$

$$n = 3 + 8\frac{z_{1-\alpha}^2}{\ln^2 \frac{1+\rho_{ck,ct}}{1-\rho_{ck,ct}}}$$

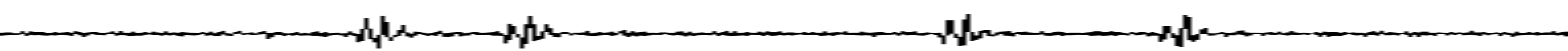
Hiding in software: AES toy example

- Insertion of random NOP instruction(s)
  - AES is implemented in such a way that 0, (0 or) 1, or (0 or) 2 NOP instructions are executed before the start.
  - Plots of the results of DPA attacks on these implementations:



Analysis

- 0 nops just gives a standard DPA attack,
- 1 nop leads to more peaks where most but one are smaller
  - This is because the DPA peak is related to an instruction where the attacked data is processed in two consequtive clock cycles!
- 2 nops leads to even more/wider peaks which are only half in height
  - Is this a coincidence?



Analysis

- Assuming the peak ct occurs with probability $\hat{p}$ the covariance (and hence the correlation) can be rewritten as

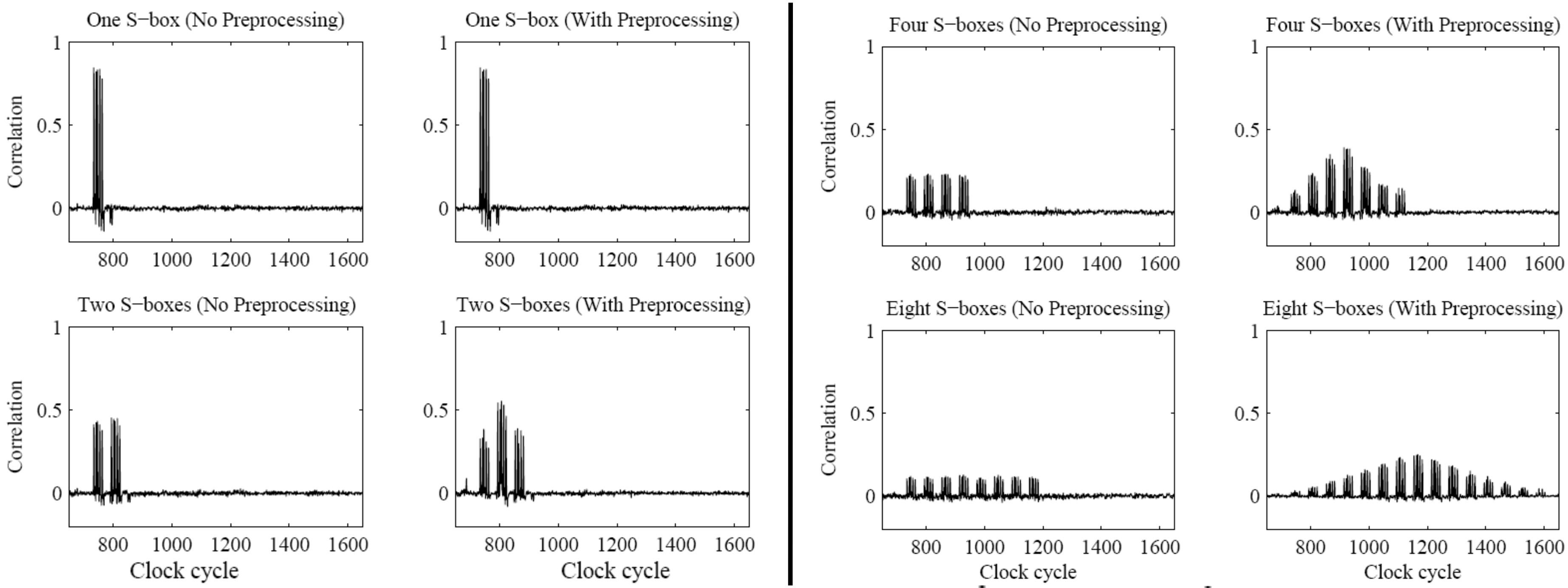
$$Cov(H_{ck}, \widehat{P_{total}}) = \hat{p} \cdot Cov(H_{ck}, P_{total}) + (1-\hat{p}) \cdot Cov(H_{ck}, P_{other})$$

$$\rho(H_{ck}, \hat{P}_{total}) = \rho(H_{ck}, P_{total}) \cdot \hat{p} \cdot \sqrt{\frac{Var(P_{total})}{Var(\hat{P}_{total})}}$$

- Maximum correlation is hence determined by probability describing the displacement and by ratio of variances

Hiding in software: another simple example for AES

- Shuffling: The sequence of the 16 S-box look-ups of the AES is changed randomly. The plots show the correlation for the correct hypothesis once without preprocessing and once with integration over the possible occurrences as preprocessing



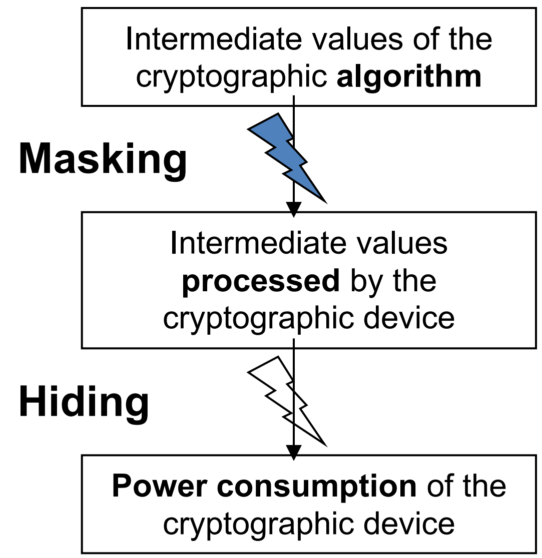
Analysis

- Misalignment leads to the DPA peak ct being `distributed‘ over a number of l clock cycles
  - Align traces to minimise misalignment
  - Integrate over l consequtive clock cycles to `sum up‘ the peaks.
    - Assuming we such l somewhat `independent´ clock cycles, and that variances in different clock cycles are equal we have that

$$\rho(H_{ck}, \sum_{i=1}^{l} P_i) = \frac{\rho(H_{ck}, P_1)}{\sqrt{l}}$$

Countermeasure: Masking

- PA-Countermeasure
  - Remove dependency between processed intermediate values (performed operations) and power consumption



- Masking
  - Process only randomized intermediate values
  - Power required to process randomized values is independent of the actual intermediate values
  - No change of power consumption characteristic of the cryptographic device necessary

Masking/Blinding

- Masking
  - Each intermediate value v is concealed by a random value m … the "mask"; not known by attacker
    - $v_m = v * m$
    - $*$ … general combination operator
      - XOR-function $\oplus$ → Boolean masking
      - Arithmetic operation like addition or multiplication → Arithmetic masking
        - » Multiplicative masking has a problem with v = 0 (attack using ZV model)!!!
- Blinding = masking in context of PK schemes
  - RSA (decrypt/sign): compute $c_m = {P_m}^d$mod n, with $P_m = P \cdot m^e \; mod \; n,$ then remove mask by multiplying with m^(-1) mod n
  - Message vs exponent blinding, SPA resistance?

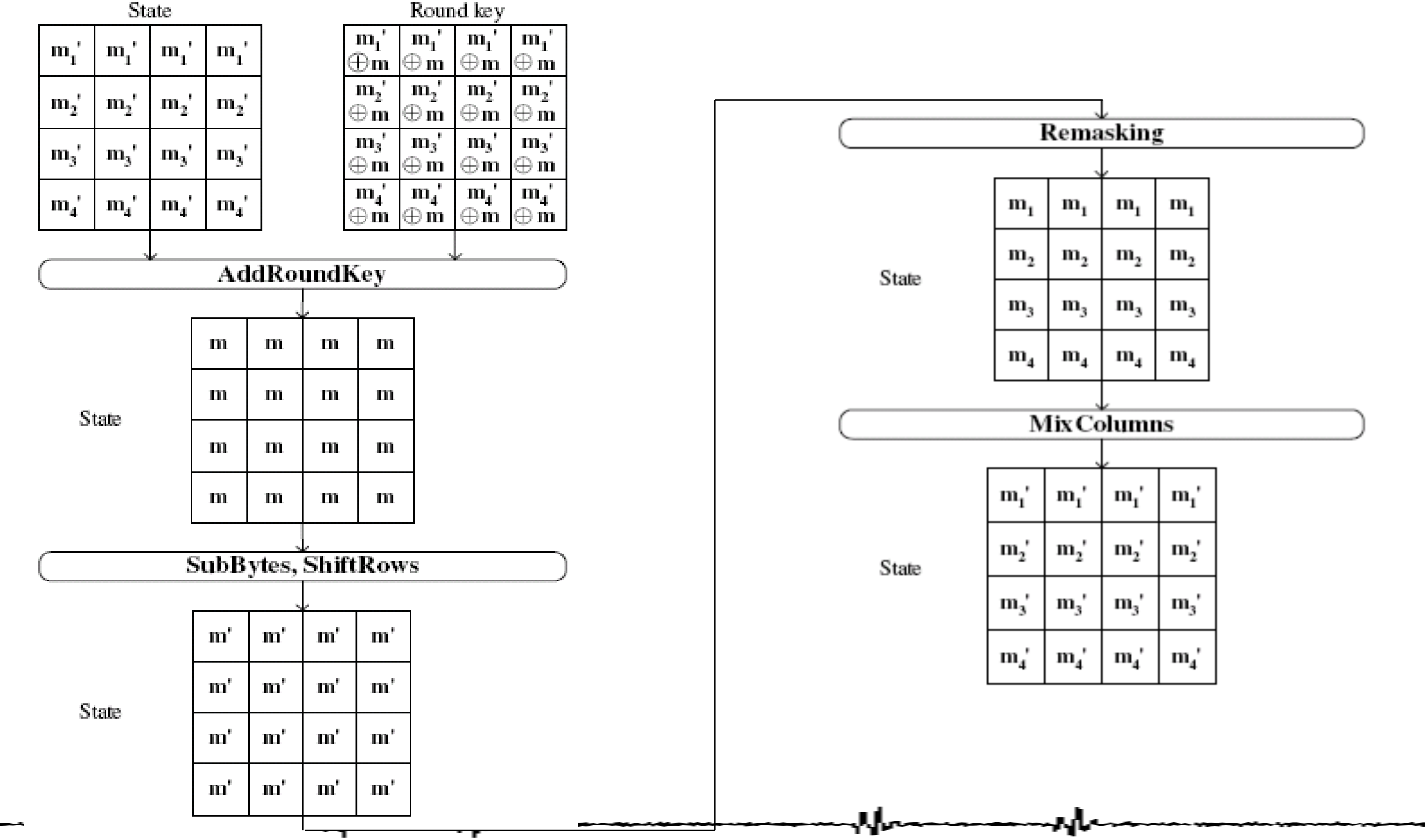
Masking AES as an example

- Use Boolean masking
  - p … $(p \oplus m)$, (m)
  - AddRoundKey ,ShiftRows, Mixcolumns: linear operations with respect to Boolean addition
    - $ARK(p \oplus m,k) = ARK(p,k) \oplus ARK(m,0)= ARK(p,k) \oplus m$
  - SubBytes: non-linear operation with respect to Boolean addition, i.e. $S(v \oplus m) \neq S(v) \oplus S(m)$
    - We pre-compute another table S_m such that $S_m(v \oplus m) = S(v) \oplus m'$, (i.e. we implicitely allow a change of mask from $m \rightarrow m'$)

Example: Management/change of masks in the flow of an AES round

- ShiftRows: all state bytes use the same mask, no change
- MixColumns: requires 2 or more masks to avoid unintentional unmasking
  - Input column masks: m_1, m_2, m_3, m_4
  - Output column masks: m_1', m_2', m_3', m_4'
- At the beginning of each round
  - Calculate S_m
  - Calculate m_1', m_2', m_3', m_4' out of m_1, m_2, m_3, m_4

Example: Management/change of masks in the flow of an AES round



Analysis

- Pre-condition: masks are chosen uniformely at random, every new encryption run
- Then we have a guarantee (in a mathematical sense) that every intermediate value that is masked is independent of the unmasked value
  - V_m is independent of m and V
  - Consequently no `first order DPA´ (i.e. a DPA attack that only uses single points in the statistical distinguisher) can succeed

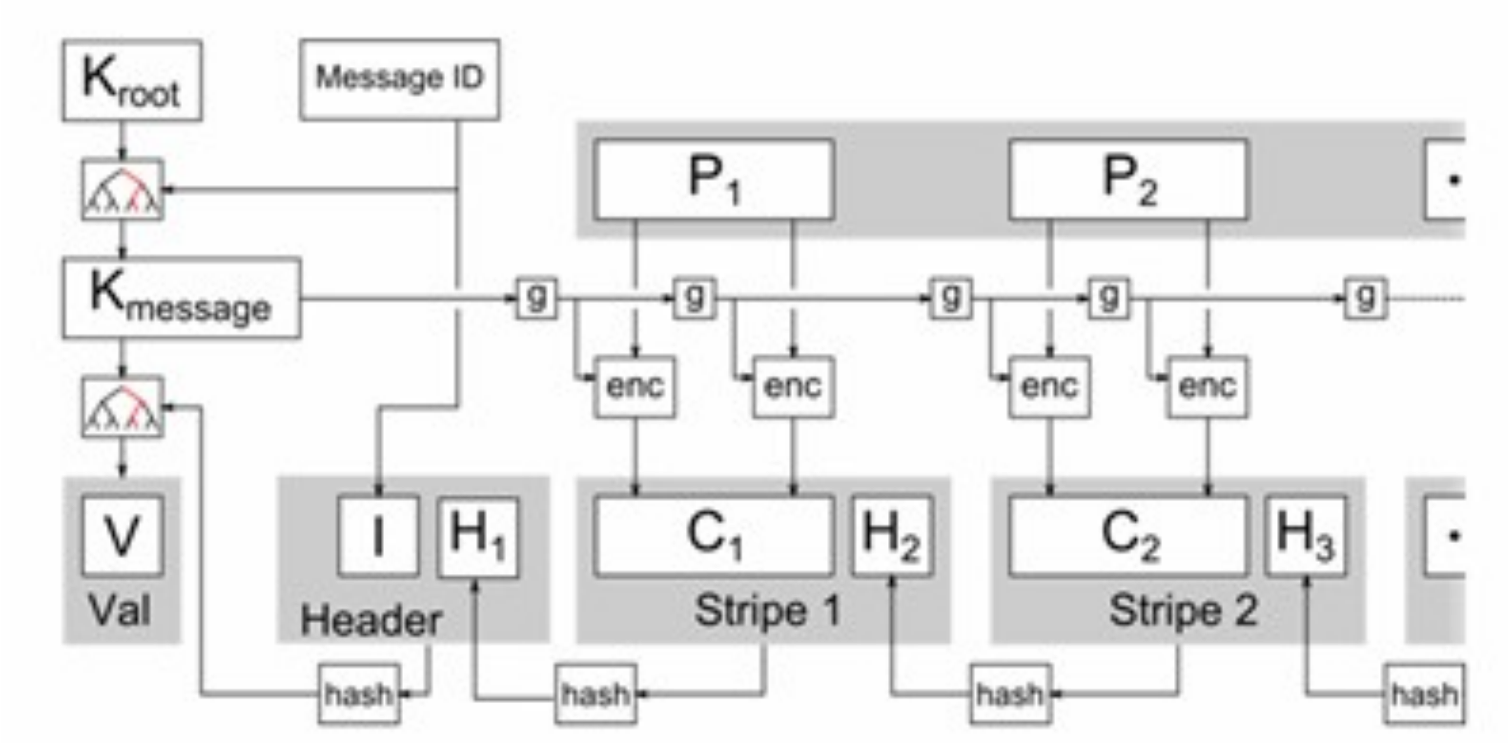
Analysis

- Consequently attacks must aim to
  - Introduce some bias into the masks
    - Either by influencing the RNG or by selecting a subset of power traces
  - Exploit some weakness in the type of masking used
    - Multiplicative masking is inherently insecure against zero-value attacks (i.e. attack that exploit that the plaintext value zero cannot be masked multiplicatively)
  - Use a distinguisher that takes 2 (or more) points of a power trace as input
    - Leading to higher-order or template-based DPA attacks

Protocol-level considerations

- All countermeasures discussed so far are used in practice, yet none of them actually gives total security against SPA or DPA
- Rather than investing a huge amount of memory/computing overhead/area into a 100% effective countermeasure, a trade off is sought that minimises leakage such that in combination with a key update procedure, the overall application is likely to meet security requirements of practical applications

Encryption with built-in authentication and key update



For each message a random ID is created which determines how to derive a (number of) session key(s) based on a secret key. This method is covered by a patent by Kocher. Pictures and further details can be found here.

Another countermeasure involving some protocol level considerations

- Fresh rekeying: same idea as before, i.e. the generation of a fresh session key preceeds any encryption
  - But the session key is produced by always updating the same secret key, wheras previous patented method would actually ‚continue‘ along the tree
- This method is hence stateless in comparison to the previous one

Fresh rekeying, cont.

- Define function g that updates a key given some public randomness:
  - Ks = g(key, rand)
  - This function g needs to be implemented such that it is resistant against SPA and DPA attacks
- Then the underlying encryption scheme works using the session key Ks:
  - C = AES(Ks, msg)
  - And the receiver gets both C and rand
- Obviously this only makes sense if g is such that protection against SPA and DPA is much easier to achieve in comparison to AES!

Summary

- There are no foolproof push-button solutions for making an implementation resilient against leakage
  - Few provably secure approaches exist (all are beyond the scope of this unit), and their proof assumptions are hard to fulfill in practice (even for a simple thing such as first order masking)
  - It may be better to implement several cheap countermeasures than to implement one very expensive one
  - All countermeasures require randomness