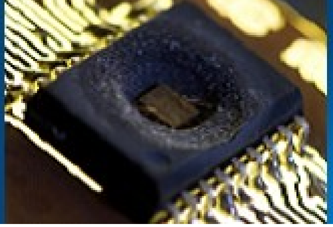
Technische Universität München

Fakultät für Elektrotechnik
und Informationstechnik



Smartcard Laboratory
Introduction to Side-Channel Analysis

Prof. Dr.-Ing. Georg Sigl
Institute for Security in Information Technology
Lecturer: M.Sc. Michael Gruber

Summer Semester 2019

**Lehrstuhl für Sicherheit in der
Informationstechnik**

Agenda

- Introduction
  - Attacks on crypto implementations
  - Classification of hardware attacks
  - Different kinds of side-channel attacks
- Side Channel Analysis
  - AES algorithm
  - Power consumption models
  - Steps of a DPA Attack
- SCA Countermeasures
  - Hiding
  - Masking
- HDF5 Format
  - Introduction to HDF5

Section 1

INTRODUCTION

Black box assumptions

Symmetric cryptography:

- AES introduced 1999 is still cryptographically secure

Asymmetric cryptography:

- RSA with 2048 Bit key will be secure for the next years
- Elliptic curves cryptography with smaller key size and equivalent security as an alternative

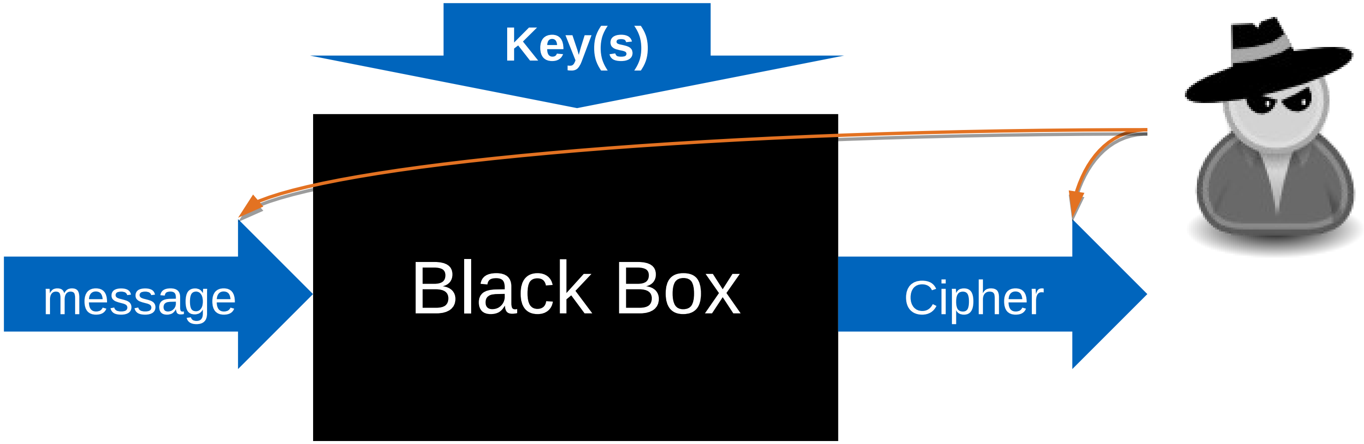
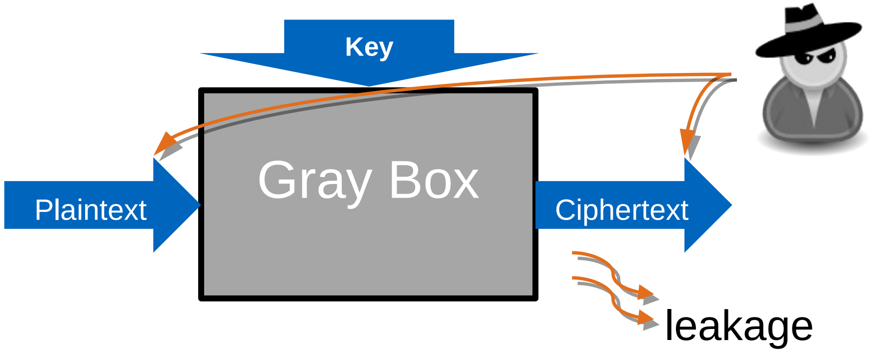


image source: http://www.opensecurityarchitecture.org/cms/library/icon-library

Grey Box or White Box

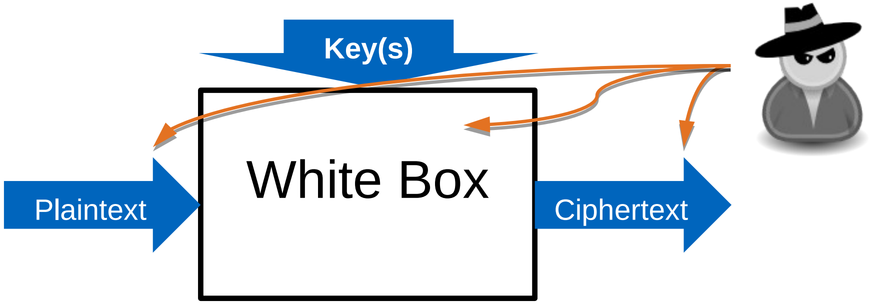


+

- Plaintext
- Ciphertext
- **Side-channel**

leakage

  - Power
  - EM
  - Timing
  - …



+

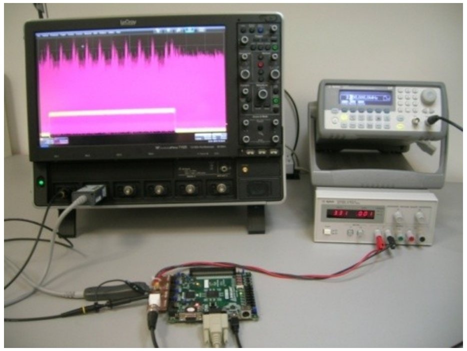
Debug
Reverse Eng.

Memory
Inspection

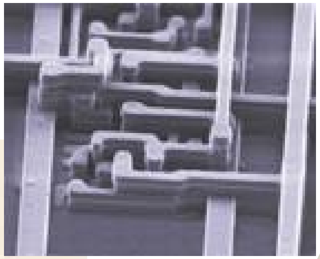
For Implementations of cryptographic algorithms the black box assumptions are no longer valid!

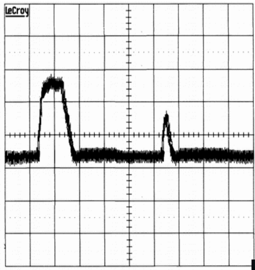
image source: http://www.opensecurityarchitecture.org/cms/library/icon-library

Which kinds of attacks on embedded systems exist?



Side Channel

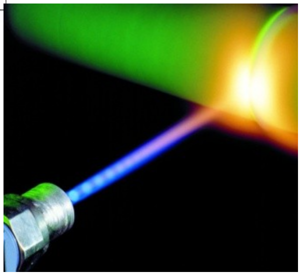




Attacks

Reverse Engineering

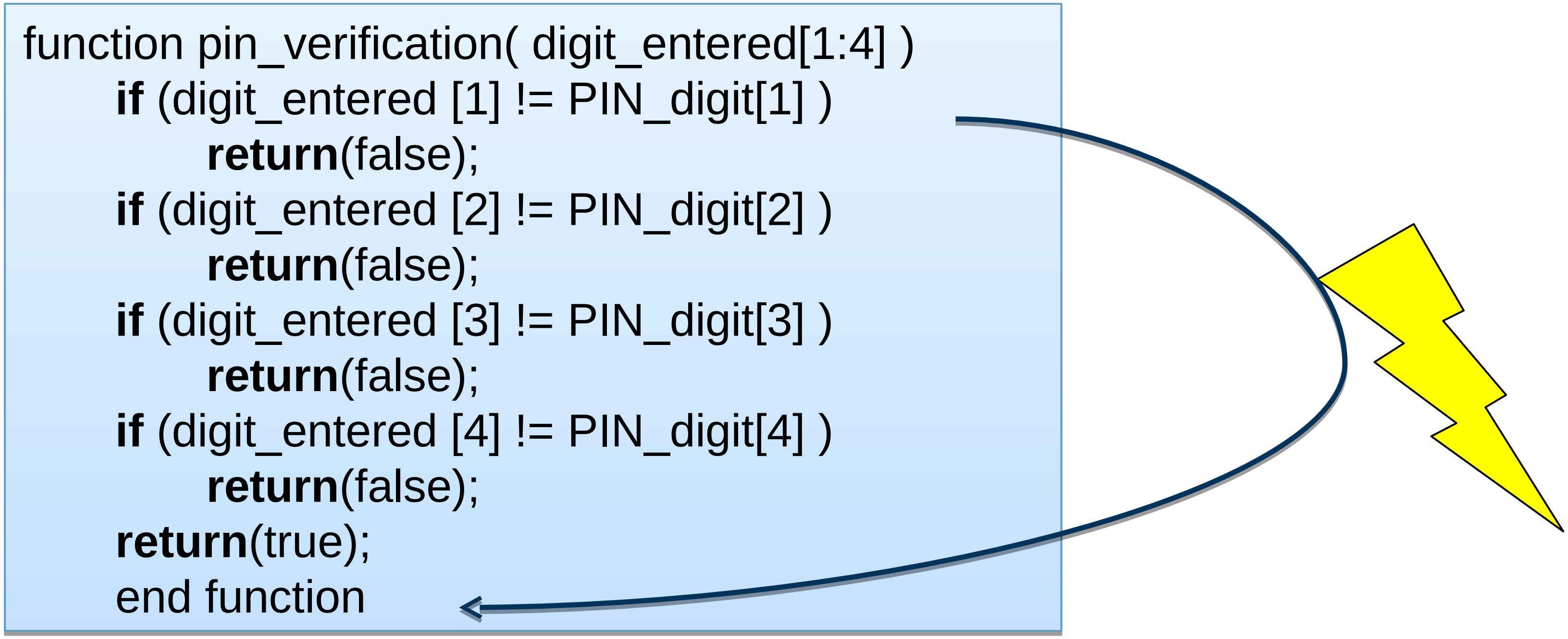
Fault Injection



Example: PIN check with 4 digits

```
function pin_verification( digit_entered[1:4] )
    if (digit_entered [1] != PIN_digit[1] )
        return(false);
    if (digit_entered [2] != PIN_digit[2] )
        return(false);
    if (digit_entered [3] != PIN_digit[3] )
        return(false);
    if (digit_entered [4] != PIN_digit[4] )
        return(false);
    return(true);
    end function
```

Fault attack

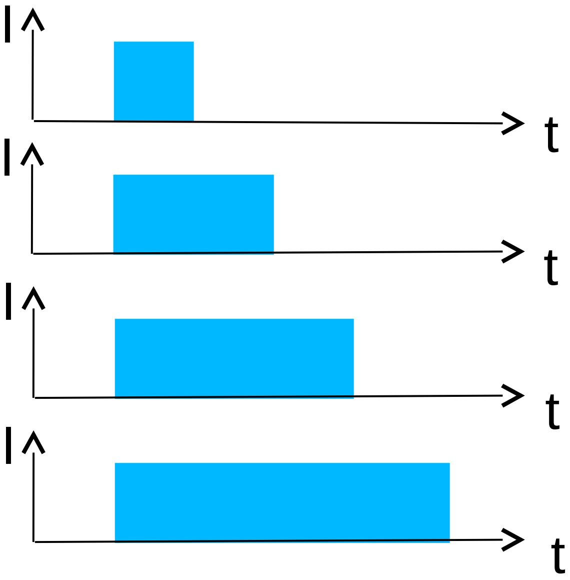


```
function pin_verification( digit_entered[1:4] )
    if (digit_entered [1] != PIN_digit[1] )
        return(false);
    if (digit_entered [2] != PIN_digit[2] )
        return(false);
    if (digit_entered [3] != PIN_digit[3] )
        return(false);
    if (digit_entered [4] != PIN_digit[4] )
        return(false);
    return(true);
    end function
```

Side-channel attack

(Timing attack in this case)

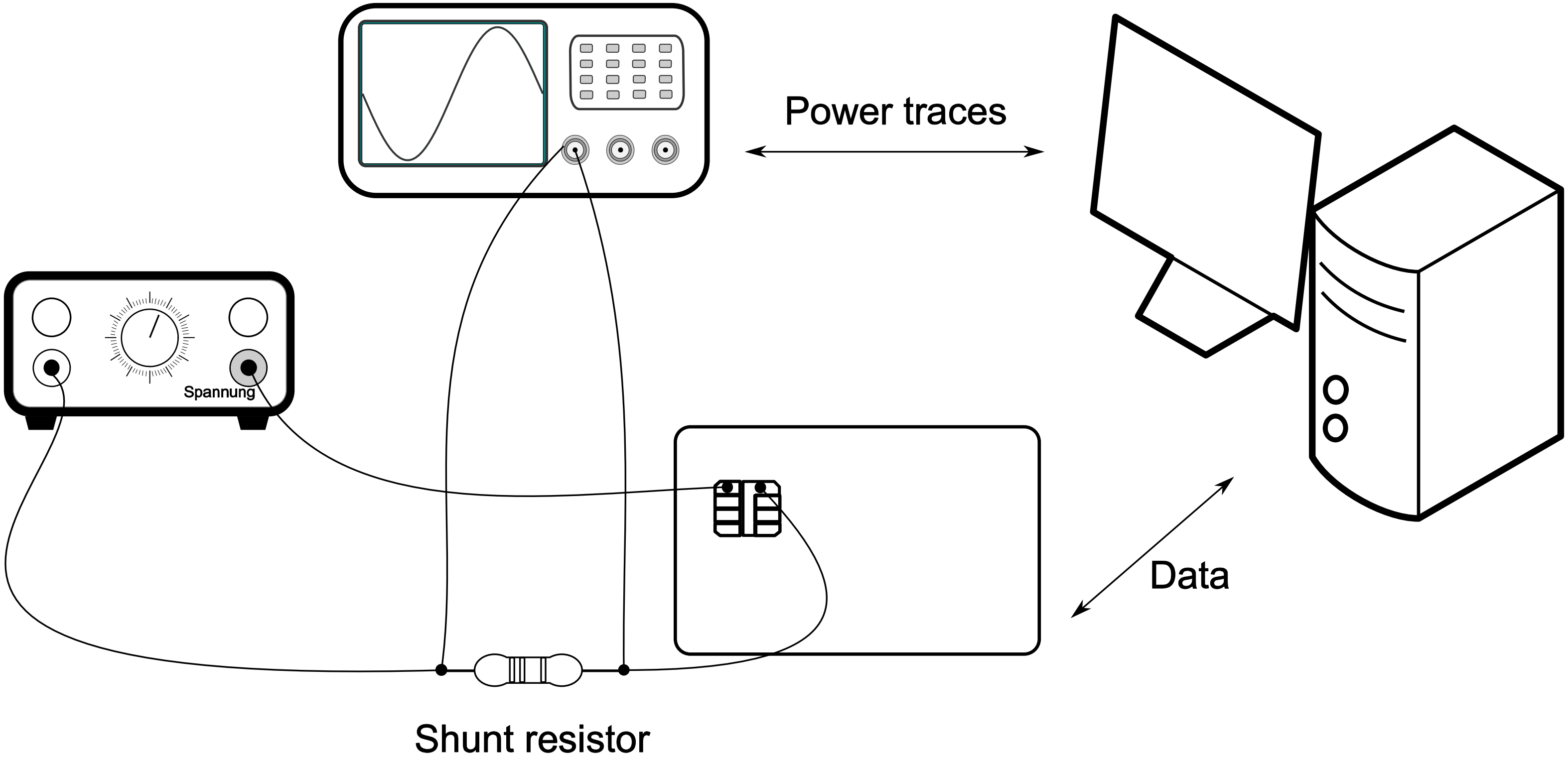
```
function pin_verification( digit_entered[1:4] )
    if (digit_entered [1] != PIN_digit[1] )
        return(false);
    if (digit_entered [2] != PIN_digit[2] )
        return(false);
    if (digit_entered [3] != PIN_digit[3] )
        return(false);
    if (digit_entered [4] != PIN_digit[4] )
        return(false);
    return(true);
    end function
```



Attack classification

	Active	**Passive**
Non-Invasive	Glitching, Temperature Change, Low Voltage, …	Side-Channel Attacks (Timing Analysis, Power Analysis, Simple EM Attacks …)
Semi-Invasive	Light Attacks, Radiation Attacks, …	Sophisticated EM Attacks, Optical inspection (ROM, …)
Invasive	Forcing, Permanent circuit changes, …	Probing Attacks, …

Power Analysis Measurement



Simple Power Analysis

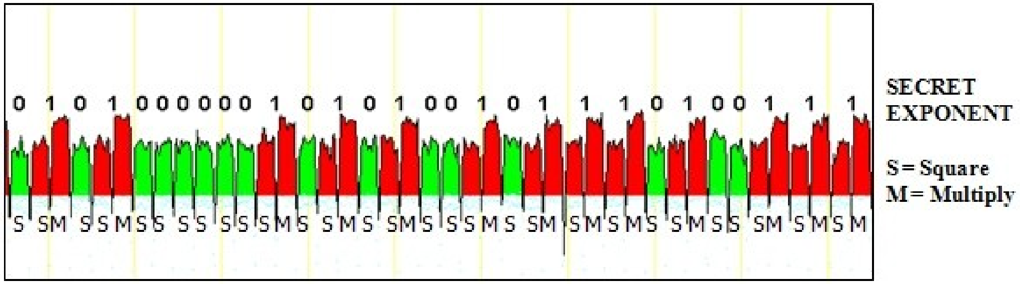


image source: eetimes.com

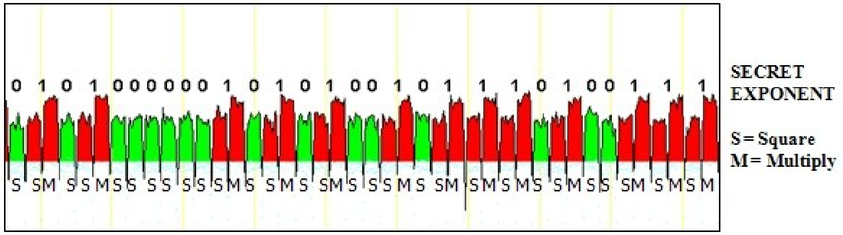
Simple Power Analysis

Idea

- Measure the power profile during a single crypto operation
- Power consumption is data dependent
- Extract information directly from the power trace
  e.g. square versus (square + multiply)

Pros

- Cheap equipment
- Low knowledge required



Cons

- Bad resolution: critical in case of low signal to noise ratio
- Modifications to the board are required
- Difficult in SoCs with a single power supply; no local information

image source: eetimes.com

Differential Power Analysis

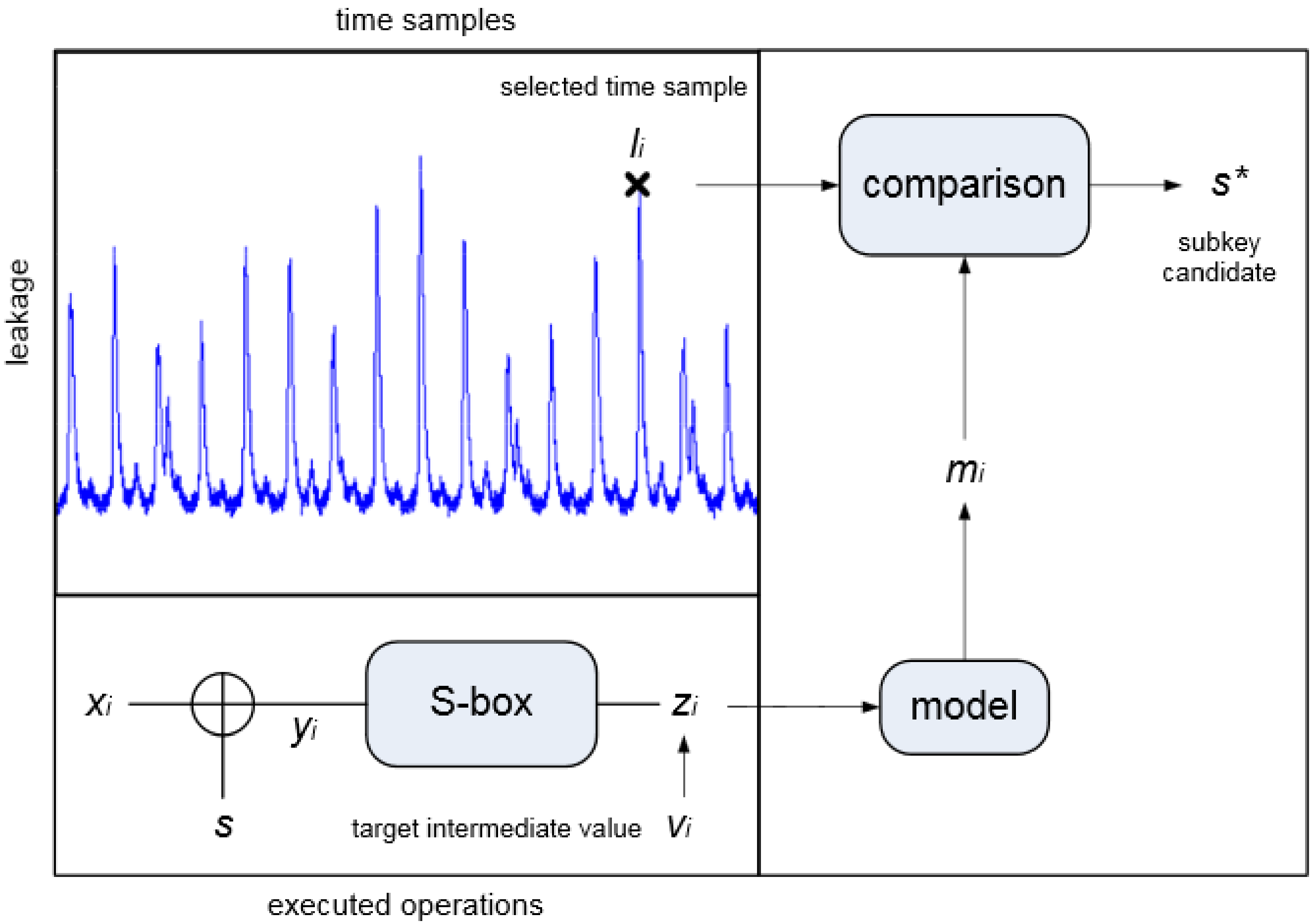


image source: Mangard, S., et al. One for all–all for one: unifying standard differential power analysis attacks. *IET Information Security*, *5*(2), 100-110.

Differential Power Analysis

Idea

- Measure the power profile during a crypto operation
- Repeat the measurements many times ($10^2 – 10^6$)
- Perform statistical correlation between the traces measured and a power model of the implementation

Pros

- Cheap equipment
- High resolution

Cons

- Modifications to the board are required
- Difficult in SoCs with a single power supply; no local information
- Expertise required

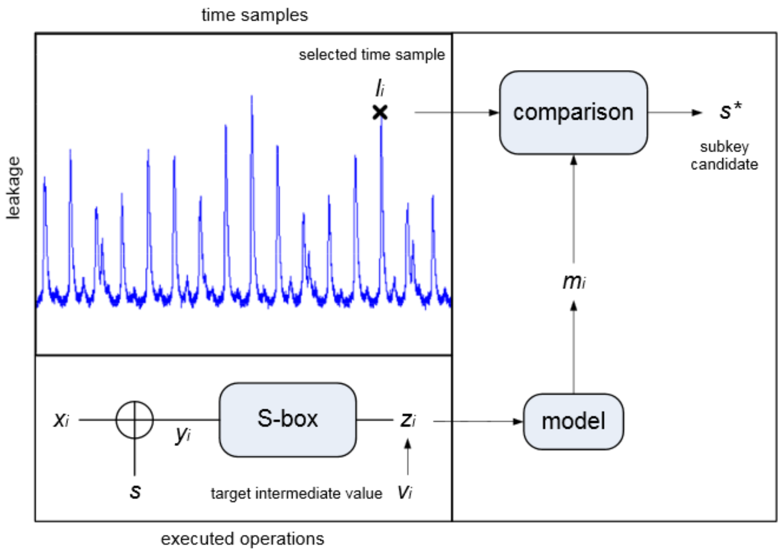
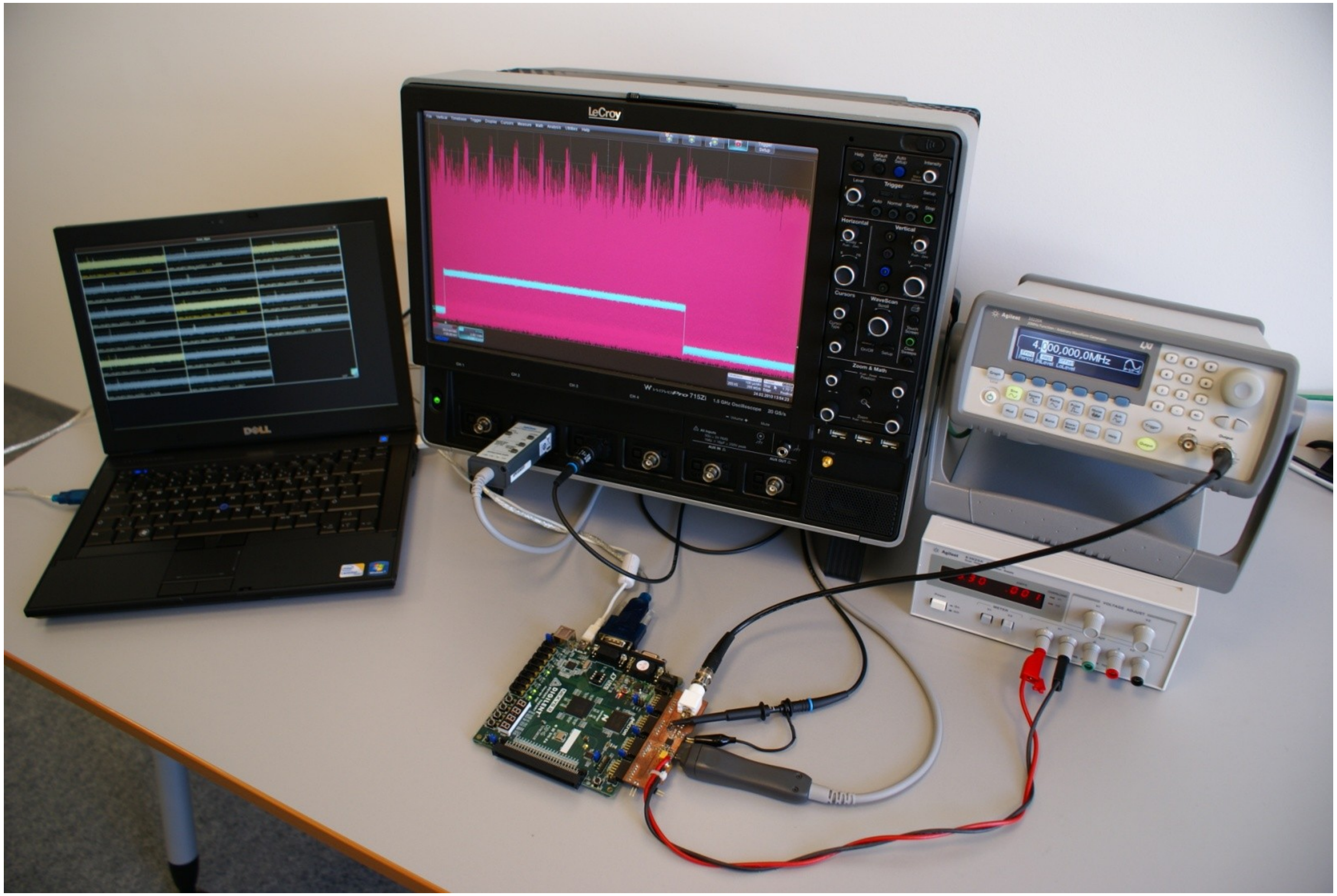


image source: Mangard, S., et al. One for all–all for one: unifying standard differential power analysis attacks. *IET Information Security*, *5*(2), 100-110.

Power measurement setup



Electromagnetic Analysis

Idea:

- Measure electromagnetic emanation during crypto operations
- Repeat this many times ($10^2 – 10^6$)
- Perform statistical correlation between the traces measured and an emission model of the implementation

Pros

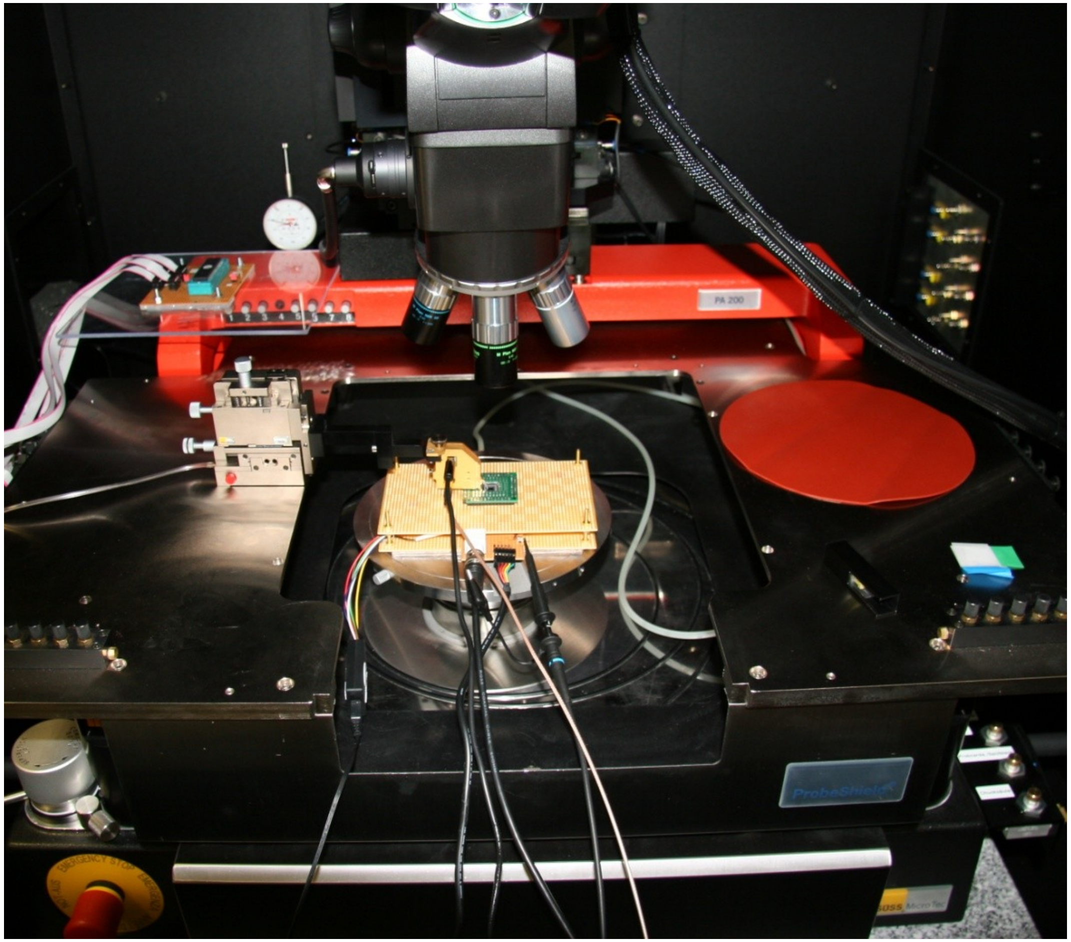
- High resolution
- No modifications to board are required
- Localized information (for localized EM)

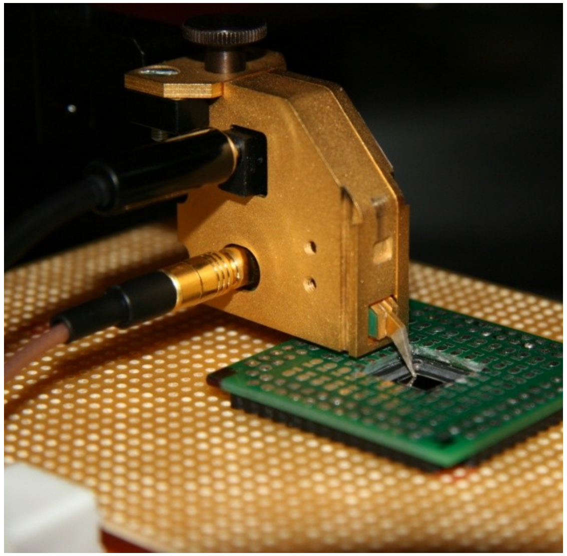
Cons

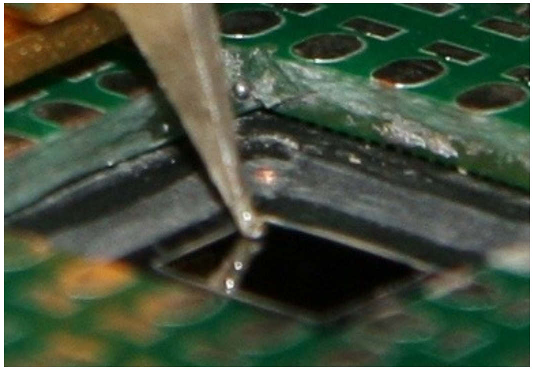
- Expertise required
- Chip de-capsulation (improves results drastically)

image source: http://aktuell.ruhr-uni-bochum.de/mam/images/pi2011/em-messung_smartcard.jpg

Localized EM Analysis (semi-invasive)







Section 2 – Side-channel Analysis

DIFFERENTIAL POWER ANALYSIS + AES

Differential Power Analysis

- DPA was published by Paul Kocher, Joshua Jaffe, Benjamin Jun in Proceedings of Crypto 1999

- Paul Kocher founded a company Cryptography Research Inc. CRI, which holds most of the patents for countermeasures against SCA (including DPA)

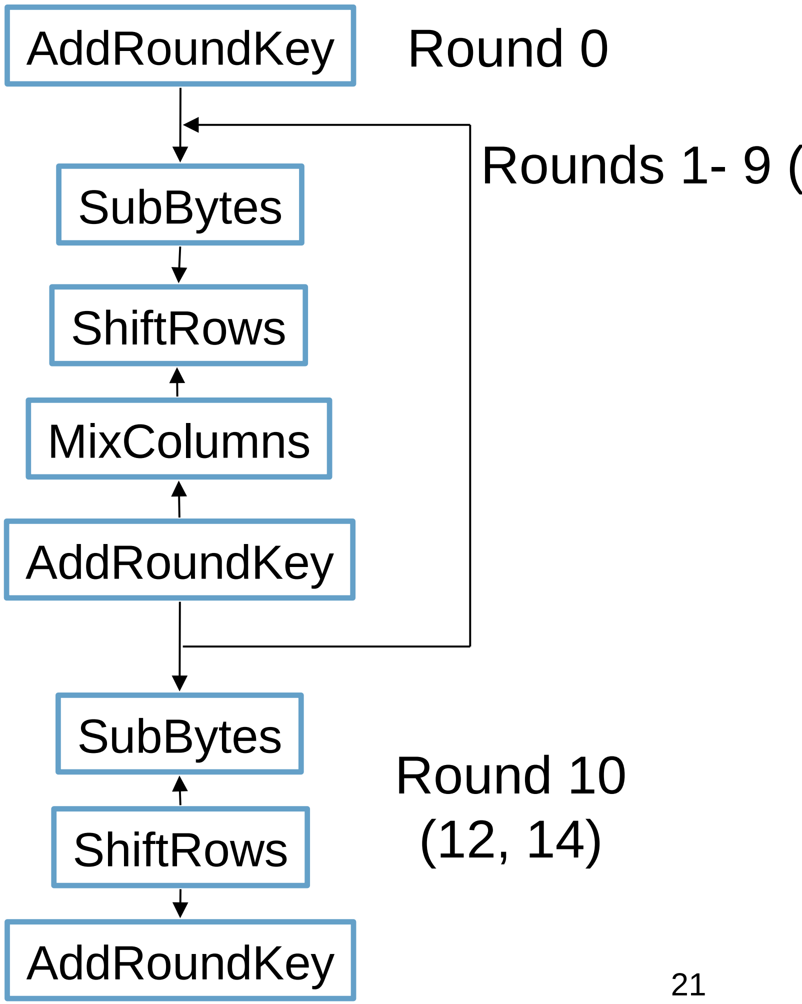
- On June 6, 2011 Cryptography Research was bought by Rambus in a deal worth $342.5 million

- For further information and nice videos about DPA see: http://www.cryptography.com/

Basic structure of the AES algorithm

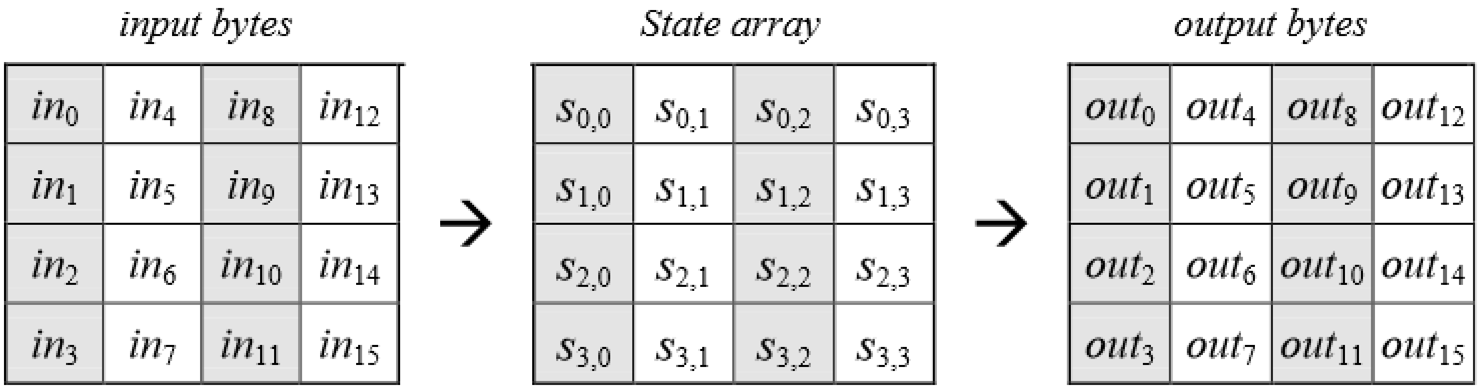
- 10, 12, or 14 rounds for key sizes 128, 192, 256 bits

- Last round is slightly different (no mixcolumns)



Advanced Encryption Standard

The AES algorithm operations are performed on a two-dimensional array of bytes called the **state**



input bytes

in_0	in_4	in_8	in_{12}
in_1	in_5	in_9	in_{13}
in_2	in_6	in_{10}	in_{14}
in_3	in_7	in_{11}	in_{15}

→

State array

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

→

output bytes

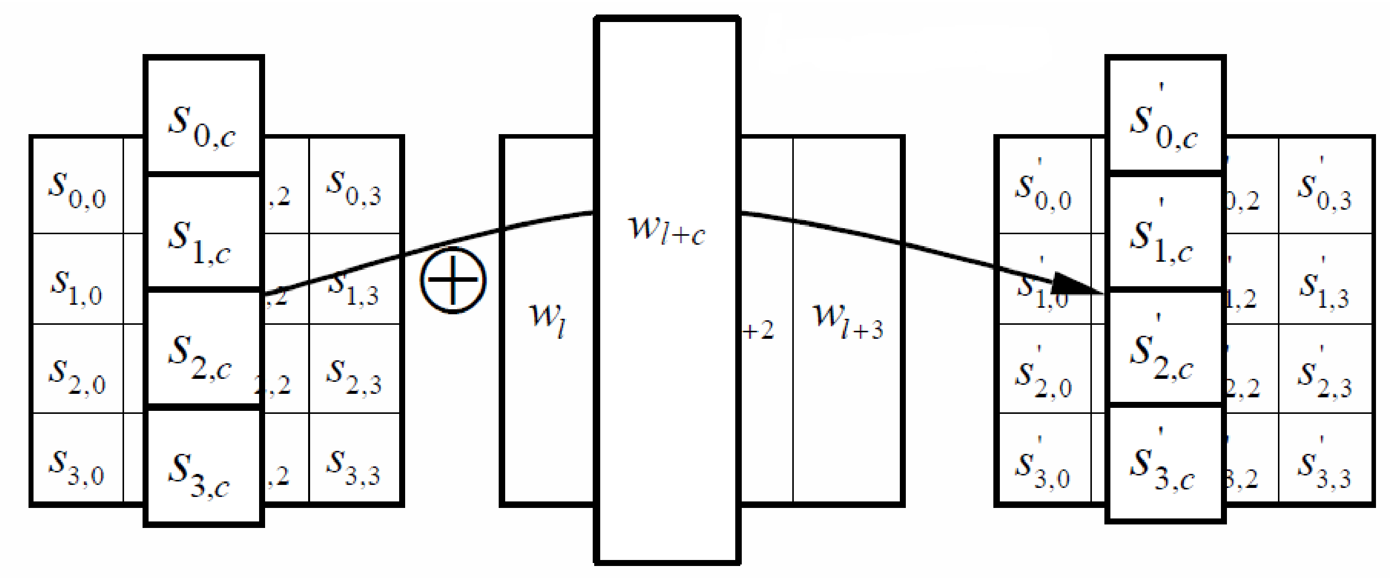
out_0	out_4	out_8	out_{12}
out_1	out_5	out_9	out_{13}
out_2	out_6	out_{10}	out_{14}
out_3	out_7	out_{11}	out_{15}

source: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Advanced Encryption Standard

AddRoundKey transformation

- The round key is added to the state
- Simple bitwise XOR operation

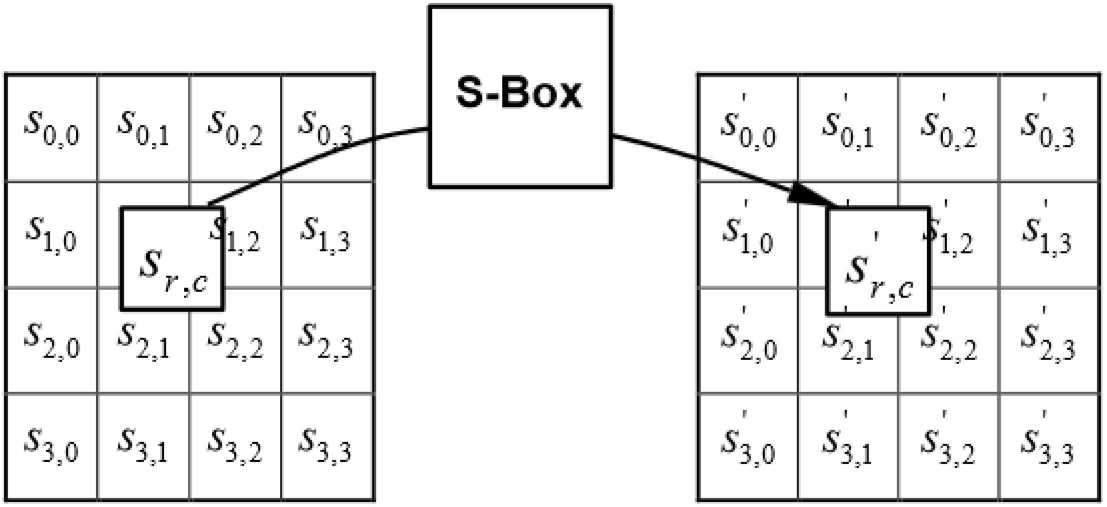


source: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Advanced Encryption Standard

SubBytes transformation

- Non-linear byte substitution
- Operates on each byte using a substitution table

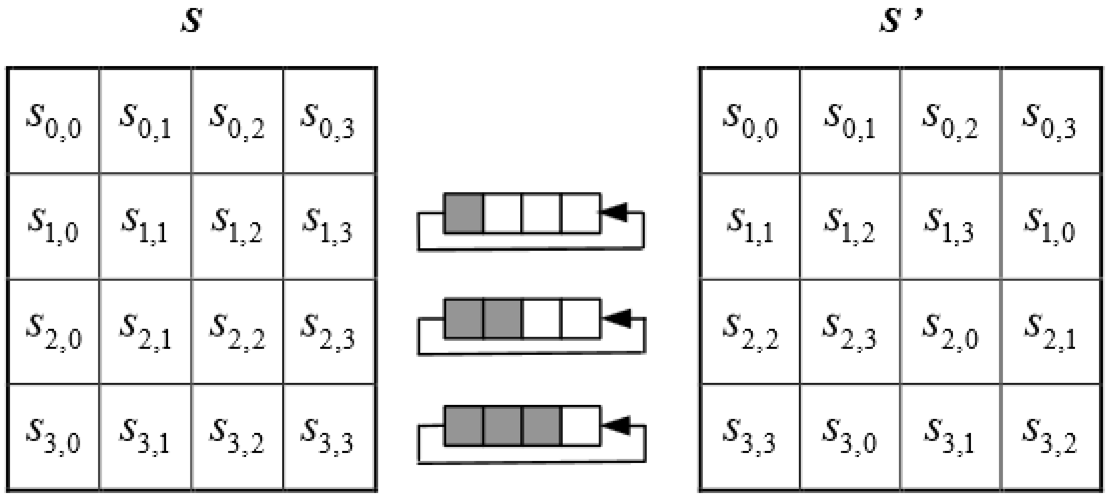

S-Box
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{r,c}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{r,c}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

source: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Advanced Encryption Standard

ShiftRows transformation

- Cyclical shift of rows
- Each row is cycled with a different offset



s

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

s'

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

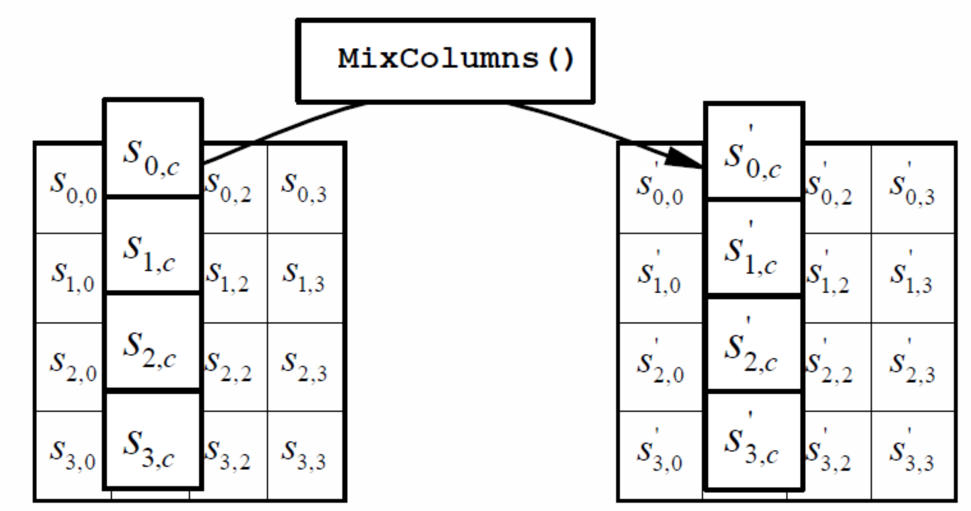
source: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Advanced Encryption Standard

MixColumns transformation

- Operates on the state one column at a time
- The columns are considered as polynomials over GF(2^8) and multiplied modulo x^4+1 with a fixed polynomial
- It can be seen as a matrix multiplication

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

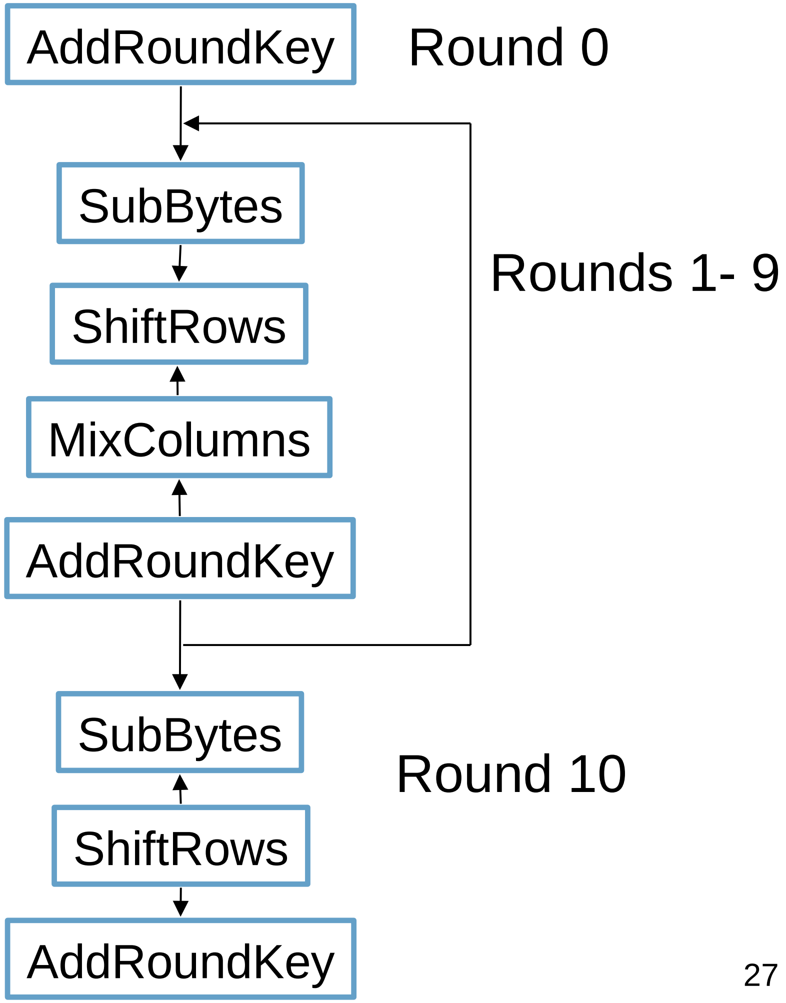

MixColumns()
$s_{0,0}$ $s_{0,c}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,c}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,c}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,c}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,c}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,c}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,c}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,c}$ $s'_{3,2}$ $s'_{3,3}$

source: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

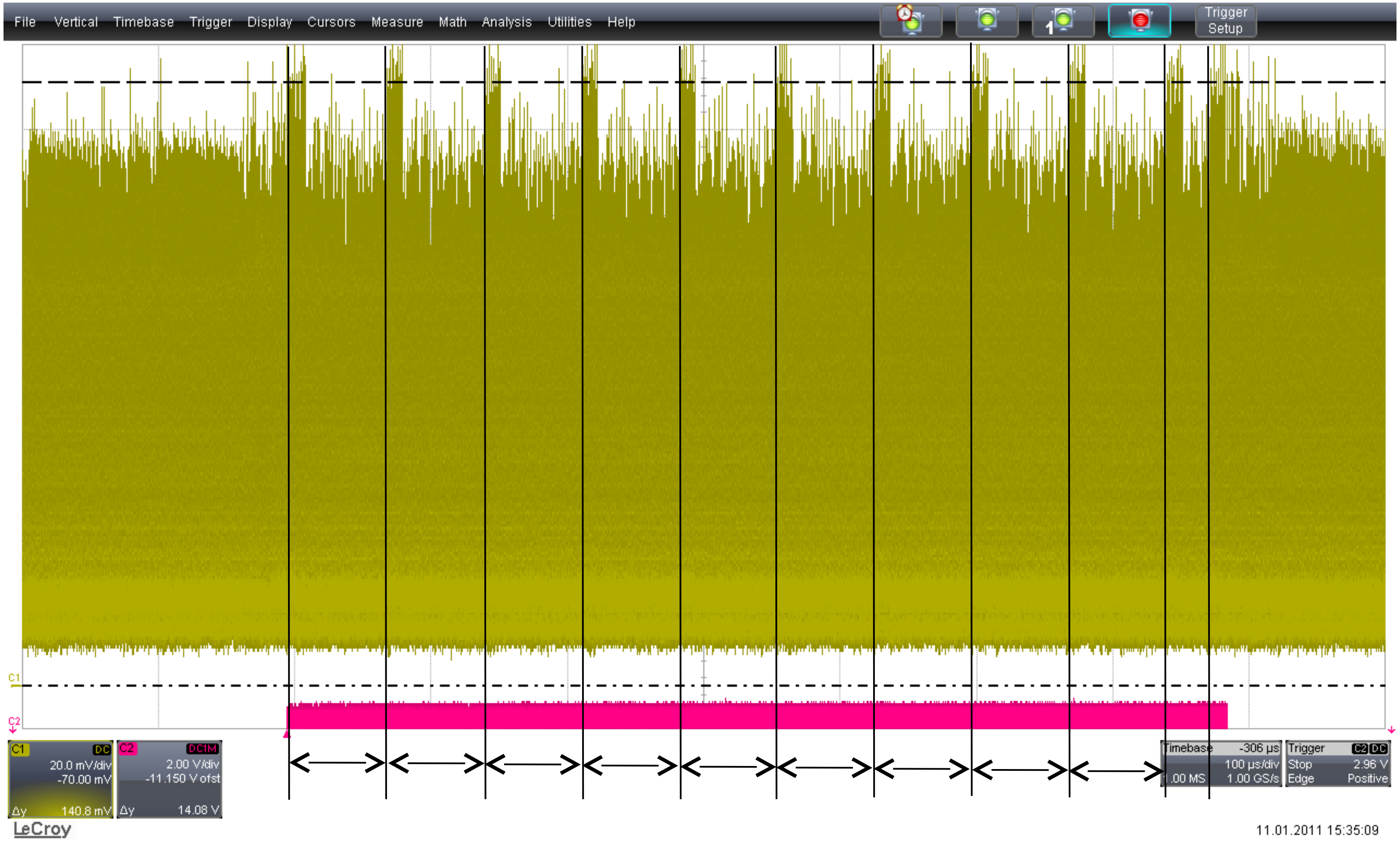
Basic structure of AES-128

AES-128

- Input size = 128 bit
- Output size = 128 bit
- Key size = 128 bit
- 10 Rounds



Power trace of an AES SW implementation



Power traces

What can be learned from power traces?

- Through visual inspection (as in SPA), we can get information about:
  - The parameters of the algorithm
    (e.g. # rounds in AES ➡ key length)
  - A closer look may identify characteristics of single instructions
  - Length differences of instructions (e.g. MUL, DIV) may help make educated guesses about the parameters
  - Differences in the instruction flow (key dependent jumps) may allow extracting secret data
  - Memory accesses often show characteristic power profiles
  - Cache hit and miss behavior can be easily identified

Differential Power Analysis

What is DPA?

- Power analysis attack which goal is to reveal the secret keys of a cryptographic device
- It assumes that the power consumption is a function of the secret data being processed
- The shape of the trace is not important (as in SPA)
- Uses a large number of power traces taken during the operation of the device
- Linear data relationships are determined by using statistical tools

Differential Power Analysis

Attack Strategy

1. Choosing an intermediate result of the executed algorithm
2. Measuring the power consumption
3. Calculating intermediate values
4. Mapping intermediate values to hypothetical power consumption values
5. Comparing the hypothetical power consumption values with the power consumption traces

Choosing an intermediate result

Choose an intermediate result of the algorithm which depends on data (known) and the key (unknown): *f* (d,k)

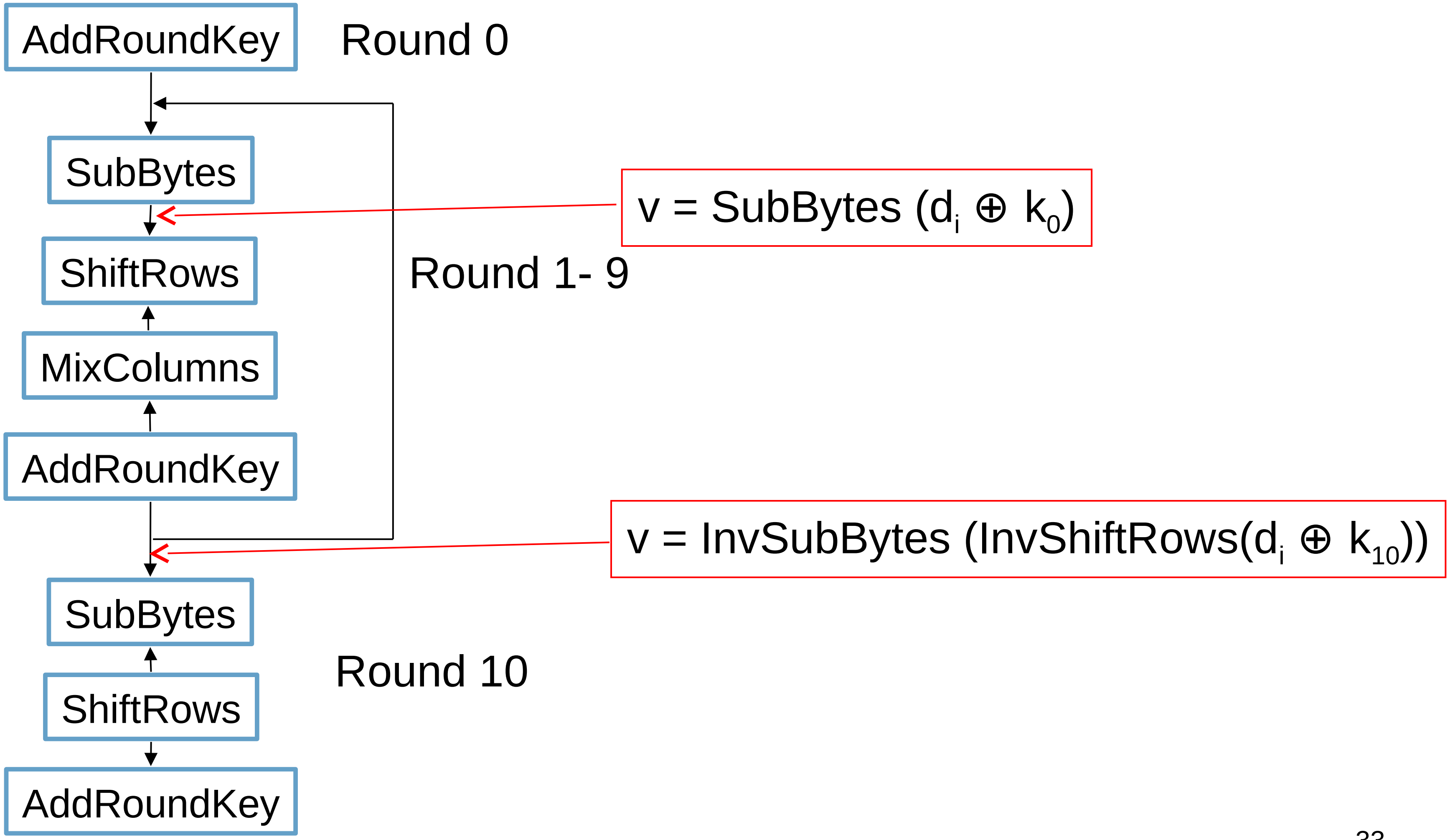
Most effective attacks on AES can be mounted at the

- S-box output of the first round (for encryption)
- S-box input of the last round (for decryption)

Why?

- Number of key hypothesis for the S-box is small (i.e. 2^8=256)
- The intermediate value depends on small part of the key *k* and known data *d*
- Non-linear elements of the S-box make the DPA more effective (i.e. a one bit difference at the input of an S-box leads to a difference of several bits at the output)

Choosing an intermediate result



v = SubBytes ($d_i \oplus k_0$)

v = InvSubBytes (InvShiftRows($d_i \oplus k_{10}$))

Measuring the power consumption

- Choose a set of D data values which will be encrypted (or decrypted). Store this values as a vector: **d** = (d_1,…d_D)’

- Measure the power consumption for each encryption of a data value d_i and store it in a power trace **t’$_i$** = ($t_{i,1}$, …$t_{i,T}$)’ with T samples

- Store all collected traces inside a matrix **T** of dimensions DxT.
- It is important to align the traces
  - All the values in a column of **T** have to belong to the same operation
  - Using a unique trigger for trace collection with the oscilloscope yields the best results

$$T = \begin{bmatrix} t_{1,1} & \cdots & t_{1,j} & \cdots & t_{1,T} \\ \vdots & \ddots & & & \vdots \\ t_{i,1} & & t_{i,j} & & t_{i,T} \\ \vdots & & & \ddots & \vdots \\ t_{D,1} & \cdots & t_{D,j} & \cdots & t_{D,T} \end{bmatrix}$$

Trace compression

Concept

- Power traces contain many points with lots of redundancy
- In order to speed up analysis it is favorable to reduce the number of points
- Such techniques receive the name of trace compression

There are many methods to compress a trace, e.g.:

- Sum of absolute values over a time interval
- Sum of squared values over a time interval
- Maximum value in a time interval

Calculating intermediate values

- List all possible key values and store it in a vector of size K (e.g. for the S-box K = 256 possible values since each key part is 8-bit long)
  k = $(k_1, \ldots, k_K)'$ (e.g. **k** = (0,1,2,… ,255)’ for an 8-bit value)
- Calculate a matrix **V** of possible intermediate results for all data and key hypothesis with elements $v_{i,j} = f(d_i, k_j)$ using the function which was chosen in the first step

$$V = \begin{bmatrix} v_{1,1} & \cdots & v_{1,j} & \cdots & v_{1,K} \\ \vdots & \ddots & & & \vdots \\ v_{i,1} & & v_{i,j} & & v_{i,K} \\ \vdots & & & \ddots & \vdots \\ v_{D,1} & \cdots & v_{D,j} & \cdots & v_{D,K} \end{bmatrix}$$

This matrix will contain a column with j = ck (correct key) where all the data elements d_i will be processed with the correct key.

Hypothetical power consumption values

- Calculate the power consumption for each element in the matrix **V** of possible intermediate results
- Make use of a power model to estimate the power consumption
  - Hamming weight
  - Hamming Distance
- This results in a matrix **H** of hypothetical power consumption:

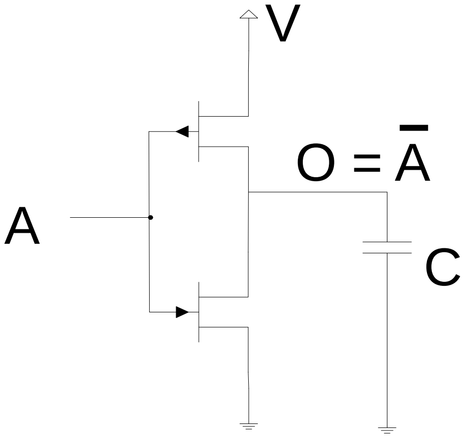
$$H = \begin{bmatrix} h_{1,1} & \cdots & h_{1,j} & \cdots & h_{1,K} \\ \vdots & \ddots & & & \vdots \\ h_{i,1} & & h_{i,j} & & h_{i,K} \\ \vdots & & & \ddots & \vdots \\ h_{D,1} & \cdots & h_{D,j} & \cdots & h_{D,K} \end{bmatrix}$$

This matrix contains a column with j = ck where all power values will be correlated with a power value in the measurements.

CMOS Power consumption model

Hamming Distance

- Power consumption proportional to the number of transitions from 0 -> 1 and 1-> 0
- Assumptions
  - Transitions from 0->1 and 1->0 consume the same power
  - Transitions from 0->0 and 1->1 do not consume power



$P = C*V^2$ if O(t-1)=0 and O(t)=1

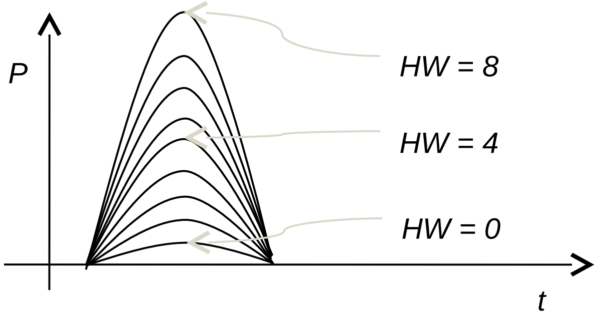
$P = 0$ if O(t-1) = O(t) or O(t-1) = 1 and O(t)=0

Hamming Distance = |A(t) – A(t-1)|

Power consumption models

Hamming Weight

- Most commonly used
- Simpler than the Hamming Distance model
- Used when the attacker only knows one data value being transferred (i.e. no information about the previous value)
- Assumption
  - Power consumption is proportional to the number of bits that are set in the processed value



Statistical Correlation

Pearson Correlation Coefficient

- The covariance is a measure of the statistical dependence of two random variables, i.e. power traces **t**=(t_i); **q**=(q_i) for i = 1,…,n

$$Cov(t,q) = E(t \cdot q) - E(t) \cdot E(q)$$

- If **t** and **q** are independent E(**t**·**q**) = E(**t**) · E(**q**) ➡ Cov(**t**,**q**) = 0
- The correlation coefficient ρ is a normalized measure for the dependence with -1 ≤ ρ ≤ 1 and ρ=0 for independent variables.

$$\rho = \frac{Cov(\mathbf{t},\mathbf{q})}{\sqrt{Var(\mathbf{t}) \cdot Var(\mathbf{q})}} \approx r = \frac{\sum_{i=1}^{n}(t_i - \bar{t}) \cdot (q_i - \bar{q})}{\sqrt{\sum_{i=1}^{n}(t_i - \bar{t})^2 \cdot \sum_{i=1}^{n}(q_i - \bar{q})^2}}$$

Statistical Correlation

In this step the correlation coefficient between the columns of the matrix of measured traces **T** and the columns of the matrix of hypothetical values **H** is calculated for all points in time.
The estimated correlation coefficient $r_{i,j}$ is calculated by taking column *i* from matrix **H** and column *j* from matrix **T**:

$$r_{i,j} = \frac{\sum_{d=1}^{D}[(t_{d,j} - \bar{t}_j) \cdot (h_{d,i} - \bar{h}_i)]}{\sqrt{\sum_{d=1}^{D}(t_{d,j} - \bar{t}_j)^2 \cdot \sum_{d=1}^{D}(h_{d,i} - \bar{h}_i)^2}}$$

The result is a KxT matrix of correlation coefficients **R**:

$$R = \begin{bmatrix} r_{1,1} & \cdots & r_{1,j} & \cdots & r_{1,T} \\ \vdots & \ddots & & & \vdots \\ r_{i,1} & & r_{i,j} & & r_{i,T} \\ \vdots & & & \ddots & \vdots \\ r_{K,1} & \cdots & r_{K,j} & \cdots & r_{K,T} \end{bmatrix}$$

Finding the most probable result

The key byte and the time when it is used in the power trace can be obtained by finding

$$r_{ck,ct} = \max_{i,j}(abs(r_{i,j})); \qquad ck = i; \qquad ct = j$$

ck is the index of the row with correct key
ct is the index of the column with the correct time

$$R = \begin{bmatrix} r_{1,1} & \cdots & r_{1,j} & \cdots & r_{1,T} \\ \vdots & \ddots & & & \vdots \\ r_{i,1} & & r_{i,j} & & r_{i,T} \\ \vdots & & & \ddots & \vdots \\ r_{K,1} & \cdots & r_{K,j} & \cdots & r_{K,T} \end{bmatrix}$$

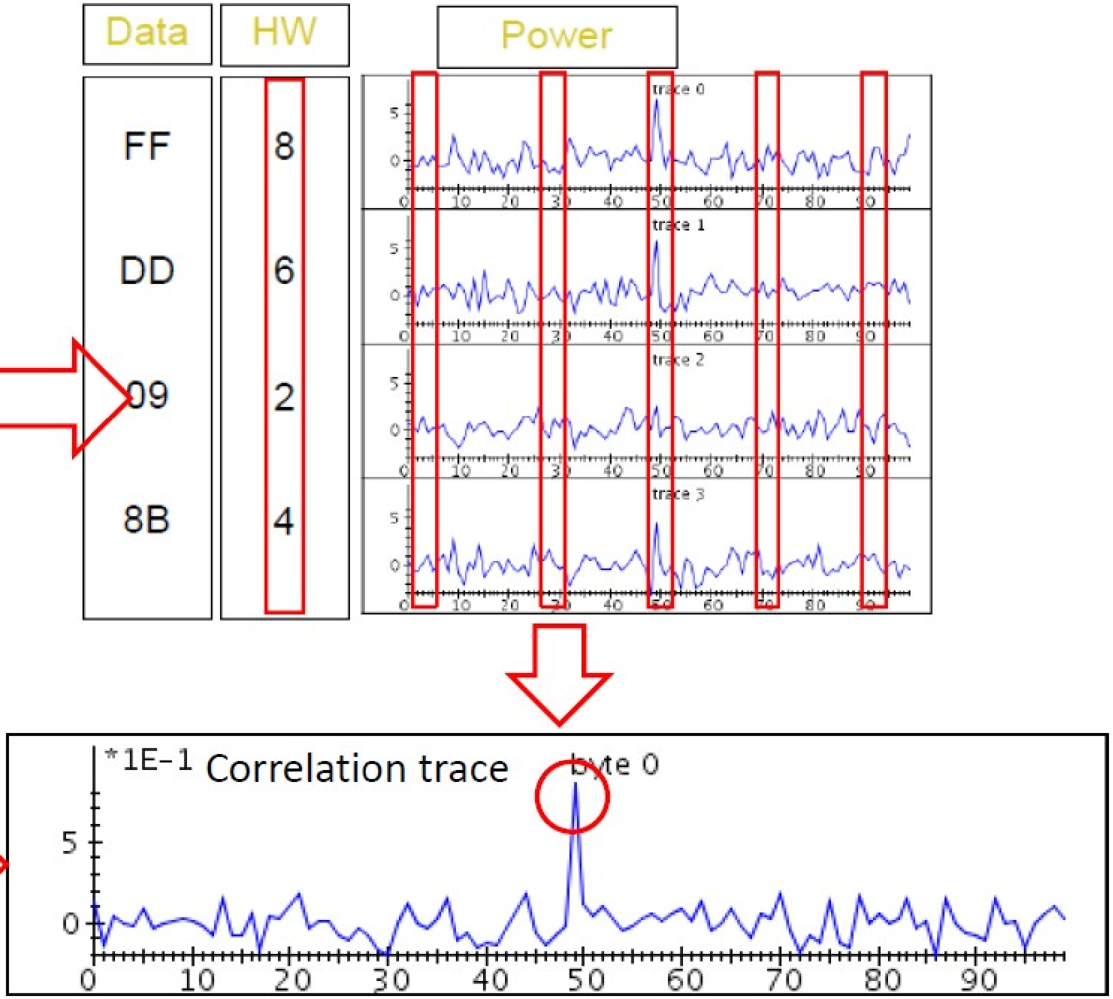
← key with
maximum correlation

↑

time of maximum correlation

Testing if a key byte is correct

Correlate the hypothetical intermediate values to the power traces

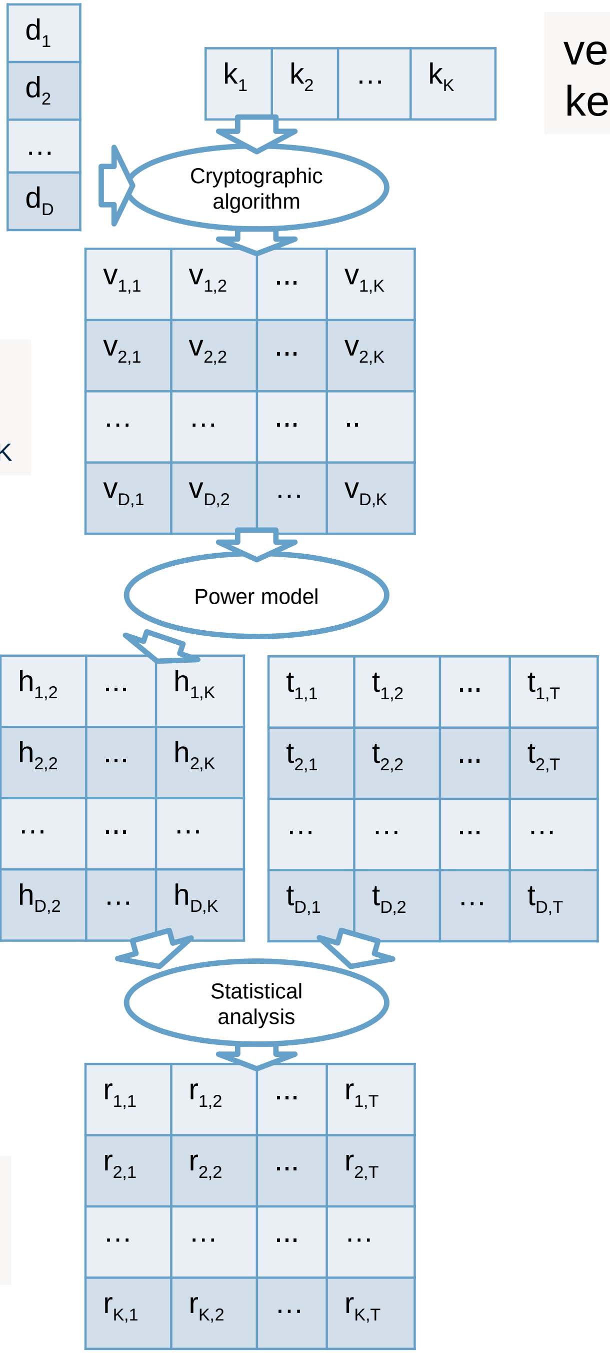


Data	HW
FF	8
DD	6
09	2
8B	4

Only the right key guess exhibits high peaks

source: Cloning 3G/4G SIM Cards with a PC and an Oscilloscope: Lessons Learned in Physical Security – Yu Yu – Black Hat 2015 vector **d** with D plaintext/ciphertexts

vector **k**‘ with K key hypothesis



d_1
d_2
…
d_D

k_1	k_2	…	k_K

$v_{1,1}$	$v_{1,2}$	…	$v_{1,K}$
$v_{2,1}$	$v_{2,2}$	…	$v_{2,K}$
…	…	…	..
$v_{D,1}$	$v_{D,2}$	…	$v_{D,K}$

intermediate values $v_{i,j}$ with all key hypothesis ➡ $\mathbf{V}_{DxK}$

$h_{1,1}$	$h_{1,2}$	…	$h_{1,K}$
$h_{2,1}$	$h_{2,2}$	…	$h_{2,K}$
…	…	…	…
$h_{D,1}$	$h_{D,2}$	…	$h_{D,K}$

hypothetical power values $h_{i,j}$ ➡ $\mathbf{H}_{DxK}$

$t_{1,1}$	$t_{1,2}$	…	$t_{1,T}$
$t_{2,1}$	$t_{2,2}$	…	$t_{2,T}$
…	…	…	…
$t_{D,1}$	$t_{D,2}$	…	$t_{D,T}$

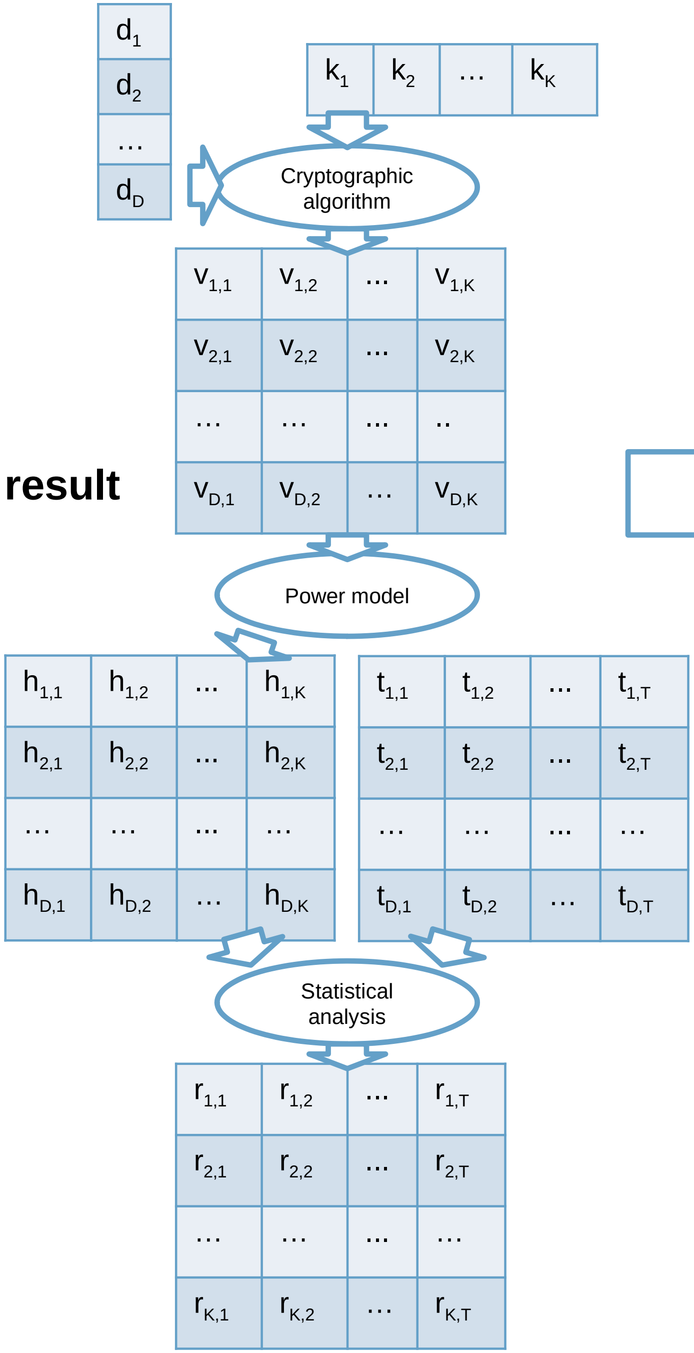
Power traces for T measuring points, D data values and alignment to same trigger ➡ $\mathbf{T}_{DxT}$

$r_{1,1}$	$r_{1,2}$	…	$r_{1,T}$
$r_{2,1}$	$r_{2,2}$	…	$r_{2,T}$
…	…	…	…
$r_{K,1}$	$r_{K,2}$	…	$r_{K,T}$

Matrix of correlation coefficients ➡ $\mathbf{R}_{KxT}$

Section 2 – Differential Power Analysis

COUNTERMEASURES



d_1
d_2
…
d_D

k_1	k_2	…	k_K

Cryptographic algorithm

$v_{1,1}$	$v_{1,2}$	…	$v_{1,K}$
$v_{2,1}$	$v_{2,2}$	…	$v_{2,K}$
…	…	…	..
$v_{D,1}$	$v_{D,2}$	…	$v_{D,K}$

Intermediate result

Power model

$h_{1,1}$	$h_{1,2}$	…	$h_{1,K}$
$h_{2,1}$	$h_{2,2}$	…	$h_{2,K}$
…	…	…	…
$h_{D,1}$	$h_{D,2}$	…	$h_{D,K}$

$t_{1,1}$	$t_{1,2}$	…	$t_{1,T}$
$t_{2,1}$	$t_{2,2}$	…	$t_{2,T}$
…	…	…	…
$t_{D,1}$	$t_{D,2}$	…	$t_{D,T}$

Power

Statistical analysis

$r_{1,1}$	$r_{1,2}$	…	$r_{1,T}$
$r_{2,1}$	$r_{2,2}$	…	$r_{2,T}$
…	…	…	…
$r_{K,1}$	$r_{K,2}$	…	$r_{K,T}$

Masking of intermediate values

Hiding power consumption

Hiding the power consumption

Increasing noise

- Time domain
  - Insertion of random wait states
  - Shuffling instructions, memory accesses, etc…
- Amplitude domain
  - Perform dummy operations in parallel to crypto algorithm
  - Dedicated noise generators on smartcards

Attenuation of the side-channel signal

- Amplitude domain
  - Special circuit design styles to achieve logic value independent power consumption
  - Filtering the power supply

Hiding in SW

Randomization

- Random wait state insertion (nop’s)
- Randomize instruction execution
- Randomize memory accesses

Generate noise

- Make use of peripherals which generate noise in parallel to the cryptographic function

Masking internal values

Description

- Make the power consumption of the device independent of the intermediate values of the cryptographic algorithm through randomization
- Avoids having to modify the power consumption characteristics of the device
- Can be implemented at the algorithm level

Concept

- Each intermediate value *v* is concealed by a random value *m*
- The value *m* varies in each execution and cannot be predicted

$$v_m = v * m$$

* = Masking operation $\oplus$, +, ·

Masking types

Types

- Boolean masking (with xor ⊕): $v_m = v \oplus m$
  - Mostly applied in symmetric cryptography i.e. AES
- Arithmetic masking (with + or ·): $v_m = v + m$ or $v_m = v \cdot m$
  - Mostly applied in asymmetric cryptography

Masking of linear functions

- $f(v_m) = f(v) * f(m)$

Masking of nonlinear functions is difficult

- $f(v_m) \neq f(v) * f(m)$
  - For example the AES S-box is nonlinear:
    $S(v \oplus m) \neq S(v) \oplus S(m)$

* = Masking operation ⊕, +, ·

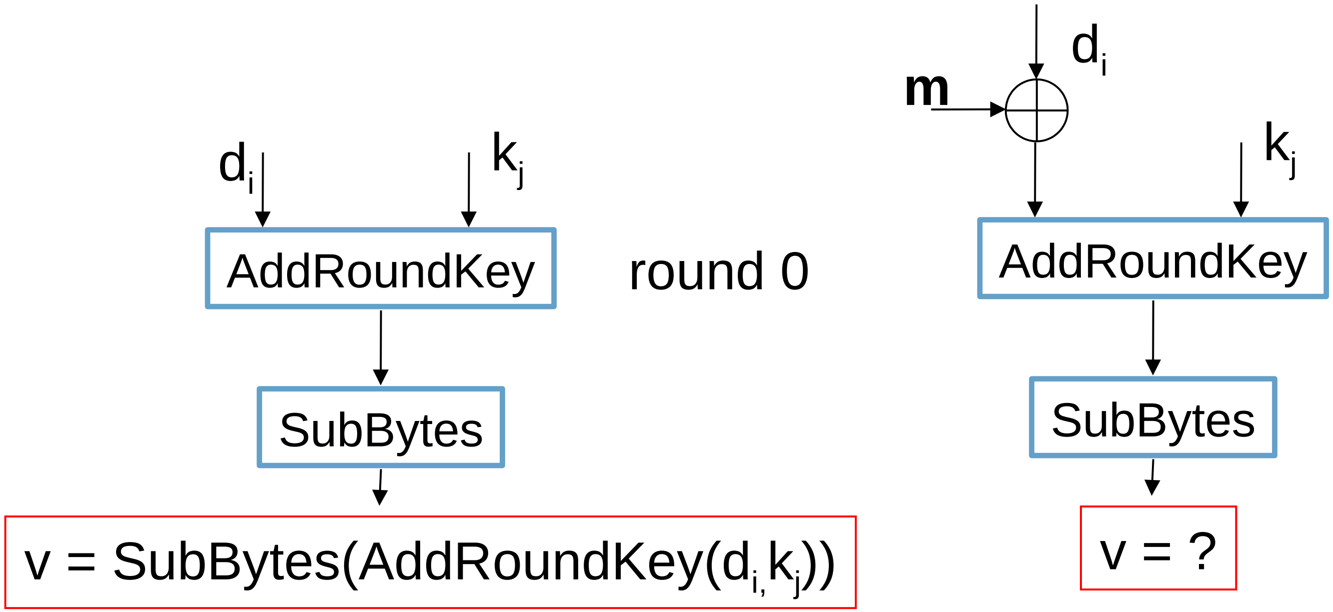
Masking Example

Masking is the most widely used countermeasure in software

In hardware masking can be implemented on any design level

What is the impact on masking with an **unknown mask m** on SCA?

- Intermediate values are concealed



Masking in software for AES

Simple masking scheme for AES

- Data (or key) are XOR‘ed with a random mask at the beginning
- During each round the transformations are applied to the masked data and the mask itself
  - AddRoundKey: apply the function only to the masked data
  - SubBytes: Need for a masked S-box table
  - Shift Rows: Shifting is applied to mask and data separately (unless all bytes in the state are masked with the same value, in that case it does not affect the masking)
  - MixColumns: Applied to mask and data separately
- After the last round, the mask is removed

AES software masking scheme

For the SW implementation 6 independent masks are used

- 2 masks *m* and *m’* for the S-box input and output
- 4 masks for each row of the state m_1,m_2,m_3,m_4 for the input of MixColumns

Pre-computation (done for every encryption)

- Generate the six masks at random
- Calculate the masked S-box, $S_m(x \oplus m) = S(x) \oplus m'$
- Execute MixColumns for the masks m_1,m_2,m_3,m_4 to create the output masks m'_1,m'_2,m'_3,m'_4

Execute the AES algorithm applying the masks on the state and round key

C. Herbst, M. E. Oswald, S. Mangard, "An AES Smart Card Implementation resistant to Power Analysis Attacks“, Applied Cryptography and Network security, 2006

AES (software) masking scheme

Rounds execution

- Mask the plaintext with [m'_1,m'_2,m'_3,m'_4]
- Mask the round key with [m'_1 *xor m*, m'_2 *xor m*, m'_3 *xor m*,m'_4 *xor m*]
- Perform AddRoundKey masks will become [*m, m, m, m*]
- Perform SubBytes and ShiftRows masks will become [*m', m', m', m'*]
- Do re-masking to end up with the input masks for MixedColumns [m_1,m_2,m_3,m_4], masks after MixedColumns [m'_1,m'_2,m'_3,m'_4]

In the final round skip re-masking and MixedColumns

Remove the masks from the ciphertext

State

m'1	m'1	m'1	m'1
m'2	m'2	m'2	m'2
m'3	m'3	m'3	m'3
m'4	m'4	m'4	m'4

Round Key

m'1⊕ m	m'1 ⊕ m	m'1 ⊕ m	m'1 ⊕ m
m'2 ⊕ m	m'2 ⊕ m	m'2 ⊕ m	m'2 ⊕ m
m'3 ⊕ m	m'3 ⊕ m	m'3 ⊕ m	m'3 ⊕ m
m'4 ⊕ m	m'4 ⊕ m	m'4 ⊕ m	m'4 ⊕ m

AddRoundKey

m	m	m	m
m	m	m	m
m	m	m	m
m	m	m	m

SubBytes, ShiftRows

m'	m'	m'	m'
m'	m'	m'	m'
m'	m'	m'	m'
m'	m'	m'	m'

Remasking

m1	m1	m1	m1
m2	m2	m2	m2
m3	m3	m3	m3
m4	m4	m4	m4

Mixcolumns

m'1	m'1	m'1	m'1
m'2	m'2	m'2	m'2
m'3	m'3	m'3	m'3
m'4	m'4	m'4	m'4

Attacks on Masking

Masking protects against DPA if

- There is no joint power consumption of the masked value and mask
- The mask is uniformly distributed

Implementation pitfalls

- If masks are not changed frequently enough, DPA is still possible
- Masks may be biased due to insufficient statistical properties of the PRNG generating the masks
- Binning of the traces may be possible if (global) mask changes can be detected
- If masks are reused, operations with values (u, v), which are masked with the same mask m, may show the plain values
  $(u \oplus m) \oplus (v \oplus m) = u \oplus v$
- Hamming Distance of a register may leak the value being protected
  $HD(v_m, m) = HW(v_m \oplus m) = HW(v)$

Power Analysis Attacks – Revealing the Secrets of Smart Cards, Stefan Mangard et al., Chapter 10, Attacks on Masking

Further Information

The book from Stefan Mangard, Thomas Popp, Elisabeth Oswald provides all necessary know how and detailed mathematical background to perform power attacks. http://www.dpabook.org/

SICA lecture[1] at TUM every winter term,
provides very good training in this topic. (taught in German)
Stefan Mangard’s material (2012) can be found under:
http://www.physical-security.org

Power analysis attacks are in the focus of the research community. Therefore new attack flavors are published every year.
www.chesworkshop.org
https://www.cosade.org/

[1] http://www.sec.ei.tum.de/lehrveranstaltungen/sichere%20implementierung%20kryptographischer%20verfahren/

HDF5 INTRODUCTION

HDF5 Format

Hierarchical Data Format 5

- File format designed to store and organize large amounts of data
- BSD-like license (minimal restrictions)
- Official support for C/C++, Fortran, Java.
- Third party support for Python, Matlab, R, Perl, LabView, Julia, etc…
- Access to resources in a POSIX-like style
  - /path/to/resource

Object type:

- Datasets: Multidimensional arrays of a homogeneous type
- Groups: Container structures, they can contain:
  - Datasets
  - Other groups
- Metadata: User-defined, named attributes. May be attached to datasets and groups

HDF5 Chunks

(In a nutshell)

Higher dimension illusion: data in a disk is stored linearly

$$\begin{bmatrix} A & B & C \\ D & F & G \end{bmatrix} = \begin{cases} [A\ B\ C\ D\ F\ G] \text{ – } \textit{row-major} \text{ ordering} \\ [A\ D\ B\ F\ C\ G] \text{ – } \textit{column-major} \text{ ordering} \end{cases}$$

locality: memory reads from a disk are generally faster when the data being accessed is all stored together.

$[\boxed{A\ D}\ B\ F\ C\ G]$ $\qquad$ $[\boxed{A}\ B\ C\ \boxed{D}\ F\ G]$

Chunking let’s you specify the n-dimentional shape that best fits your access pattern.

HDF5 cheat sheet

Python (h5py)

Opening a file (read):

```
f = h5py.File("name.h5", "r")
```

Listing the group members (keys)

```
f.keys()
```

Displaying all the attributes of the root object

```
f.values()
```

Creating a group (file must be open as append or write)

```
group = f.create_group("/some/long/path")
```

Creating a dataset

```
data = f.create_dataset(“dataset1", (10, 10))
```

Reading a previously generated dataset

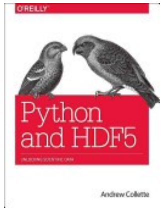
```
data = f[“ciphertext”]
```

Closing a file:

```
f.close()
```

References:

- https://opac.ub.tum.de/search?bvnr=BV041778278
- http://docs.h5py.org/en/latest/



HDF5 Cheat sheet

Matlab

Importing data from a file

```
data = h5read(filename, datasetname)
data = h5read(‘name.h5’,’/dataset1’)
```

Creating a dataset

```
h5create(filename,datasetname,size,Name,Value)
```

Writing data to a file

```
h5write(filename, datasetname, data)
h5create(‘name.h5','/dataset1',[10 20])
```

Reference:

- *http://de.mathworks.com/help/matlab/high-level-functions.html*

Questions?

THANK YOU FOR YOUR ATTENTION!