Intro to Physical Side Channel Attacks

Thomas Eisenbarth

15.06.2018

Summer School on Real-World Crypto & Privacy

Šibenik, Croatia

UNIVERSITÄT ZU LÜBECK
STIFTUNGSUNIVERSITÄT
SEIT 2015

WPI

Outline

- **Why physical attacks matter**
- Implementation attacks and power analysis
- Leakage Detection
- Side Channel Countermeasures

Train Theft of MoD Laptop

Train theft of MoD laptop with fighter secrets alarmed Pentagon:

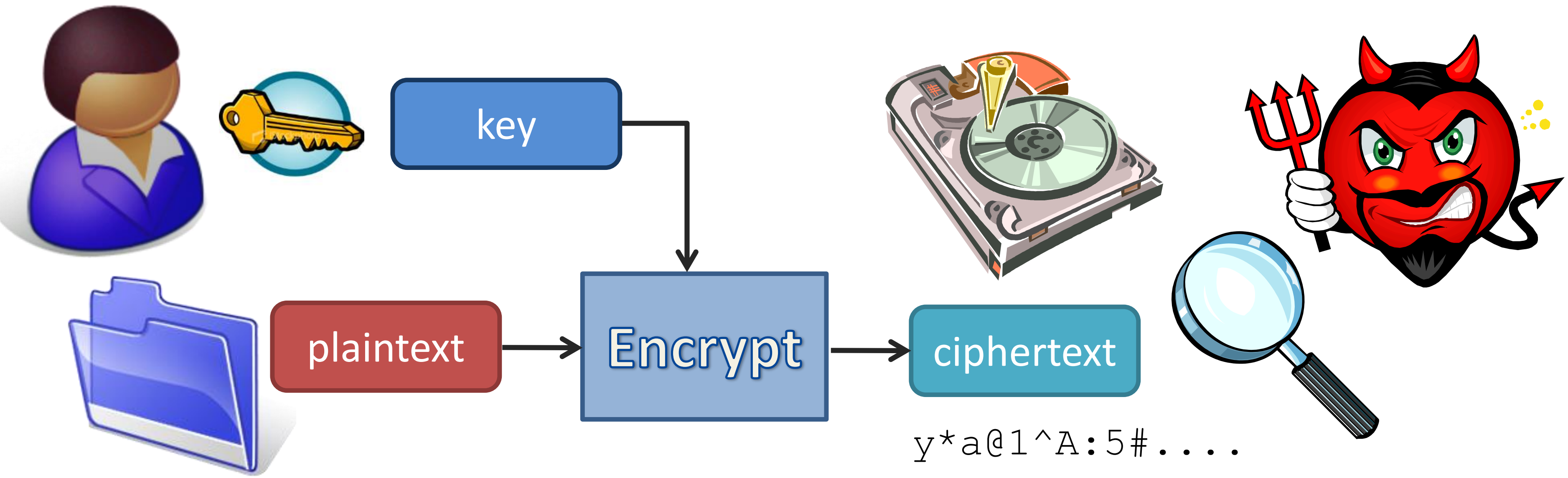
[…] **a laptop was stolen containing secrets of the biggest military procurement project ever launched** […]. It held details of progress on the development of the United States' supersonic joint strike fighter. […]

A petty thief stole the laptop from a British military officer at Paddington station in London last May. **It had been left on the luggage rack on a train**. […]

The computer is believed to have passed through several hands before it was returned to the Ministry of Defence. The thief was caught and later convicted. […]

The Guardian, Tuesday 6 February 2001:

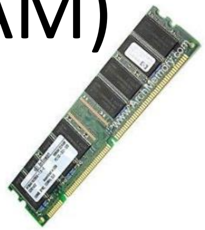
Solution: Hard Disk Encryption



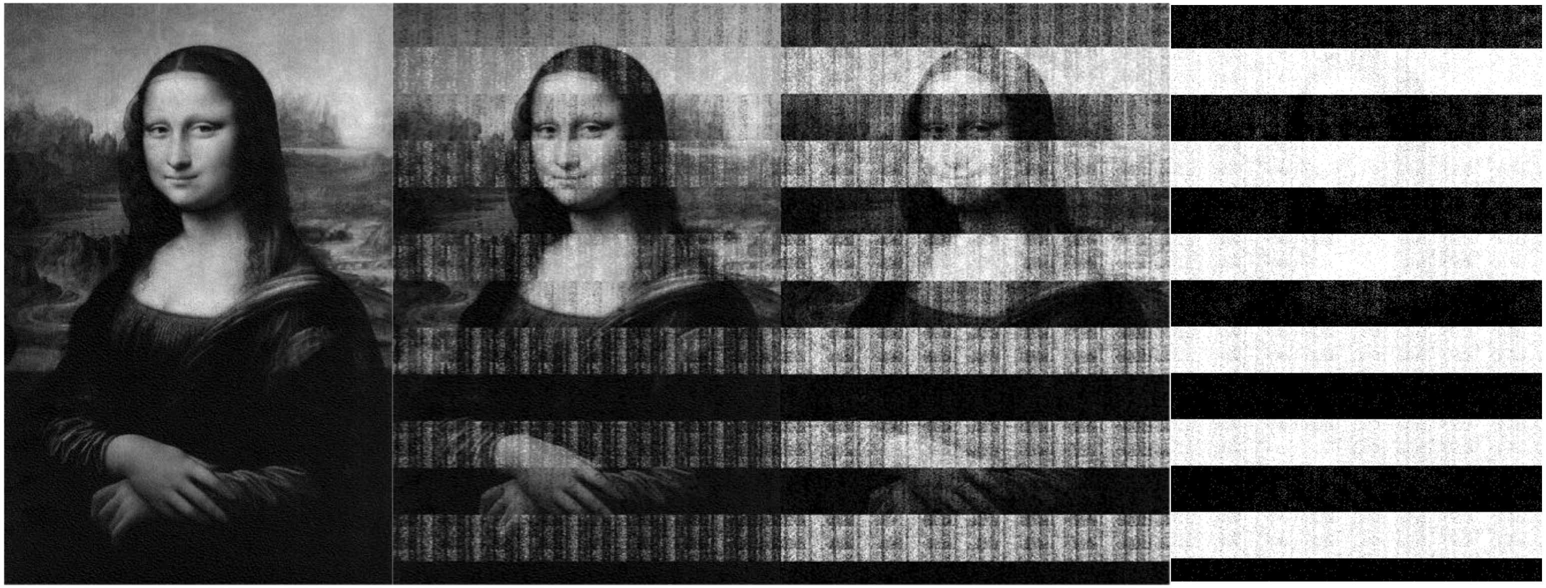
- Hard Disk Encryption available on all major OSs
- Enabled by default on mobile phones
- Solves Problem: Good password sufficient for secure storage

Problem: Physical Attacks

Problem: your key is stored in memory (DRAM)



This happens if you cut power:



5 secs | 30 secs | 60 secs | 300 secs

Cold Boot Attacks

Lunchtime Attack:

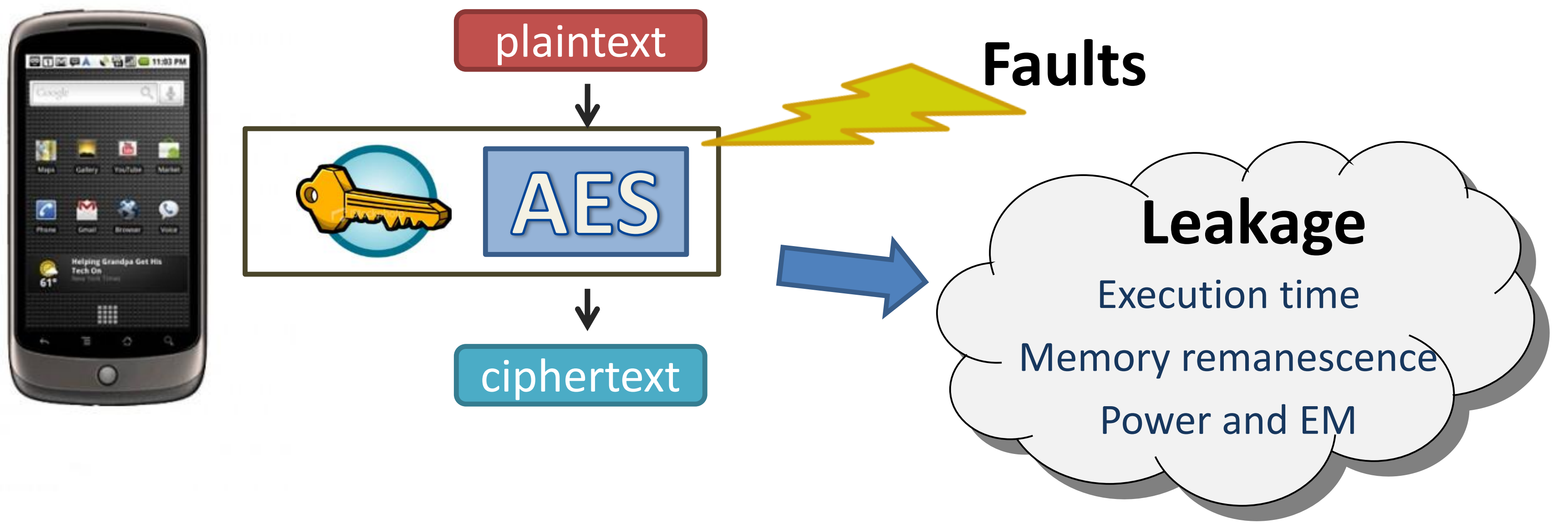
- data will persist for minutes if chips are cooled
- Keys easily recovered from memory content

Physical Access is needed

Halderman; Schoen; Heninger; Clarkson; Paul; Calandrino; Feldman; Appelbaum; Felten: *Lest We Remember: Cold Boot Attacks on Encryption Keys,* USENIX Security 2008

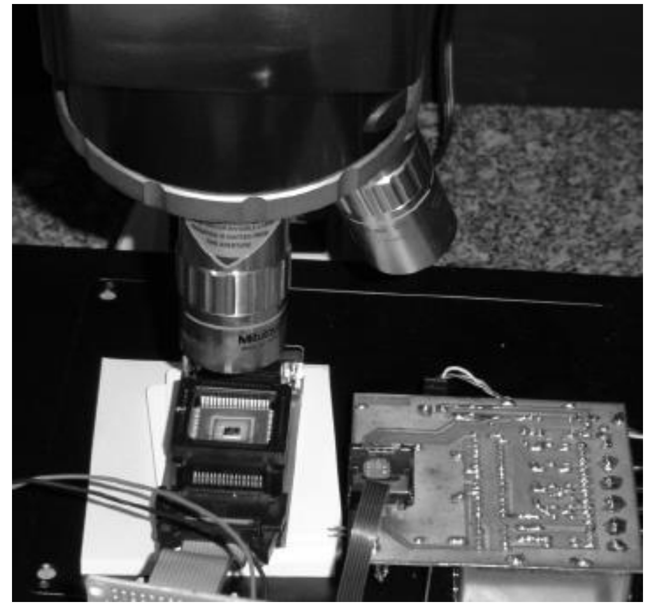
Implementation Attacks

Implementation Attacks



- Critical information leaked through side channels
- Adversary can extract critical secrets (keys etc.)
- Usually require physical access (proximity)

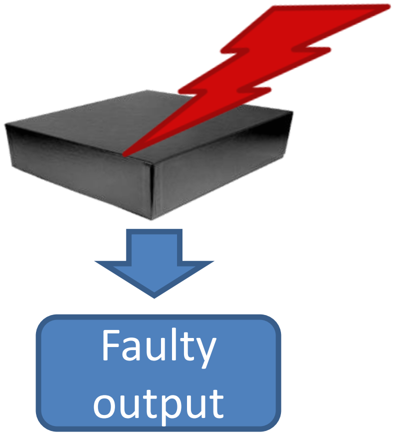
Physical Attacks



- Invasive Attacks
  - Probing Attacks
- Semi-invasive
  - Fault Injection Attacks
- Non-invasive
  - Timing Attacks (cf. Tuesday talk)
  - Physical side channel attacks:
  - Power, EM, Sound, Light

Fault Attacks

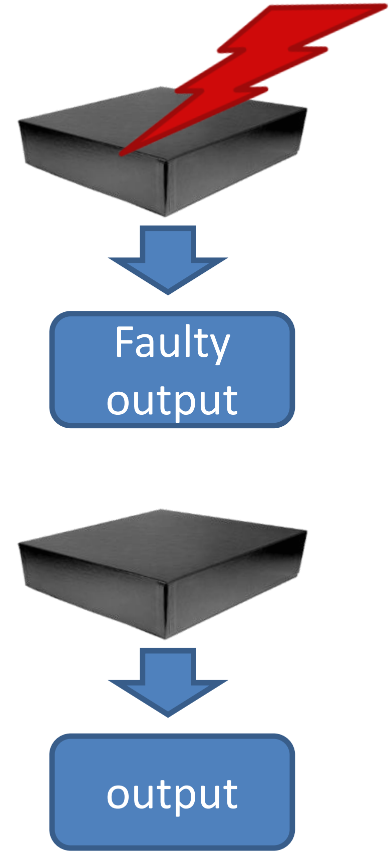
- Very powerful and not that difficult to implement
- **Approach:**
  - Induce faults during crypto computation (e.g. power or clock glitch, shine laser, EM etc.)
  - Use corrupt data output to recover keys
- **Countermeasures:**
  - Strong error detection through coding or repeat computation
  - Tamper resilient hardware
- **Example:** single faulty output of RSA-CRT can reveal entire RSA key [BDL97,Len96]



[BDL97] Boneh, DeMillo, Lipton. "*On the importance of checking cryptographic protocols for faults*. CRYPTO 97
[Len96] Lenstra AK. *Memo on RSA signature generation in the presence of faults*. 1996.

Types of fault attacks

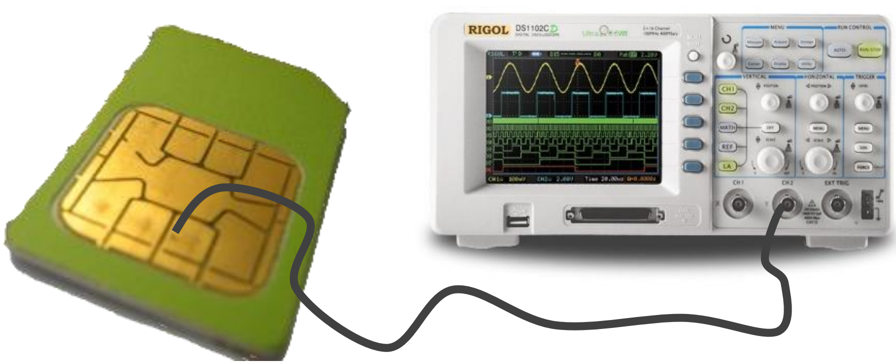
- Differential Fault Analysis [BS96]:
  - Analyze difference between correct and faulty output: knowledge about fault position and/or value reveals (partial) key
- Simple fault analysis:
  - only faulty output given; additional statistical knowledge about fault behavior needed.
  - Fault sensitivity analysis [LSG10]: only certain internal states can be faulted: faulty behavior→that state occured

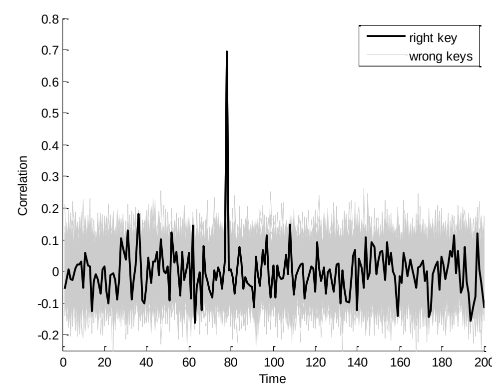


[BS96] Biham, Shamir. *Differential fault analysis of secret key cryptosystems*, CRYPTO 96
[LSG+10] Li, Sakiyama, Gomisawa, Fukunaga, Takahashi, and Ohta, *Fault sensitivity analysis*, CHES 2010

Information Leakage through Power

- **Key Observation:** Power Consumption of ICs depends on processed data
- First exploited to recover **cryptographic keys** from smart cards in 1999





Power Consumption of CMOS

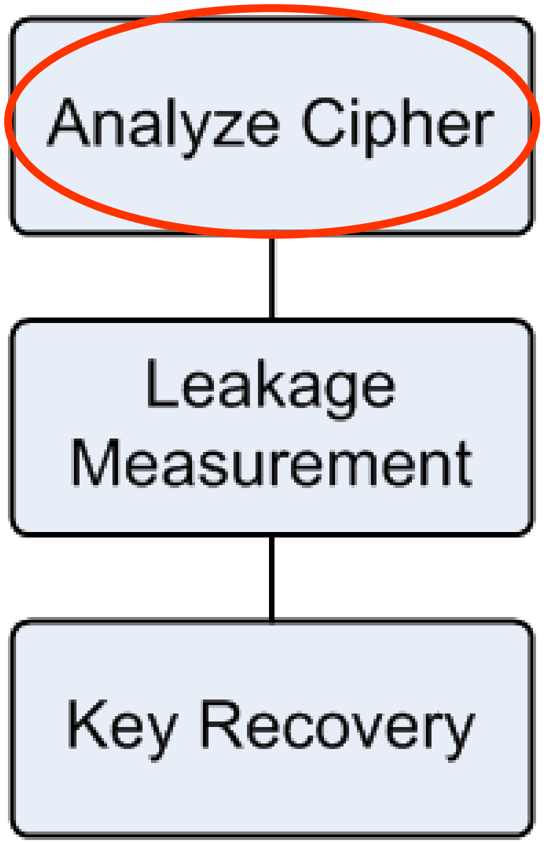
- Information stored as voltage levels –Hi =1/Lo=0
- Signal transitions dissipate power:

$$P = \underbrace{\alpha \cdot C \cdot V^2 \cdot f}_{dynamic} + \underbrace{V \cdot I_{leak}}_{static}$$

Activity factor α is determined by data

→ **Power Consumption / EM emanation depends on processed data!**

A Simple Power Analysis Attack



1. Find a suited predictable intermediate value in the cipher

2. Perform power measurements and post processing

3. Recover Secret Key

Modular Exponentiation for RSA

Basic principle: Scan exponent bits from left to right and square/multiply operand accordingly → **Exponent is secret key**

Algorithm: Square-and-Multiply

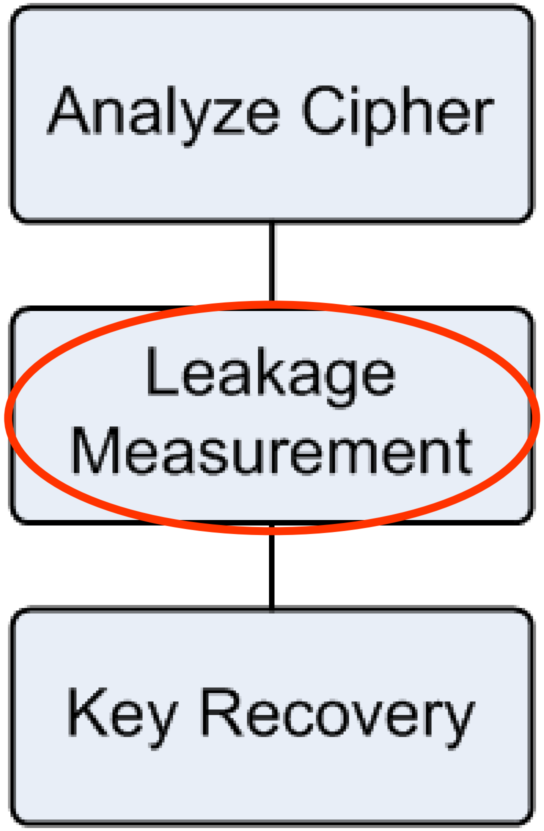
Input: Exponent H, base element x, Modulus N

Output: $y = x^H \bmod N$

1. Determine binary representation $H = (h_t, h_{t-1}, \ldots, h_0)_2$
2. **FOR** $i = t\text{-}1$ **TO** 0
3. $y = y^2 \bmod N$
4. **IF** $h_i = 1$ **THEN**
5. $y = y * x \bmod N$
6. **RETURN** y

→ **Execution of multiply depends on secret**

A Simple Power Analysis Attack



1. Find a suited predictable intermediate value in the cipher

2. Perform power measurements and post processing

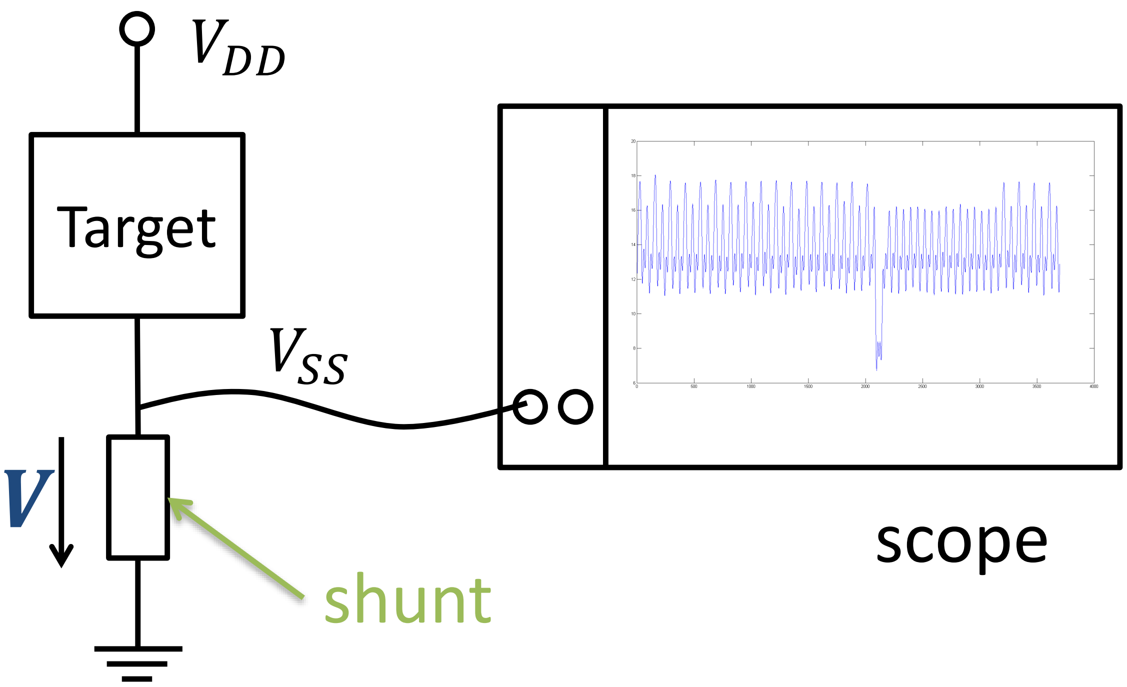
3. Recover Secret Key

Measurement setup

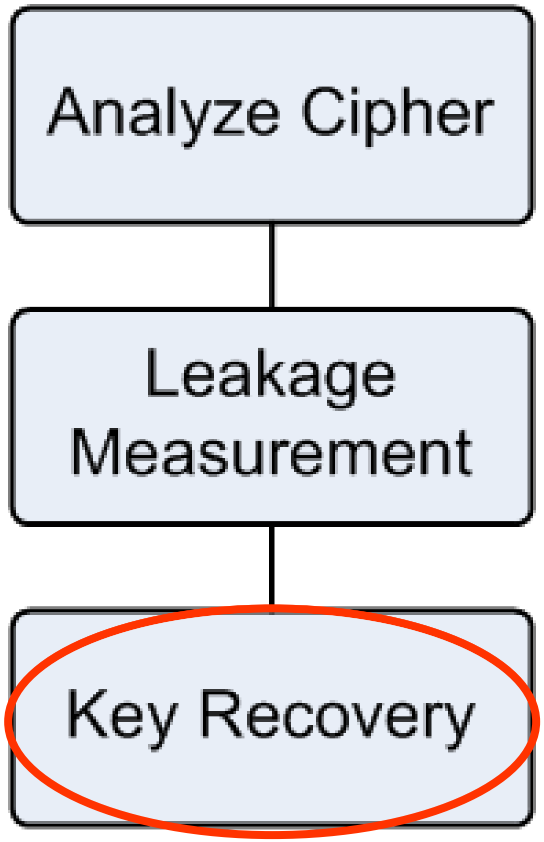
- Oscilloscope measures power or EM from target crypto device
- Usually PC to control setup

SPA Measurement Setup

- Voltage drop over shunt resistor ~ power

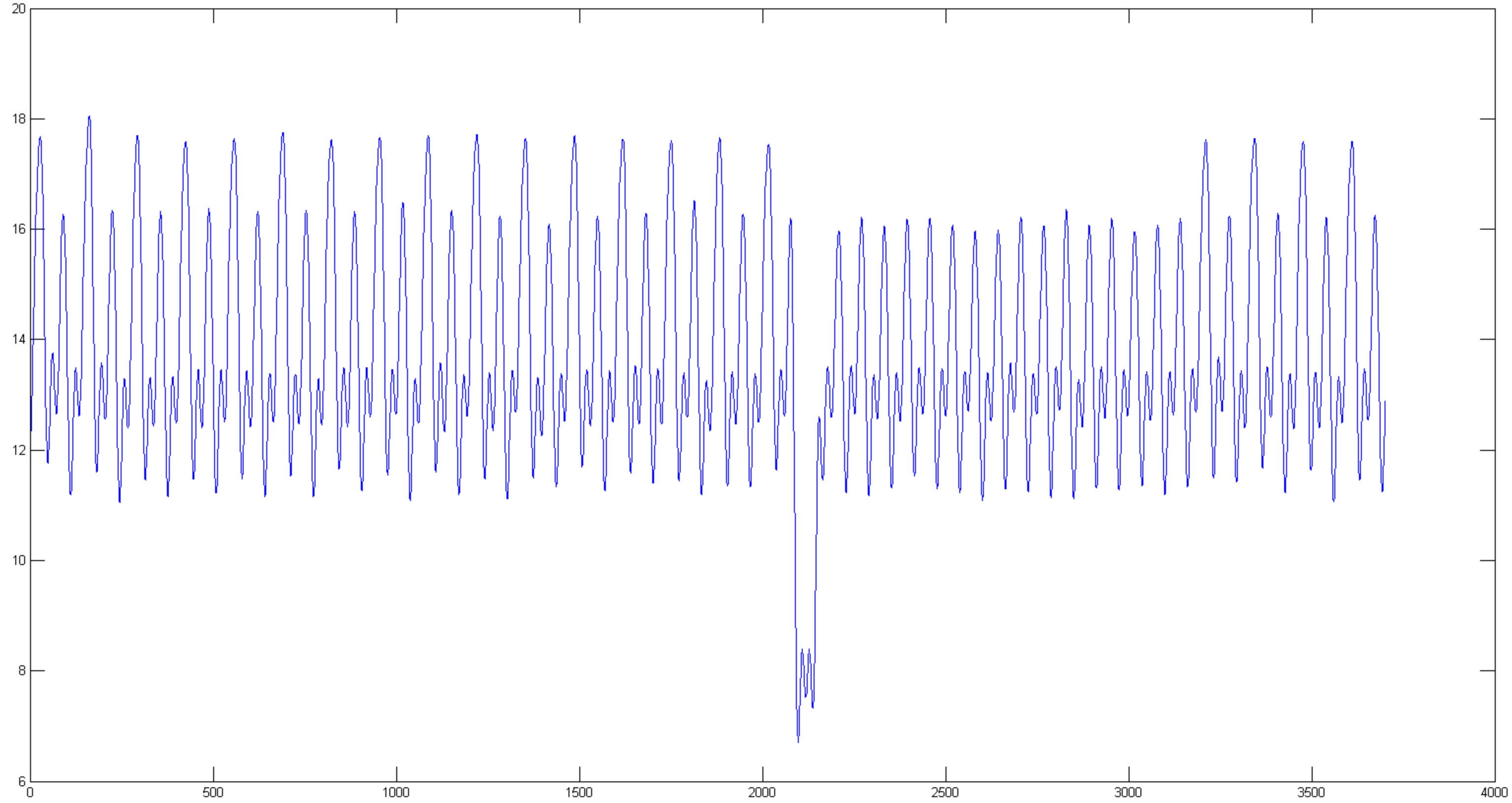


A Simple Power Analysis Attack



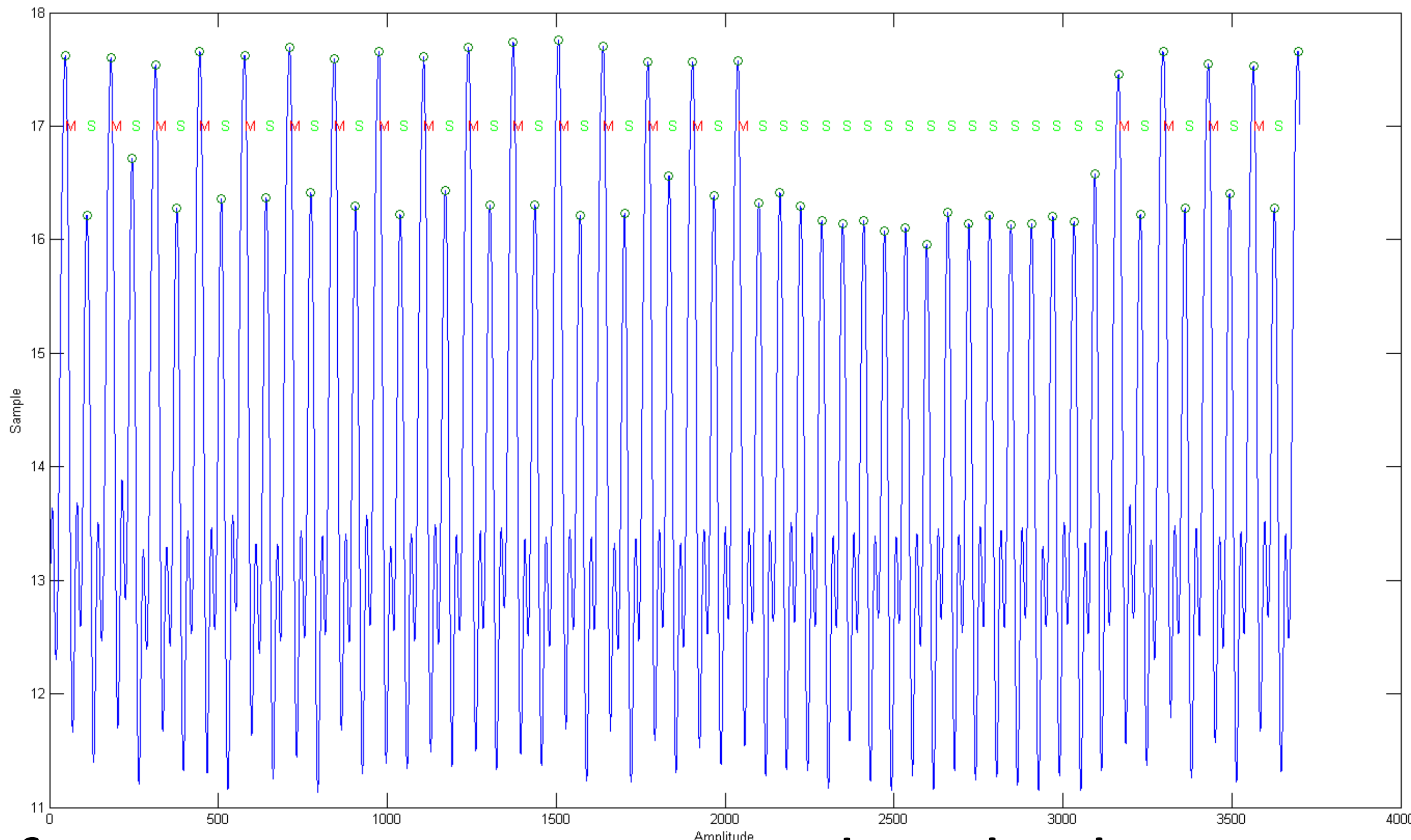
1. Find a suited predictable intermediate value in the cipher
2. Perform power measurements and post processing
3. Recover Secret Key

RSA Power trace



Where are the squares, where are the multiplies?

Detecting key bits



- After zoom-in, squares and multiplies are easily distinguishable

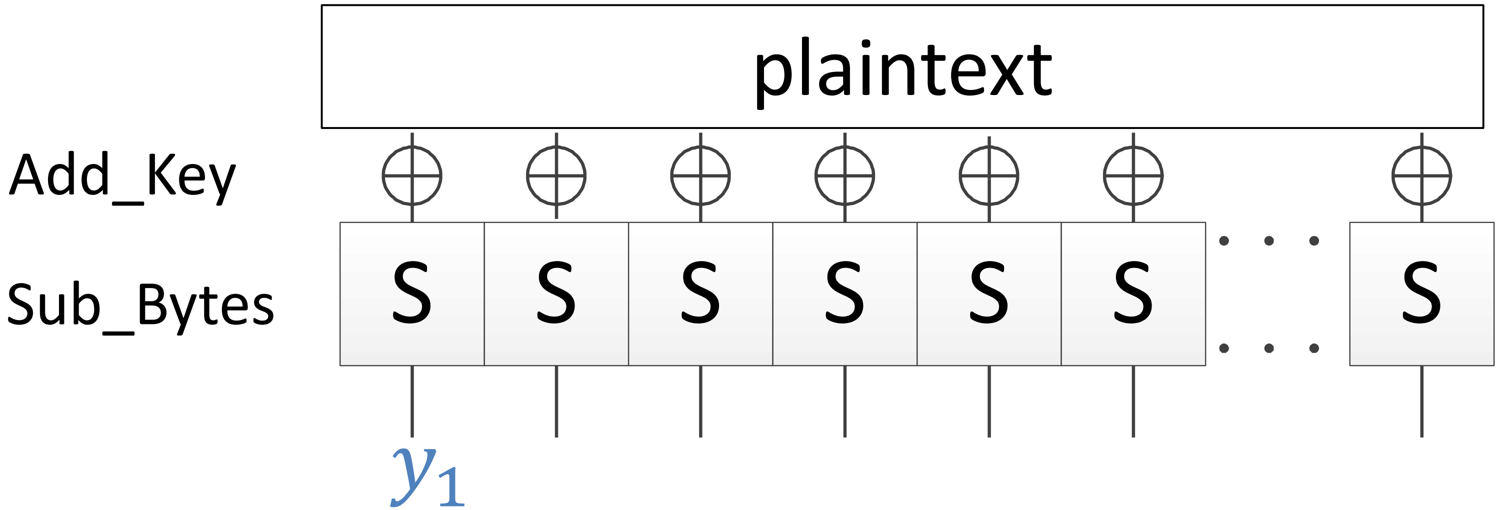
Differential Power Analysis

- **Key idea:** use statistical information from many observations
- Recall Password Timing Example:

$$time = f(input, secret)$$

- Leakage exists, how to exploit it?
  - some variations may be predicted
  - variations may be tiny,
  - only small parts of implementation need be predicted

AES: predicted value



Predicted state: $y_1 = S(x_1 \oplus key_1)$

Single-bit DPA: Predict only one bit of state:

$$h = \text{LSB}(y_1)$$

DPA on AES on Controller

Assumption: Controller leaks **HW(y_1)** during S-box lookup

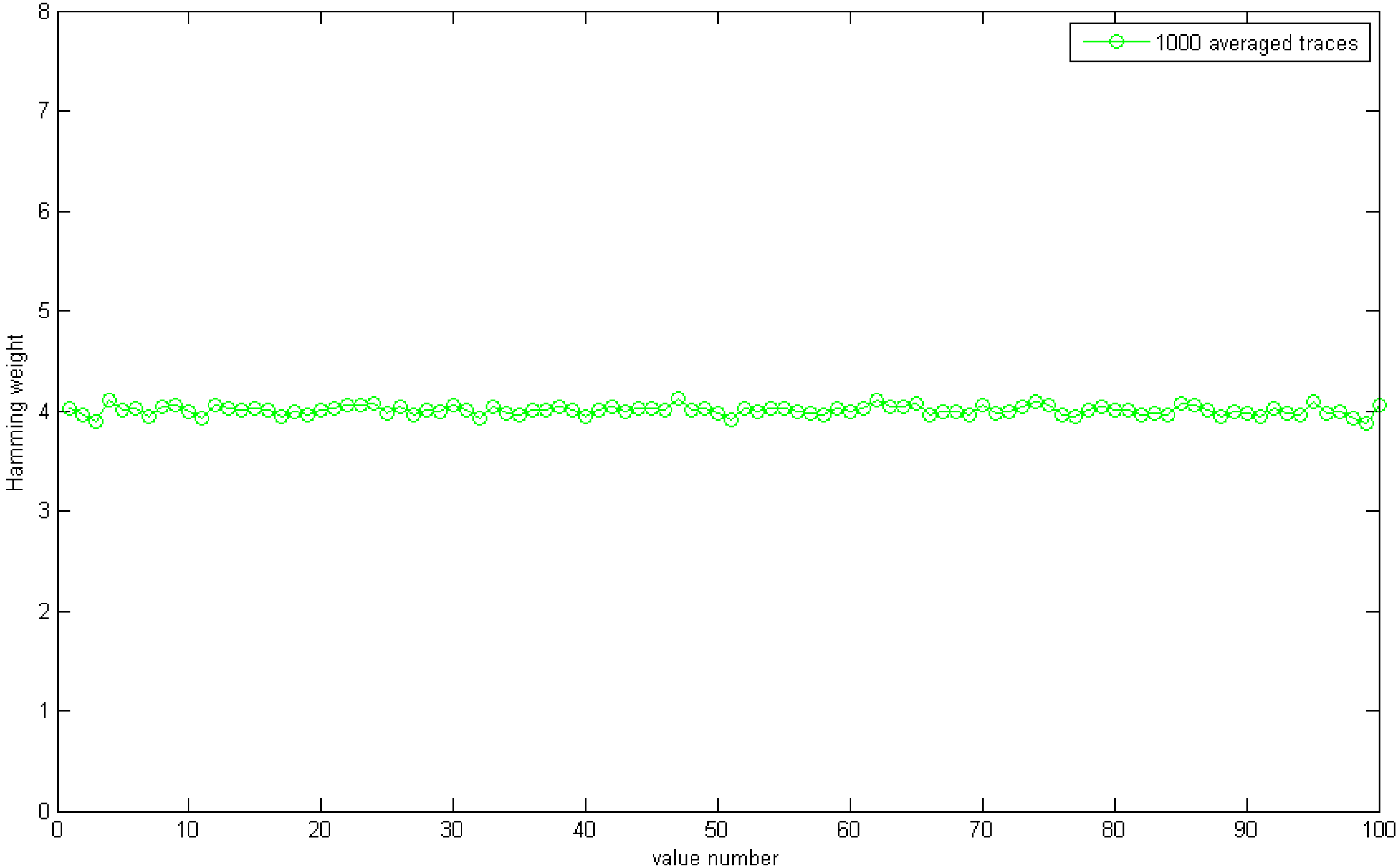
1. Measure $P_i(t)$ and store all $\langle P_i(t), in_i \rangle$
2. **Sort** traces based on $h = \text{LSB}(y_1)$ and average

$$\mu_0 = \overline{P_i(t)}|(h = 0) \qquad \mu_1 = \overline{P_i(t)}|\,(h = 1)$$

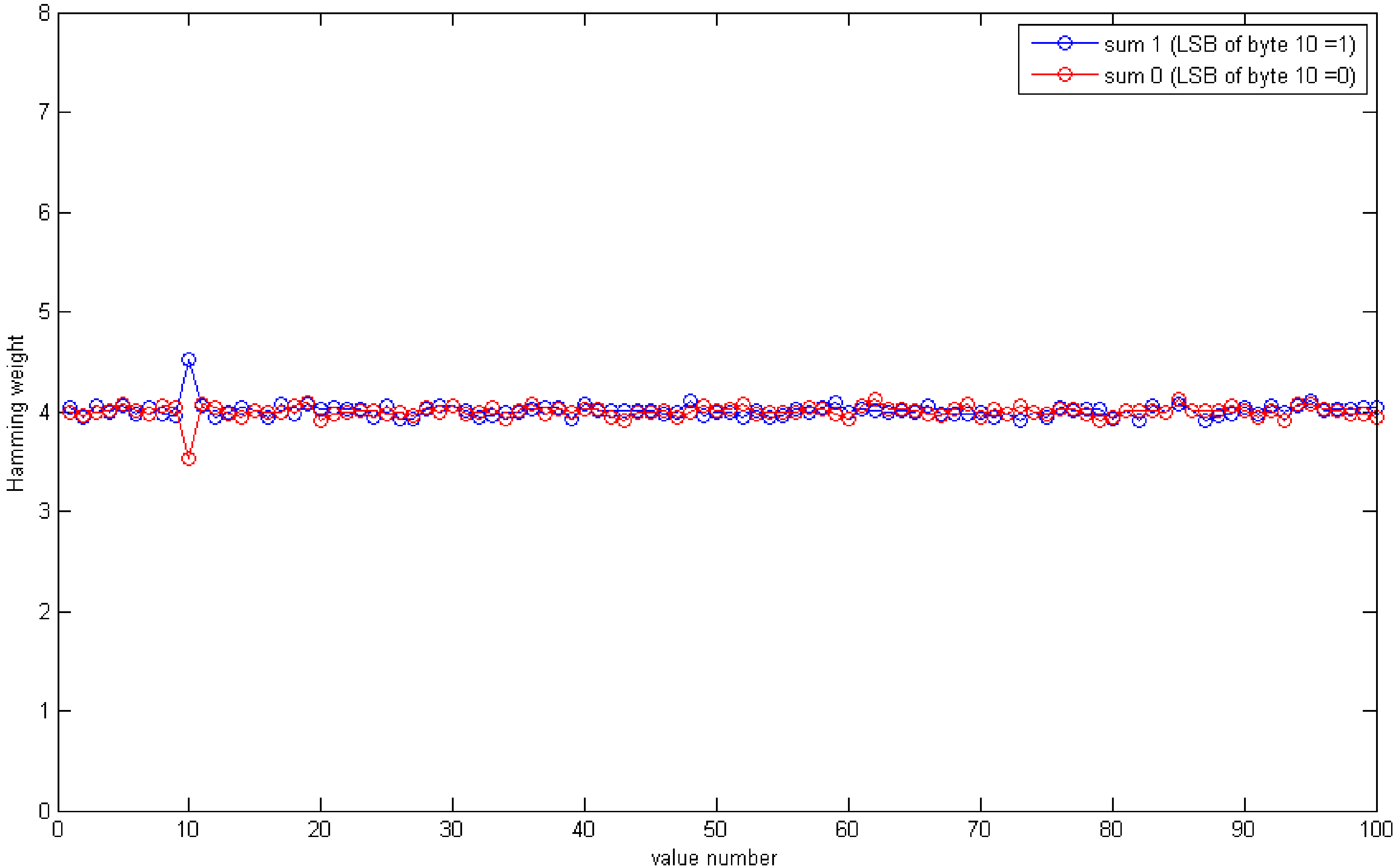
3. Compute difference of means:

$$\Delta = \boldsymbol{\mu_1} - \boldsymbol{\mu_0}$$

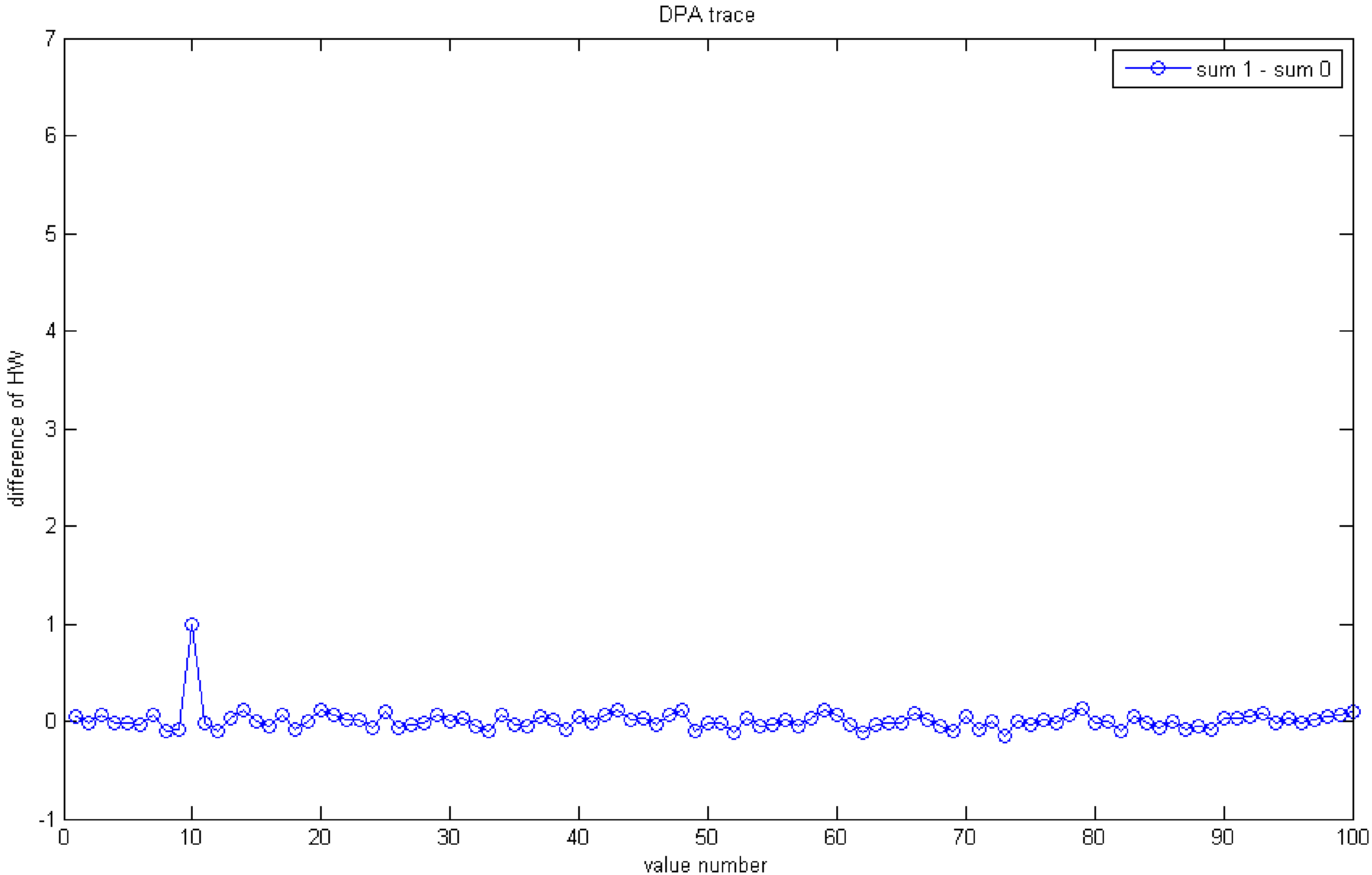
Average of 1000 HWs



Sorted Traces (based on h)



Result of the Distance of Means Attack

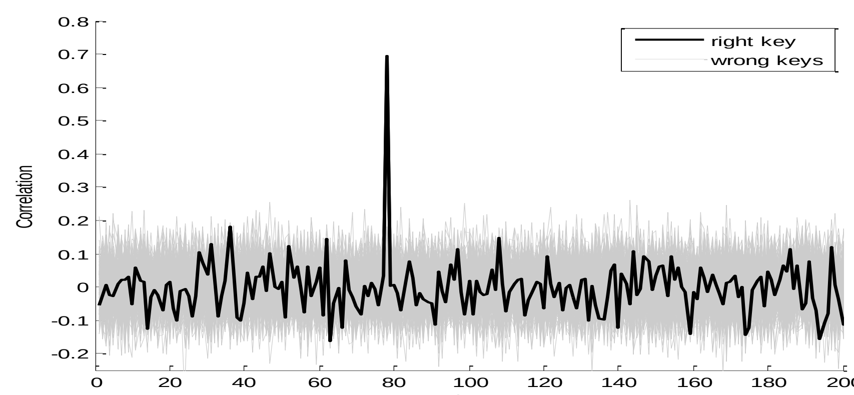


Side Channel Attacks Classification

- **Non-Profiled Attacks**
  - Need some knowledge of implementation and (approximate) leakage model (or build it on the fly)
    - Difference of Means (Classic DPA)
    - Correlation Power Analysis (CPA)
    - Mutual Information Attack (MIA)
    - Collision Based Attacks
- **Profiled Attacks:**
  - Two-step process: 1) profile leakage, 2) use learned leakage model to extract information
  - Usually more effective in exploitation due to better modeling
    - Template Attack
    - Linear Regression

Single-bit DPA

- Simple yet effective attack:
  - Very generic leakage model: only needs slight difference for one bit
  - Many more powerful, but less generic attacks exist
- $\Delta \approx 0$ for wrong key and wrong time points
- Reveals both **correct key** AND time point of leakage



Leakage Detection

Methods for Leakage Detection?

Goal: Simple test to detect any leakage in implementation

- Profiled vs. Non-profiled?
  - **MIA:** strong but slow convergence; Depends strongly on parameter choices: how to describe and sample pdfs?
  - **Templates:** very powerful, but costly to build and also model-dependent: Which variable to template?
  - Good choices for **leakage quantification**
- **CCA (Correlation Collision Attack)[MME10]:**
  - Basically univariate self-profiling attack
  - Already widely used as leakage detection tool
  - Disadvantage: does not work for single-bit leakages
- Above proposed as attacks. More generic solution?

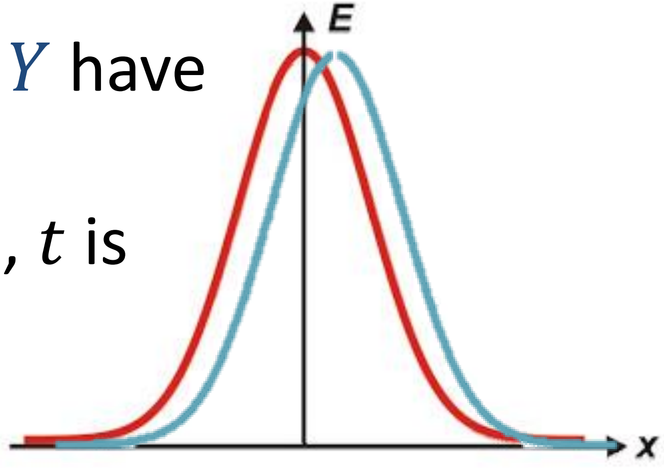
[MME10] Moradi, Mischke, Eisenbarth *Correlation-enhanced power analysis collision attack*—CHES 2010

Leakage Detection: TVLA Test [GJJR11]

- Builds on **T-Test**: test to check matching means for two distributions
- T-Test returns confidence for non-leakage hypothesis
- Non-profiled, DPA derived
- Originally proposed for automated test suite
  - Given cipher-specific test vectors, check implementation correctness and ensure observed leakage traces do not break test
- Comes in two (three) flavors

[GJJR11] Goodwill, Jun, Jaffe, Rohatgi: "A testing methodology for side-channel resistance validation", NIST Workshop, 2011.

Welch's T-Test



- Checks if two normal distributions X, Y have the same mean
- With sample mean $\bar{x}$ and variance s_x^2, t is given as: $t = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{s_x^2}{n_x} + \frac{s_y^2}{n_y}}}$,
- If X, Y have the same mean, then t follows a student distribution and thus $|t|$is small:
  $\Pr(|t_{df=v>1000}| > 4.5) < 0.00001$
- Hence, if no leakage exists, the probability of $|t| > 4.5$ is sufficiently small

Fixed vs Random Test
Non-Specific T-Test

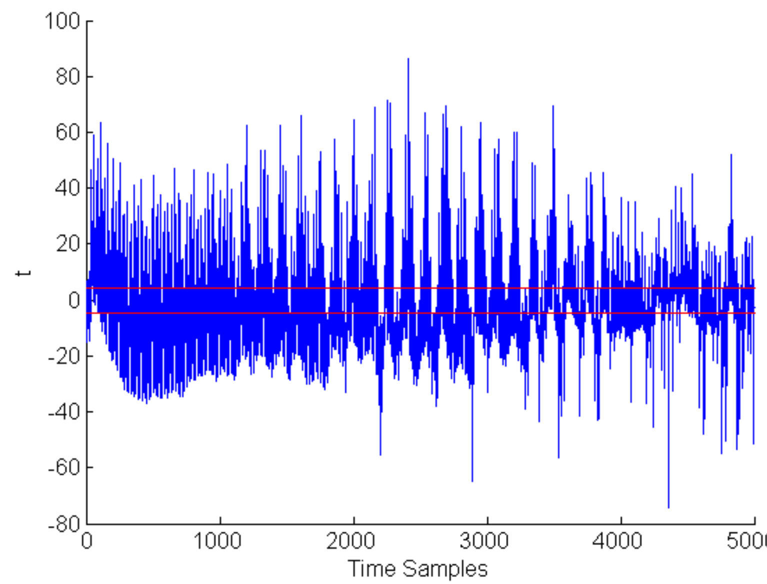
Two sets of measurements:

- **Fixed:** external variables (plaintext, key) are fixed
- **Random:** external variable (e.g. plaintext) is random (others, e.g. key, as before)
- Both sets compared with T-test

→If (mean of) leakage distributions differ, device leaks

Properties:

- Non-specific: Works on all intermediate states (that differ from mean)
- Not every found leakage might be exploitable



Random vs. Random
Specific T-Test

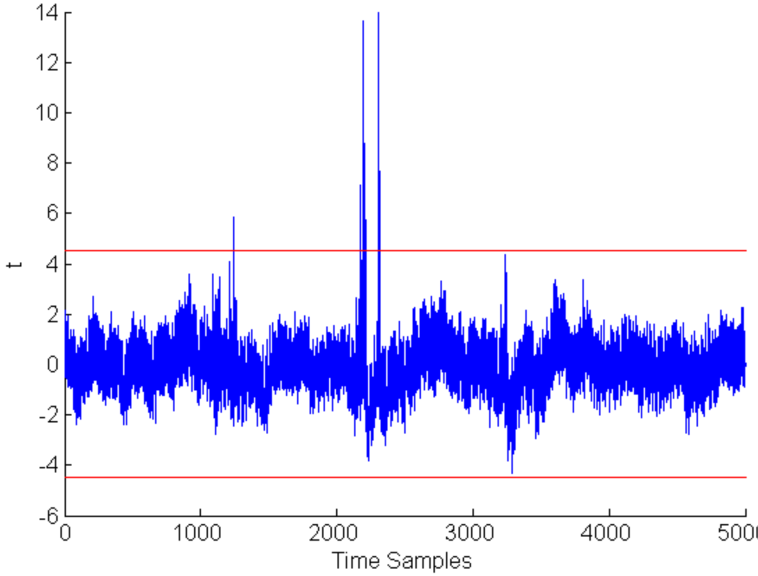
Kocher's DPA as a Test:

- Key is known and fixed, input is random
- Measurements split in two sets according to known intermediate state
- Both sets compared with T-test

→If (mean of) leakage distributions differ, **specific intermediate state** leaks

Properties:

- Specific: Works on predicted intermediate state
- Only finds expected leakages
- Shows an attack



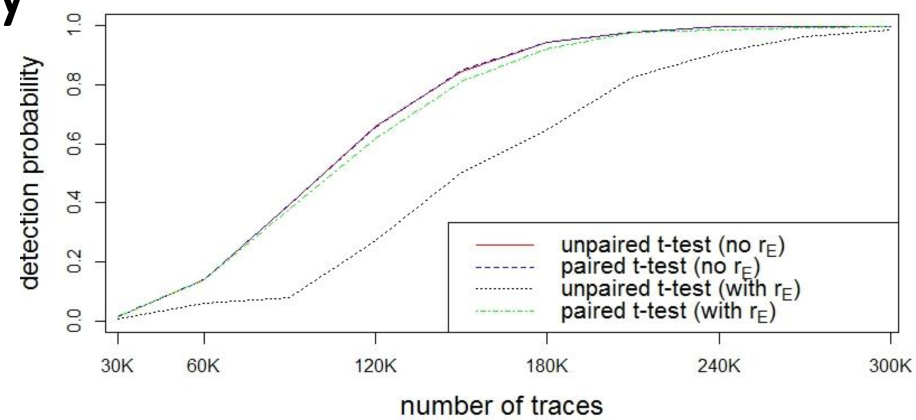
Practical Considerations

- Test is influenced by measurement setup:
  - Both sets should be **randomly** interleaved, to ensure initial state is not biased
  - FvR: plaintext is fixed in one set, but not other: marks hiding countermeasures as insecure
- Semi-Fixed vs Random Test:
  - Fixes partial intermediate state for semi-fixed case
  - Inputs and outputs still seem random
  - Avoids FvR problem above

Susceptibility to Common Noise

- Drifts decrease sensitivity
- Remedy: **Paired T-test**

$$t_p = \frac{D}{\sqrt{\frac{s_D^2}{n}}}, \text{ with } D = x_i - y_i$$



- Common noise of paired observations vanishes
- Also works for higher order analysis with *moving averages*

$$x' = (x - \mu_x)^d \rightarrow x' = \left(x - \mu_{x,local}\right)^d$$

- Less susceptible to noise and easier to compute

[DCE16] Ding, Chen, Eisenbarth *Simpler, Faster, and More Robust T-test Based Leakage Detection* –COSADE 2016

Side Channel Countermeasures

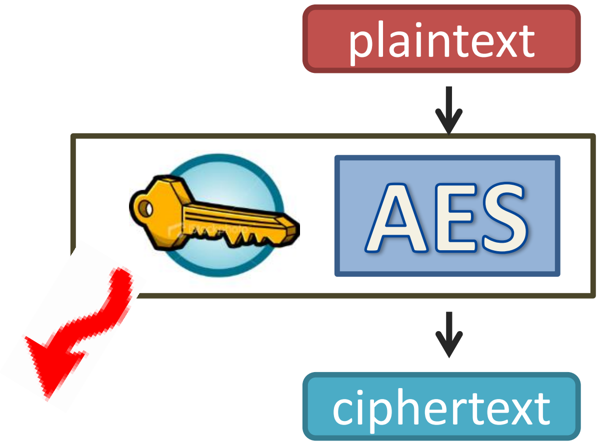
Preventing Side Channel Attacks

Goal: Prevent inference from observable state

- **Hiding:** lowers signal to noise ratio
  - Noise generator, randomized execution order, dual-rail/asynchronous logic styles…
- **Masking:** (secret sharing) splits state into shares; forces adversary to recombine leakage
  - Boolean or arithmetic masking, Higher-order masking
- **Leakage Resilience:** prevents leakage aggregation by updating secret

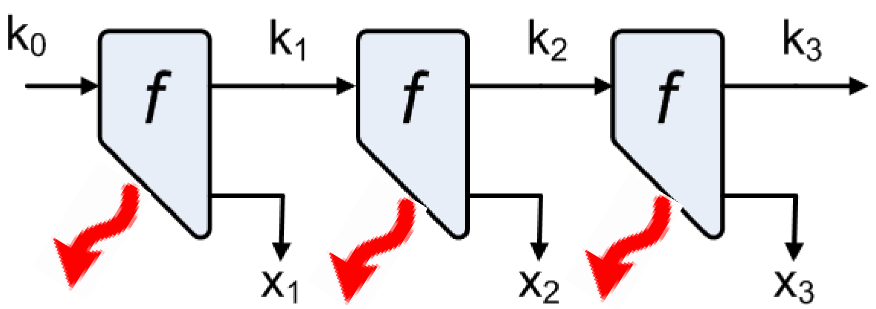
Key usage in Cryptography

Classic Method:



- Same key leaks for every execution of crypto
- Unlimited observations per key

Leakage Resilience (Concept):



- Key changes at each iteration
- Only one (few) observation per key

Leakage Resilience: Key Update

Key needs *update* with every usage:

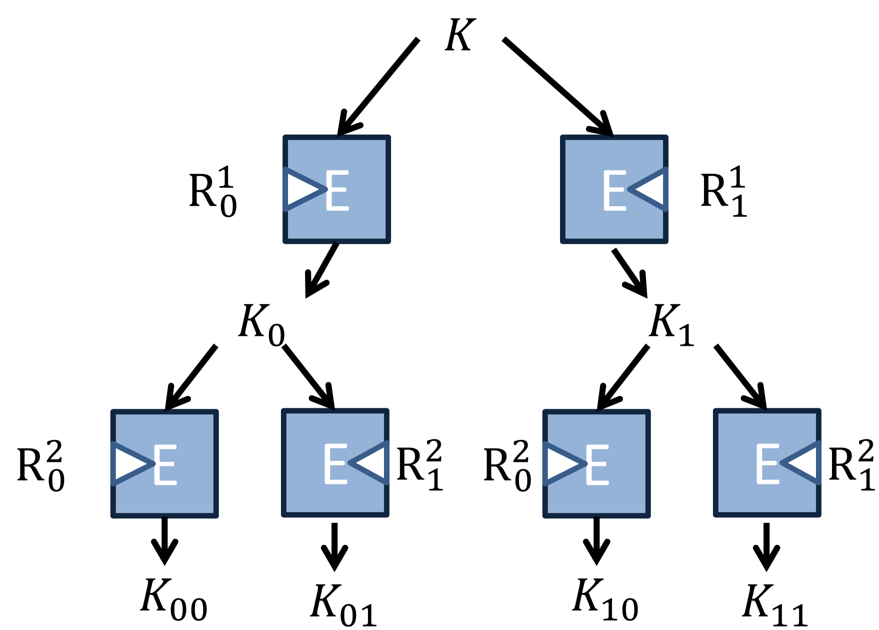
- Stateful design
  - Key owner updates key before each usage
  - **Problem:** Multiple key owners (symmetric crypto) need to stay synchronized
- Stateless design
  - Highly desirable for many symmetric applications
  - First practical proposals exist, e.g. [MSJ12] and [TS13]

[**MSJ12]** M. Medwed, F.-X. Standaert, A. Joux. *Towards Super-Exponential Side-Channel Security with Efficient Leakage-Resilient PRFs*. CHES 2012
[TS13] M. Taha and P.Schaumont. *Side-channel countermeasure for SHA-3 at almost-zero area overhead.* HOST 2013

Stateless Key Updates

GGM Construction:

- Nonce bits decide path
- R_i: public randomness
- One encryption per nonce bit (128 Enc)
- Final key K_{nonce} used!



- Great leakage properties:
  At most two observations per key!
- Big performance overhead: 128 Encryptions to derive key

Masking:
Threshold Implementation

Threshold Implementation [NRR06]

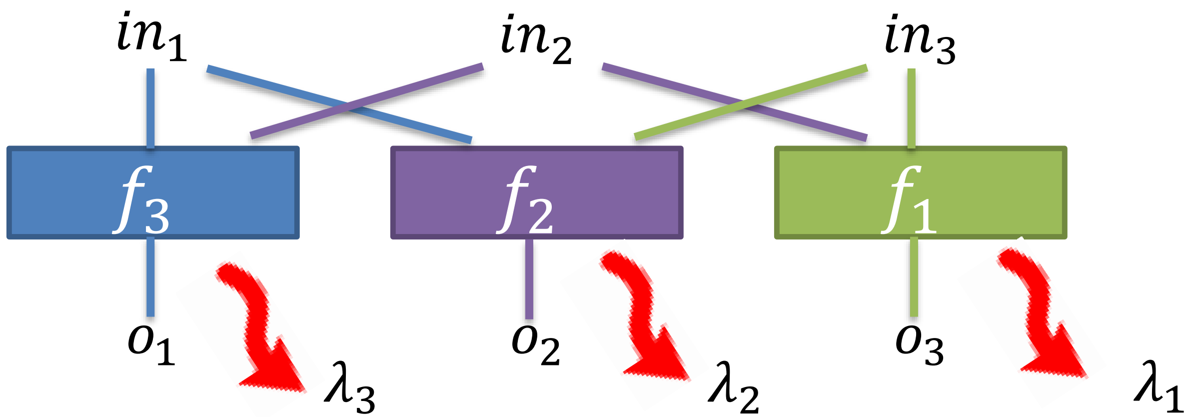
Applies xor-secret sharing (Boolean masking) to thwart SCA:

1. Share inputs, states, outputs as $x = x_1 + x_2 + \cdots$, where $x_i \in \{0,1\}$ and x_i must be **uniformly distributed** →**uniformity property**
2. Perform arithmetic on shares without leaking secret; Output shares must be independent of at least one input share → **non-completeness property**
3. The correct output is the xor-sum of the shares → **correctness property**

- Solves the **glitches** issue: any RTL block is independent of at least one share
- Ensures constant means→ prevents 1^{st} order DPA leakage

[NRR06] Nikova, Rechberger, and Rijmen: *Threshold Implementations Against Side-Channel Attacks and Glitches,* ICICS 2006

TI: Parallel vs. Sequential



- Each f_i lacks one share i → cannot leak about input

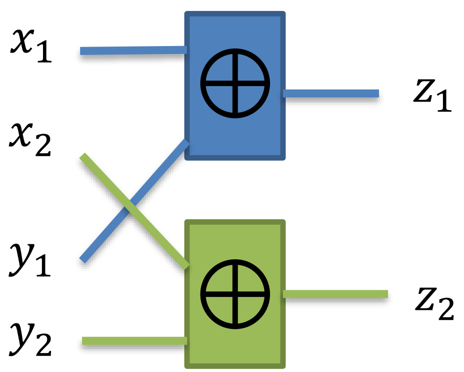
How about parallel leakage? $\lambda = \sum_i \lambda_i$

- **Uniformity** ensures input-independent mean:
  - First order DPA prevented
  - Aggregate leakage also input-independent mean (as long as λ_i are linearly combined (summed))

TI: Secure XOR

Exercise:

- Given $x = x_1 + x_2$ and y $= y_1 + y_2$, compute $z = z_1 + z_2 = x + y$ without breaking uniformity, non-completeness or correctness?



Solution: $z_1 = x_1 + y_1$

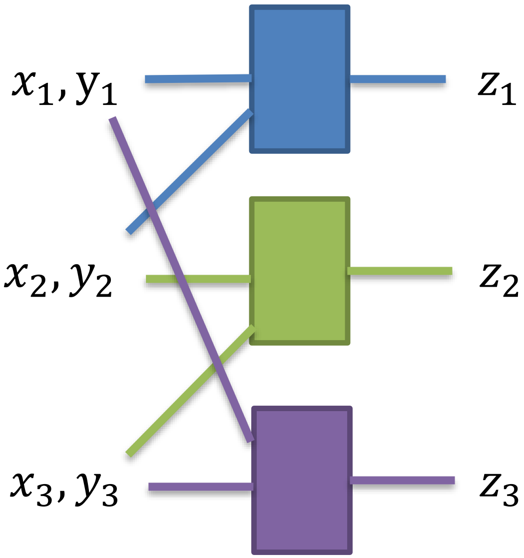
$z_2 = x_2 + y_2$

- **Correctness:** $z = z_1 + z_2 = x + y$
- **Non-Completeness**: i share does not depend on non-i shares
- **Uniformity:** z_i is uniform if either y_i or x_i is uniform

TI: Secure AND

Exercise:

- Given sharing of x and y, find minimum number of shares and method to compute $z = xy$ without breaking uniformity, non-completeness or correctness?



Solution:

$$z_1 = x_1y_1 + x_1y_2 + x_2y_1$$
$$z_2 = x_2y_2 + x_3y_2 + x_2y_3$$
$$z_3 = x_3y_3 + x_3y_1 + x_1y_3$$

- **Correctness:**
  $z = z_1 + z_2 + z_3 = xy$
- **Completeness**:
  i share independent of share j $\neq i$
- **Uniformity: not fulfilled!!!**

Uniformity needs more shares or masking variable

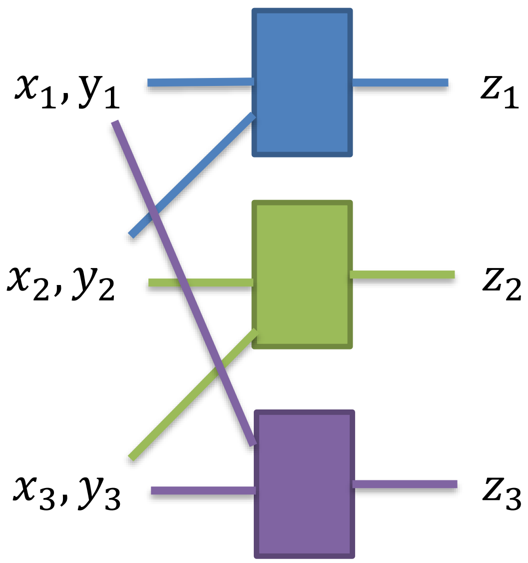
Secure AND: Re-masking

Restoring uniformity:

- **Add randomness**:
  e.g. $r_1, r_2 \leftarrow \{0,1\};\ r_3 = r_1 + r_2$

  Then:
  $$z_1 = x_1y_1 + x_1y_2 + x_2y_1 + \boldsymbol{r_1}$$
  $$z_2 = x_2y_2 + x_3y_2 + x_2y_3 + \boldsymbol{r_2}$$
  $$z_3 = x_3y_3 + x_3y_1 + x_1y_3 + \boldsymbol{r_3}$$
  →Each z_i is uniformly distributed, non-complete and correct, but additional randomness needed
- **Adapt function:**
  $z = xy + w$, (w is properly shared, i.e. uniform):

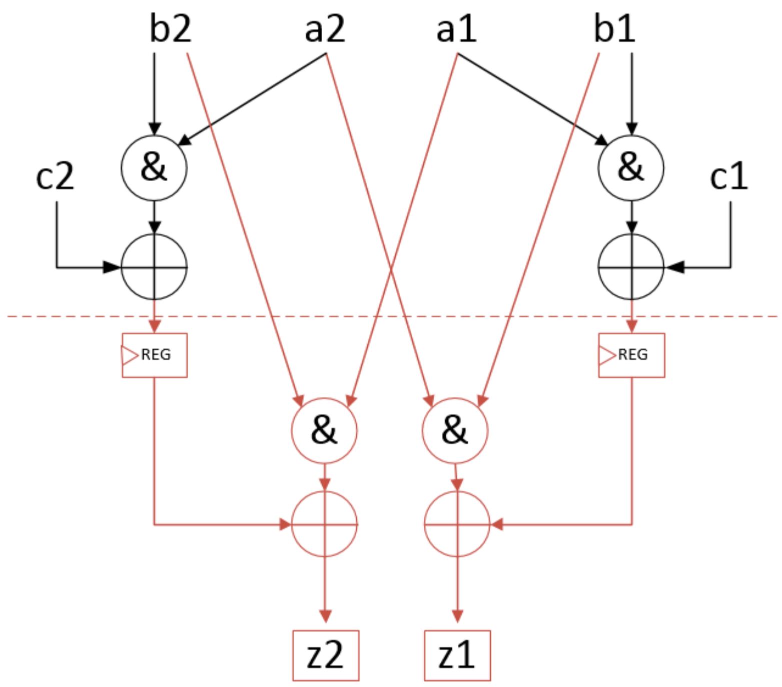
  Then:
  $$z_1 = x_1y_1 + x_1y_2 + x_2y_1 + \boldsymbol{w_1}$$
  $$z_2 = x_2y_2 + x_3y_2 + x_2y_3 + \boldsymbol{w_2}$$
  $$z_3 = x_3y_3 + x_3y_1 + x_1y_3 + \boldsymbol{w_3}$$
  →Each z_i is uniformly distributed, non-complete and correct; randomness of w re-used



From 3-share to 2-share

Non-linear function: $z = a \cdot b + c$

$z_2 = \boxed{(a_2 \cdot b_2 + c_2)} + a_1 \cdot b_2$ $\quad$ $z_1 = \boxed{(a_1 \cdot b_1 + c_1)} + a_2 \cdot b_1$

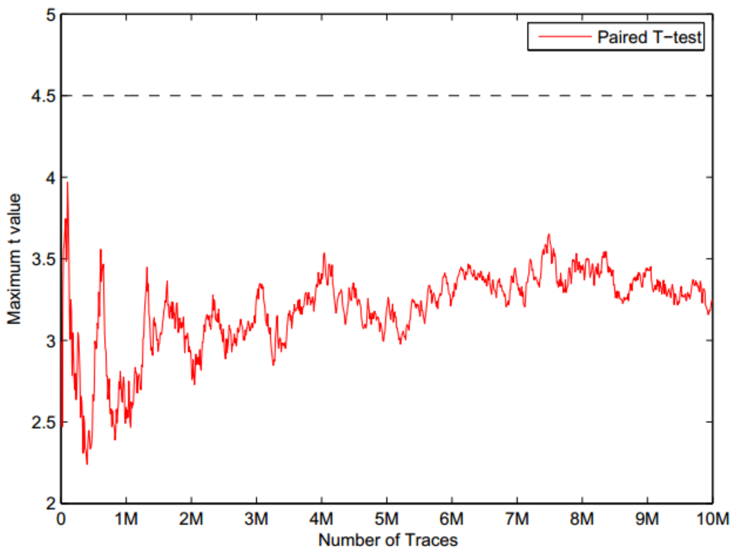


Pipelining!

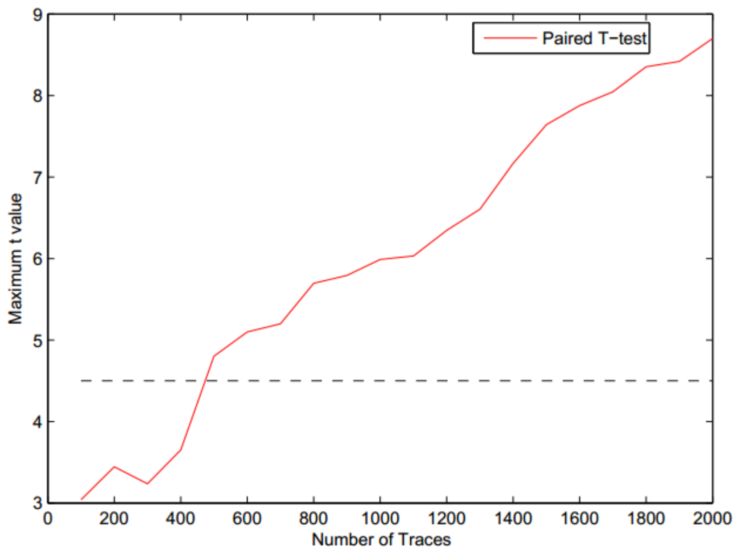
Correct;
Non-Complete;
Uniform;

Compared with 3-share:
- Less randomness
- Fewer logic operations
- Two extra flip-flops
- Two stages

Leakage Detection on 2-TI Simon Implementation



(a) 1st order t-test



(b) 2nd order t-test

Conclusions

- Physical access gives rise to many possible attacks
- Protection against physical attacks is possible, but neither easy nor cheap
  - Perfect protection is not possible
  - device compromise may not result in system compromise
- IoT will ensure interest for years to come


Thank You!
vernam.wpi.edu
its.uni-luebeck.de
CONCORDIA DOMI FORIS PAX
thomas.eisenbarth@uni-luebeck.de