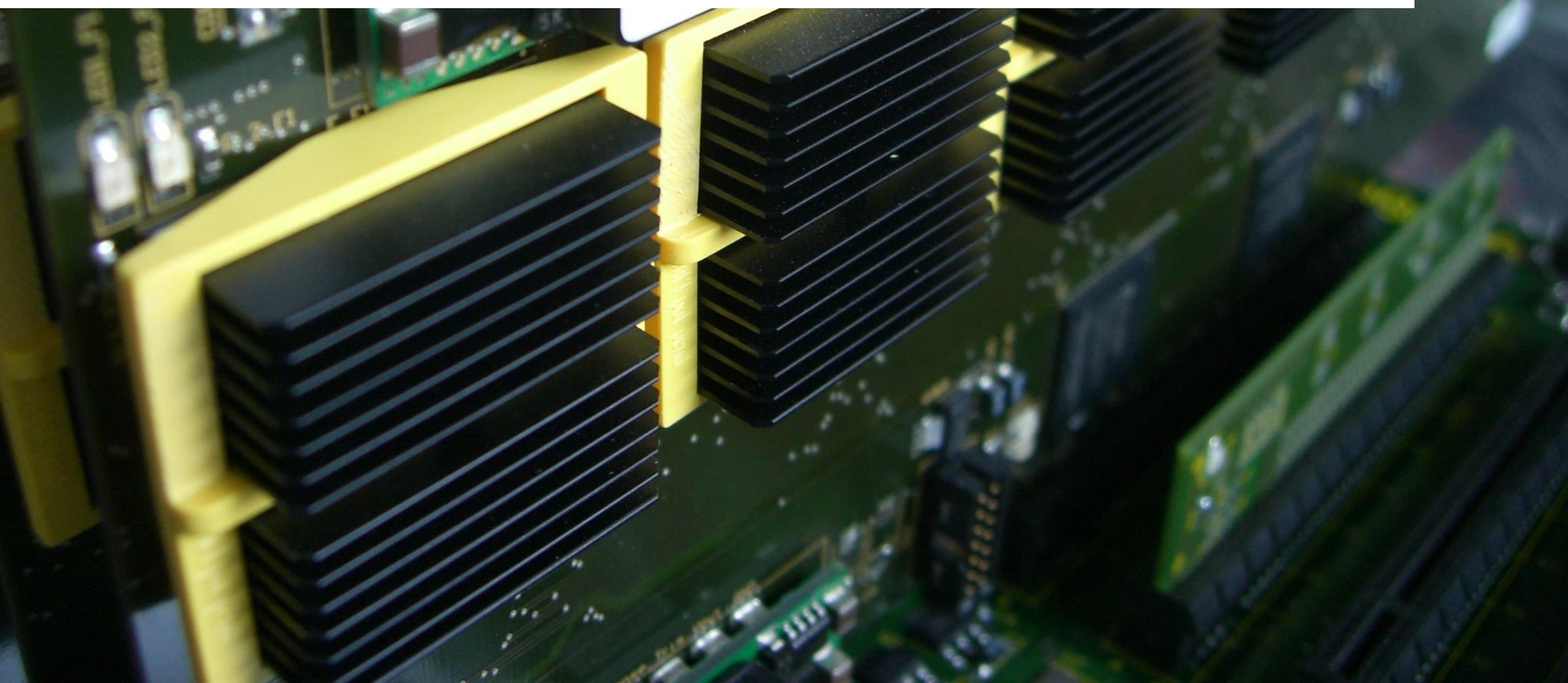
RUHR-UNIVERSITÄT BOCHUM

AFFINE EQUIVALENCE AND ITS APPLICATION TO TIGHTENING THRESHOLD IMPLEMENTATIONS

PASCAL SASDRICH, AMIR MORADI, TIM GÜNEYSU

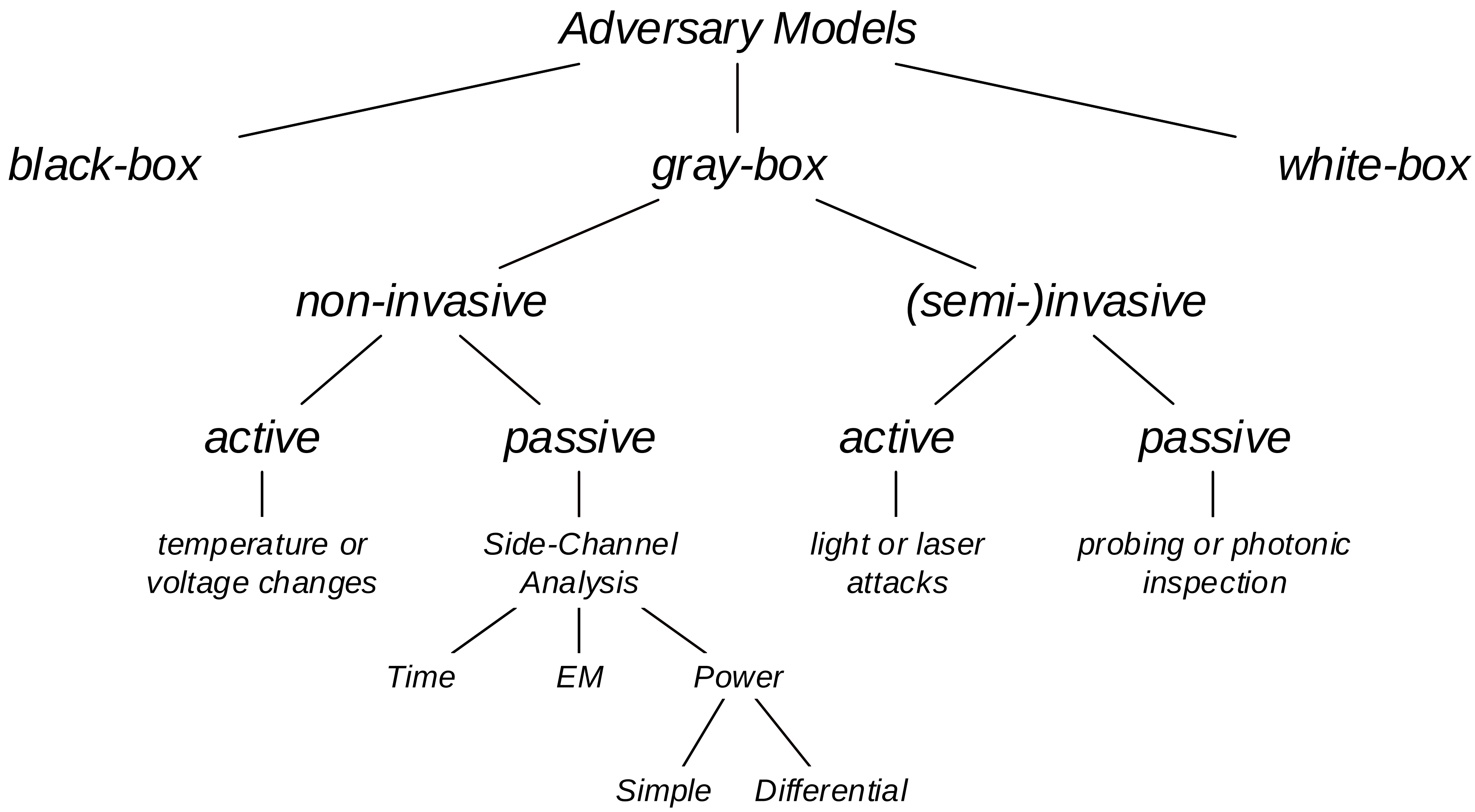
22ND INT. CONFERENCE ON SELECTED AREAS IN CRYPTOGRAPHY, SACKVILLE, NB, CANADA AUGUST 13, 2015



WHAT ARE THRESHOLD IMPLEMENTATIONS?

***Threshold Implementations* are a countermeasure against Side-Channel Analysis such as Differential Power Attack.**

SIDE-CHANNEL ANALYSIS (SCA)



DIFFERENTIAL POWER ANALYSIS (DPA)

General: Measure multiple power traces of an encryption with *same key* but *different plaintexts*.

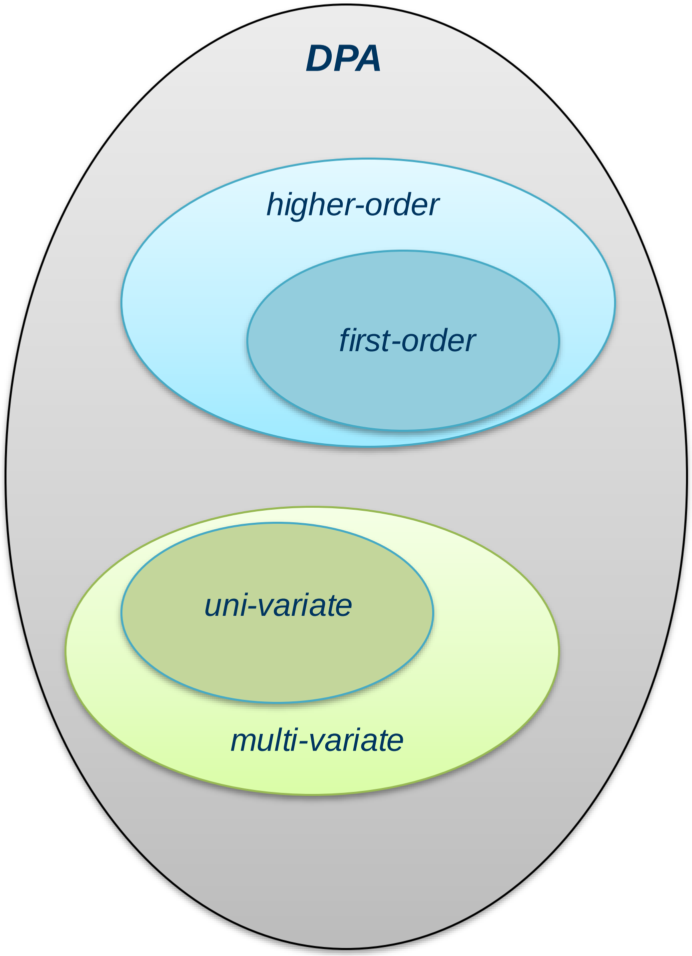
Idea: Each signal transition will consume a different amount of power.

- 0 → 0: *low*
- 0 → 1: *high*
- 1 → 0: *high*
- 1 → 1: *low*

The leakage of an encryption $E_k(m)$ will create a unique fingerprint in the power consumption.

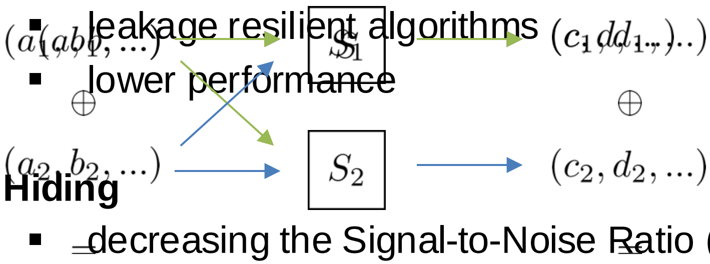
Statistical analysis will help to reveal the secret encryption key k.

Analysis is simplified using divide-and-conquer strategies (e.g. only observing S-box computation)



COUNTERMEASURES AGAINST DIFFERENTIAL POWER ANALYSIS

1. **Limitation of the key invocation**
   - key distribution is a challenge
   - leakage resilient algorithms
   - lower performance



2. **Hiding**
   - decreasing the Signal-to-Noise Ratio (SNR)
   - decrease signal (e.g. power equalization, logic styles)
   - increase noise (e.g. shuffling, dummy executions)

3. **Masking**
   - randomizing the leakage
   - secret sharing approach
   - multi-party computations

unshared	shared	HW	mean	var
0	$(0,0)$ $(1,1)$	0 2	1	2
1	$(0,1)$ $(1,0)$	1 1	1	0

✓ first-order DPA security

X second-order DPA security

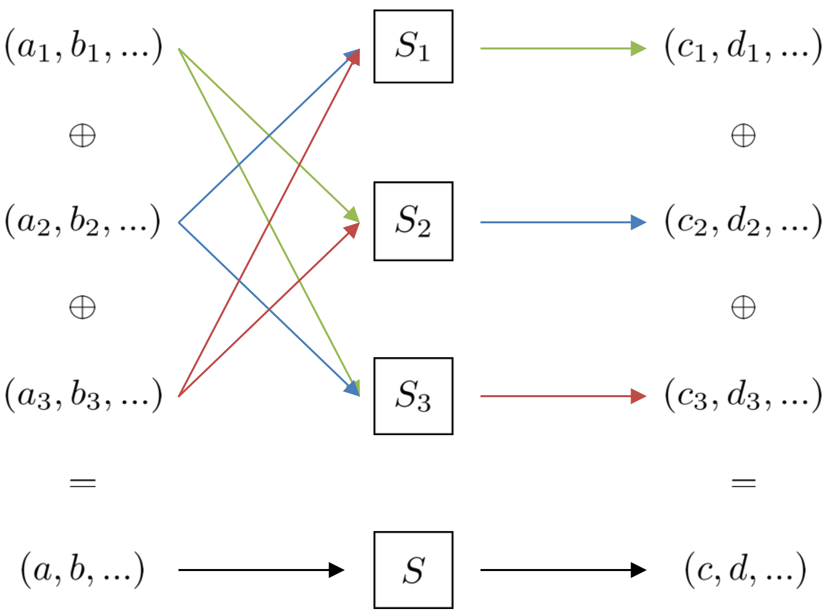
THRESHOLD IMPLEMENTATION

Threshold Implementation:

- efficient *countermeasure* against (*first-order*) Side-Channel Analysis
- introduced in 2006 by Nikova et al.
- provides provable security even in a glitch circuit

Concept and properties:

- uniform masking
- non-completeness
- correctness
- uniform sharing of function outputs (each set of output pairs occurs with same probability)

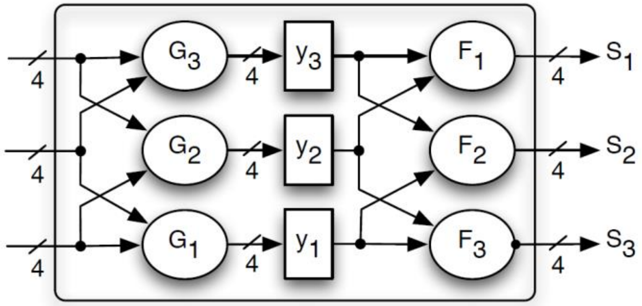


Note: *The number of input and output shares depends on the function S.*

WHAT IS THE STORY OF THIS WORK?

- **Side-Channel Analysis (SCA):** attacks exploit information leakage of cryptographic devices
- **Threshold Implementation (TI):** countermeasure based on Boolean masking and multi-party computation

Problem: TI counteracts *first-order attacks*, but is vulnerable to *higher-order attacks* (using higher-order statistical moments).



Different approaches to encounter this problem:

- **Higher-order Threshold Implementations (HO-TI)**
  - *might be* restricted to univariate settings
  - area overhead *might be* problematic
  - finding uniform representations *might be* challenging
- **Stay with 1st-order secure TI and make *higher-order attacks* harder**
  - increase the noise
  - reduce the signal

Our contribution: Increase the noise by introducing structured randomness into a 1st-order secure TI.

NOISE ADDITION

Started a **case study** on PRESENT cipher:

- particularly investigated the PRESENT S-box $y = S(x)$
- S-box has algebraic degree of 3, at minimum 4 shares
- alternatively, S-box can be decomposed into quadratic functions
- thanks to classification in

Bilgin, Nikova, Nikov, Rijmen, Tokareva, Vitkup: Threshold implementations of small S-boxes. Cryptography and Communications 7(1): 3-33 (2015)

we know that PRESENT S-box $S : A' \circ C_{266}^4 \circ A$ can be decomposed in 7 different ways

$(\mathcal{Q}_{12} \circ \mathcal{Q}_{12})$, $(\mathcal{Q}_{293} \circ \mathcal{Q}_{300})$, $(\mathcal{Q}_{294} \circ \mathcal{Q}_{299})$, $(\mathcal{Q}_{299} \circ \mathcal{Q}_{294})$, $(\mathcal{Q}_{299} \circ \mathcal{Q}_{299})$,
$(\mathcal{Q}_{300} \circ \mathcal{Q}_{293})$, $(\mathcal{Q}_{300} \circ \mathcal{Q}_{300})$

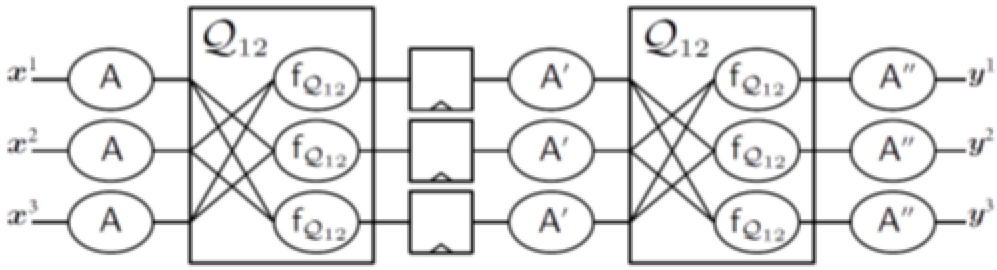
- it means, e.g.: $S : A'' \circ \mathcal{Q}_{12} \circ A' \circ \mathcal{Q}_{12} \circ A$ with three affine functions (A, A', A'')

Idea: Randomly change affine functions on-the-fly to introduce structured (random) noise.

SOME NOTES ON AFFINE FUNCTIONS

Question: How can we implement, e.g. $S : A'' \circ \mathcal{Q}_{12} \circ A' \circ \mathcal{Q}_{12} \circ A$ with random affine functions?

Solution: Uniform TI of $\mathcal{Q}_{12}$ is easily made by *direct sharing*



Question: How many of such 3-tuple affine functions exist (depending on the decomposition)?

Decomposition	# of (A, A', A'')
$\mathcal{Q}_{12} \circ \mathcal{Q}_{12}$	147 456
$\mathcal{Q}_{294} \circ \mathcal{Q}_{299}$	229 376
$\mathcal{Q}_{299} \circ \mathcal{Q}_{294}$	229 376
$\mathcal{Q}_{299} \circ \mathcal{Q}_{299}$	200 704

Note: We exclude those decompositions with $\mathcal{Q}_{300}$, as its uniform TI needs (at least) two stages.

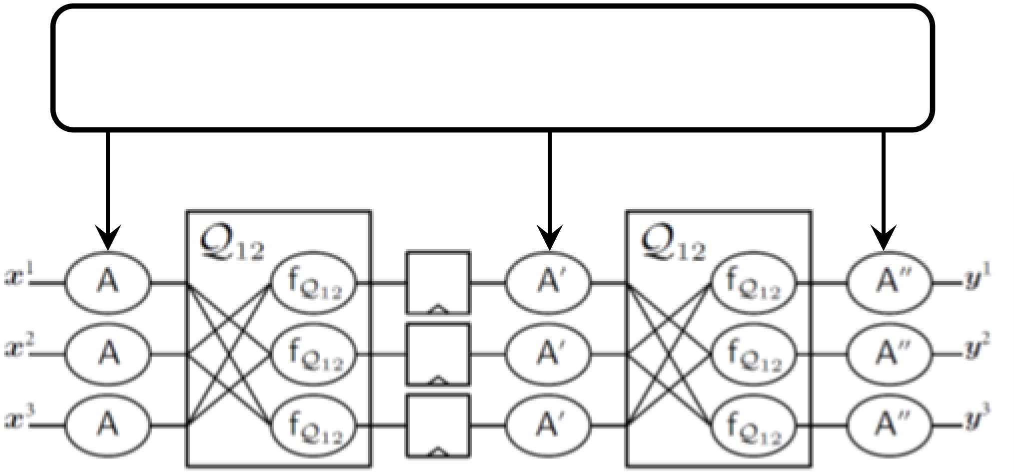
CHANGING THE AFFINE FUNCTIONS ON-THE-FLY

Implementation platform: Spartan-6 FPGA of SAKURA-G side-channel board.

Focused on decomposition $S: A'' \circ \mathcal{Q}_{12} \circ A' \circ \mathcal{Q}_{12} \circ A$ with 147 456 different 3-tuple affine functions.

Hope: If we change the affine functions dynamically, this will introduce random noise to our design and make *second-order attacks* harder (but it should not affect the *third-order vulnerability)*.

Challenge: How can we implement a circuit that allows us to select a random 3-tuple affine function of the set of all possible (e.g. 147 456) affine functions?



OPTION 1: SAVE ALL AFFINE TRANSFORMATIONS

Naïve approach that precomputes and stores all affine transformations on the target device.

Single affine transformation: 4×4 binary matrix and 4-bit constant (20 bit)

All affine transformations: $3 \times 147\,456 \times 20$ bit $= 8640$ kbit

Problem: Spartan-6 LX75 FPGA (*XC6SLX75*) has only 3096 kbit dedicated block memory (BRAM).

Solution: Precompute and store only a fraction of all possible affine triples. For example, 16384 affine triples would occupy 60 BRAMs.

Disadvantages:

- approach is extreme costly in terms of area (memory) requirements
- only covers a fraction of all possible affine functions which may reduce the security

OPTION 2: SEARCH AFFINE TRANSFORMATIONS ON-THE-FLY

This approach just implements the searching algorithm to precompute the affine triples in hardware.

Advantages:

- pretty efficient in terms of area (memory) overhead
- covers all possible (e.g. $147\,456$) options to select an affine triple

Disadvantages:

- affine triples are not found with a constant rate (i.e. algorithm is not time-invariant)
- several affine triples are found sequentially and for a long time no new affine triple may be found
- it may happen that multiple encryptions are performed with a fixed set of affine functions (contradiction with our goal)

OPTION 3: GENERATE AFFINE TRANSFORMATIONS ON-THE-FLY

This approach uses some interesting observations to reduce the number of affine triples to be stored.

Observations for $S : A'' \circ \mathcal{Q}_{12} \circ A' \circ \mathcal{Q}_{12} \circ A$ **:**

- only 384 different input affine functions A
- only 384 different output affine functions A''
- $384 \times 384 = 147456$ different combinations of A and A''
- set of 384 affine functions is made of 48 linear functions combined with 8 different constants

Idea: Store only input and output affine functions and compute middle the affine function on-the-fly.

Approach:

- compute the middle affine using the inverses of A and A''
- $48 \times 2 \times 16$ bit storage for linear and inverse of A (same for A'')
- $2 \times 48 \times 2 \times 16 = 3$ kbit in total, which fits into a single BRAM
- Some extra logic to compute middle affine

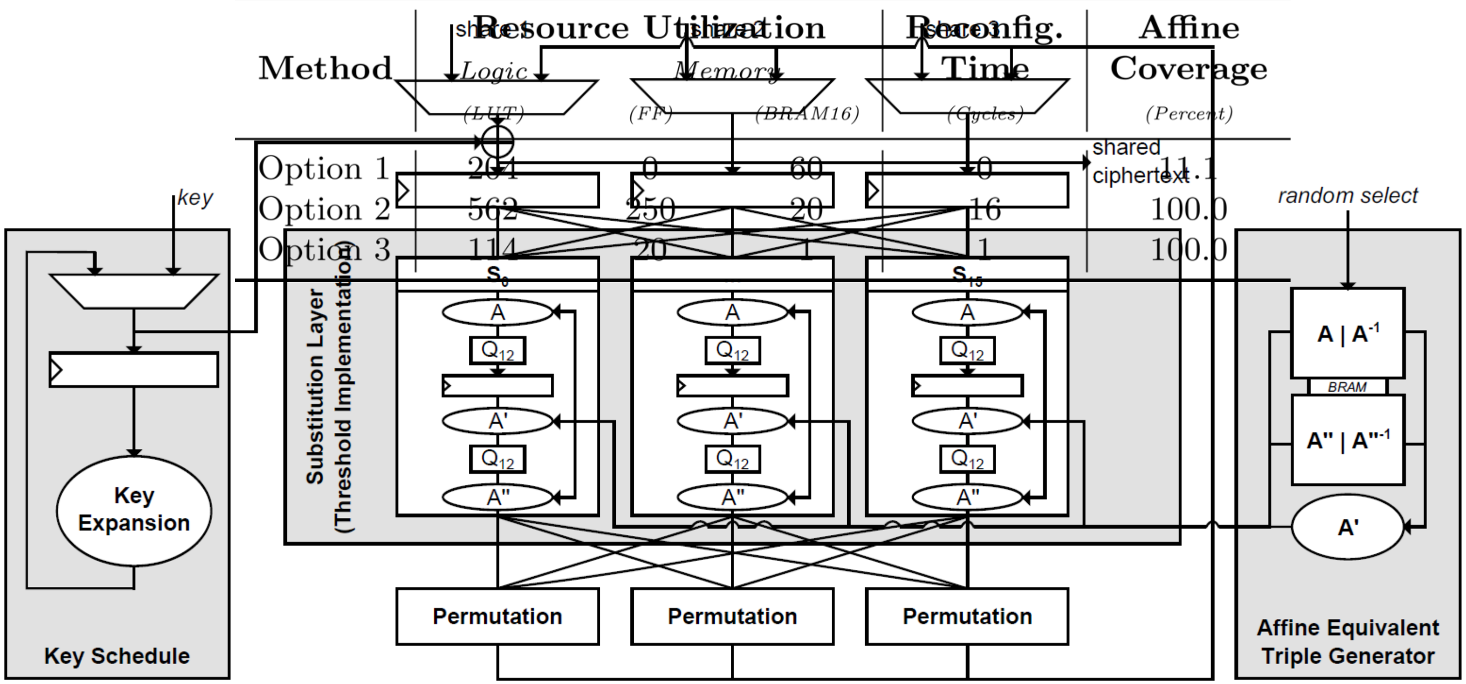
IMPLEMENTATION OF THE CASE STUDY

We implemented PRESENT-128 following a round-based fashion.

- pipeline with two stages, due to the middle register in the decomposed S-box
- 33 clock cycles latency with two full encryptions

We also provide a comparison between the options to realize the random affine selection:

Method	Resource Utilization			Reconfig. Time	Affine Coverage
	Logic (*LUT*)	*Memory* (*FF*)	*Memory* (*BRAM16*)	(*Cycles*)	(*Percent*)
Option 1	201	0	60	0	11.1
Option 2	562	250	20	16	100.0
Option 3	114	20	1	1	100.0

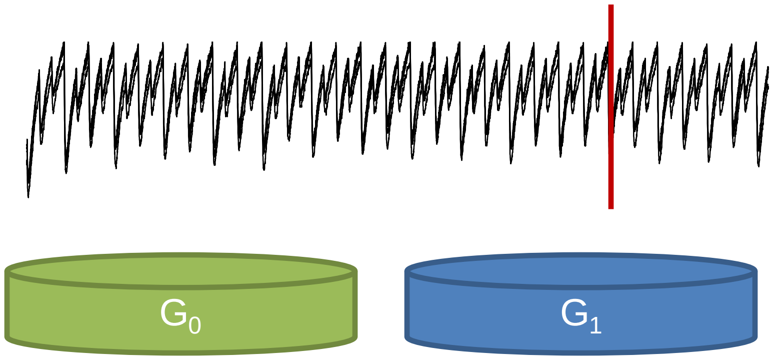


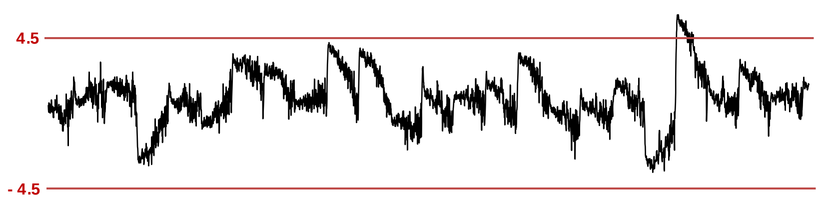
EVALUATION BY WELCH'S *t*-TEST

- measure power traces with digital oscilloscope
- determine distinguisher, e.g.:
  - fix vs. random plaintext (*non-specific t*-test)
- group traces depending on distinguisher
- compute *sample mean* for each point in time
- compute *sample variance* for each point in time
- determine *t*-statistic for each point in time:

$$t = \frac{\mu(T \in G_1) - \mu(T \in G_0)}{\sqrt{\frac{\delta^2(T \in G_1)}{|G_1|} + \frac{\delta^2(T \in G_0)}{|G_0|}}}$$

where μ denotes the *sample mean* and δ denotes the *sample variance*.



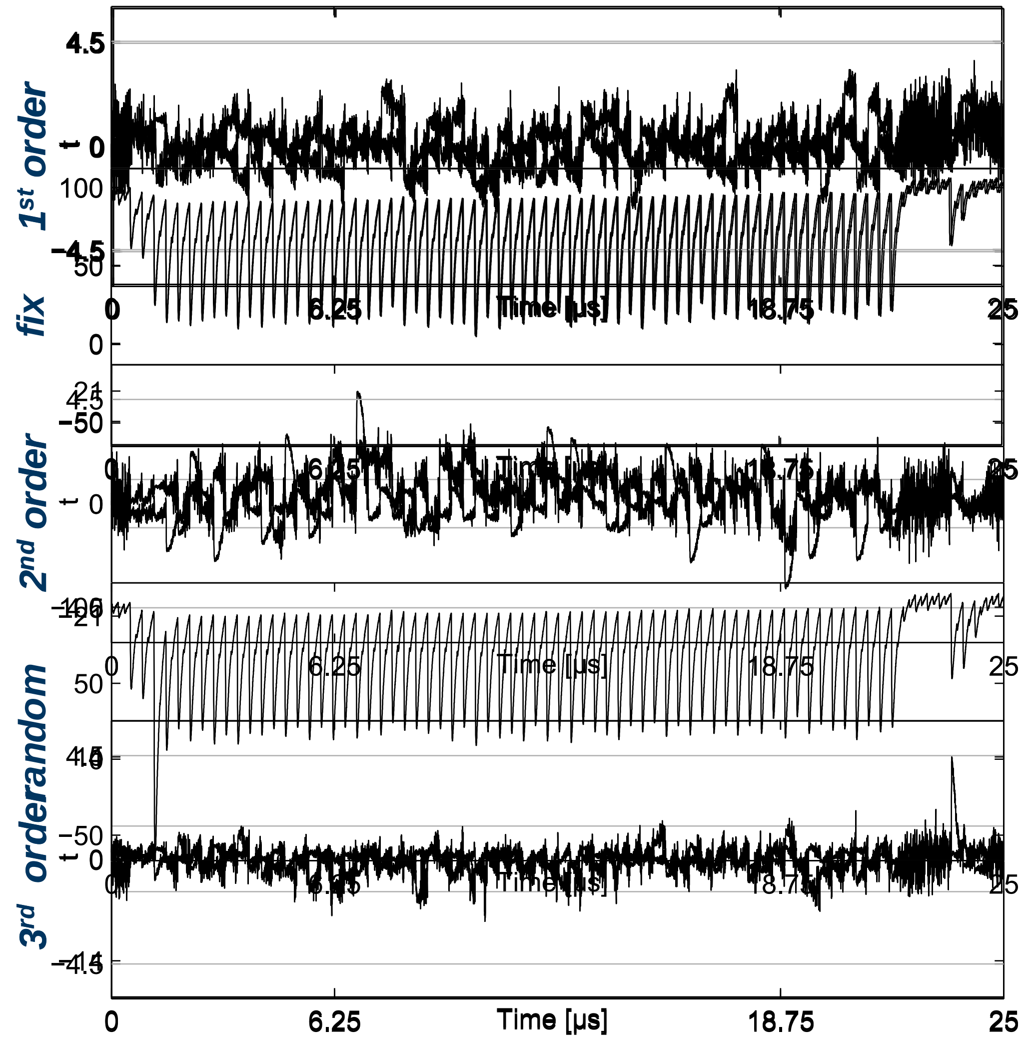


Fail/Pass Criteria: If there is any point in time for which the t-statistic exceeds a threshold of $\pm$ 4.5 the device under test fails.

More info: "Leakage Assessment Methodology - a clear roadmap for side-channel evaluations", Cryptology ePrint Archive, Report 2015/207

RESULTS

- Sample trace for:
  - fix affine triples
  - random affine triples
- first-order, second-order and third-order *non-specific t*-test:
  - fixed affine triples (50 million traces)
  - random affine triples (200 million traces)
- as expected no first-order leakage detected (TI)



CONCLUSION:

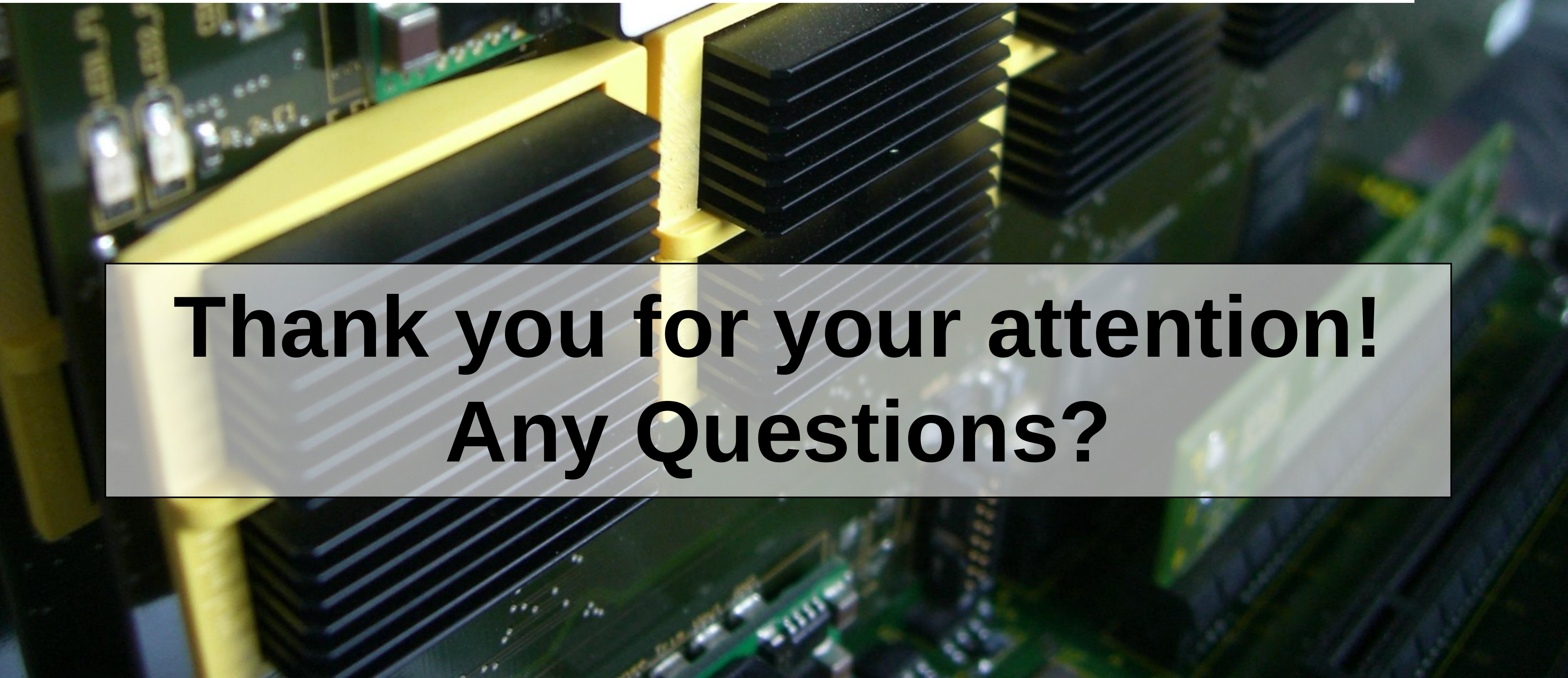
Changing the affine triples randomly could avoid detectable second- and third-order leakage.

RUHR-UNIVERSITÄT BOCHUM

AFFINE EQUIVALENCE AND ITS APPLICATION TO TIGHTENING THRESHOLD IMPLEMENTATIONS

pascal.sasdrich@rub.de

22ND INT. CONFERENCE ON SELECTED AREAS IN CRYPTOGRAPHY, SACKVILLE, NB, CANADA AUGUST 13, 2015



Thank you for your attention!

Any Questions?