Designing cryptographic algorithms with physical attack resistance in mind

Silvia Mella
Institute for Computing and Information Sciences, Radboud University

June 5, 2025

Radboud Universiteit

WHO AM I?

- Assistant professor @Radboud University
  - Design and analysis of cryptographic permutations
  - Hardware implementations
  - Side-channel attacks
- Post-doc @Radboud University
  - Same stuff
- Cryptographer @STMicroelectronics, Italy
  - Hardware accelerators for public-key crypto
  - Design, verification, pre-silicon side-channel and fault attacks evaluations
- PhD in CS @University of Milano, Italy
  - Crypto
- Bachelor and Master in Math @University of Milano, Italy

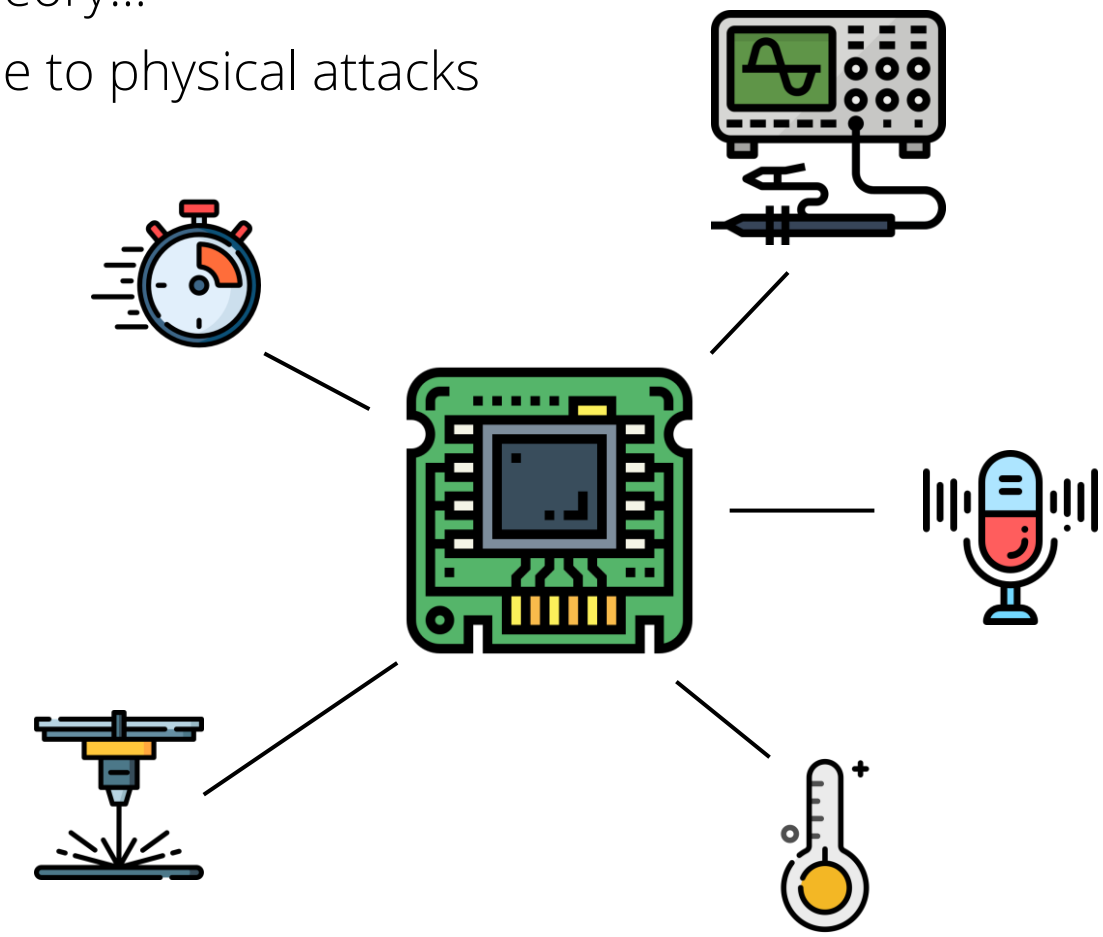
BEFORE WE START

- This presentation focuses on SCA
- And symmetric crypto
- Actually, permutation-based crypto

Who is familiar with permutation-based crypto?

PHYSICAL ATTACKS

- Cryptographic algorithms can be secure in theory…
- …but their implementations can be vulnerable to physical attacks
- Adversaries can extract secrets using:
  - Power consumption
  - EM emissions
  - Timing differences
  - Fault injections



COUNTERMEASURES

- Side-channel and fault countermeasures often require:
  - Duplicated computations
  - Redundant logic
  - Extra randomness
- This results in
  - Higher execution time
  - Bigger code size
  - More silicon area
- Protected implementations may still leak if not correctly crafted!

DESIGN-LEVEL RESISTANCE

Goal

Reduce complexity
of
secure implementations

How

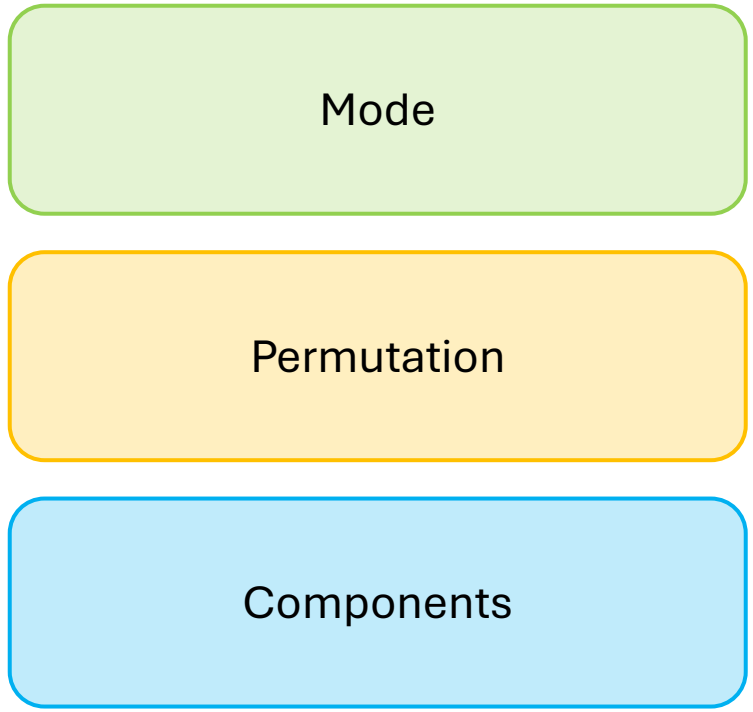
By considering
physical attacks
during design

Benefit

Easier to protect securely
Less room for errors
Lower implementation cost

ROADMAP OF TODAY

- Bottom-up with examples for each level of the design hierarchy
  - Inherently masking friendly components
  - Order of operations in the permutation
  - Modes that can either reduce the attack surface or prevent them
- Examples from real-world ciphers
- Open questions

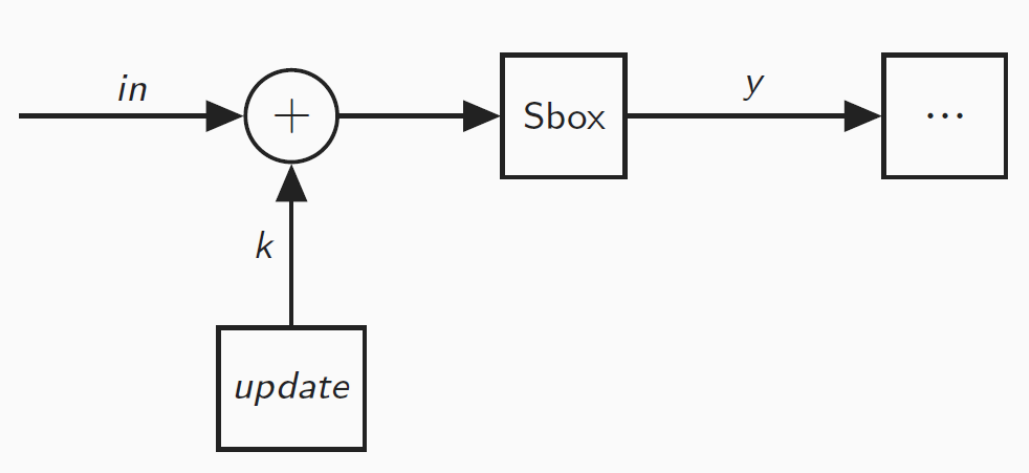


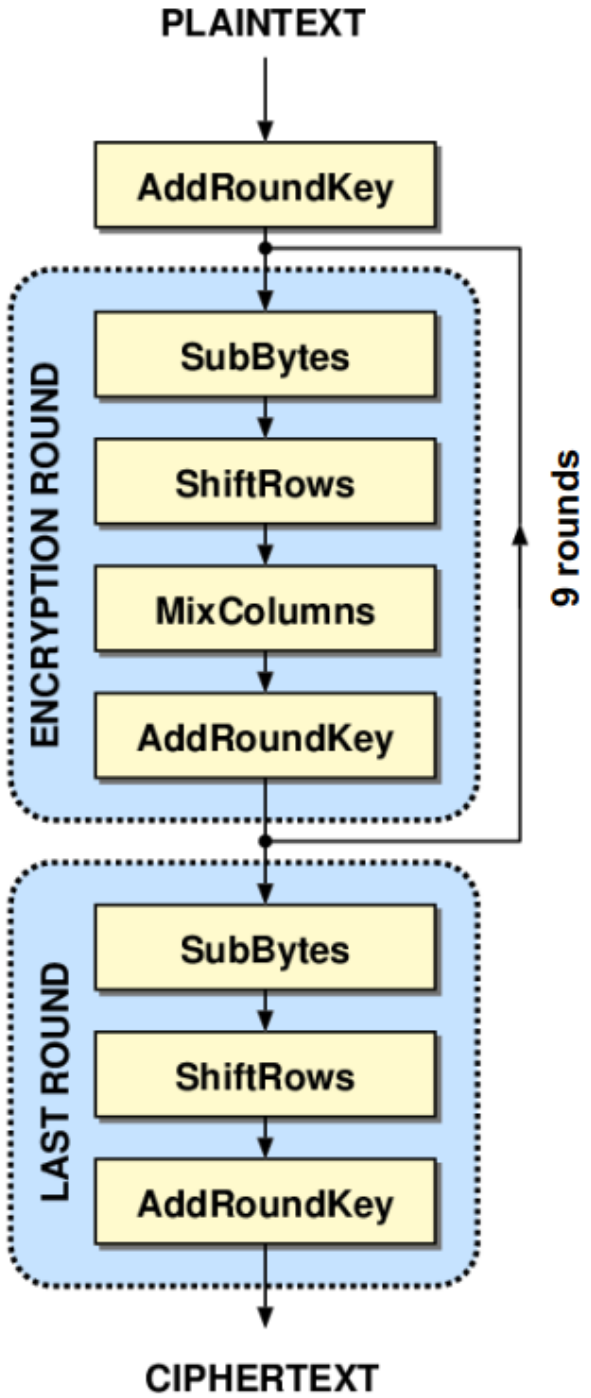
DPA - MAIN CONCEPTS

- Differential Power Analysis (DPA) is potentially feasible when there exists a sensitive variable that depends non-linearly on
  - something we want to learn (usually the key), which is fixed across multiple executions
  - something we know (usually the message), which changes across multiple executions
- Main steps:
  1. Choose your sensitive variable (for which exhaustive key search is possible)
  2. Collect measurements, known plaintext/ciphertext
  3. Predict (hypothetical) intermediate values by making sub-key guesses
  4. Decide on the leakage model
  5. Recover the key by statistical or other means using a side-channel distinguisher

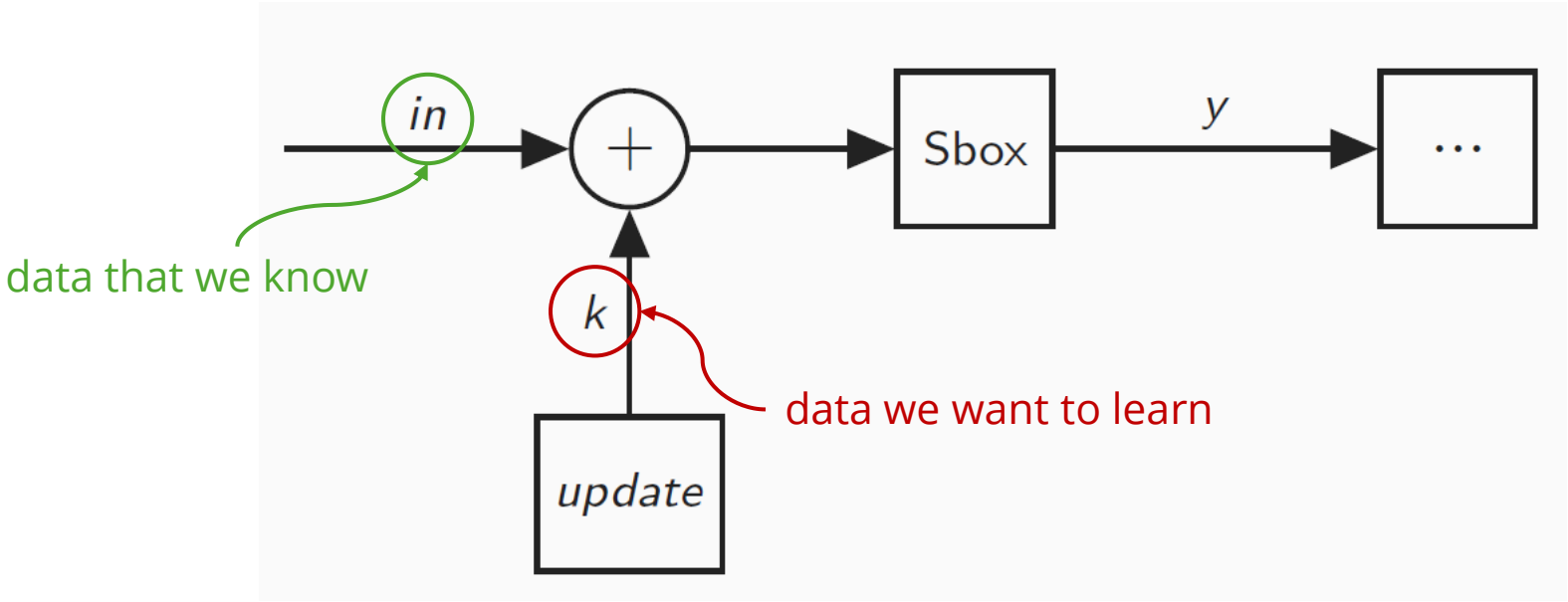
DPA ON AES

- The sensitive variable must be fairly small to make exhaustive key-search possible
- A common choice is one S-box output, i.e., y = S-box(in ⊕ k)
- Examples of leakage model are single-bit model, Hamming weight, and Hamming distance
- Examples of distinguisher are difference of means (DPA) and Pearson correlation (CPA)



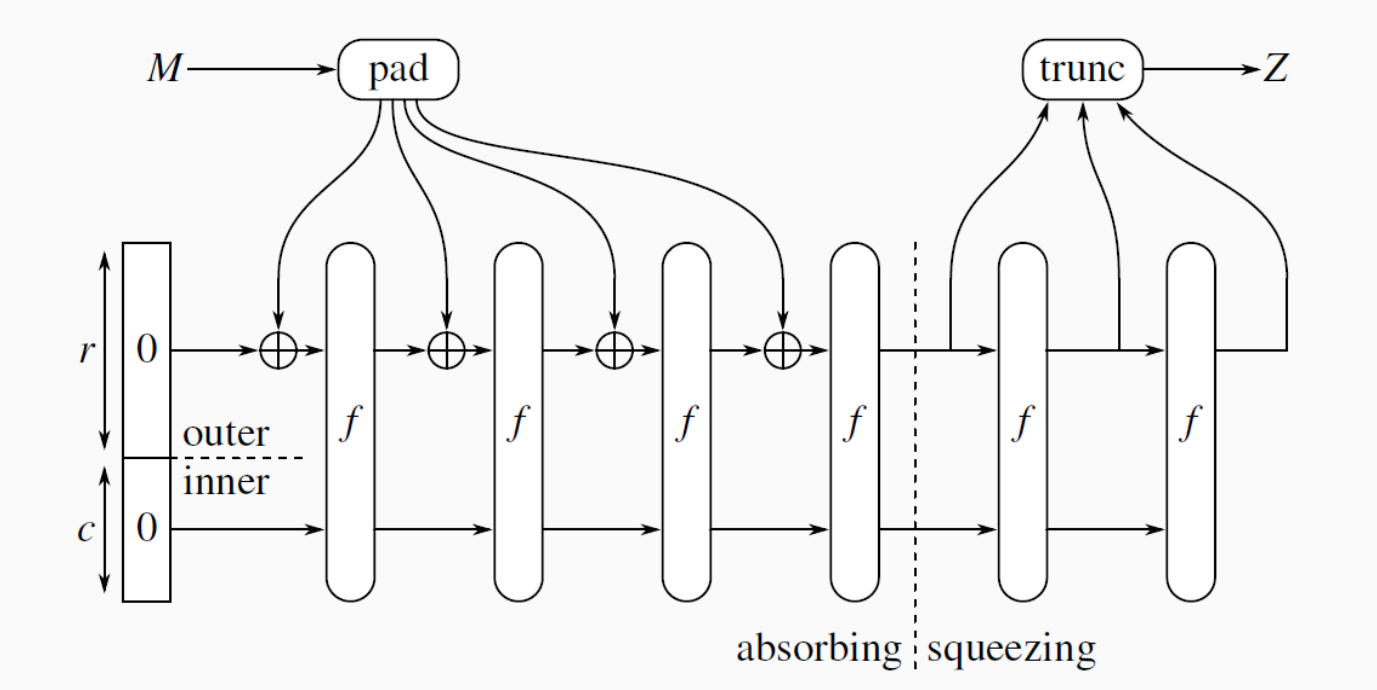


GENERALIZATION



- in can be any data that we know
- k is not always the secret key, but it can be something derived from it or in general any data we want to learn

THE SPONGE CONSTRUCTION*

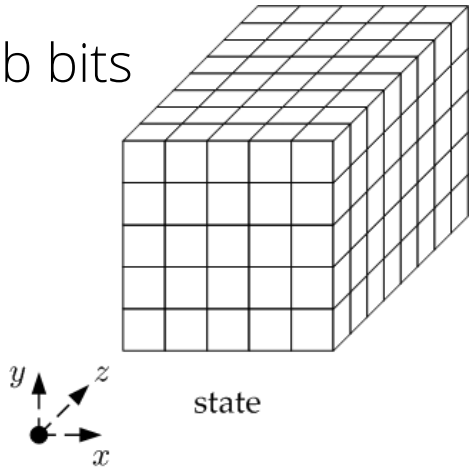


- Used in KECCAK (SHA-3), standardized by NIST in 2015
- Arbitrary-length input and output, r-bit rate, b = r + c, b-bit permutation, c-capacity
- Keyed mode: part of the input is secret key
- Security relies on secrecy of inner state

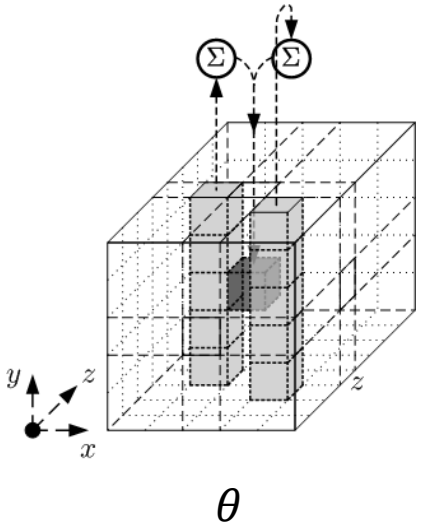
* G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. On the Indifferentiability of the Sponge Construction. EUROCRYPT 2008.

KECCAK-*f* [b]

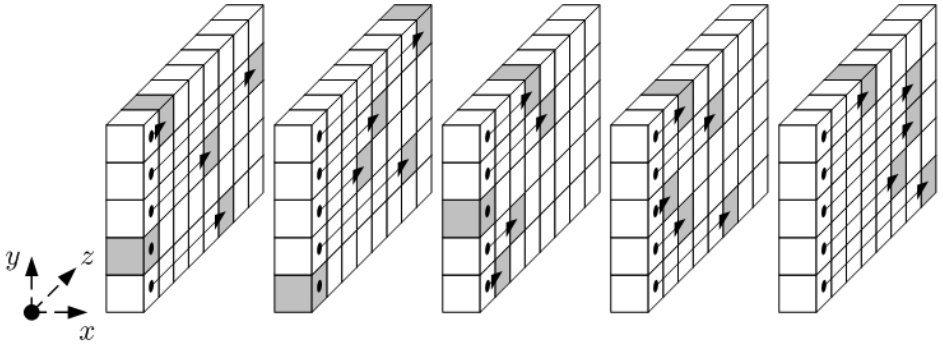
- Operates on 3D state of b bits
  - (5 x 5)-bit slices
  - 2^l-bit lanes
  - $0 \leq l < 7$



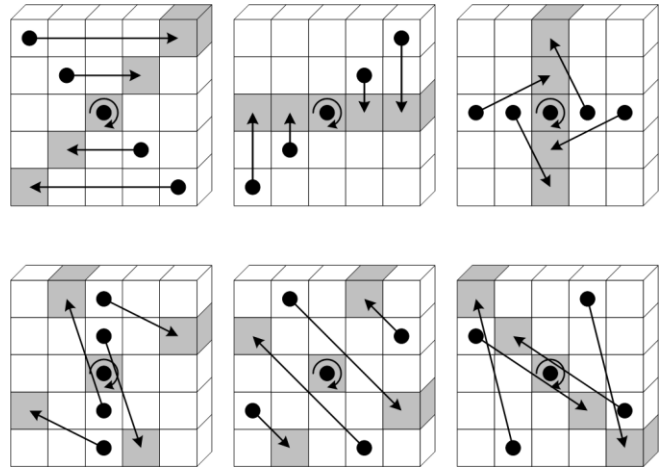
- Round function R with 5 steps:
  - θ : mixing layer (on columns)
  - ρ : bit transposition (intra lanes)
  - π : bit transposition (inter lanes)
  - χ : non-linear layer (on rows)
  - ι : round constants (on lane (0,0))
- Number of rounds: $12 + 2 \cdot l$ for $b = 2^l \cdot 25$
  - 12 rounds in KECCAK-*f* [25]
  - 24 rounds in KECCAK-*f* [1600]



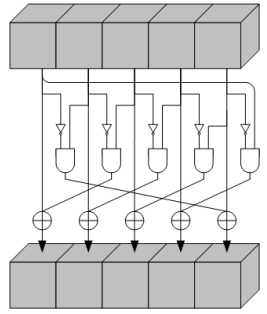
θ



ρ

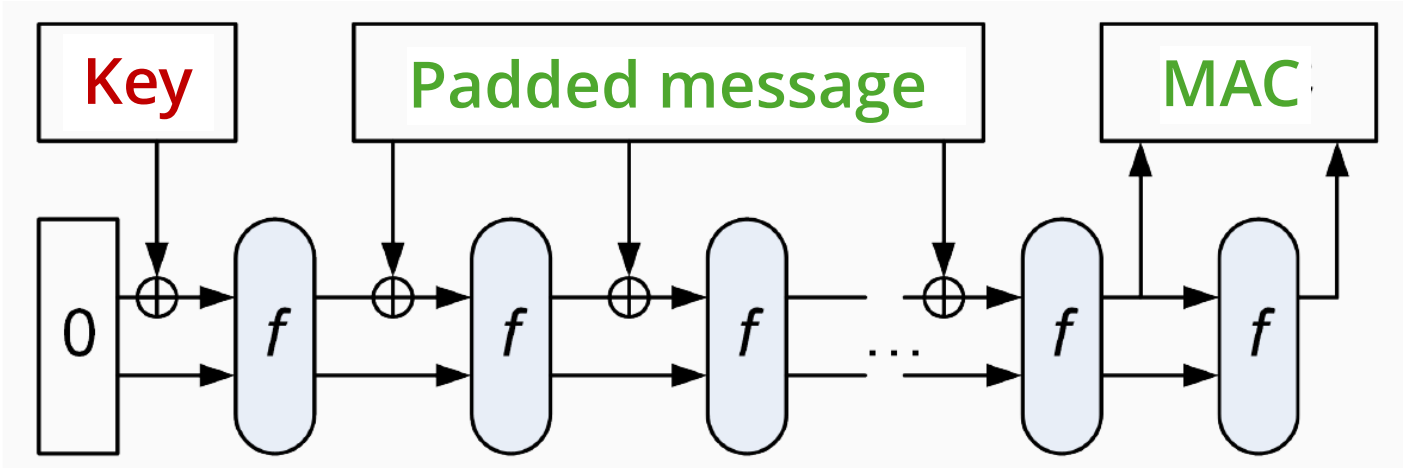


π



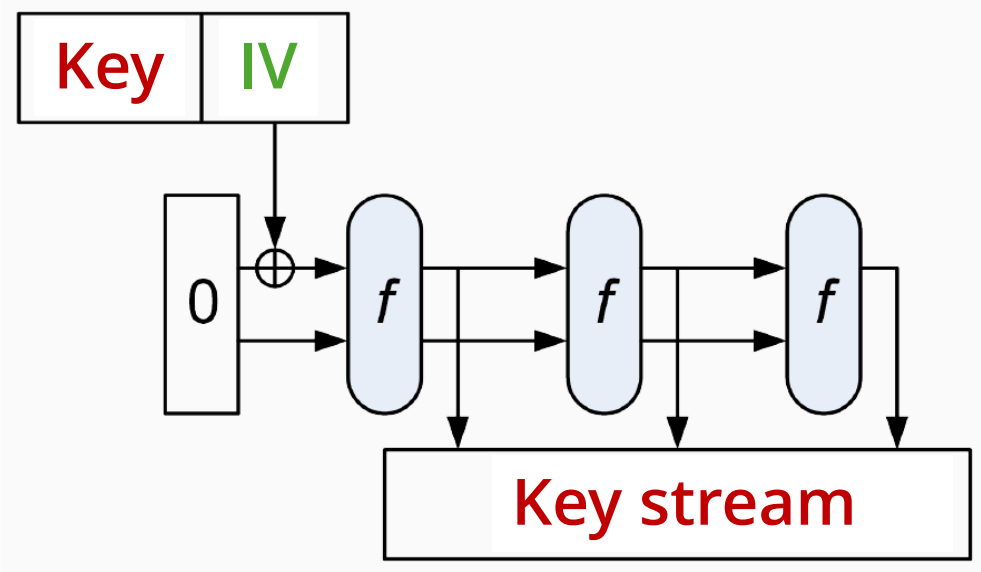
χ_5

USING SPONGE FOR MAC



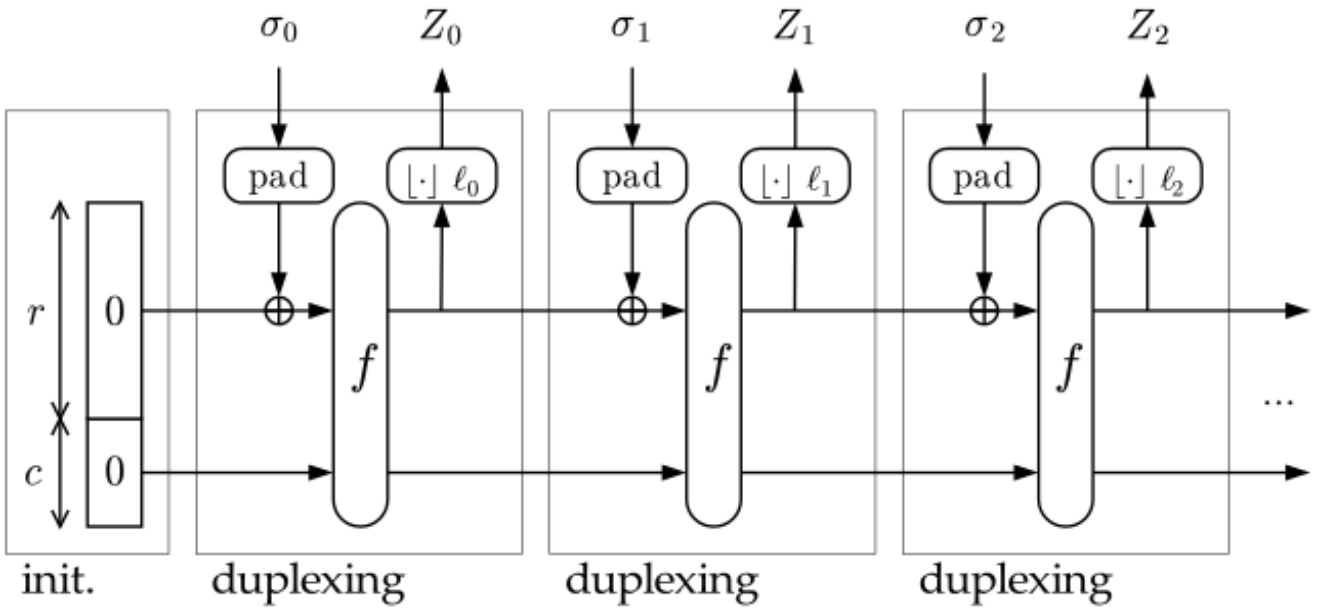
How can we attack it with DPA?

USING SPONGE FOR KEY STREAM



How can we attack it with DPA?

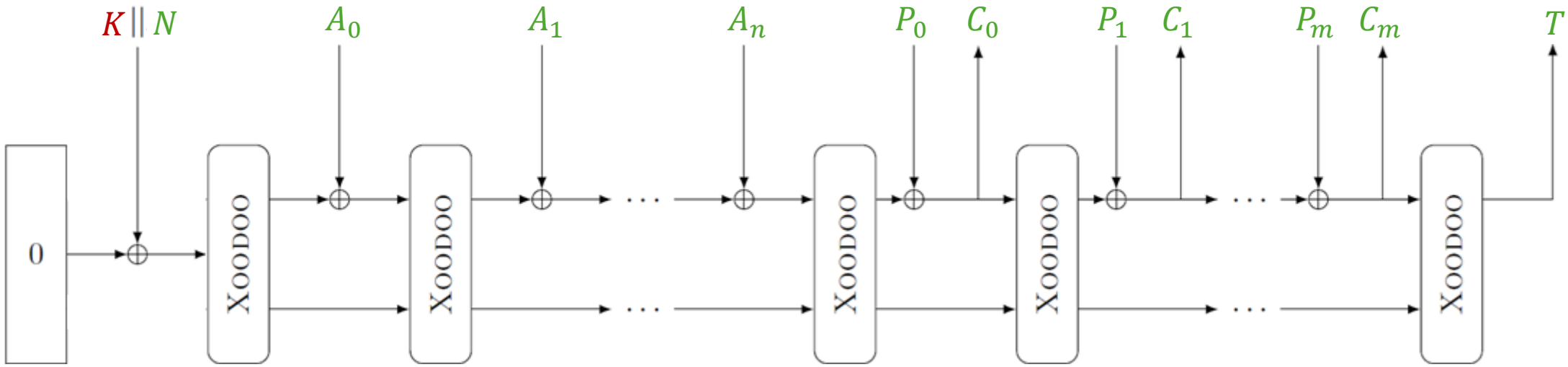
THE DUPLEX CONSTRUCTION*



- Allows the alternation of input and output blocks at the same rate as the sponge construction
- Input blocks are padded, and output blocks can be truncated
- Security equivalent to that of the sponge construction
- Keyed mode: part of the input is secret key

* G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. On the Indifferentiability of the Sponge Construction. EUROCRYPT 2008.

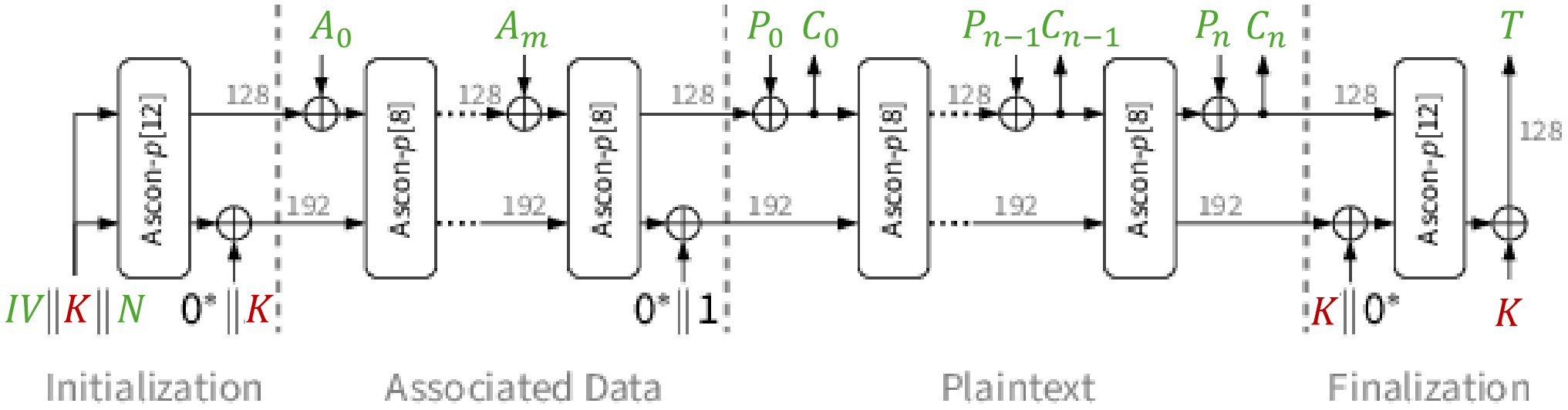
USING DUPLEX FOR AUTHENTICATED ENCRYPTION: XOODYAK*



How can we attack it with DPA?

* Joan Daemen, Seth Hoffert, Silvia Mella, Michaël Peeters, Gilles Van Assche, Ronny Van Keer. Xoodyak, a final update. Publication to NIST Lightweight Cryptography Standardization Process (round 3), 2022.

USING DUPLEX FOR AUTHENTICATED ENCRYPTION: ASCON*



How can we attack it with DPA?

* C. Dobraunig, M. Eichlseder, F. Mendel, M. Schlaffer. Ascon v1.2: Lightweight Authenticated Encryption and Hashing. J. Cryptol., 2021.

SCA COUNTERMEASURES

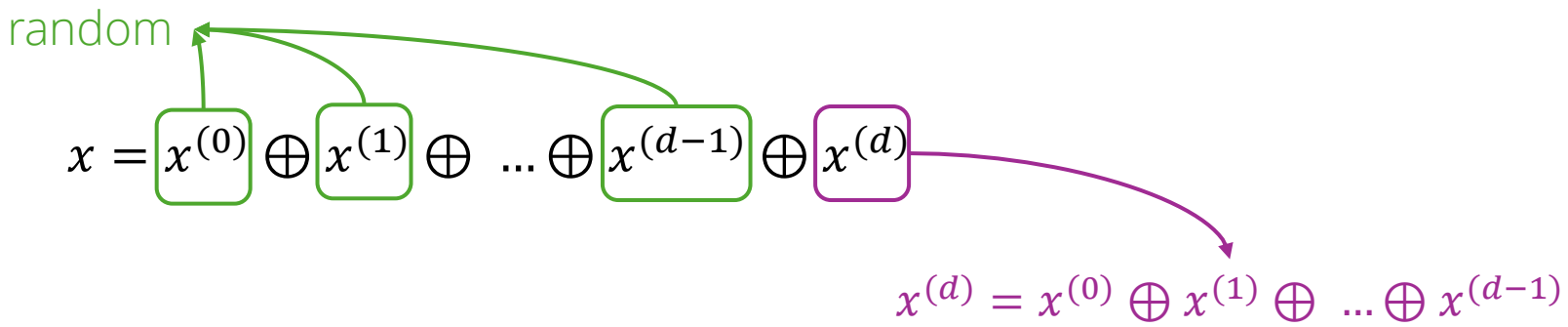
- Countermeasures can be applied on different levels
  - Transistor-level: logical gates and circuits are built in such a way that the information leakage is reduced
  - Program-level: dummy instructions, randomized order, etc. to make the alignment of traces more difficult
  - Algorithmic level: operations are computed in a way that reduces information leakage
  - Protocol level: limits the number of computations an attacker can perform with a given key
- No 100 % security
- Robustness: combine countermeasures at different levels
- Cost: area, energy and power consumption increase, loss of speed, . . .

MASKING

- Implemented at algorithmic level
- Purpose: breaking the link between sensitive variables and power consumption
- Principle: randomizing intermediate values with a secret sharing scheme
  - Random masks hide the native intermediate values
  - The power consumption depends on the randomized values on which the computation is performed (and not the native values)
- Each sensitive variable is split into multiple shares
- When the splitting operation is an Exclusive OR (XOR), we refer to it as a Boolean masking
- Many Boolean masking schemes have been proposed over the years: ISW, TI, CMS, DOM, etc.

BOOLEAN MASKING — BASIC PRINCIPLES

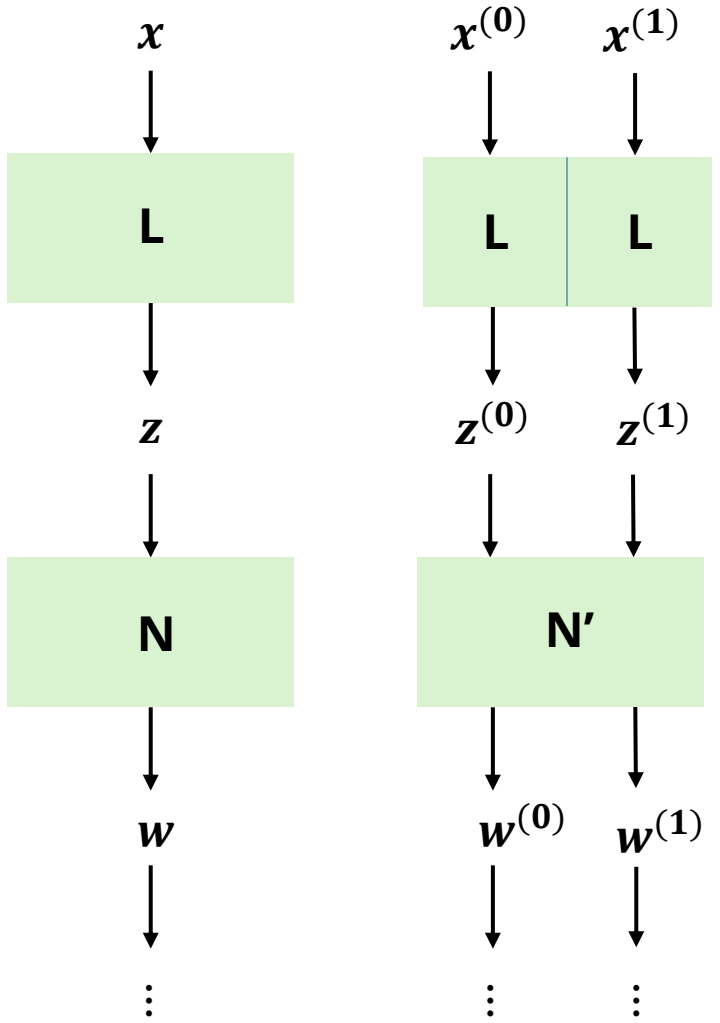
- A dth-order (Boolean) masking scheme splits an internal sensitive value x into d + 1 shares



- Only the combination of all shares reveals x
- Any set of at most d shares should not leak information about x
- The number of traces required for a successful attack grows exponentially w.r.t. the security order d

BOOLEAN MASKING WITH 2 SHARES

$$x = (x_0, x_1, \dots, x_{n-1}) \in \mathbb{F}_2^n$$



$$x = x^{(0)} \oplus x^{(1)}$$

$$z = z^{(0)} \oplus z^{(1)}$$

$$w = w^{(0)} \oplus w^{(1)}$$

From linear algebra: linear maps preserve operations

$$f(u+v) = f(u) + f(v), \quad f(cu) = cf(u)$$

$$L(x) = L\left(x^{(0)} \oplus x^{(1)}\right) = L\left(x^{(0)}\right) \oplus L\left(x^{(1)}\right)$$

$$z_0 = L\left(x^{(0)}\right)$$
$$z_1 = L\left(x^{(1)}\right)$$

$$N(x) = N\left(x^{(0)} \oplus x^{(1)}\right) \neq N\left(x^{(0)}\right) + N\left(x^{(1)}\right)$$

$$w_0 = N_0'\left(x^{(0)}, x^{(1)}\right)$$
$$w_1 = N_1'\left(x^{(0)}, x^{(1)}\right)$$

EXAMPLE: AND GATE WITH 2 SHARES

- Inputs: $a = a^{(0)} \oplus a^{(1)}$, $b = b^{(0)} \oplus b^{(1)}$
- Goal: Compute $z = a \cdot b$ in a masked way
- Naïve masked AND (not secure):

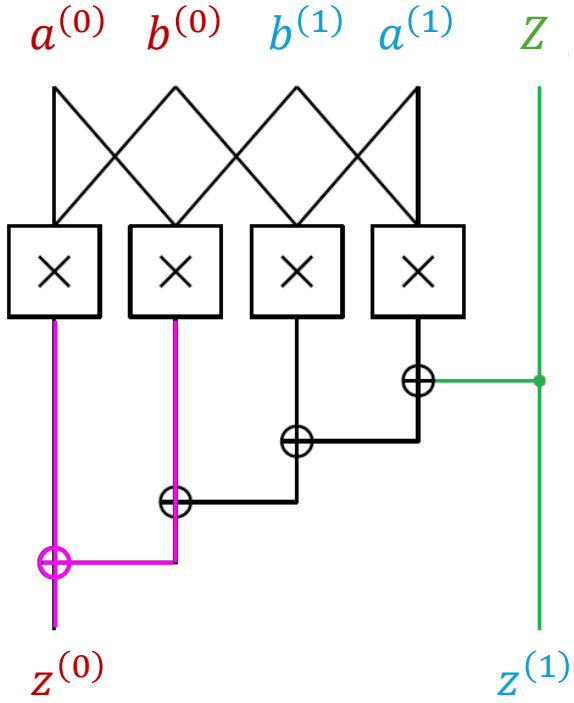
$$\begin{aligned} z &= \left(a^{(0)} \oplus a^{(1)}\right) \cdot \left(b^{(0)} \oplus b^{(1)}\right) \\ &= a^{(0)} \cdot b^{(0)} \oplus a^{(0)} \cdot b^{(1)} \oplus a^{(1)} \cdot b^{(0)} \oplus a^{(1)} \cdot b^{(1)} \end{aligned}$$

- 1 AND operation is replaced by 4 AND operations and 3 XOR operations

AND WITH CLASSICAL MASKING

$$z = a^{(0)} \cdot b^{(0)} \oplus a^{(0)} \cdot b^{(1)} \oplus a^{(1)} \cdot b^{(0)} \oplus a^{(1)} \cdot b^{(1)}$$

- It is fundamental to keep computed variables independent from native variables
- All partial products are independent of a and b
- The resulting sum is not
- A fresh random share Z needs to be added to break the dependency between the intermediates
- If the intermediate result signal reach the XOR gate before Z then $z^{(0)}$ is not independent to the value of b

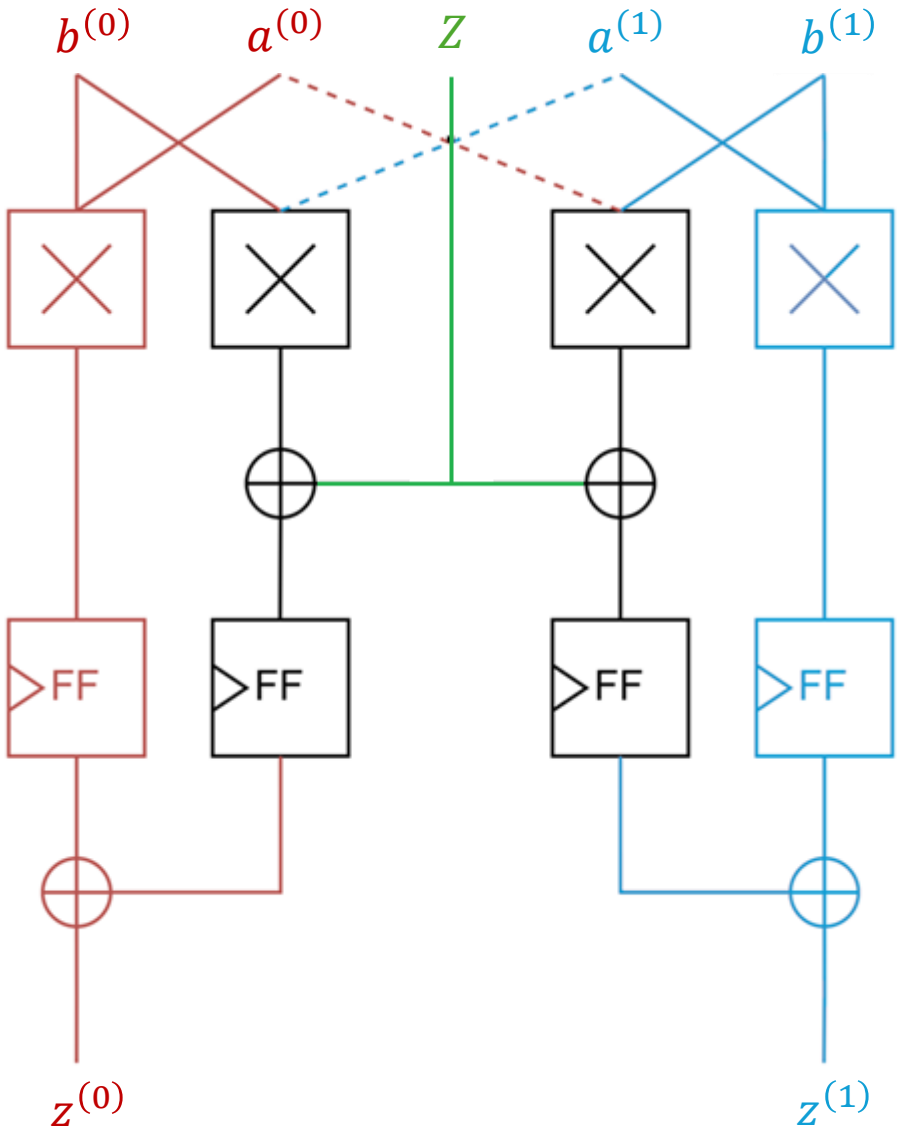


$$a^{(0)} \cdot b^{(0)} \oplus a^{(0)} \cdot b^{(1)} = a^{(0)} \cdot \left(b^{(0)} \oplus b^{(1)}\right) = a^{(0)} \cdot b$$

AND WITH DOMAIN ORIENTED MASKING (DOM)

- Unintended interactions between shares can cause 1st order leakage
- It is fundamental to keep computed variables independent from native variables
- DOM idea:
  - keep the shares of all domains independent from shares of other domains
  - secure domain crossings by adding a fresh random share Z and by using a register in order to prevent that glitches propagate from one domain to the other

$$z^{(0)} = a^{(0)} \cdot b^{(0)} \oplus a^{(1)} \cdot b^{(0)} \oplus Z$$
$$z^{(1)} = a^{(1)} \cdot b^{(1)} \oplus a^{(0)} \cdot b^{(1)} \oplus Z$$



EXAMPLE: AND GATE WITH 3 SHARES

- Inputs: $a = a^{(0)} \oplus a^{(1)} \oplus a^{(2)}$, $b = b^{(0)} \oplus b^{(1)} \oplus b^{(2)}$
- Goal: Compute $z = a \cdot b$ in a masked way
- Naïve masked AND (not secure):

$$
\begin{aligned}
z &= \left(a^{(0)} \oplus a^{(1)} \oplus a^{(2)}\right) \cdot \left(b^{(0)} \oplus b^{(1)} \oplus b^{(2)}\right) \\
&= a^{(0)} \cdot b^{(0)} \oplus a^{(0)} \cdot b^{(1)} \oplus a^{(0)} \cdot b^{(2)} \oplus a^{(1)} \cdot b^{(0)} \oplus a^{(1)} \cdot b^{(1)} \oplus a^{(1)} \cdot b^{(2)} \\
&\quad \oplus a^{(2)} \cdot b^{(0)} \oplus a^{(2)} \cdot b^{(1)} \oplus a^{(2)} \cdot b^{(2)}
\end{aligned}
$$

- 1 AND operation is replaced by 9 AND operations and 8 XOR operations

EXAMPLE: MONOMIAL OF DEGREE 3 WITH 2 SHARES

- Inputs: $a = a^{(0)} \oplus a^{(1)}$, $b = b^{(0)} \oplus b^{(1)}$, $c = c^{(0)} \oplus c^{(1)}$
- Goal: Compute $z = a \cdot b \cdot c$ in a masked way
- Naïve masked AND (not secure):

$$
\begin{aligned}
\boldsymbol{z} &= \left(a^{(0)} \oplus a^{(1)}\right) \cdot \left(b^{(0)} \oplus b^{(1)}\right) \cdot \left(c^{(0)} \oplus c^{(1)}\right) \\
&= \left(a^{(0)} \cdot b^{(0)} \oplus a^{(0)} \cdot b^{(1)} \oplus a^{(1)} \cdot b^{(0)} \oplus a^{(1)} \cdot b^{(1)}\right) \cdot \left(c^{(0)} \oplus c^{(1)}\right) \\
&= a^{(0)} \cdot b^{(0)} \cdot c^{(0)} \oplus a^{(0)} \cdot b^{(1)} \cdot c^{(0)} \oplus a^{(1)} \cdot b^{(0)} \cdot c^{(0)} \oplus a^{(1)} \cdot b^{(1)} \cdot c^{(0)} \\
&\quad \oplus a^{(0)} \cdot b^{(0)} \cdot c^{(1)} \oplus a^{(0)} \cdot b^{(1)} \cdot c^{(1)} \oplus a^{(1)} \cdot b^{(0)} \cdot c^{(1)} \oplus a^{(1)} \cdot b^{(1)} \cdot c^{(1)}
\end{aligned}
$$

- 1 AND operation is replaced by 16 AND operations and 7 XOR operations

GENERALIZING

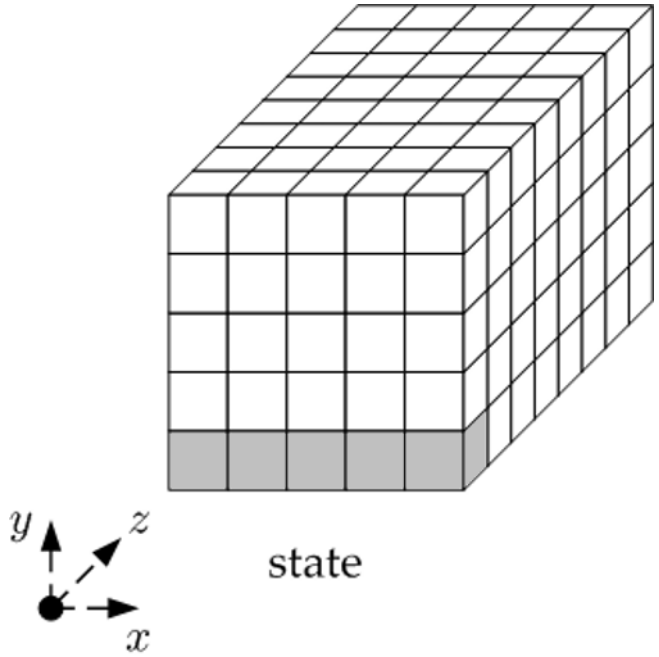
- Linear operations can simply be duplicated and performed on each share independently, i.e. a linear increase in the area/time
- Non-linear parts grow exponentially
- To protect against dth-order DPA, the minimal number of shares is d + 1
- The number of expansion monomials of a multiplication is then $(d + 1)^2$
- A monomial of degree t expands to $(d + 1)^t$ monomials

Designing algorithms with simpler nonlinear operations can ease masking

THE χ_n FUNCTION*

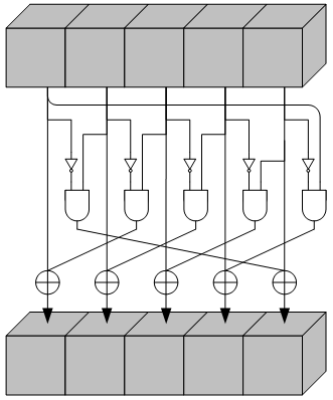
$$\chi_n : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$$

$$x_i \leftarrow x_i \oplus \overline{x_{i+1}} \cdot x_{i+2}$$



- Example: KECCAK (SHA-3) uses χ_5

$$\begin{aligned}
x_0 &\leftarrow x_0 \oplus \overline{x_1} \cdot x_2 \\
x_1 &\leftarrow x_1 \oplus \overline{x_2} \cdot x_3 \\
x_2 &\leftarrow x_2 \oplus \overline{x_3} \cdot x_4 \\
x_3 &\leftarrow x_3 \oplus \overline{x_4} \cdot x_0 \\
x_4 &\leftarrow x_4 \oplus \overline{x_0} \cdot x_1
\end{aligned}$$



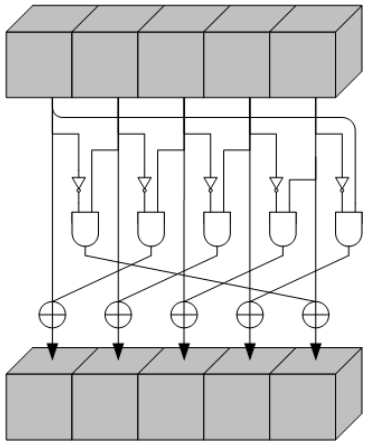
* Joan Daemen. Cipher and hash function design, strategies based on linear and differential cryptanalysis, PhD Thesis. K.U.Leuven, 1995. http://jda.noekeon.org/.

χ_n IS MASKING-FRIENDLY

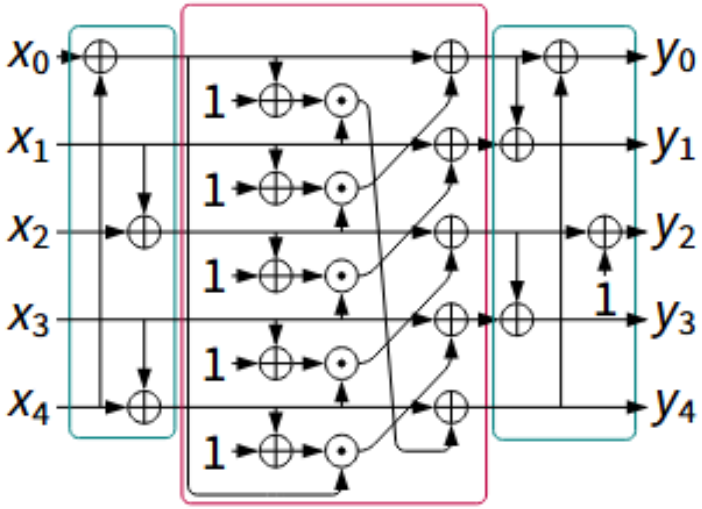
- Bitwise operation → naturally suited to masking
- Algebraic degree 2 → low number of expansion monomials
- Local regular structure → parallelizable
- Of course, good cryptographic properties and implementation advantages

χ_n is often used because it's naturally friendly to side-channel protections

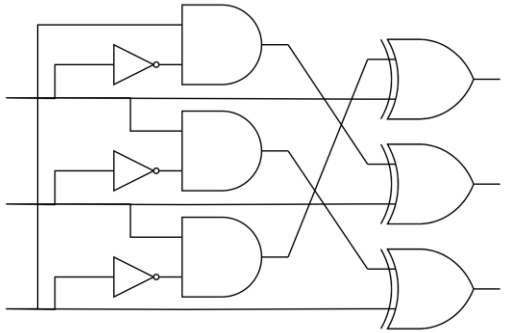
χ_n IN MANY LIGHTWEIGHT DESIGNS



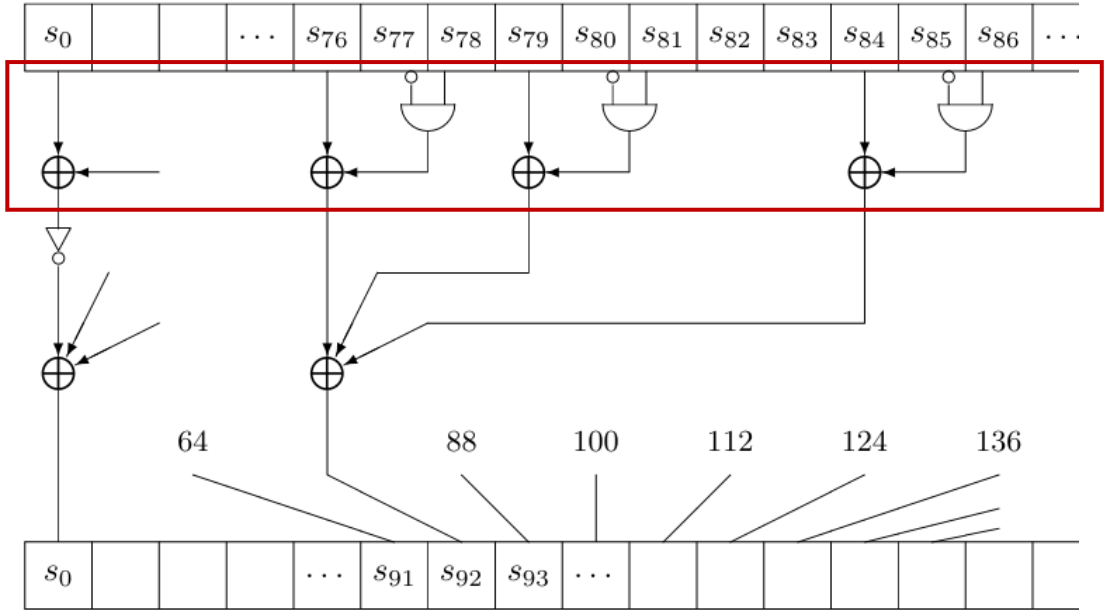
χ_5 in KECCAK



ASCON S-box based on χ_5



χ_3 in XOODOO



χ_{257} in SUBTERRANEAN

MASKING χ_n

- $\chi : x_i \leftarrow x_i \oplus (x_{i+1} \oplus 1) \cdot x_{i+2}$ becomes:

$$\begin{aligned} z &= \left(x_i^{(0)} \oplus x_i^{(1)}\right) \oplus \left(x_{i+1}^{(0)} \oplus x_{i+1}^{(1)} \oplus 1\right)\left(x_{i+2}^{(0)} \oplus x_{i+2}^{(1)}\right) \\ &= x_i^{(0)} \oplus x_i^{(1)} \oplus x_{i+1}^{(0)} \cdot x_{i+2}^{(0)} \oplus x_{i+1}^{(0)} \cdot x_{i+2}^{(1)} \oplus x_{i+1}^{(1)} \cdot x_{i+2}^{(0)} \oplus x_{i+1}^{(1)} \cdot x_{i+2}^{(1)} \oplus x_{i+2}^{(0)} \oplus x_{i+2}^{(1)} \end{aligned}$$

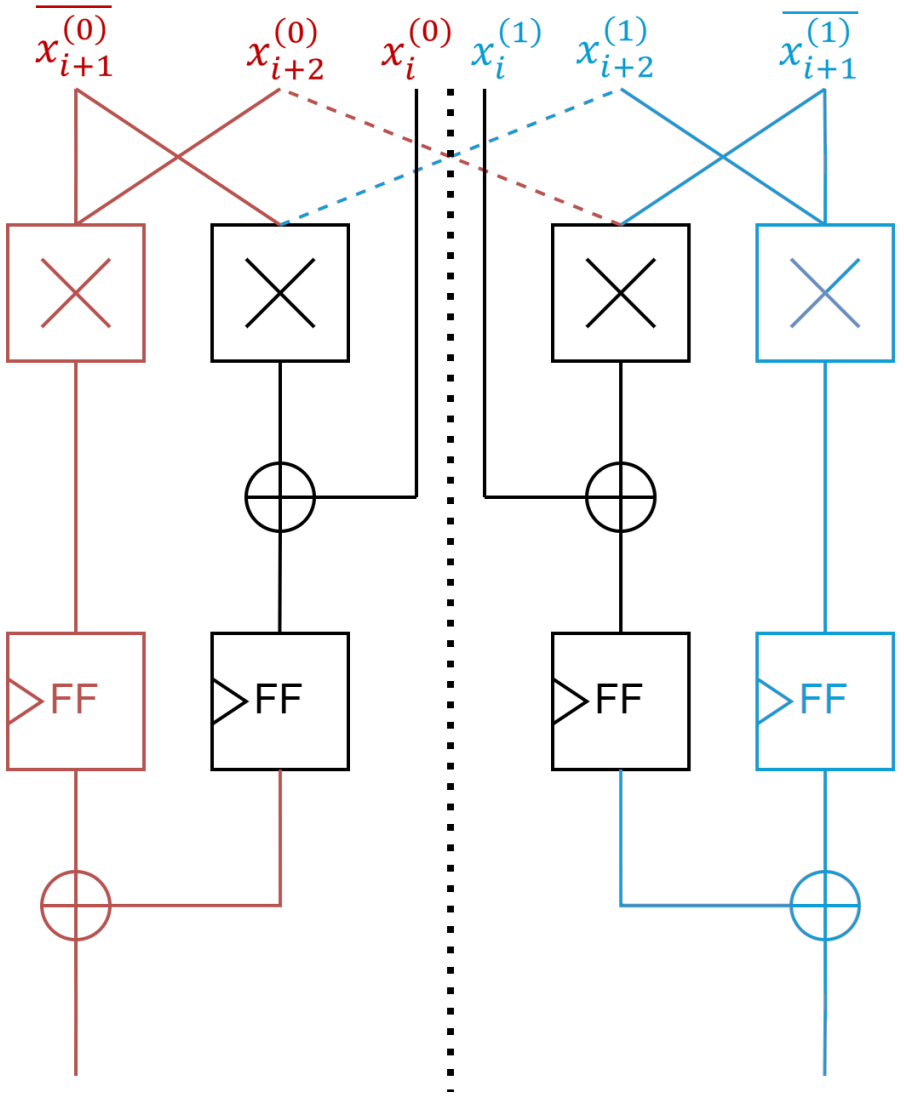
- We distribute the terms, e.g.:

$$\begin{aligned} x_i^{(0)} &= x_i^{(0)} \oplus (x_{i+1}^{(0)} \oplus 1) \cdot x_{i+2}^{(0)} \oplus x_{i+1}^{(0)} \cdot x_{i+2}^{(1)} \\ x_i^{(1)} &= x_i^{(1)} \oplus (x_{i+1}^{(1)} \oplus 1) \cdot x_{i+2}^{(1)} \oplus x_{i+1}^{(1)} \cdot x_{i+2}^{(0)} \end{aligned}$$

- Cost: 1 XOR, 1 AND and 1 NOT replaced by 4 XOR, 4 AND and 2 NOT

MASKING χ_n WITH DOM

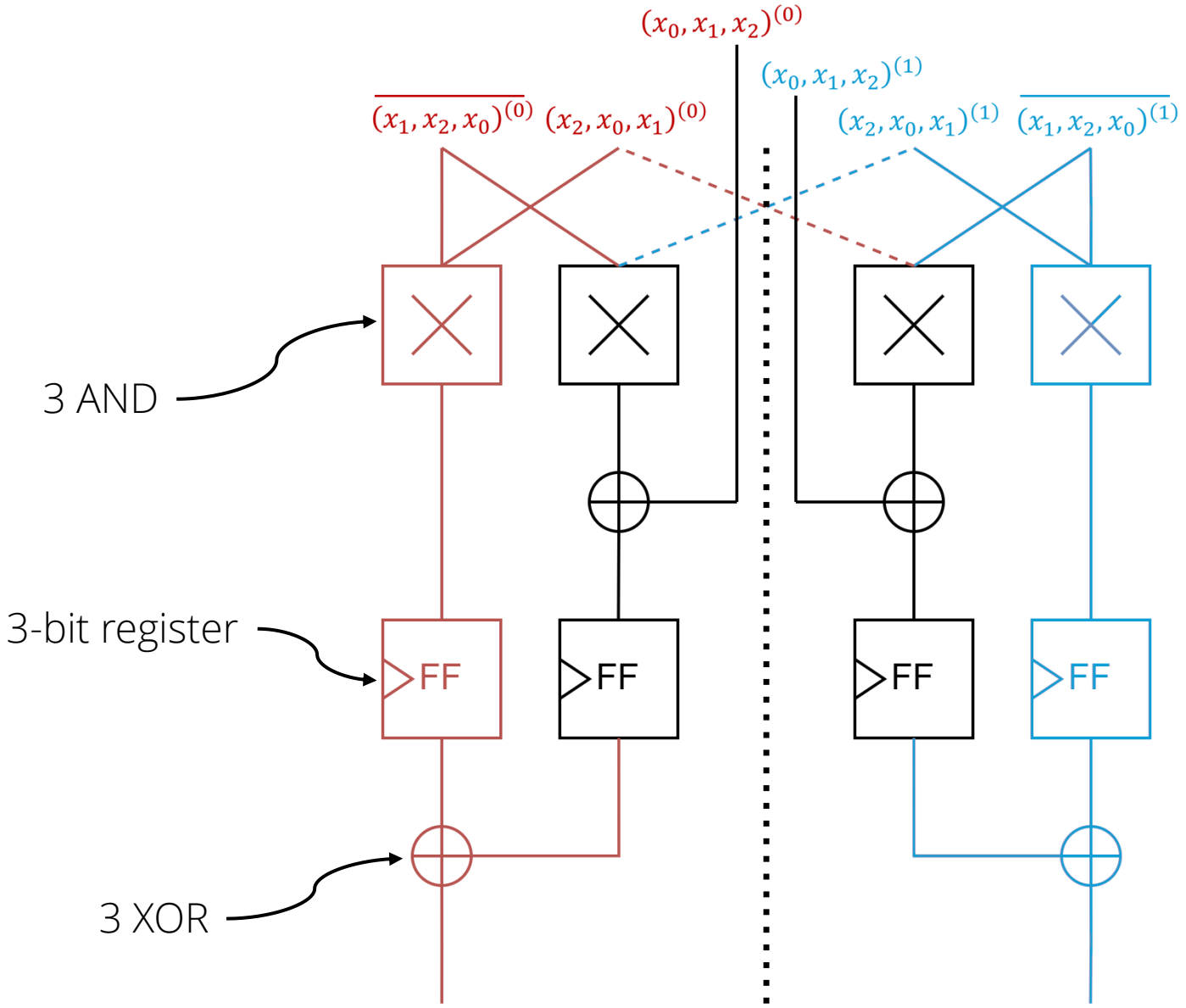
- $x_i \leftarrow x_i \oplus \overline{x_{i+1}} \cdot x_{i+2}$
- $x_i^{(0)}$ and $x_i^{(1)}$ play the role of fresh random Z



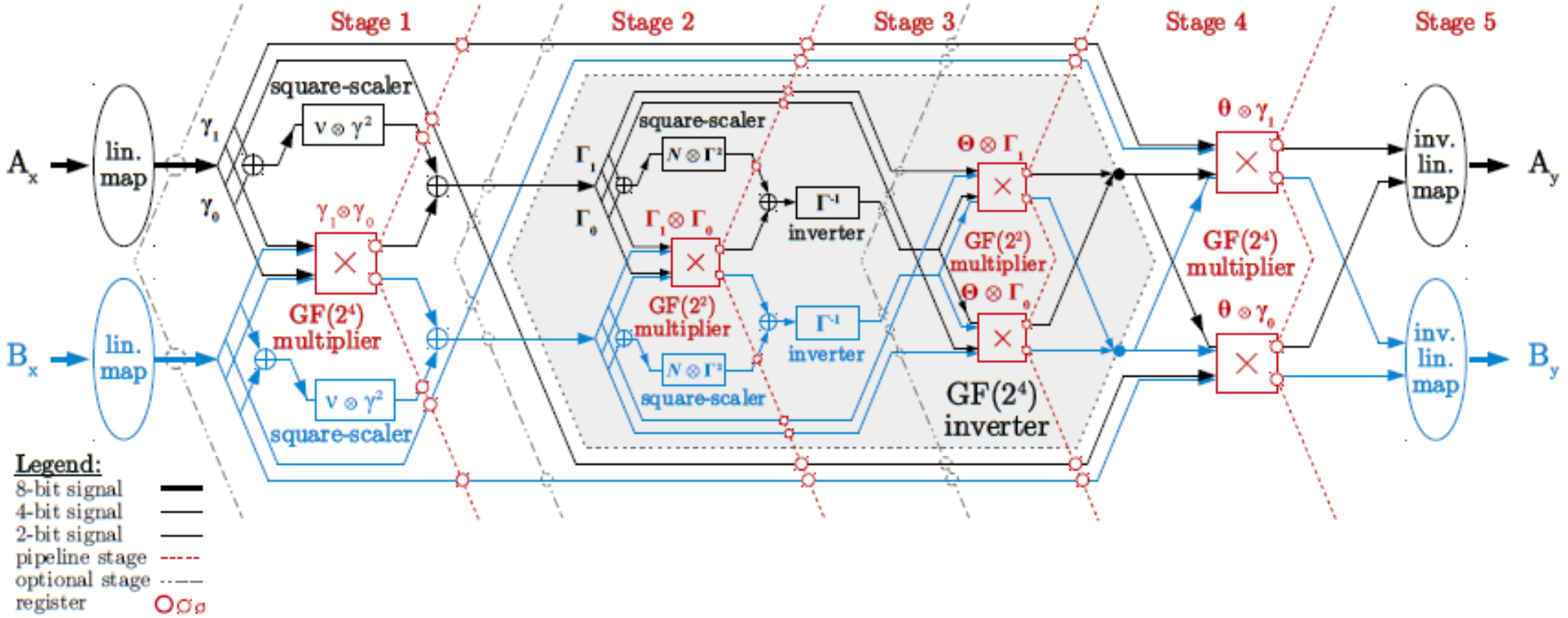
EXAMPLE: χ_3

$\chi_3 : \mathbb{F}_2^3 \rightarrow \mathbb{F}_2^3$

$$x_0 \leftarrow x_0 \oplus \overline{x_1} \cdot x_2$$
$$x_1 \leftarrow x_1 \oplus \overline{x_2} \cdot x_0$$
$$x_2 \leftarrow x_2 \oplus \overline{x_0} \cdot x_1$$



AES S-BOX PROTECTED WITH DOM



Hannes Gross and Stefan Mangard and Thomas Korak. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. ACM Workshop on Theory of Implementation Security 2016

XOODOO PROTECTED WITH DOM

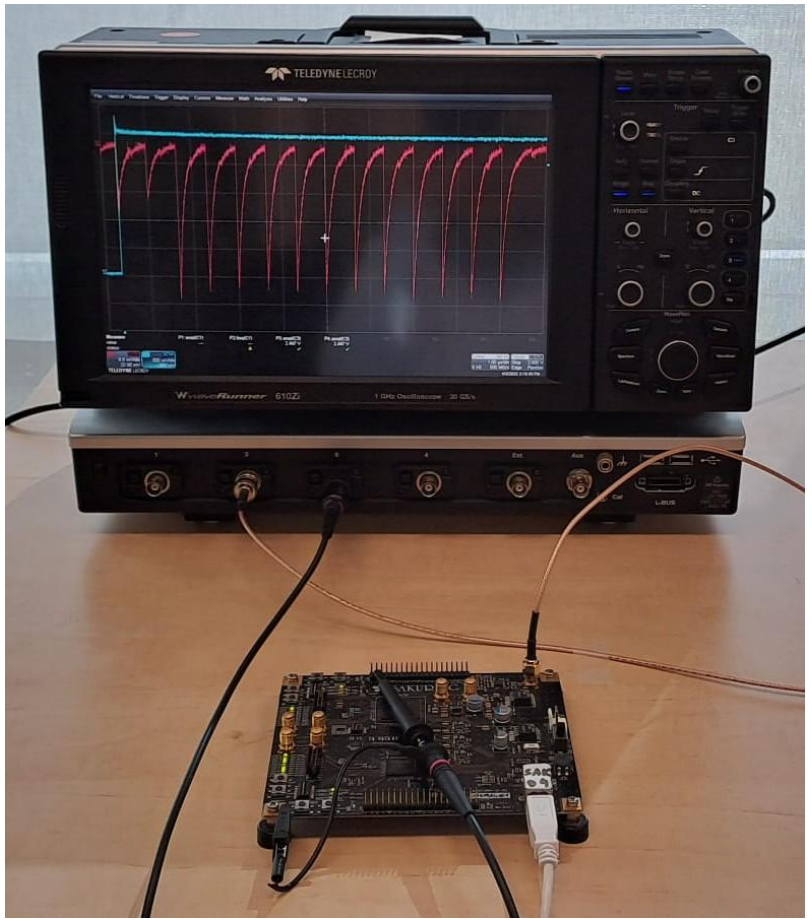
Protection order	Area (μm2)	GE	Ratio
Unprotected	7572.22	9489	1.00
1st	25802.80	32334	3.41
2nd	49237.66	61700	6.50
3rd	79923.16	100154	10.55
4th	118729.36	148784	15.68

Table: ASIC synthesis figures for NanGate 45nm @100MHz

Parisa Amiri Eliasi, Silvia Mella, Léo Weissbart, Lejla Batina, Stjepan Picek. A comprehensive evaluation of side-channel resistance of Xoodyak hardware implementations. Journal of Circuits, Systems, and Computers, 2025.

SCA SETUP

Sakura-G[1] evaluation board equipped with a Xilinx Spartan-6 FPGA, and Lecroy WaveRunner 610Zi oscilloscope



[1] https://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html

TVLA ON UNPROTECTED IMPLEMENTATION

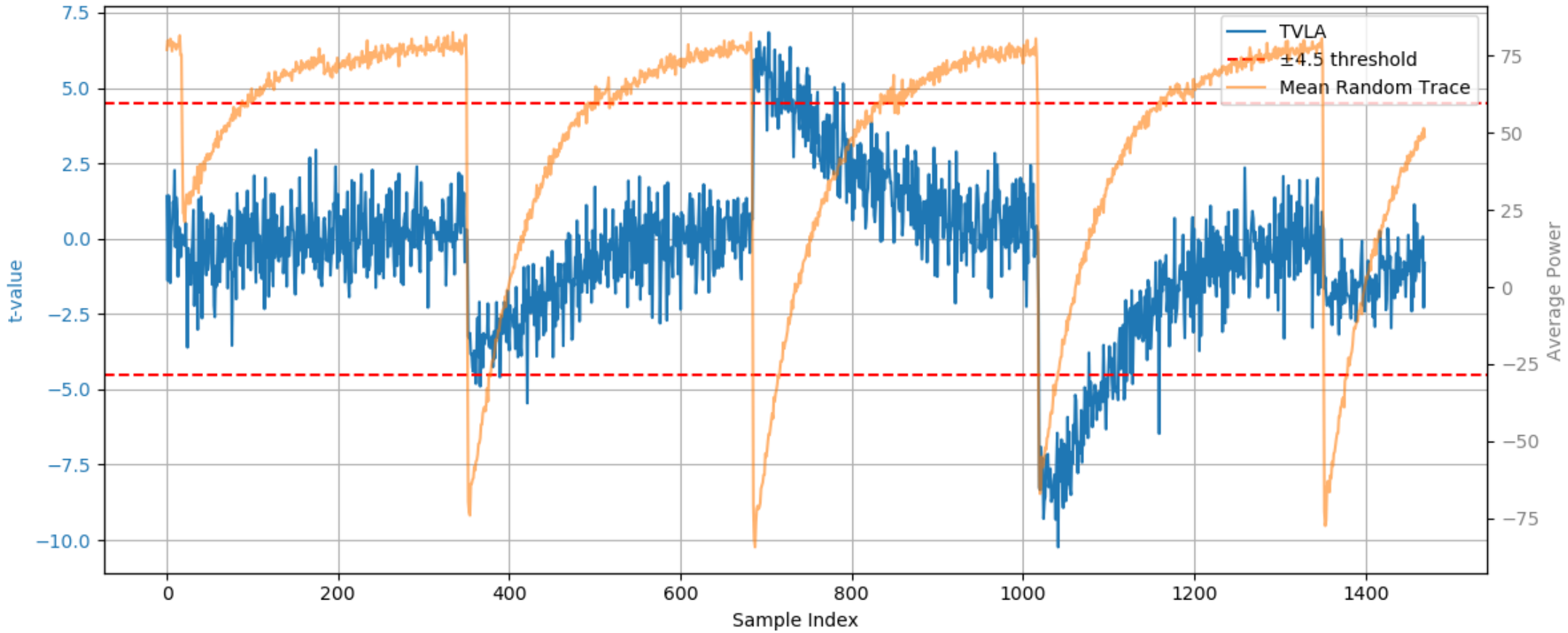


Figure: First-order TVLA on unprotected implementation using 50 power traces

TVLA ON PROTECTED IMPLEMENTATION – 1ST ORDER

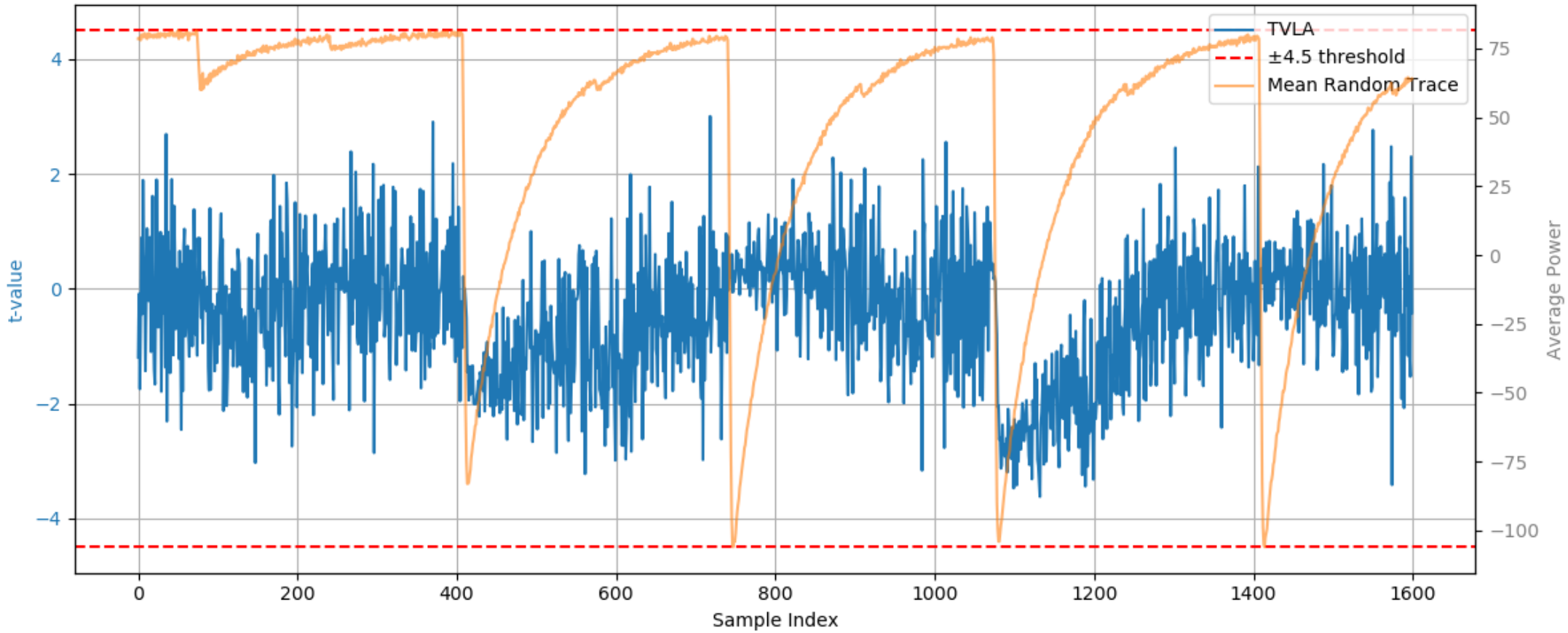


Figure: First-order TVLA on first-order protected implementation using 2.5 million power traces

TVLA ON PROTECTED IMPLEMENTATION – 2ND ORDER

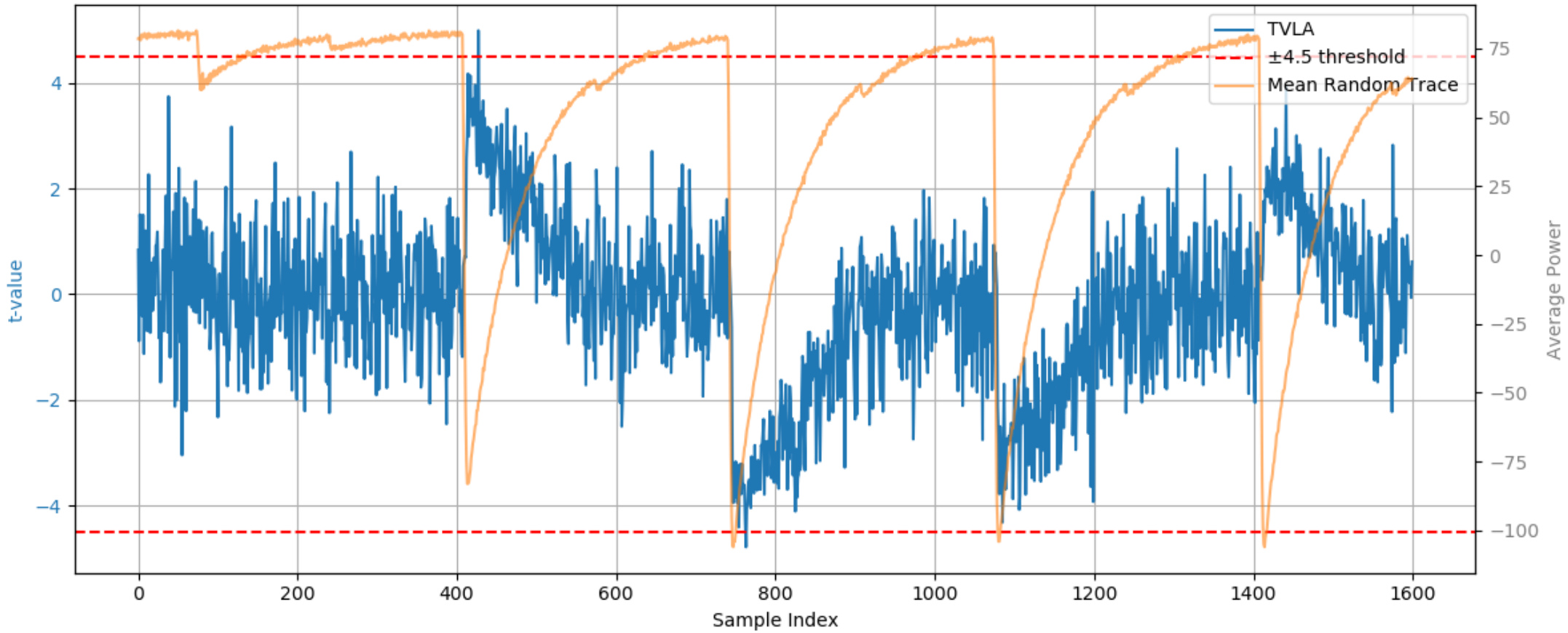
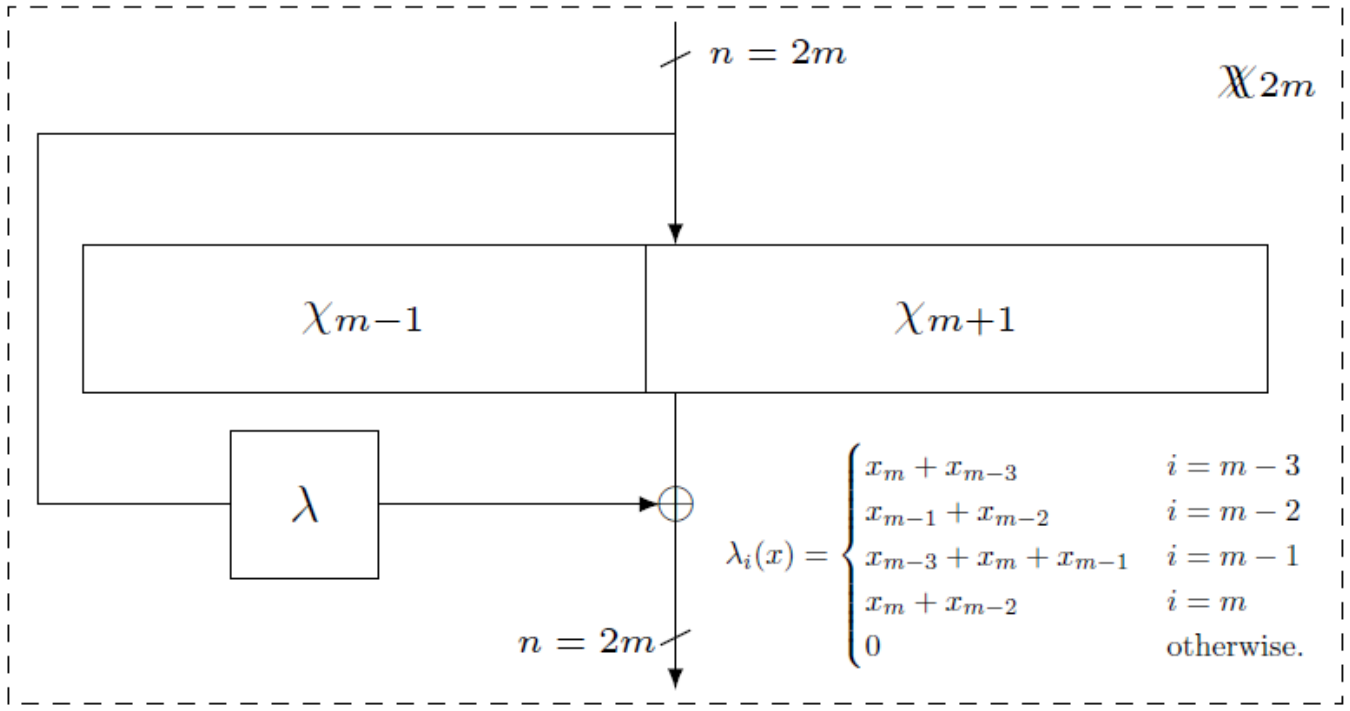


Figure: Second-order TVLA on first-order protected implementation using 2.5 million power traces

$\mathbb{X}$: A GENERALIZATION OF χ_n

- χ_n is a permutation if and only if n is odd
- What can we do when n is even?



Fact

$\mathbb{X}$ is EAE to two parallel χ and inherits its properties

Theorem

For m even, $\mathbb{X}2m$ is a bijection

Y. Belkheyar, P. Derbez, S. Ghosh, G. Leander, S. Mella, L. Perrin, S. Rasoolzadeh, L. Stennes, S. Sun, G. Van Assche, D. Vizár. ChiLow and ChiChi: new constructions for code encryption. EUROCRYPT 2025.

OPEN QUESTIONS

- Can we prove that χ is bijective for any m ?
- Can we generalize it while maintaining or improving the properties of χ ?

THE ORDER OF LAYERS

- Cryptographic permutations usually consist of
  - One non-linear layer (usually S-box layer)
  - Mixing layer
  - Rotations
  - Round constant addition
- In XOODOO: $\rho_{east} \circ \chi \circ \iota \circ \rho_{west} \circ \theta$
  - mixing layer → rotation → const. add → non-linear layer → rotation
- In ASCON-*p*: $p_L \circ p_S \circ p_C$
  - const. add → non-linear layer → mixing layer

Does the order of operations have an impact on SCA ?

THE GASTON* AND GASTON-R PERMUTATIONS

- Same components but different order of layers
  - State like in ASCON-p
  - Mixing layer inspired by KECCAK
  - Non-linear layer is χ_5
- GASTON (original design): mixing layer before non-linear layer
- GASTON-R (modified by us): mixing layer after non-linear layer

Permutation	Area (μm2)	GE	Ratio
ASCON-p	4956.38	6211	1.000
GASTON	5244.72	6572	1.058
GASTON-R	5028.73	6302	1.015

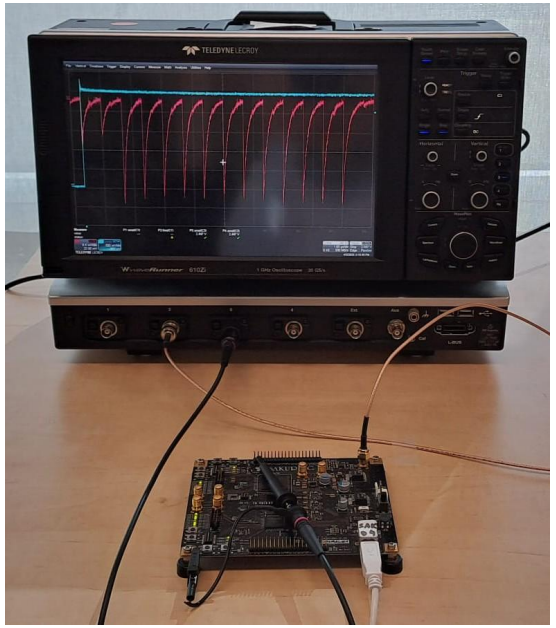
Table: ASIC synthesis figures for NanGate 45nm @1GHz

* Solane El Hirch, Joan Daemen, Raghvendra Rohit, Rusydi H. Makarim. Twin Column Parity Mixers and Gaston - A New Mixing Layer and Permutation. CRYPTO 2023.

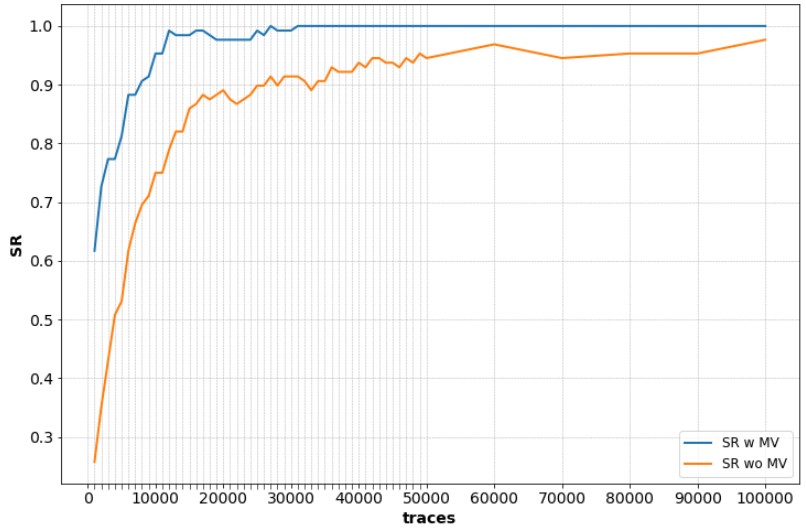
CPA RESULTS*

Permutation	Number of key bits recovered (tot. 128)
ASCON-*p*	100% with 31,000 traces
GASTON	< 90% with 100,000 traces
GASTON-R	50% with 18,000 traces†

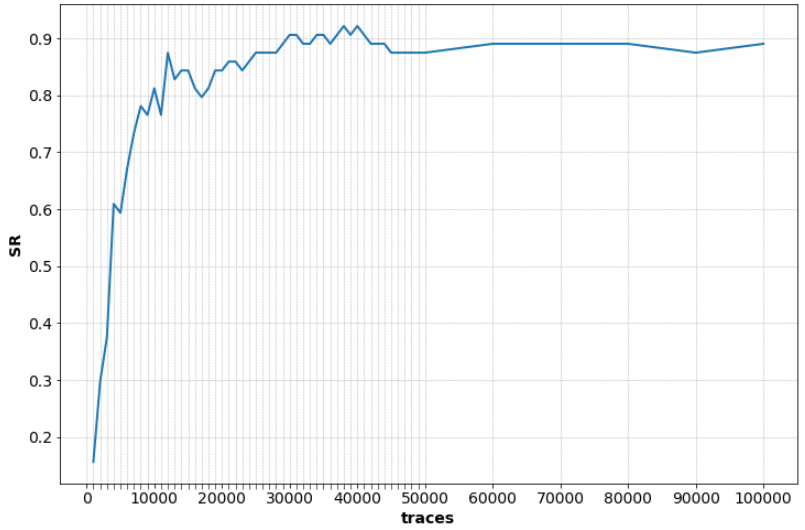
† no non-linear term involving first half of the key and nonce



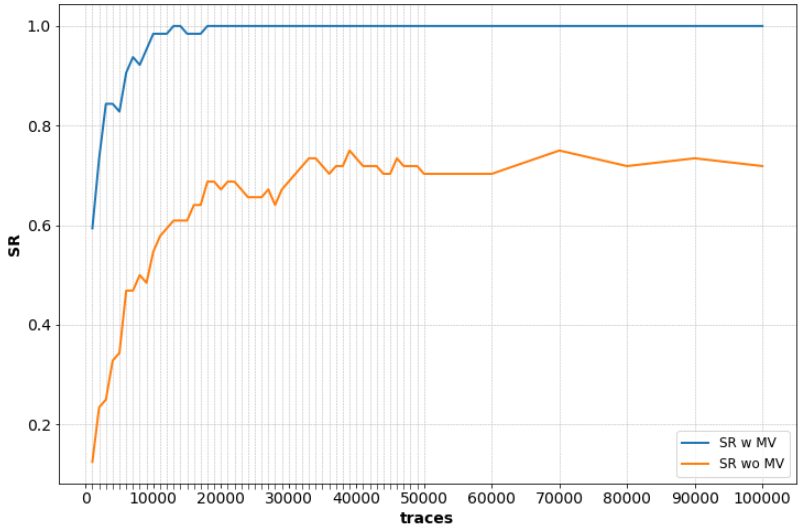
ASCON-*p*



GASTON



GASTON-R

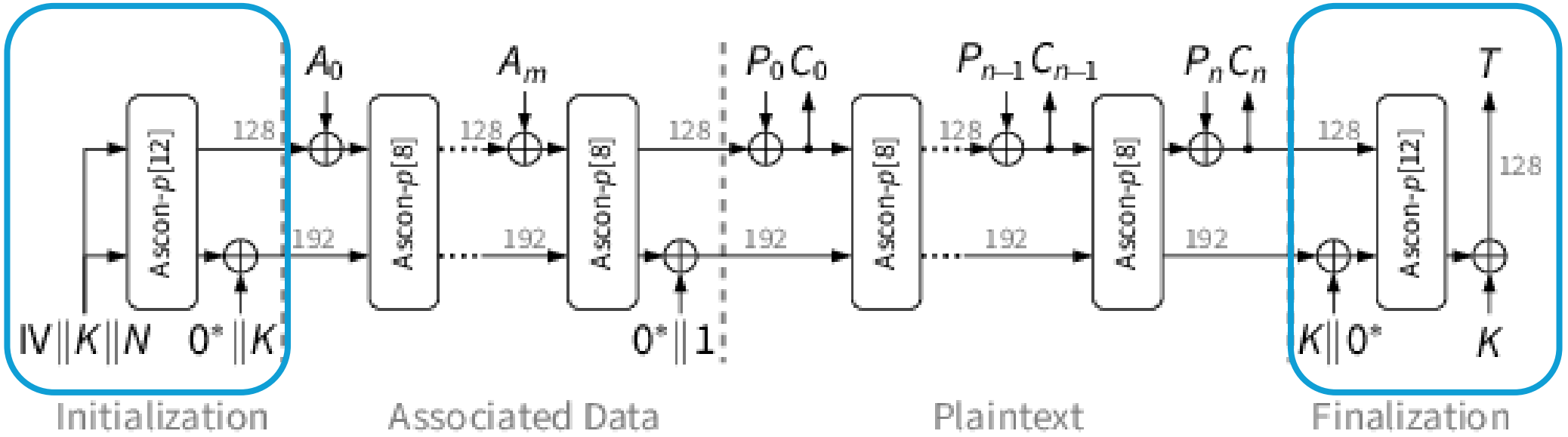


* Under submission

OPEN QUESTIONS

- What is the impact on other permutations/block ciphers?
- What is the impact on other types of SCA (like TA, collision attacks, etc.)?
- Can the Success Rate on Gaston-R be improved by attacking 2nd round?

ASCON MODE



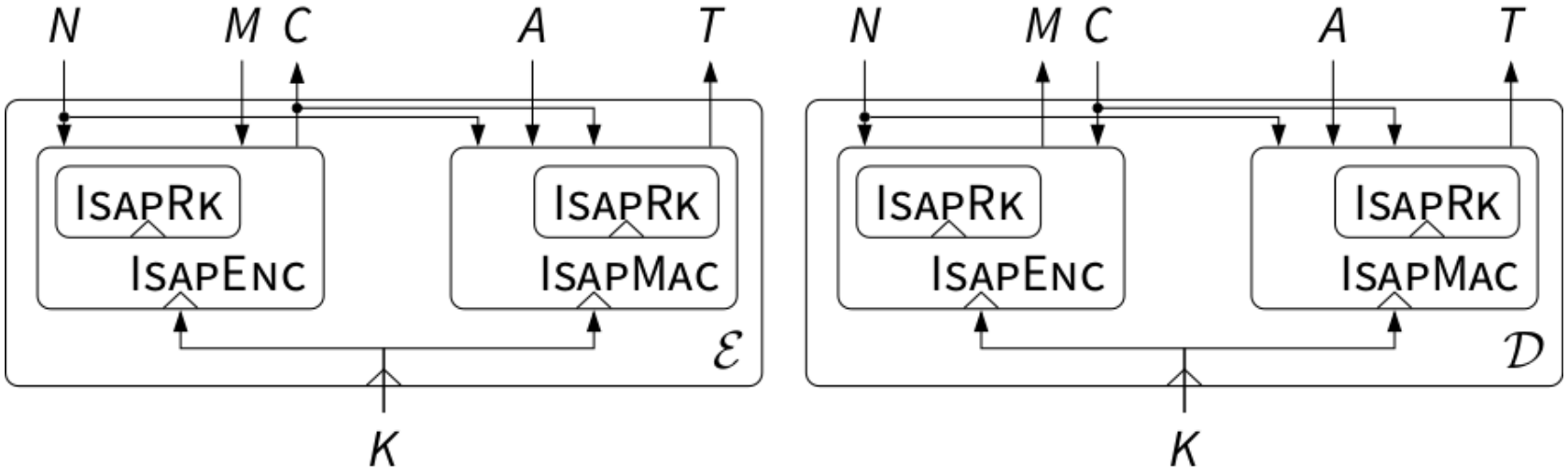
- Recovering the state during AD and P processing does not directly lead to key recovery or forgeries
- This allows more efficient protection against DPA with leveled implementations
  - The degree of algorithmic countermeasures can be reduced for certain parts of a cryptographic computation
  - Data can be processed at higher speed with a lower protection level

NECESSARY CONDITIONS FOR DPA/CPA

- We can collect measurements corresponding to multiple executions where
  - The secret data (usually the key) is fixed
  - The known data (usually the message) changes per execution
- If we can remove these conditions, then DPA/CPA are prevented
- Ideas:
  - Refresh the key at each execution → the target keeps moving
  - Limit the known data that can be exploited

EXAMPLE : ISAP*

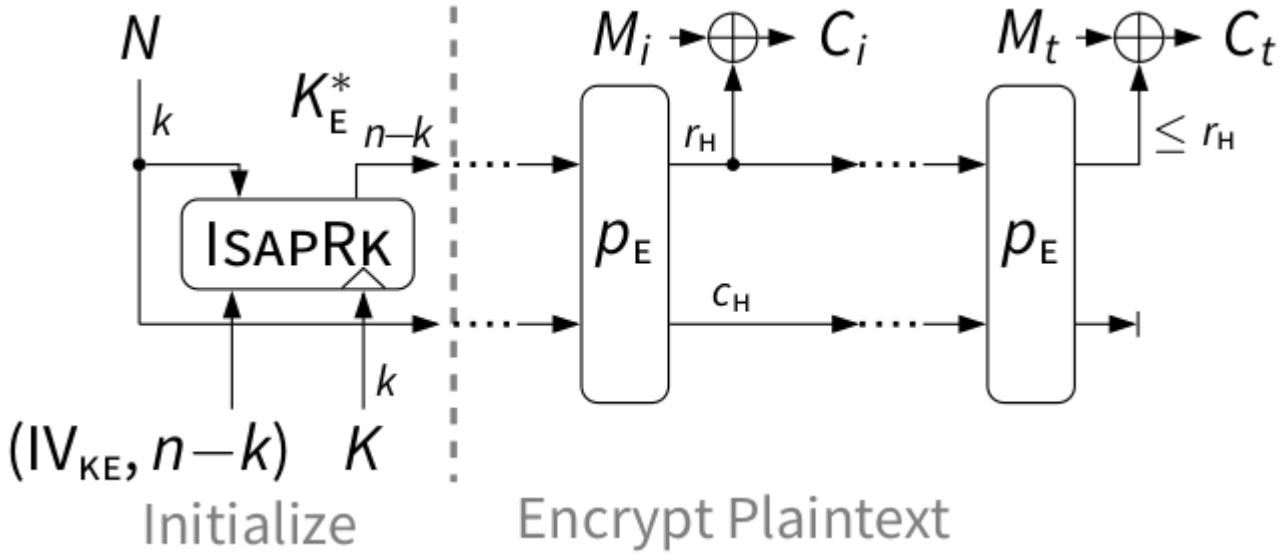
- Submission to the NIST lightweight competition
- Sponge-based mode of operation for authenticated encryption
- Can be instantiated with either Ascon-p or Keccak-p[400] as the underlying permutation
- Encrypt-then-MAC paradigm



encryption $\mathcal{E}$ decryption $\mathcal{D}$

* Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas and Thomas Unterluggauer. ISAP v2.0 Submission to the NIST Lightweight Cryptography competition. https://isap.isec.tugraz.at/publications.html

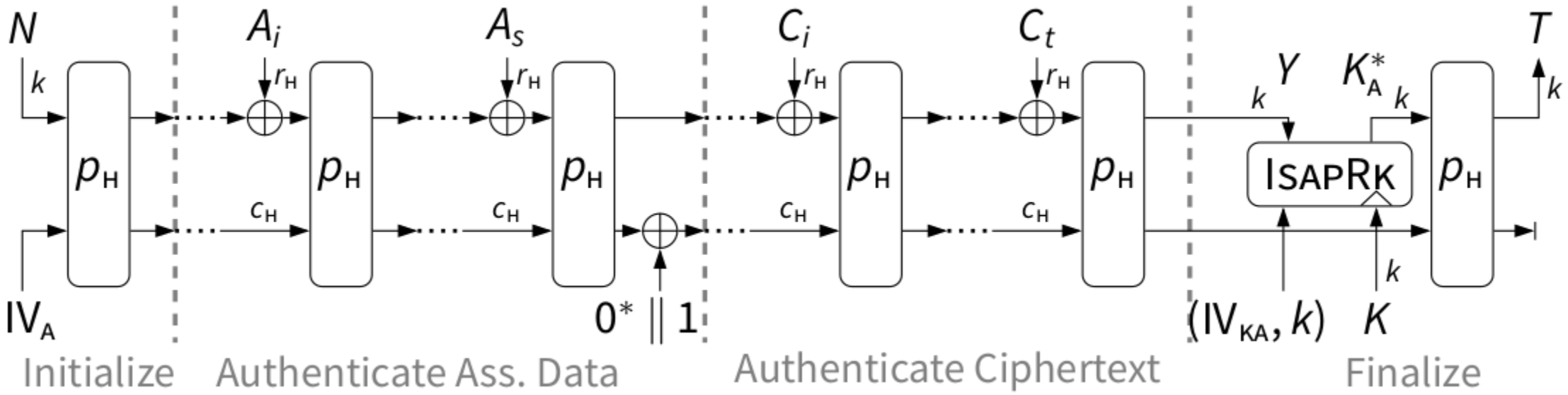
IsapEnc



We cannot collect multiple measurements with the same fixed key

- Stream encryption with keyed sponge
- Encryption Subkey K_E^* generated by IsapRk and different for every N
- Decryption is identical with M and C swapped
  - An adversary could exploit multiple decryptions with the same nonce N
  - To prevent such a DPA scenario, verification is performed prior to decryption
    - If verification fails, decryption does not start

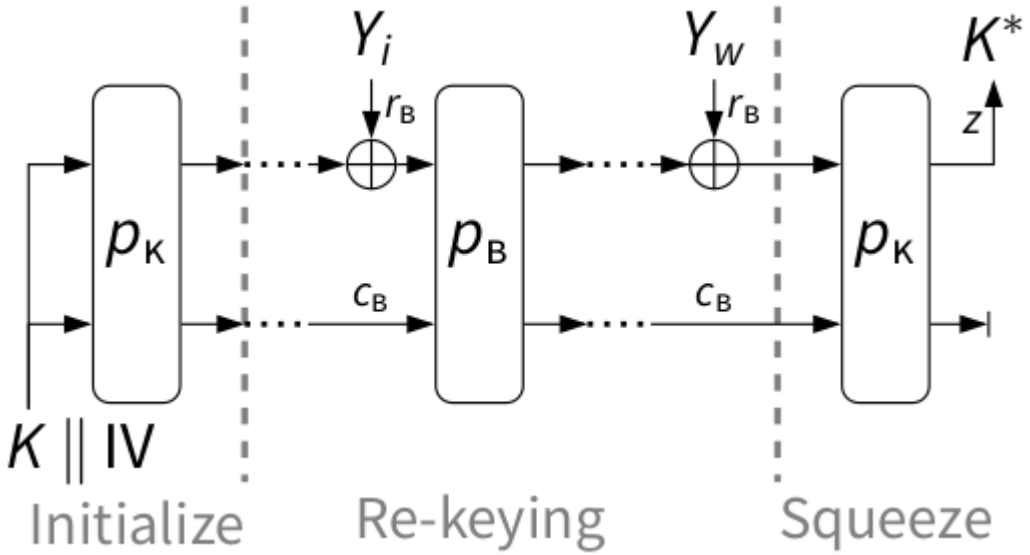
IsapMac



We cannot collect multiple measurements with the same fixed key

- Sponge-based hash function to build a suffix-MAC
- Subkey K_A^* generated by IsapRk and different for every call
- For verification, the tag is re-computed in the same way and compared with the received tag T

IsapRk



We cannot collect enough measurements with different known data

- DPA made infeasible by reducing the input data complexity exploitable
  - If $r_B = 1$, only two possible traces
- Sponge-based equivalent of GGM[1] mode for AES
- Tradeoff: rate highly restricted ↔ reduced number of rounds of p_B

[1] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. Journal of the ACM. doi: 10.1145/6490.6503

SUMMARY

- Side-channel and fault countermeasures often result in bigger area and higher execution time
- Protected implementations may still leak if not correctly crafted!
- Algorithm structure can simplify protections by
  - Using masking-friendly components
  - Combining operations such that the attack complexity increases
  - Using modes that can either reduce the attack surface or prevent them

Thank you for your attention!