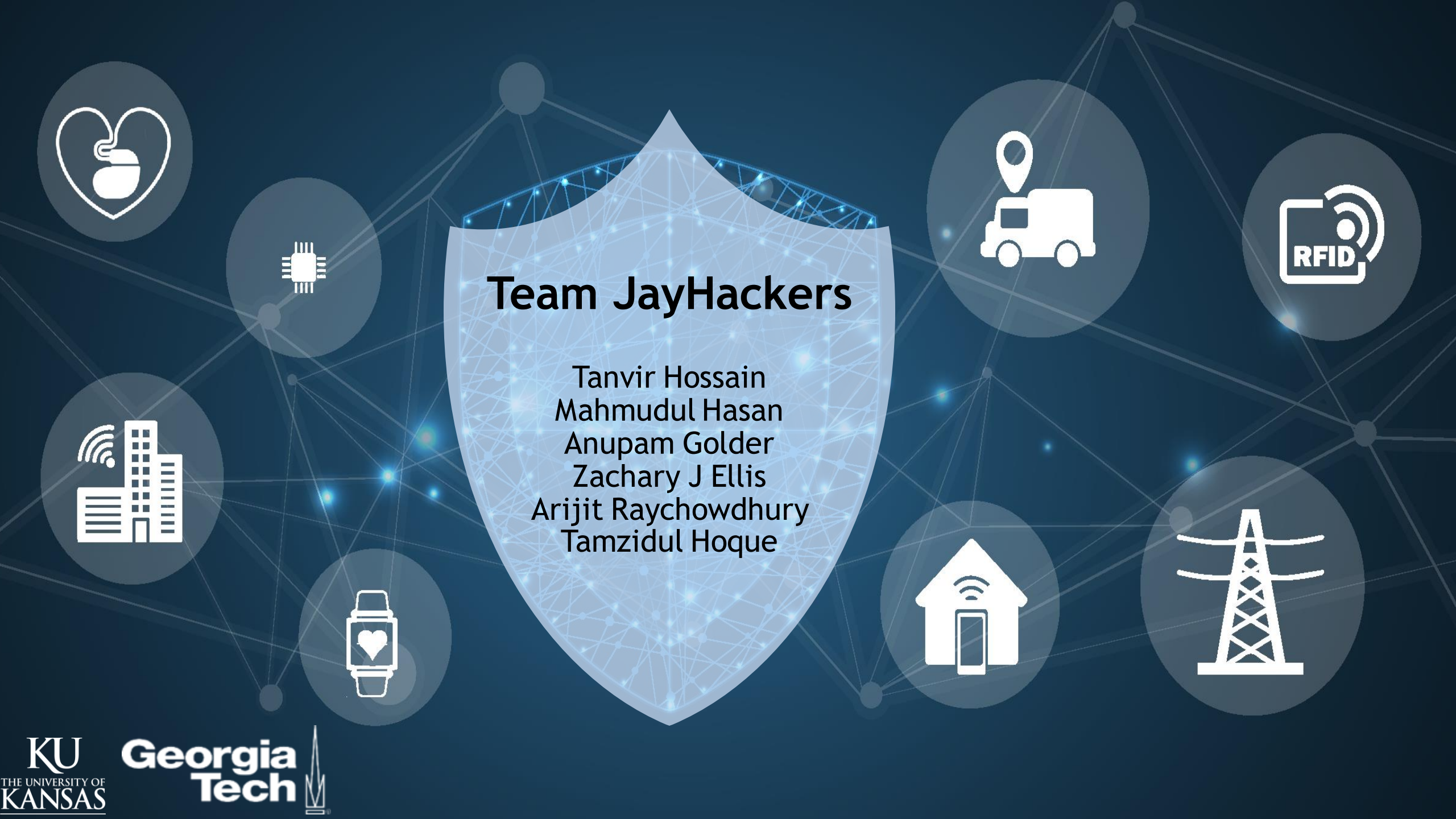


Team JayHackers

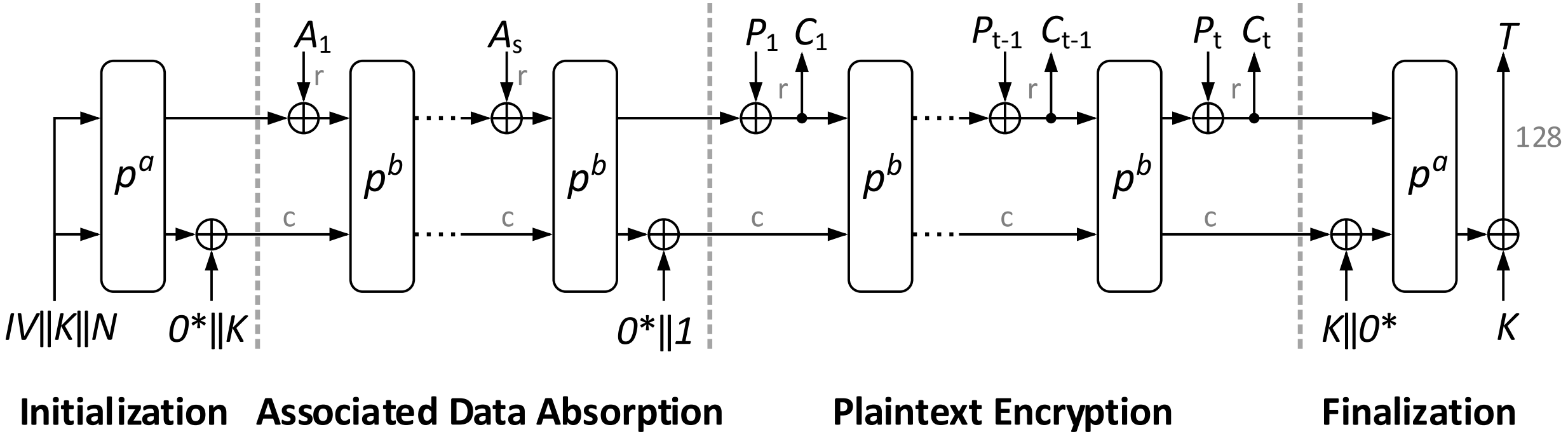
Tanvir Hossain
Mahmudul Hasan
Anupam Golder
Zachary J Ellis
Arijit Raychowdhury
Tamzidul Hoque

KU THE UNIVERSITY OF KANSAS

Georgia Tech

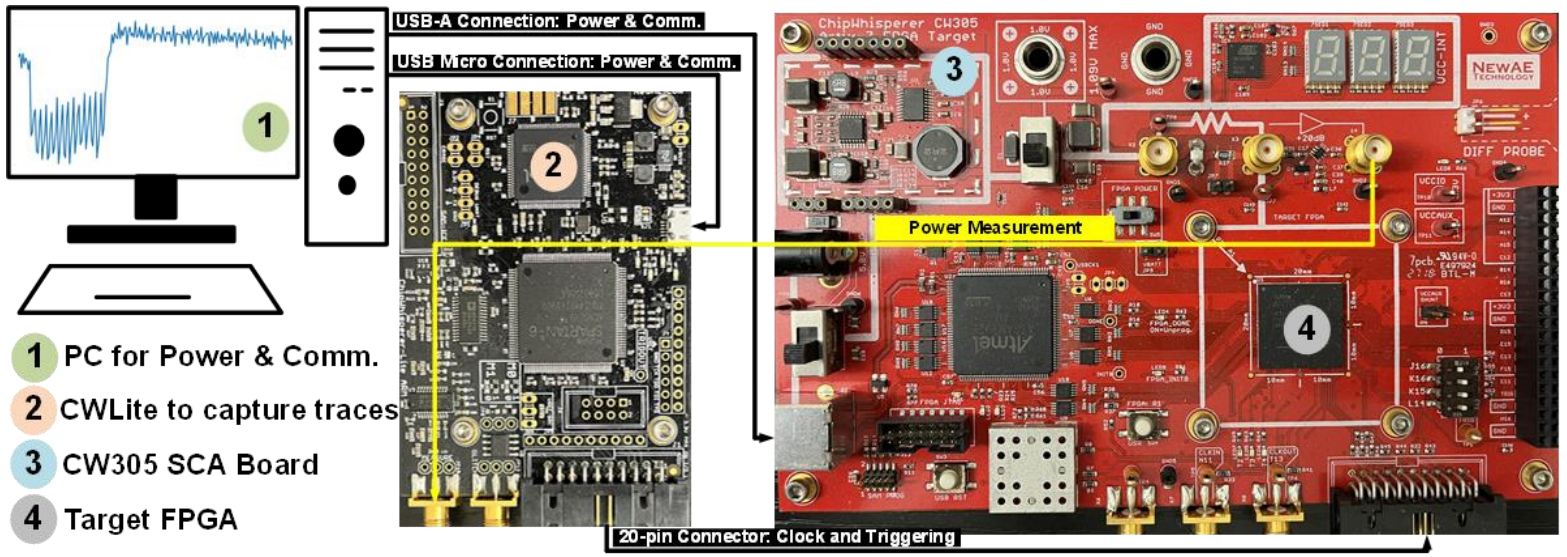
ASCON's Authenticated Encryption with Associated Data (ASCON_AEAD) [ASCON21]

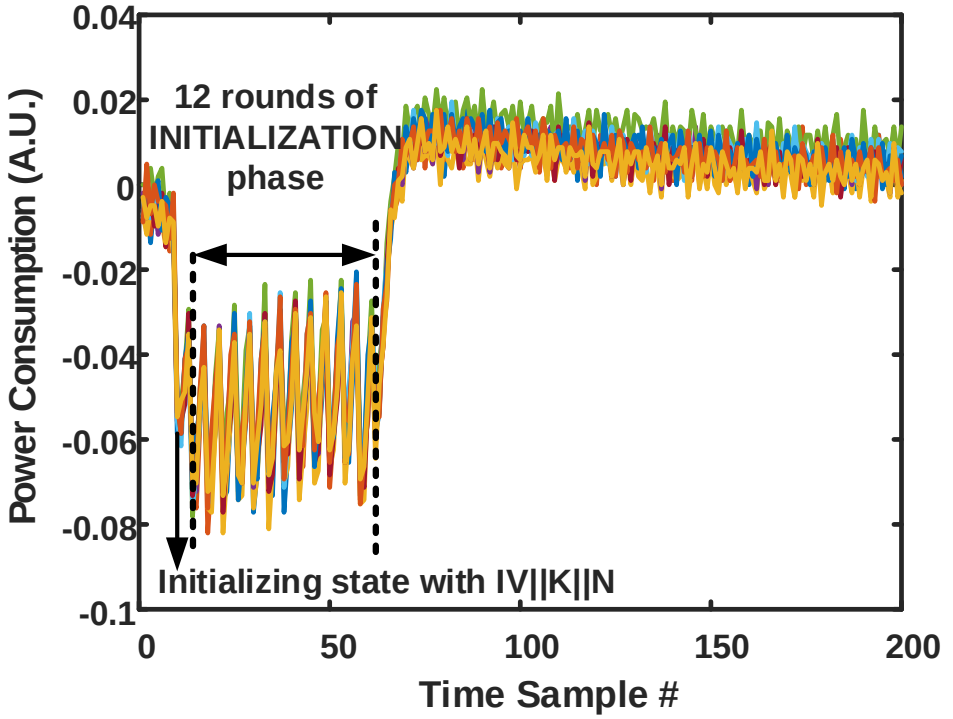
- **4 phases** of **AEAD**:
  - Keyed **INITIALIZATION**
  - Associated Data **ABSORPTION**
  - Plaintext **ENCRYPTION**
  - Keyed **FINALIZATION**



Simple Power Analysis (SPA) of INITIALIZATION Phase of ASCON_AEAD

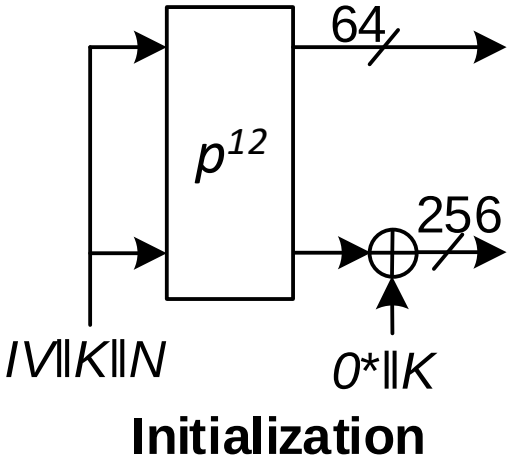
- **Measurement setup:** ChipWhisperer toolchain, AMD/Xilinx **Artix-7** FPGA-based SCA evaluation board, **10MHz** CryptoCore frequency, **40MHz sampling** frequency
- **Visible peaks** during **register updates** → **Cycle count**
- **Parallel execution** on 320-bit state + **Constant-time** implementation → **Visually indistinguishable** data-dependent leakage from single measurement

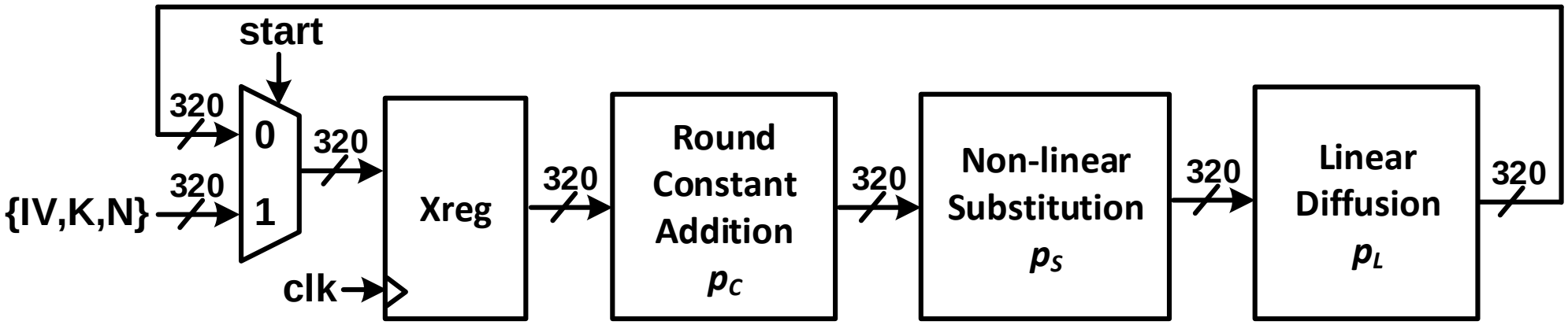




Differential Power Analysis (DPA) Attack on INITIALIZATION Phase of ASCON_AEAD [S16]: Threat Model

- **INITIALIZATION** phase →state initialized with a **64-bit** fixed and known initialization vector IV, a **128-bit secret key** K, and a **128-bit** random but public **nonce** N.
- **Assumption:** non-repeating but **attacker controllable nonce** N (i.e., nonce-respecting scenario).
- **Attack point:** The **first round of permutation** of **INITIALIZATION** phase





DPA Attack on INITIALIZATION Phase of ASCON_AEAD [S16]: Selection Functions

- x_1 and x_2: secret key K register bits
- x_3 and x_4: nonce N register bits
- x_0 : IV register bits

$$(y_0, y_1, y_2, y_3, y_4) = Sbox(x_0, x_1, x_2, x_3, x_4)$$
$$y_0 = x_0 \oplus x_2 \oplus x_1x_2 \oplus x_3 \oplus x_1 \oplus x_0x_1 \oplus x_1x_4$$

- Highlighted in blue → **Constant** register **activity**

$$y_0 = x_3 \oplus x_1x_4$$

- Intermediate value → **2 bits** from two **nonce** registers, **1 bit** from **key** register x_1

$$\sum_{l_0,r_0}(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28)$$
$$S_i(N, K^*) = k_i^*(n_{64+i}) \oplus n_i \oplus k^*_{i-19+64}(n_{64+i-19+64}) \oplus n_{i-19+64} \oplus k^*_{i-28+64}(n_{64+i-28+64}) \oplus n_{i-28+64}$$

- **Effect of diffusion layer → Selection function $S_i(N, K^*)$** to split traces based on ***i*-th** bit of x_1 register update depends on **3 bits** of **key** register x_1 and **6 bits** of **nonce** registers x_3 and x_4.
- **Total hypothesis search space: 8×64** for key register x_1

DPA Attack on INITIALIZATION Phase of ASCON_AEAD [S16]: Selection Functions

- **Remaining hypothesis** search space: 2^{64} for key register x_2
- S-box output with **non-linear term** → only y_1 has a result with a **quadratic term** containing x_2.

$$y_1 = x_1 \oplus x_3 \oplus x_1x_3 \oplus x_2x_3 \oplus x_0 \oplus x_4 \oplus x_2 \oplus x_1x_2$$

- Highlighted in blue → **Constant** register **activity**

$$y_1 = x_3 \oplus x_1x_3 \oplus x_2x_3 \oplus x_4 = x_3(1 \oplus x_1 \oplus x_2) \oplus x_4 = x_3(1 \oplus x_{12}) \oplus x_4$$

- x_1 and x_2 → **indistinguishable** due to XOR operation → **Merged** bit $x_{12} = x_1 \oplus x_2$

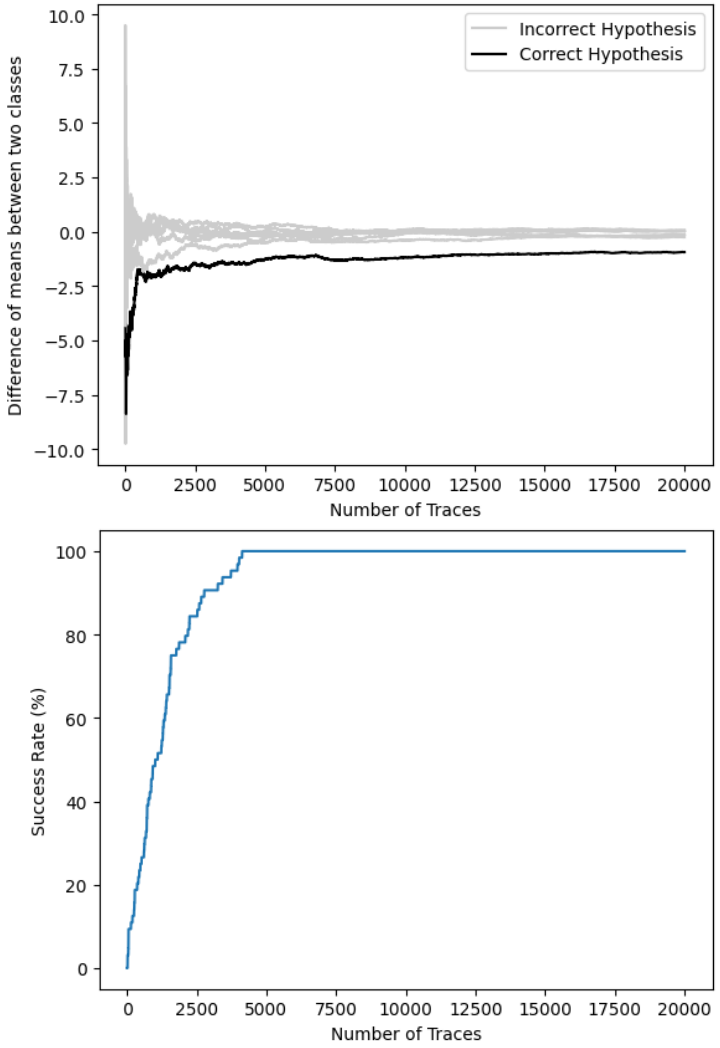
$$\sum_{l_1,r_1}(x_1) = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39)$$

$$S_i(N, K^*)$$
$$= n_i(1 \oplus k_i^*) \oplus n_{64+i} \oplus n_{i-61+64}(1 \oplus k_{i-61+64}) \oplus n_{64+i-61+64} \oplus n_{i-39+64}(1 \oplus k^*_{i-39+64}) \oplus n_{64+i-39+64}$$

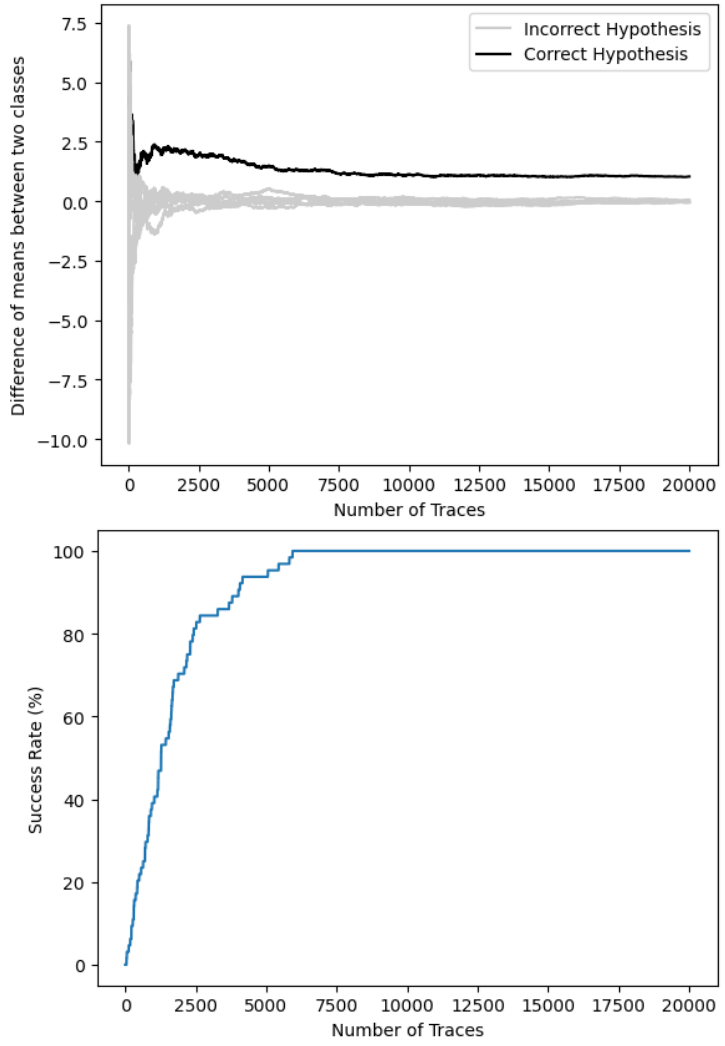
- **Effect of diffusion layer → Selection function $S_i(N, K^*)$** to split traces based on ***i*-th** bit of x_{12} register update depends on **3 bits** of **key** register x_{12} and **6 bits** of **nonce** registers x_3 and x_4.
- **Total hypothesis search space: 8×64** for merged key register x_{12}
- **Recovering x_2 register:** $x_2 = x_{12} \oplus x_1$

DPA Attack on INITIALIZATION Phase of ASCON_AEAD [S16]: Simulation Results

Recovering Key Reg 1

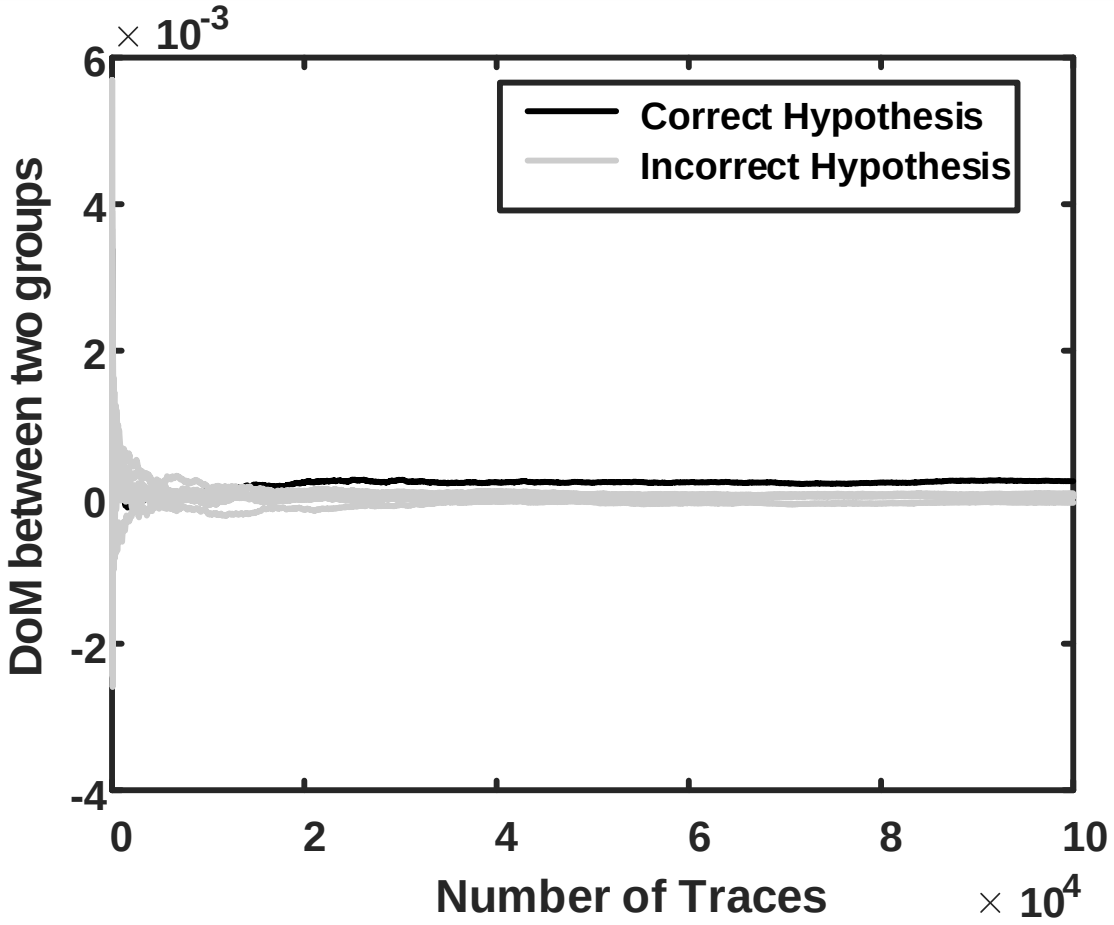


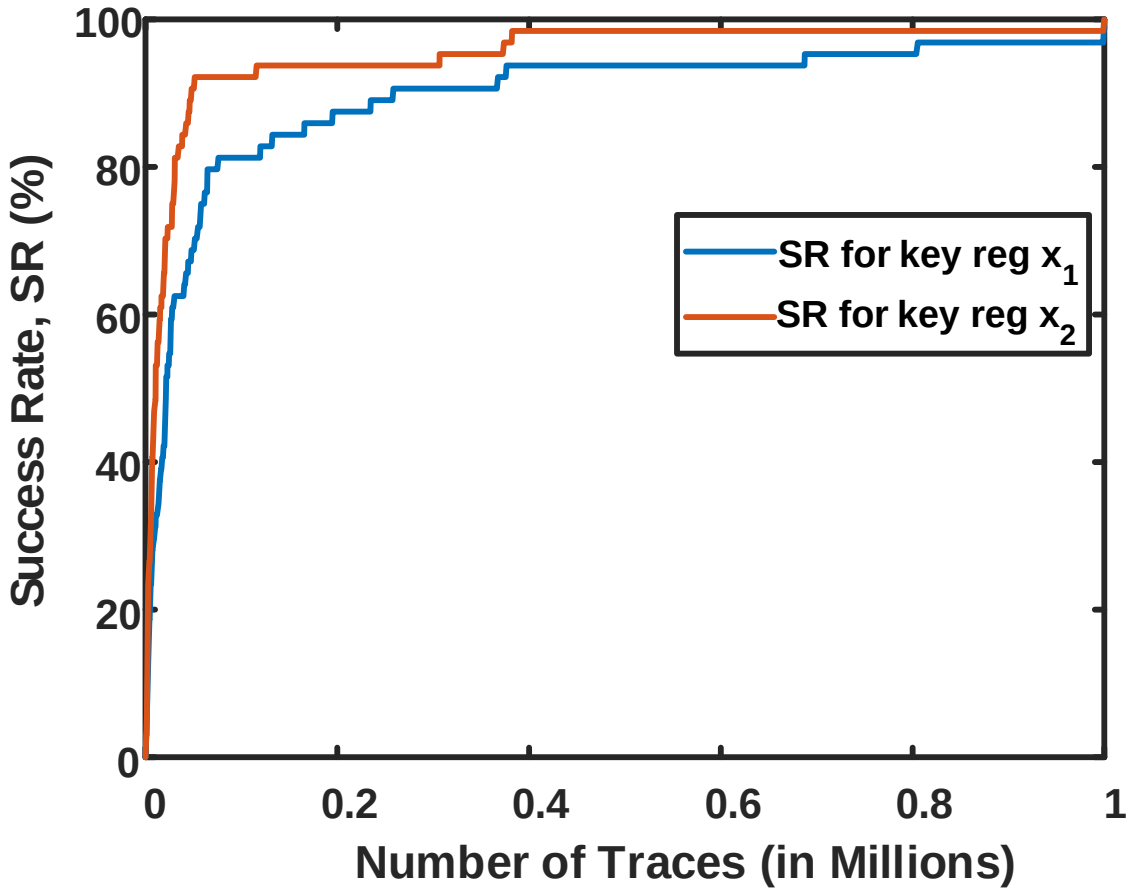
Recovering Key Reg 2



DPA Attack on INITIALIZATION Phase of ASCON_AEAD [S16]: Measured Results

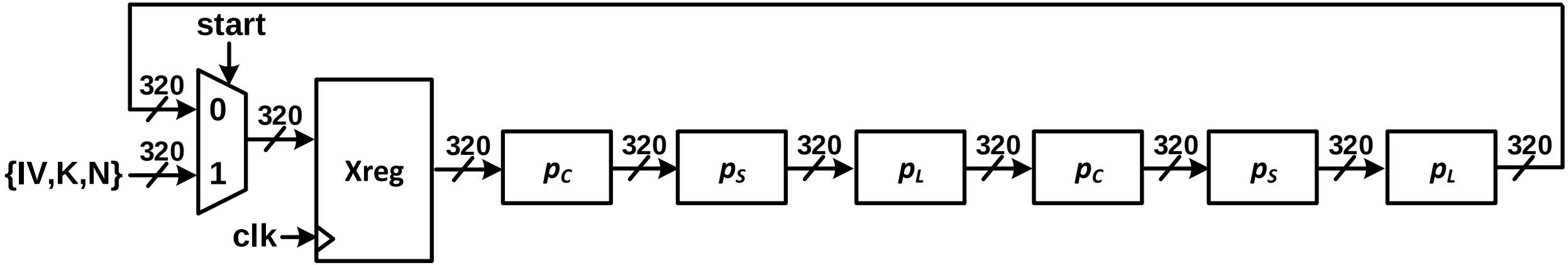
- **Distinguisher:** Difference of Means (**DoM**)
- **Number of Traces: 1M**
- **Success rate (SR):** $> 96\%$.





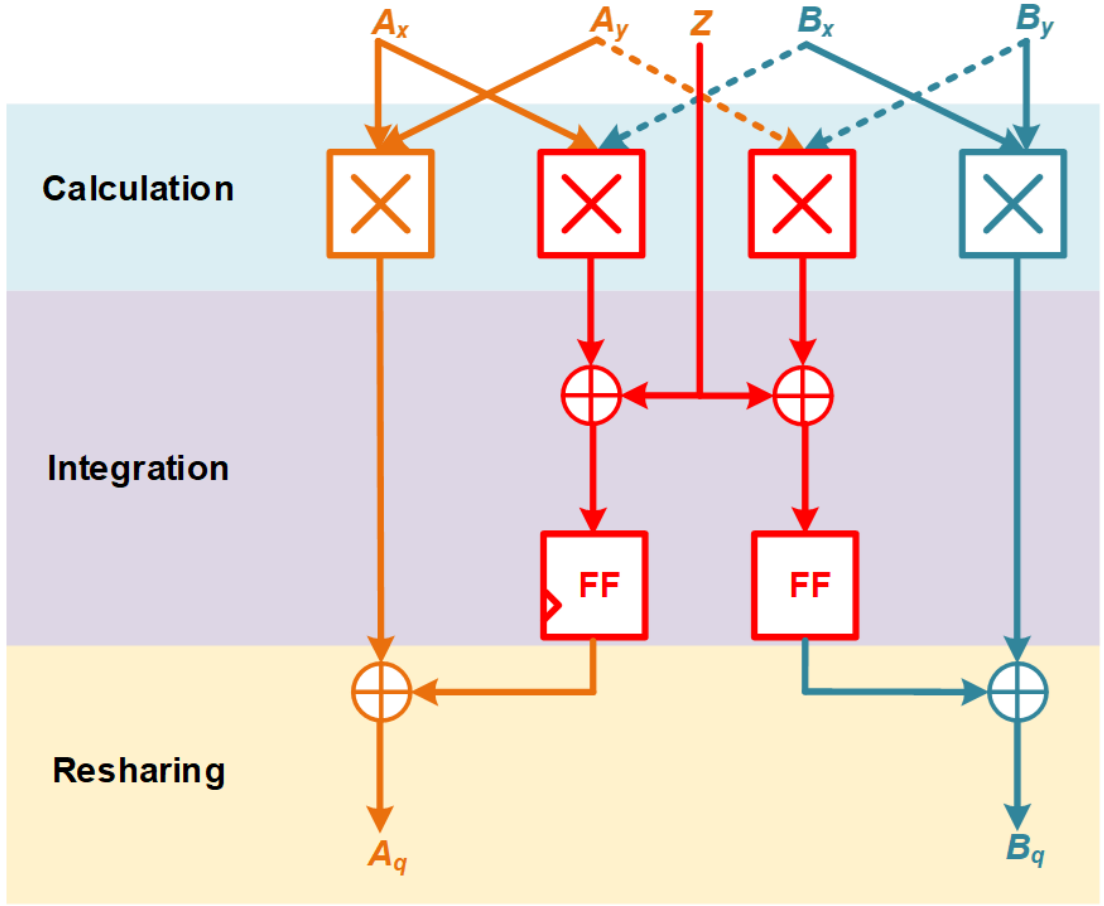
DPA Countermeasure: Round unrolled Implementation [BSL+10,M21]

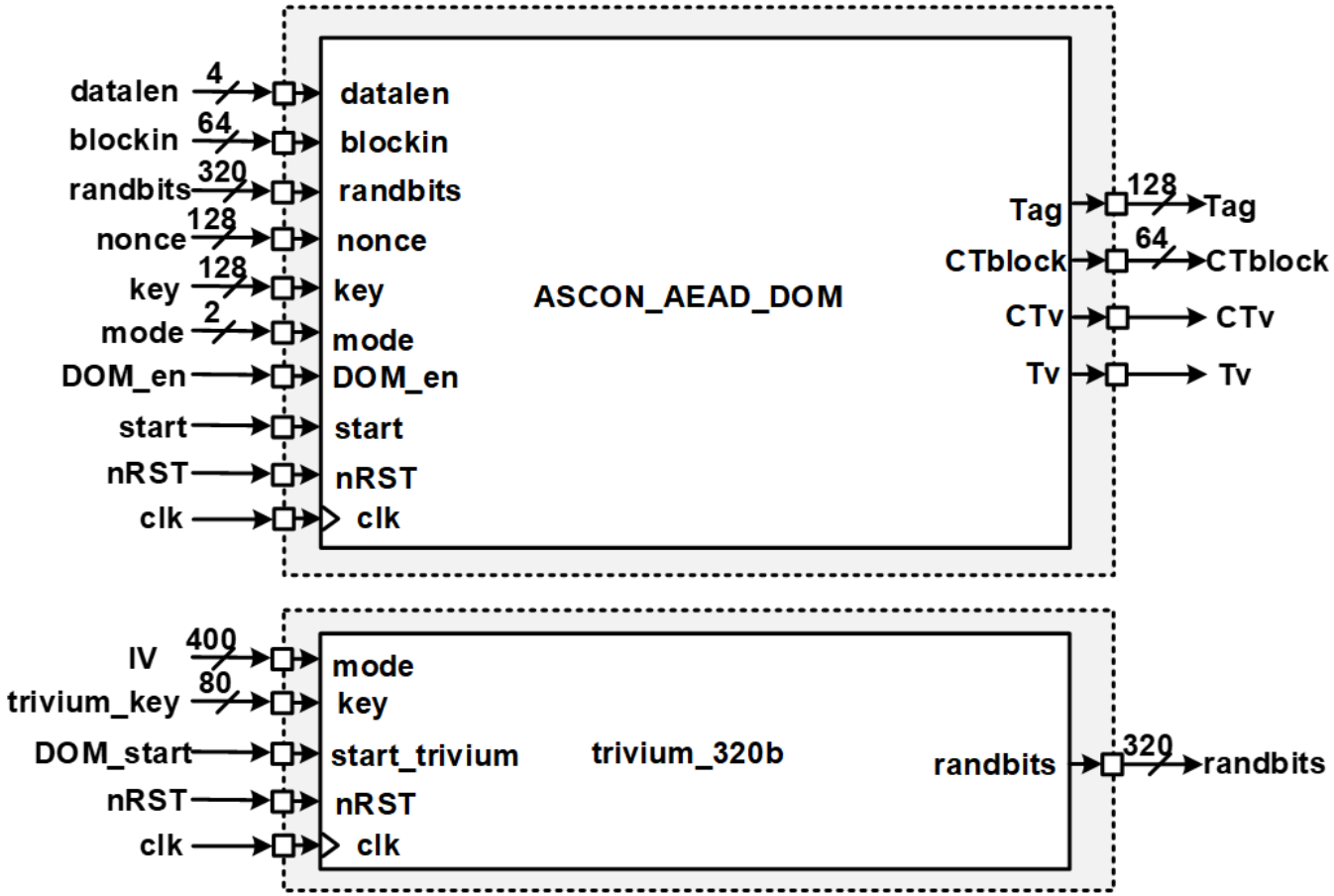
- **Key idea:**
  - Perform **multiple rounds** in each **cycle**
  - **Increase** hypothesis **search space** due to deeper **diffusion of the key**
- **Hardware overhead: Doubling** of Combinational datapath
- **Performance: Increase** in **latency** (due to longer critical path), **Increase** in **throughput** (due to removal of MUXing logic)



DPA Countermeasure: Domain-Oriented Masking (DOM) [GMK16]

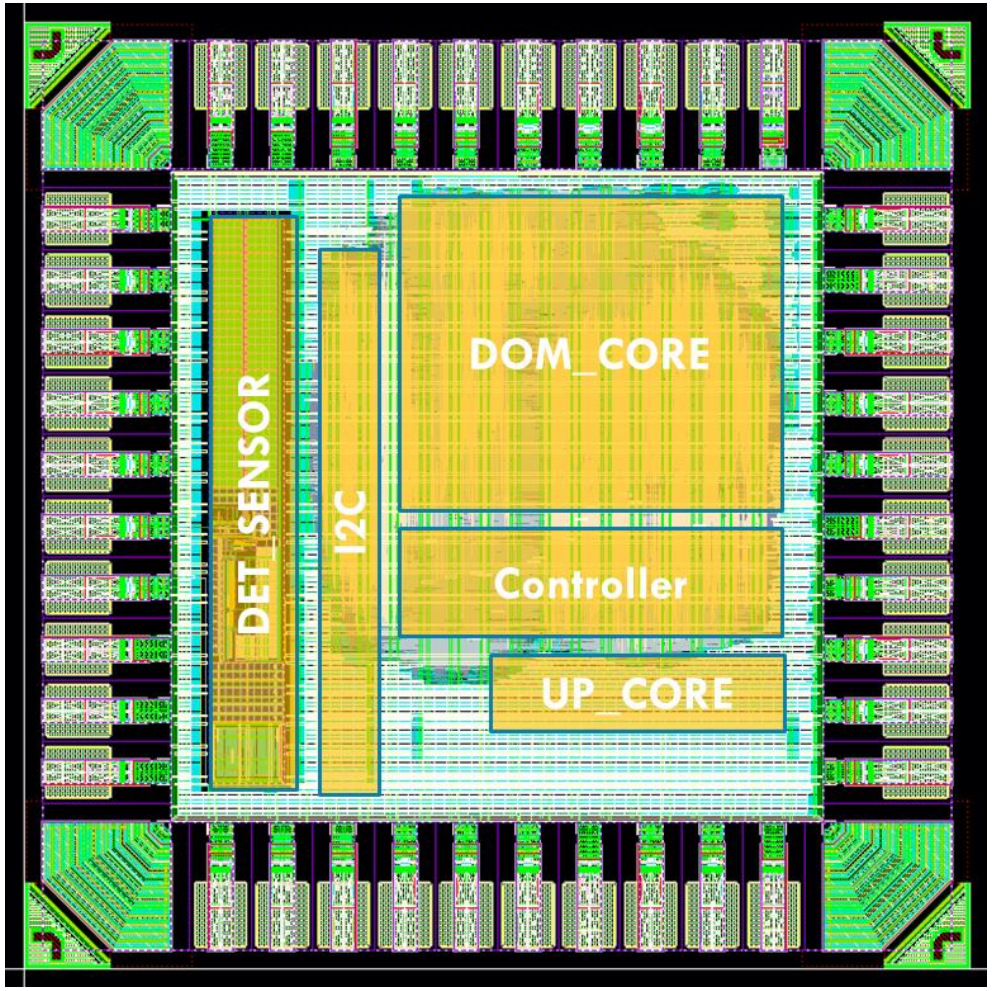
- DOM implementation requires **2 shares** for **protection** against **1st order DPA.**
- DOM requires **internal registers** to avoid **glitching.**
- **1 fresh random bit** is required per **masked AND gate.**



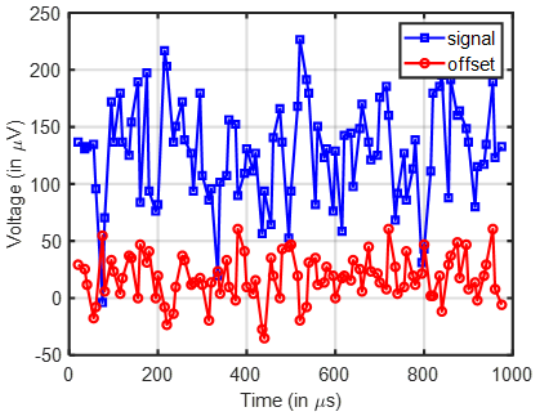


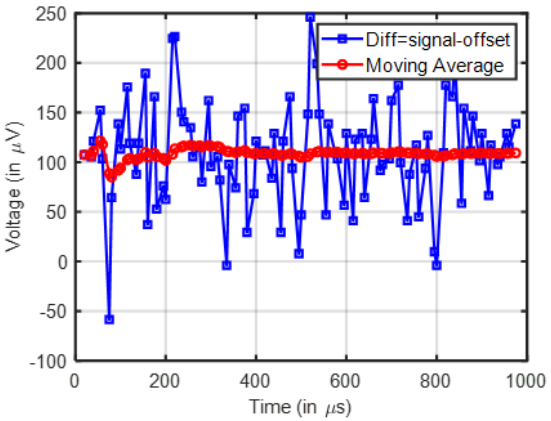
DPA Countermeasure: DOM + Detection Sensor Countermeasure ASIC

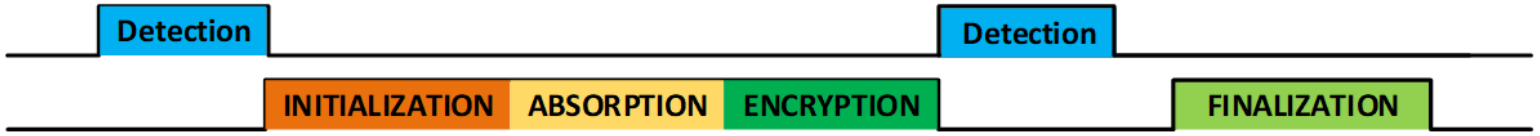
- **Detection Sensor** can detect any **change** in **resistive impedance** in the **power delivery network.**
- **DOM mode** is turned **on** if the sensor **flags an attack** and turns **off** to **save power** if there is **no attack.**



Module	Description	Area ($\mu m \times \mu m$)
UP_CORE	Unprotected ASCON	400×50
DOM_CORE	DOM ASCON	400×300
DET_SENSOR	Detection Sensor	600×80

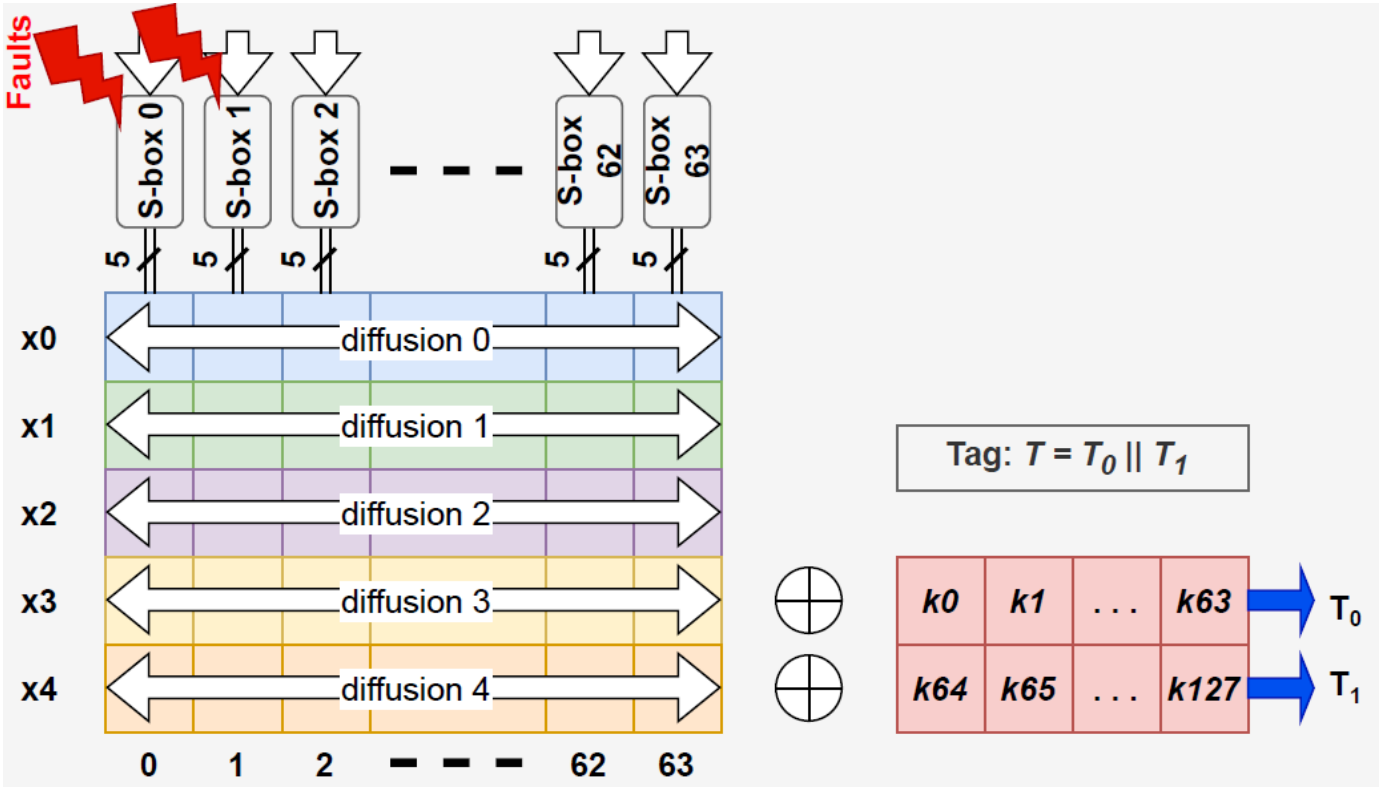






Simple Fault Analysis

- Random **fault injection** is induced subsequent to the S-box operation in the **last round** of the **FINALIZATION** phase.
- A **random** value was **ANDed** with the two selected S-box outputs (**random-AND model**).
- Due to the fault injection, the generated tags may or may not change (ineffective fault) for the same key and nonce value.



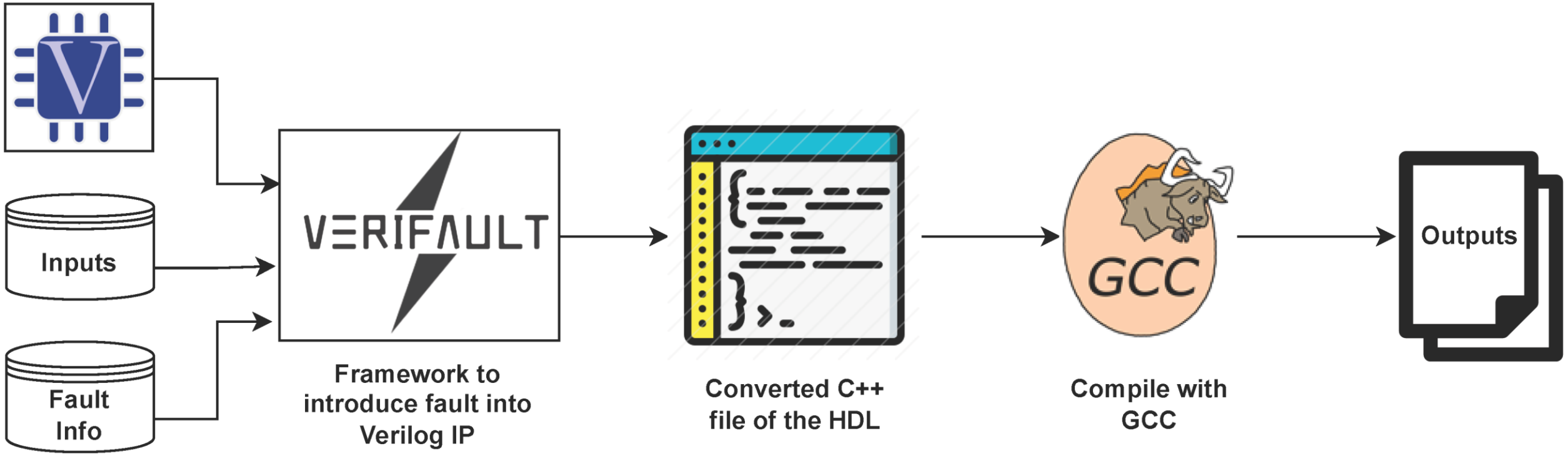
Fault Generation Method

- **Traditional Method**
  - Overclocking
  - Power supply manipulation
  - Thermal stress
  - Mechanical stress
  - Electromagnetic interference

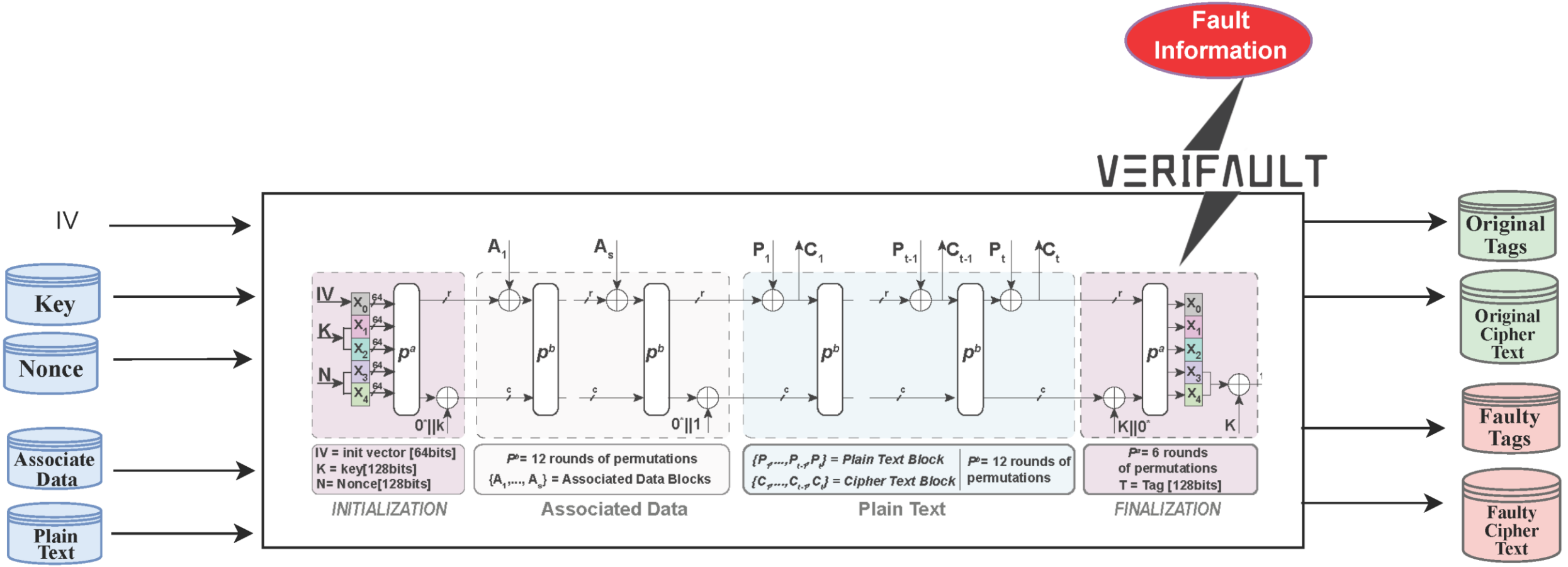
- **Problems with Traditional Method**
  - Exhaustive and need controlled conditions
  - Possibility of damaging hardware
  - Hinder the analysis which is important
  - Need unique implementation for each variant

- **Simulation-Based Method**
  - Software-based solution that directly simulates faults using Verilog hardware designs
  - Easy to simulate for any given condition with expected outcome
  - A framework is created for inducing fault in ASCON at any stage
  - Verilator environment seems most suitable as it can declare internal nets or registers as public
  - It can assign any operations on public parameter, e.g., flipping a bit randomly or in a deterministic fashion and propagate that fault in next stages
  - Further, developed countermeasure in RTL level is tested by applying random faults

Proposed Differential Fault Analysis (DFA) simulator based on open source verilator tool.



Fault Attack on ASCON using VeriFault

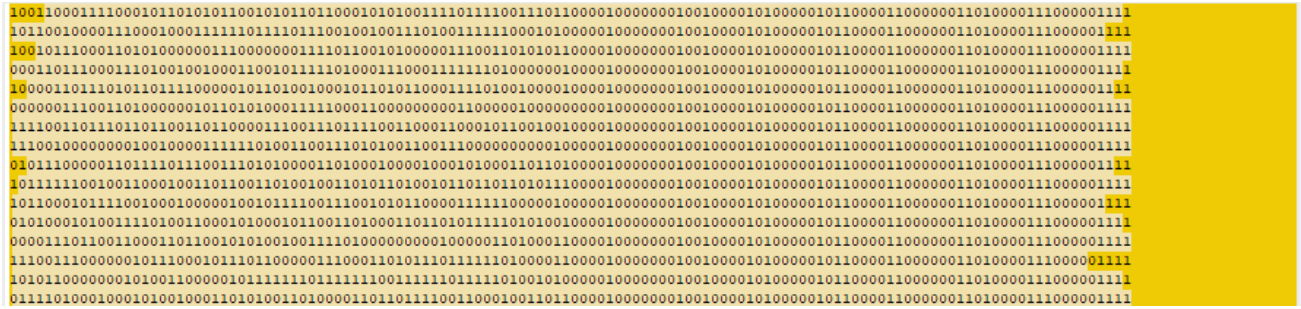


Original and Faulty Tag

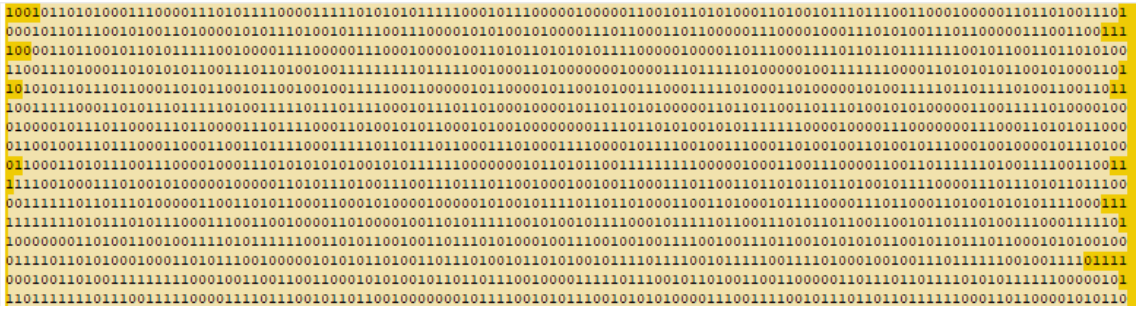
- If the attacker induces **stuck-at-zero (bit-reset) fault** at the **input of an S-box**, the output of the S-box is 00100.
- As in the **diffusion layer**, 0th bit of x_3 depends on columns 0, 54, and 47.
- Setting inputs to this S-box triple to 0s ensures that the 0th bit of x_1 being XORed with the key is 0.
- This means the 0th **tag bit** is now **equal** to the 0th **key bit.**
- Similarly, the rest of the bits can be recovered. For the 0th bit x_4, the corresponding columns will be 0, 57, and 23.

Fault Simulation

Faulty Tags



Fault Free Tags



DFA Countermeasure: Error Correction Coding based Parity Signature Generation [JMR22]

- **Key idea:**
  - Generate **Interleaved, Even,** and **Odd Parity Signatures**
  - Match the **parity signatures** with the **corresponding S-box** input.
- **Hardware overhead: Lowest**
- **Performance overhead: Highly Accurate** in terms of **error detection.**

Logic Implementation of ASCON Sbox

Output	Expression
y_0	$x_0 \oplus x_4 \oplus x_2 \oplus x_1x_2 \oplus x_3 \oplus x_4 \oplus x_1 \oplus x_0x_1 \oplus x_4x_1$
y_1	$x_1 \oplus x_4 \oplus x_1x_3 \oplus x_2x_3 \oplus x_0 \oplus x_4 \oplus x_2 \oplus x_1x_2$
y_2	$1 \oplus x_2 \oplus x_1 \oplus x_4 \oplus x_3x_4$
y_3	$x_3 \oplus x_0 \oplus x_4 \oplus x_0x_3 \oplus x_0x_4 \oplus x_2 \oplus x_1$
y_4	$x_3 \oplus x_4 \oplus x_1 \oplus x_0x_1 \oplus x_4x_1$

Logic Implementation of Parity Signatures

$$p_0 = y_0 \oplus y_1 \oplus y_2 \oplus y_3 \oplus y_4$$
$$p_1 = y_0 \oplus y_2 \oplus y_4$$
$$p_2 = y_1 \oplus y_3$$

TABLE I
LUT BASED REPRESENTATION OF NON-LINEAR SBOX OF ASCON AND THE CORRESPONDING INTERLEAVED(p_0), EVEN(p_1) AND ODD(p_2) PARITY SIGNATURES OF THE ECC COUNTERMEASURE.

Sbox Input	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	10	11	12	13	14	15	16	17	18	19	1a	1b	1c	1d	1e	1f
Sbox Output	4	b	1f	14	1a	15	9	2	1b	5	8	12	1d	3	6	1c	1e	13	7	e	0	d	11	18	10	c	1	19	16	a	f	17
p_0	1	1	1	0	1	1	0	1	0	0	1	0	0	0	0	1	0	1	1	1	0	1	0	0	1	0	1	1	1	0	0	0
p_1	1	1	1	0	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0	1	1	1	1	0	0	0	0	1
p_2	0	0	0	0	0	0	1	1	0	0	1	1	1	1	1	1	0	1	1	0	0	1	0	1	0	1	0	1	1	0	0	1

References

[W23]	Witteman M., "Security Highlight: Ascon", 2023, URL: https://www.riscure.com/security-highlight-ascon/
[NIST23]	NIST LWC selection announcement, 2023, URL: https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon, https://www.nist.gov/news-events/news/2023/02/nist-selects-lightweight-cryptography-algorithms-protect-small-devices
[NIST18]	NIST Call for LWC, 2018, URL: https://www.nist.gov/news-events/news/2018/04/nist-issues-first-call-lightweight-cryptography-protect-small-electronics
[Rambus23]	ASCON IP from Rambus, 2023, URL: https://www.rambus.com/blogs/rambus-ip-solution-supports-new-nist-lightweight-cryptography-algorithm/
[HN23]	HackerNews, 2023, URL: https://thehackernews.com/2023/02/nist-standardizes-ascon-cryptographic.html
[ASCON21]	ASCON final specification, URL: https://ascon.iaik.tugraz.at/specification.html , https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf
[B19]	Buchanan B., "One of the greatest advancements in Cybersecurity: The Sponge Function (Keccak, SHAKE and ASCON)", 2019, URL: https://medium.com/asecuritysite-when-bob-met-alice/one-of-the-greatest-advancements-in-cybersecurity-the-sponge-function-keccak-and-shake-6e6c8e298682
[Keccak]	URL: https://keccak.team/
[T22]	Tezcan C., "ASCON Authenticated Encryption Scheme", 2022, URL: https://www.youtube.com/watch?v=YljiTB9ZxYw

References

[SP21]	Steinegger, S., & Primas, R. (2021, January). A fast and compact RISC-V accelerator for ascon and friends. In Smart Card Research and Advanced Applications: 19th International Conference, CARDIS 2020, Virtual Event, November 18–19, 2020, Revised Selected Papers (pp. 53-67). Cham: Springer International Publishing. URL: https://www.youtube.com/watch?v=rC4lb6T-qm4 , https://github.com/Steinegger/riscv_asconp_accelerator
[P22]	Primas, R., 2022, URL: https://github.com/ascon/ascon-hardware
[RD22]	Rezvani, B., Diehl W., 2022, URL: https://github.com/vtsal/ascon_lwc_aead_hash
[NGK22]	Nagpal, R., Gaj, K., Kaps, J.P., 2022, URL: https://github.com/GMUCERG/Ascon
[MGK22]	Mohajerani, K., Gaj, K., Kaps, J.P., 2022, URL: https://github.com/kammoh/bluelight/tree/api3/Ascon
[GWD+15]	Gross, H., Wenger, E., Dobraunig, C., & Ehrenhöfer, C. (2015, August). Suit up!--Made-to-Measure Hardware Implementations of ASCON. In 2015 Euromicro Conference on Digital System Design (pp. 645-652). IEEE. URL: https://github.com/IAIK/ascon_hardware/tree/master/caesar_hardware_api_v_1_0_3/ASCON_ASCON
[KKA21]	Kaur, J., Kermani, M. M., & Azarderakhsh, R. (2021). Hardware constructions for error detection in lightweight authenticated cipher ASCON benchmarked on FPGA. IEEE Transactions on Circuits and Systems II: Express Briefs, 69(4), 2276-2280.
[KHK+17]	Kumar, S., Haj-Yahya, J., Khairallah, M., Elmohr, M. A., & Chattopadhyay, A. (2017). A comprehensive performance analysis of hardware implementations of CAESAR candidates. Cryptology ePrint Archive.

References

[GIB18]	Groß, H., Iusupov, R., & Bloem, R. (2018). Generic low-latency masking in hardware. *IACR transactions on cryptographic hardware and embedded systems*, 1-21.
[GM18]	Groß, H., & Mangard, S. (2018). A unified masking approach. Journal of cryptographic engineering, 8, 109-124.
[S16]	Samwel, N. (2016). Side-Channel Analysis of Keccak and Ascon.
[TS14]	Taha, M., & Schaumont, P. (2014, May). Side-channel countermeasure for SHA-3 at almost-zero area overhead. In 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST) (pp. 93-96). IEEE.
[M20]	Moos, T. (2020). Unrolled cryptography on silicon: a physical security analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems, 416-442.
[BSL+10]	Bhasin, Shivam, Sylvain Guilley, Laurent Sauvage, and Jean-Luc Danger. "Unrolling cryptographic circuits: A simple countermeasure against side-channel attacks." In Topics in Cryptology-CT-RSA 2010: The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings, pp. 195-207. Springer Berlin Heidelberg, 2010.
[JMR22]	J. Kaur, M. Mozaffari Kermani and R. Azarderakhsh, "Hardware Constructions for Error Detection in Lightweight Authenticated Cipher ASCON Benchmarked on FPGA," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 69, no. 4, pp. 2276-2280, April 2022, doi: 10.1109/TCSII.2021.3136463.
[KPQ19]	K. Ramezanpour, P. Ampadu and W. Diehl, "A Statistical Fault Analysis Methodology for the Ascon Authenticated Cipher," 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 2019, pp. 41-50, doi: 10.1109/HST.2019.8741029.