

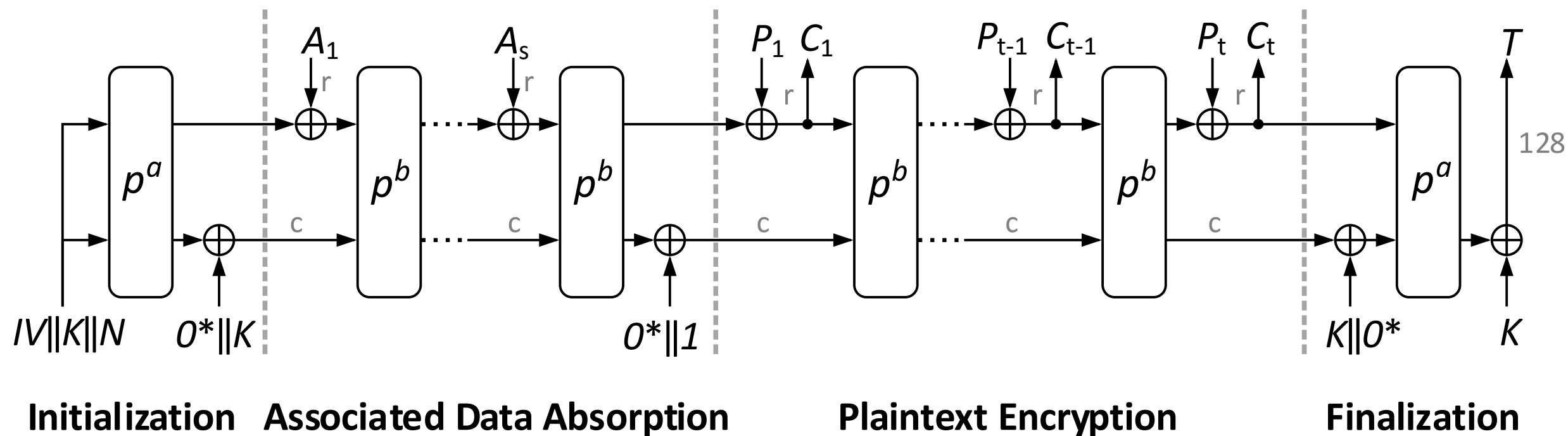
The background features a dark blue network diagram with white lines connecting various nodes. A central shield, composed of a light blue mesh of lines and dots, contains the team's name and members. Surrounding the shield are several circular icons representing different IoT applications: a heart with a pulse line, a microchip, a location pin over a truck, an RFID tag, a power line tower, a house with a Wi-Fi signal, a smartwatch with a heart, and a building with a Wi-Fi signal.

## Team JayHackers

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

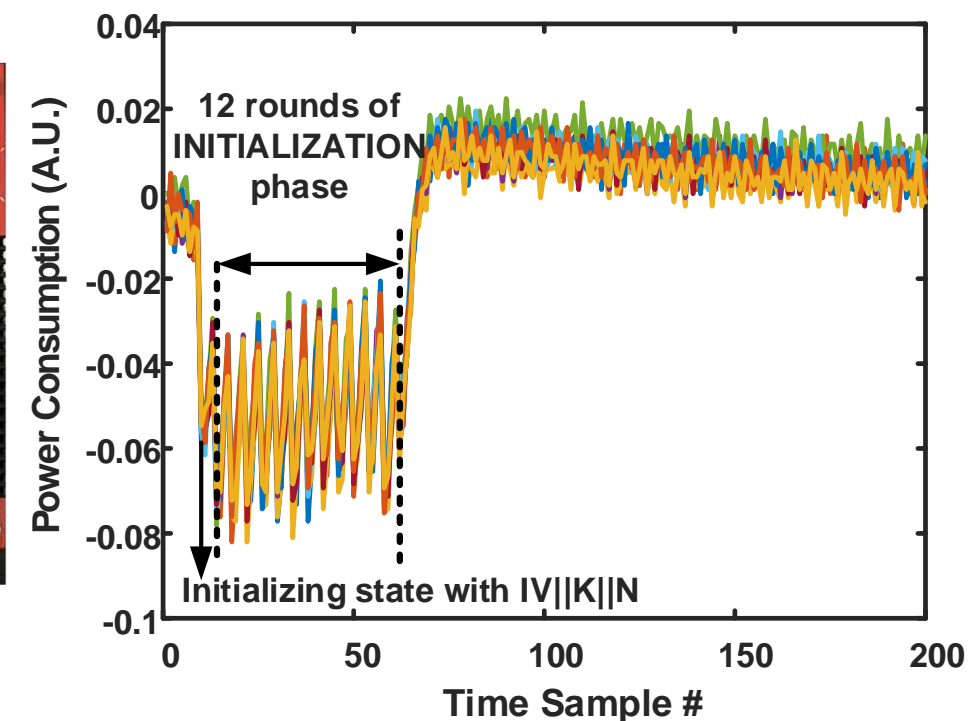
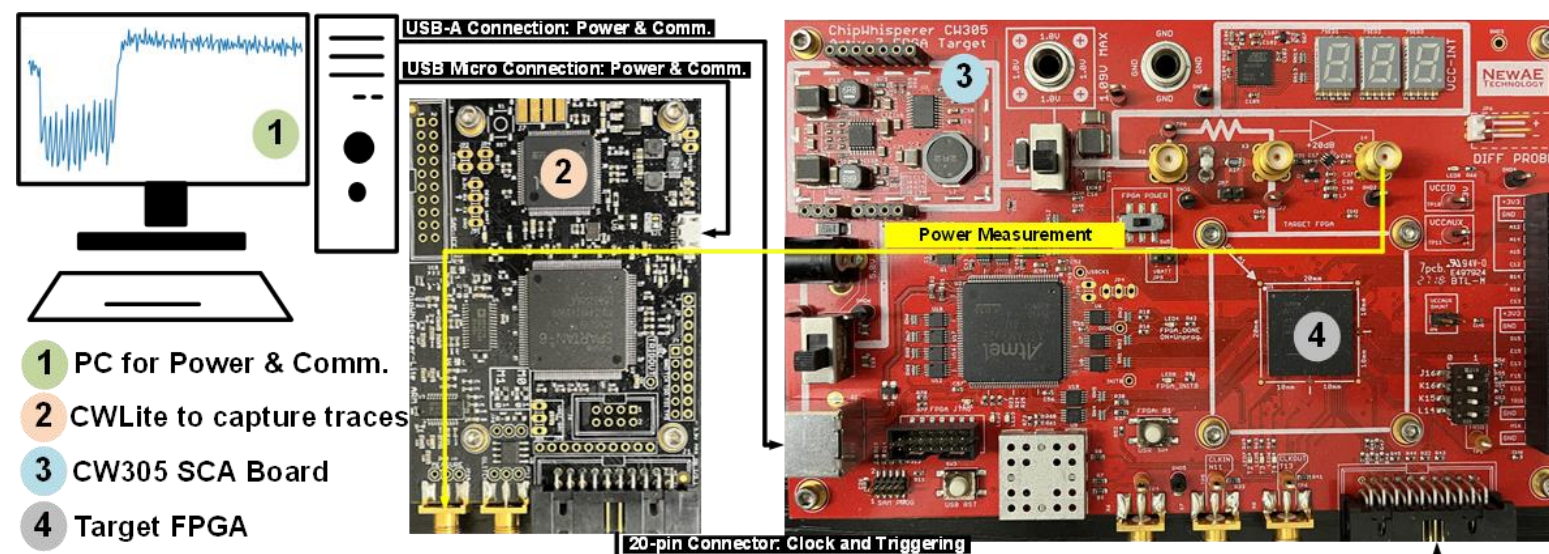
## ASCON's Authenticated Encryption with Associated Data (ASCON\_AEAD) [ASCON21]

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



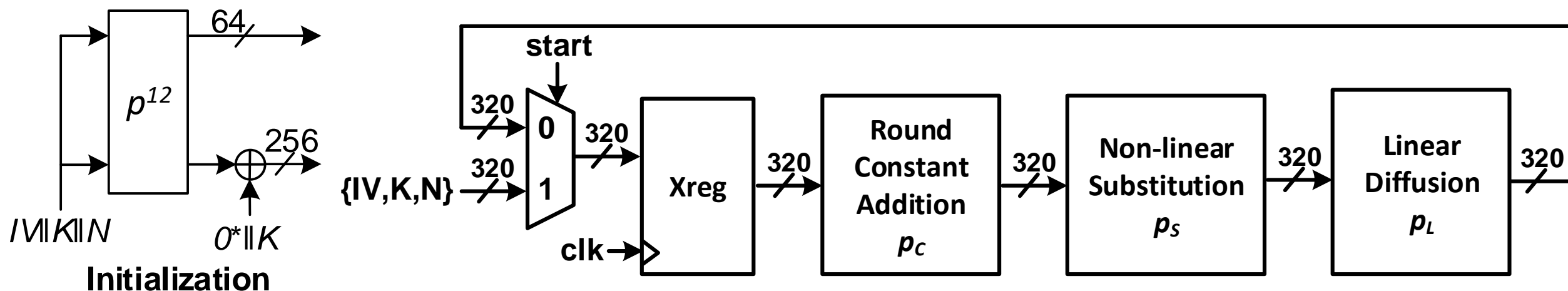
## Simple Power Analysis (SPA) of INITIALIZATION Phase of ASCON\_AEAD

- **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  $\rightarrow$  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

$$(y_0, y_1, y_2, y_3, y_4) = Sbox(x_0, x_1, x_2, x_3, x_4)$$

$$y_0 = x_0 \oplus x_2 \oplus x_1 x_2 \oplus x_3 \oplus x_1 \oplus x_0 x_1 \oplus x_1 x_4$$

- Highlighted in blue → Constant register activity

$$y_0 = x_3 \oplus x_1 x_4$$

- 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:  $8 \times 64$  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**  $\rightarrow$  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  $\rightarrow$  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  $x_2 \rightarrow$  **indistinguishable** due to XOR operation  $\rightarrow$  **Merged** bit  $x_{12} = x_1 \oplus x_2$

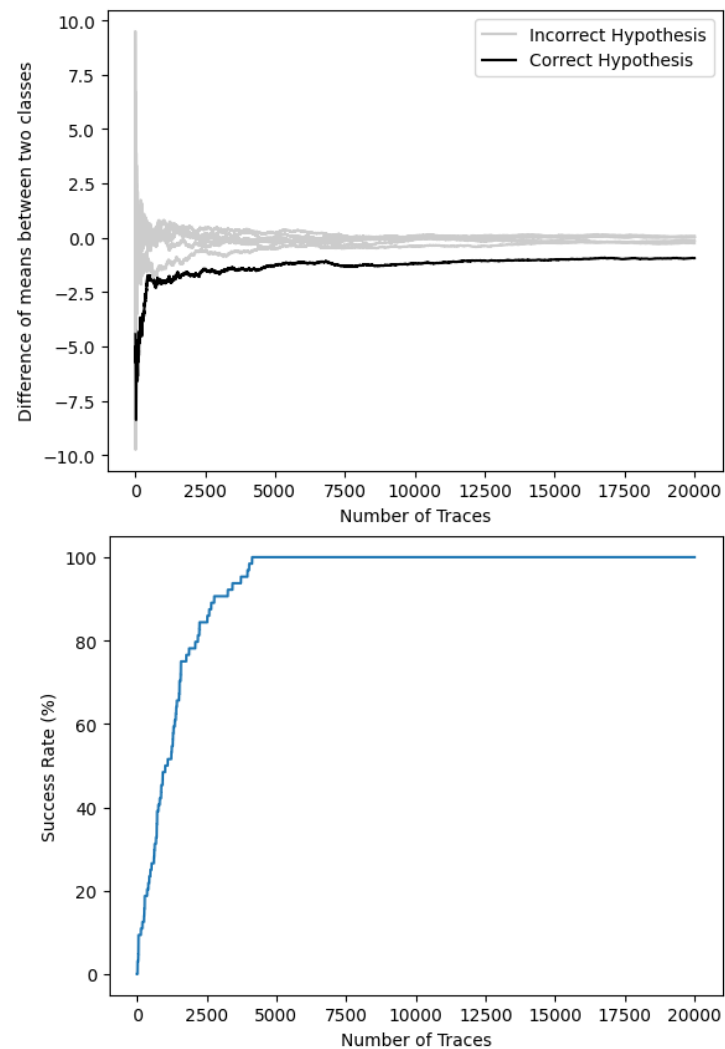
$$\sum_{l_1, r_1} (x_1) = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39)$$

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

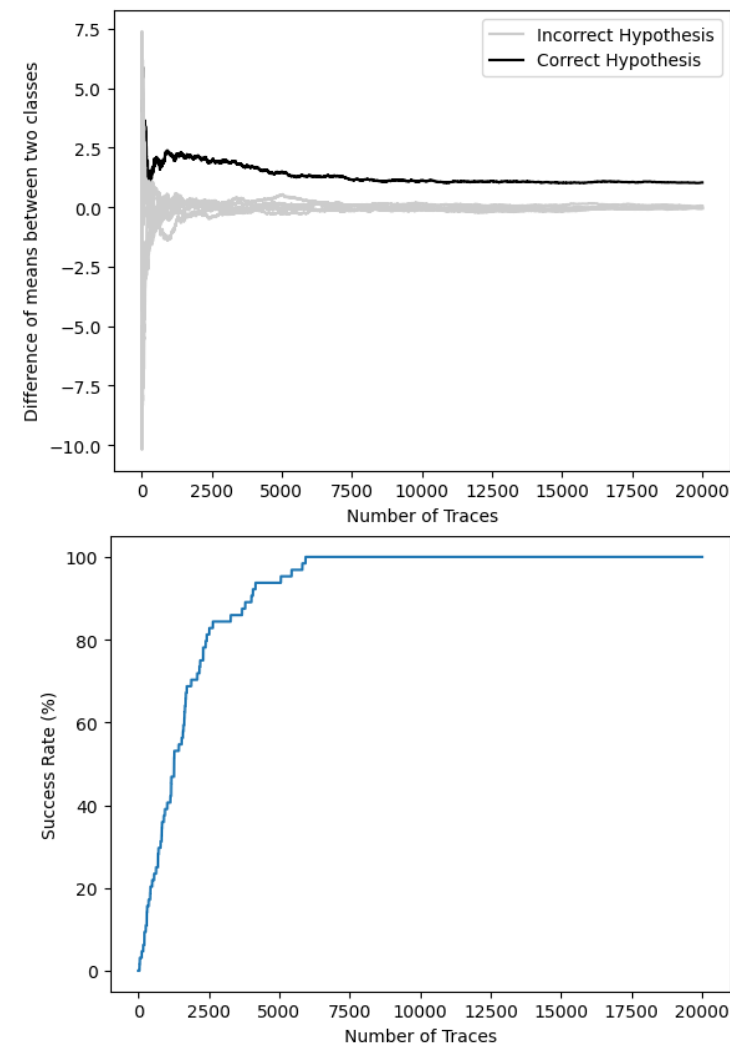
- **Effect of diffusion layer  $\rightarrow$  Selection function  $S_i(N, K^*)$**  to split traces based on  **$i$ -th** bit of  $x_{12}$  register update depends on **3 bits** of **key** register  $x_{12}$  and **6 bits** of **nonce** registers  $x_3$  and  $x_4$ .
- **Total hypothesis search space: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

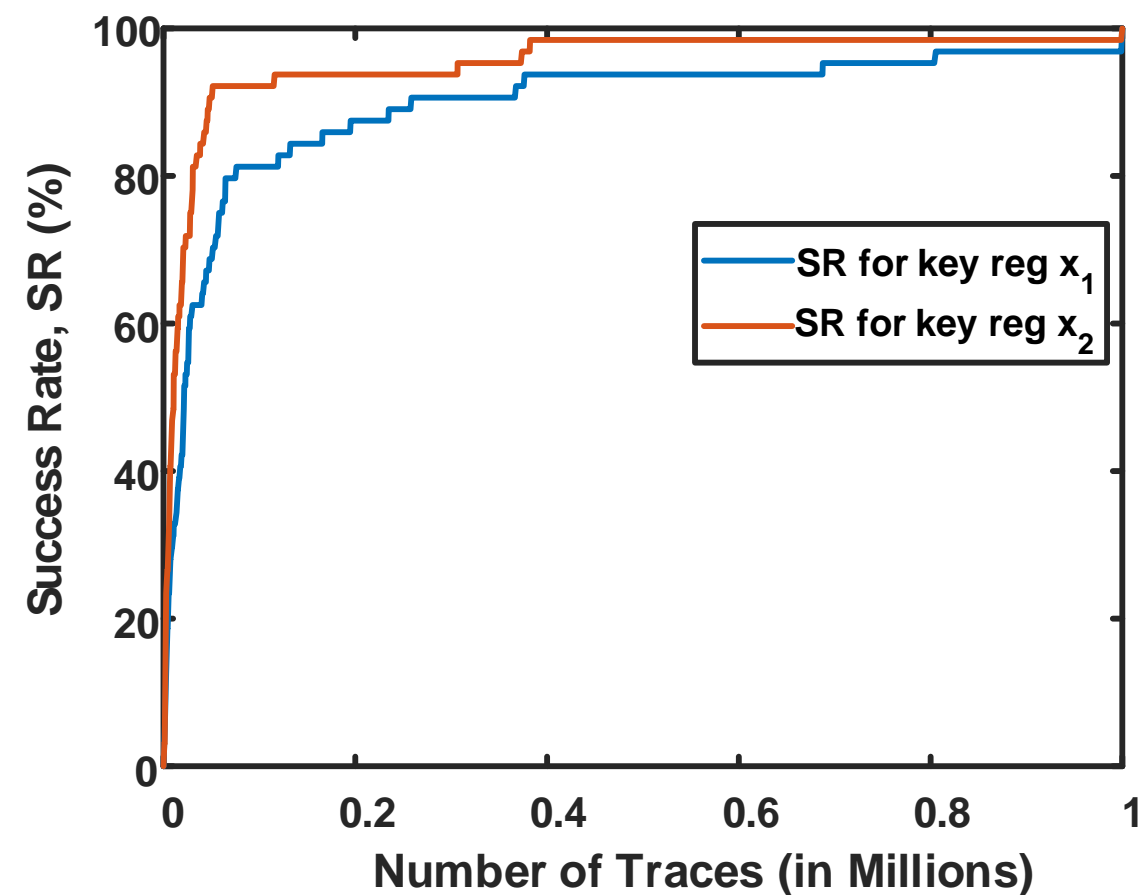
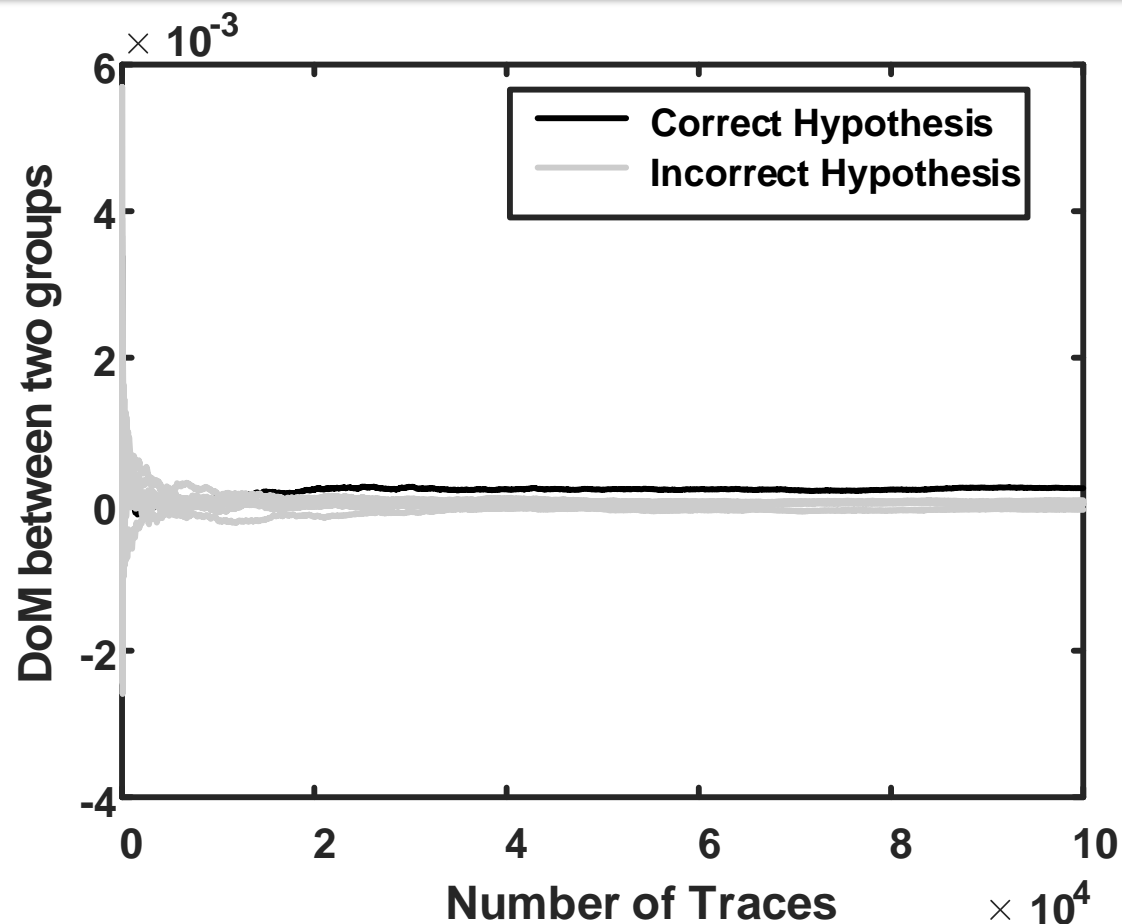


### Recovering Key Reg 2



## DPA Attack on INITIALIZATION Phase of ASCON\_AEAD [S16]: Measured Results

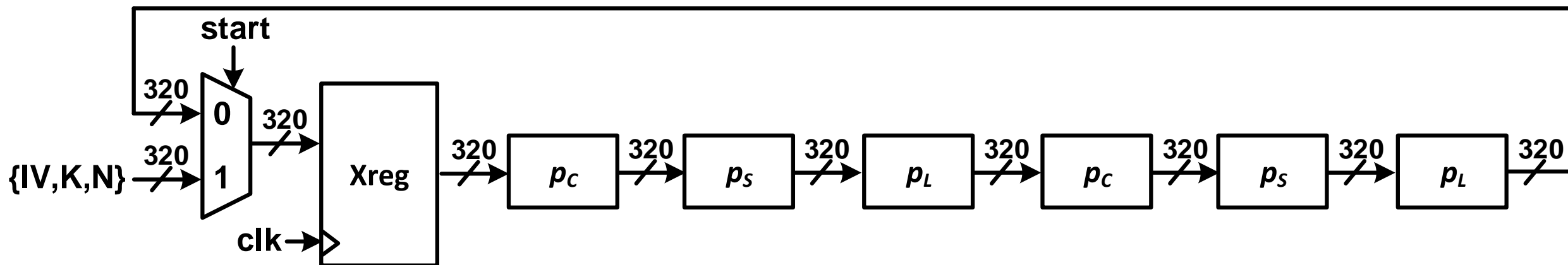
- **Distinguisher:** Difference of Means (**DoM**)
- **Number of Traces:** 1M
- **Success rate (SR):**  $> 96\%$ .





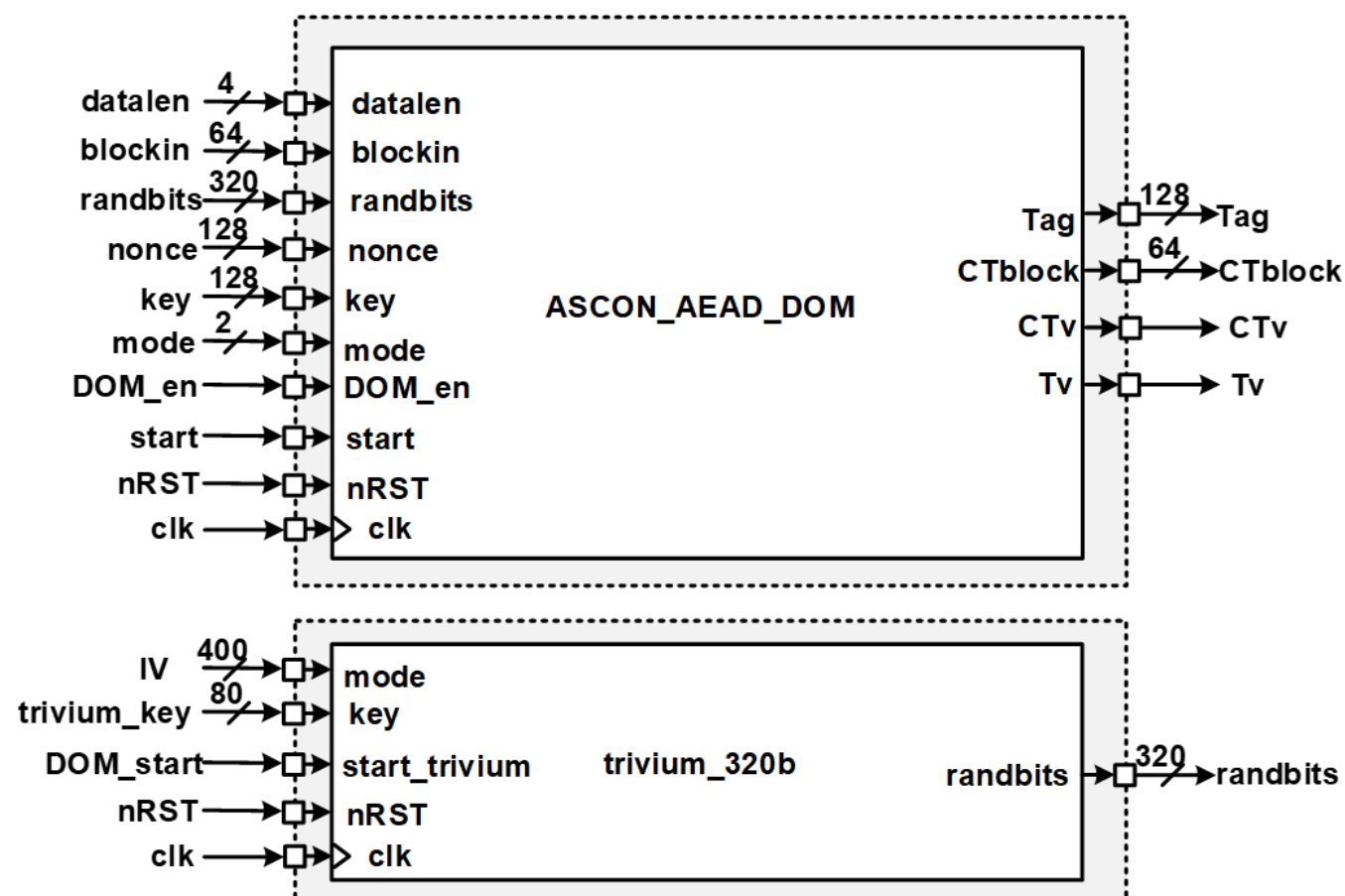
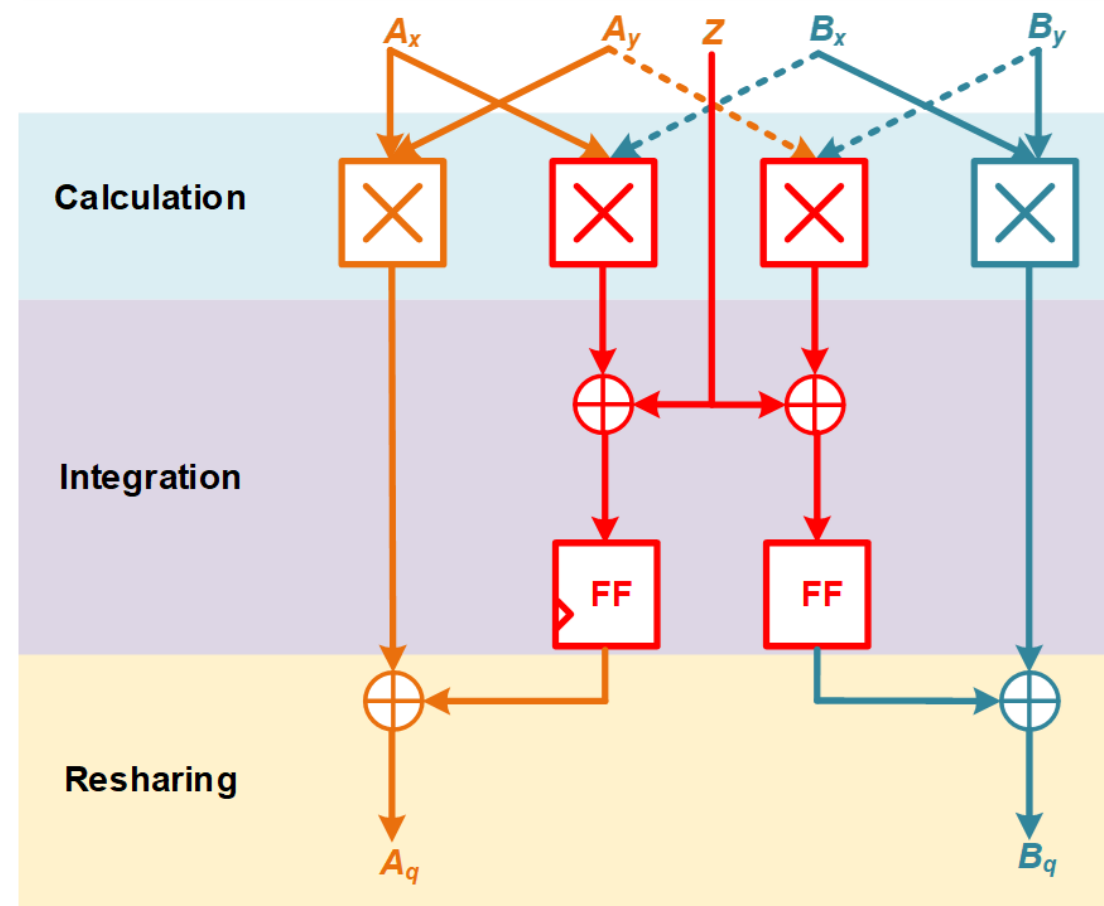
## DPA Countermeasure: Round unrolled Implementation [BSL+10,M21]

- **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)



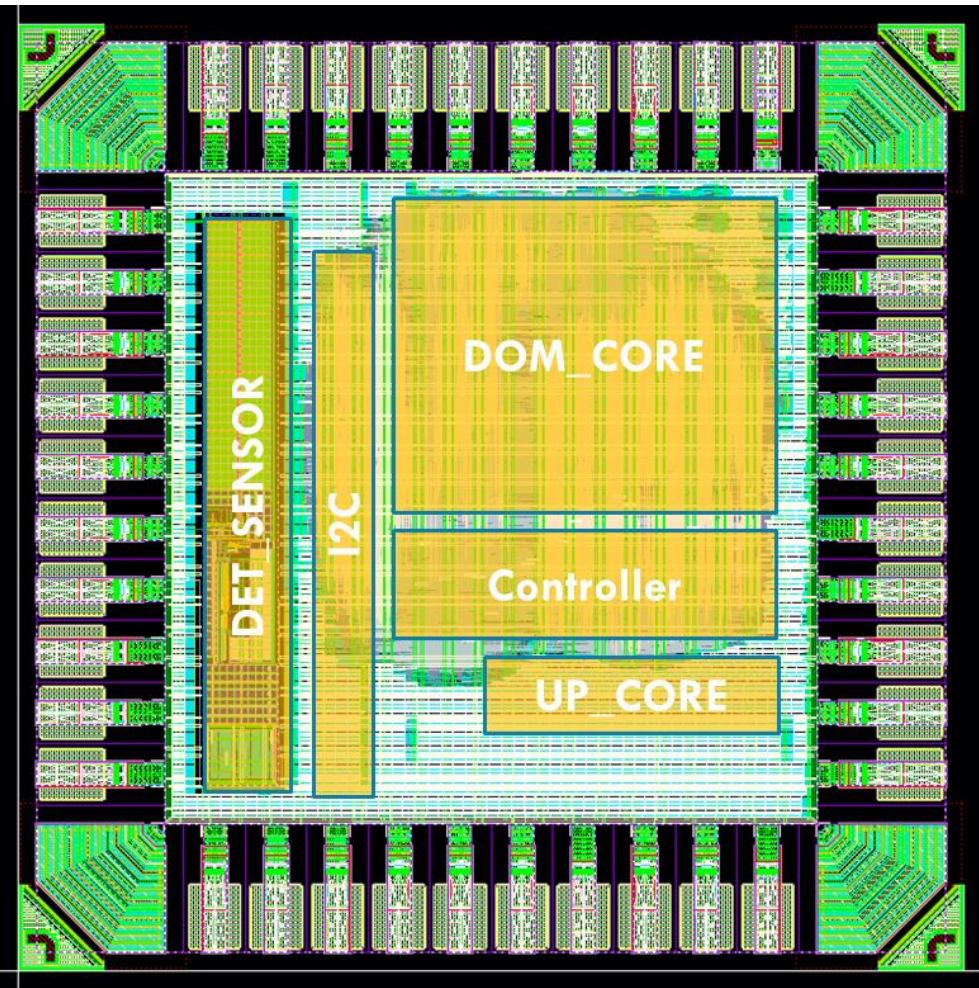
## DPA Countermeasure: Domain-Oriented Masking (DOM) [GMK16]

- 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**.

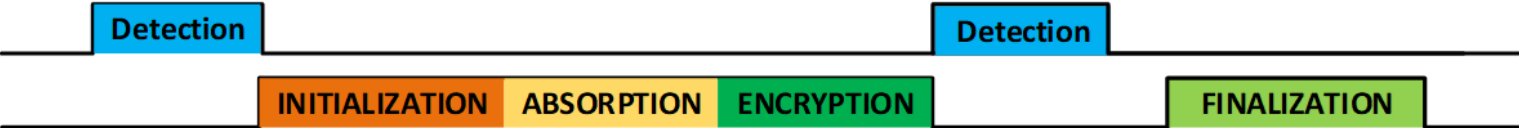
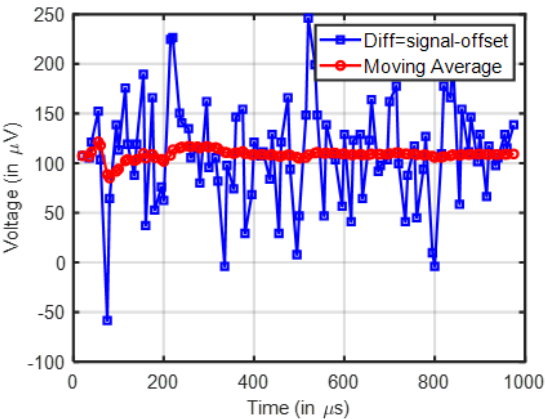
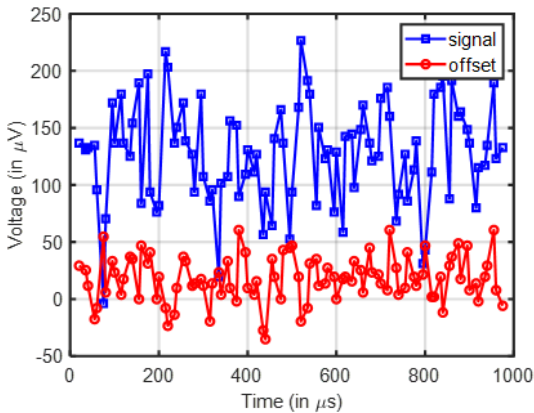


DPA Countermeasure: DOM + Detection Sensor Countermeasure ASIC

- **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**.

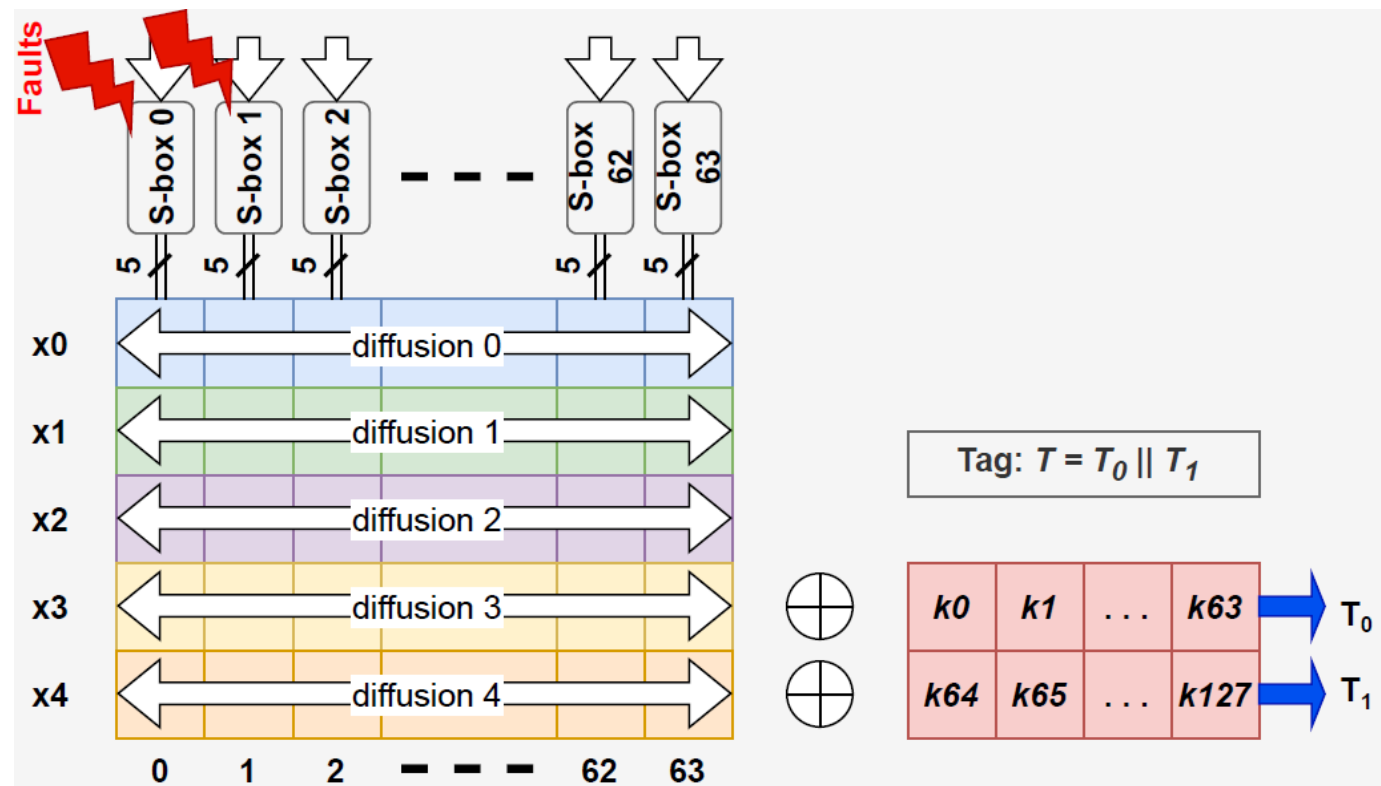


Module	Description	Area ( $\mu\text{m} \times \mu\text{m}$ )
UP_CORE	Unprotected ASCON	$400 \times 50$
DOM_CORE	DOM ASCON	$400 \times 300$
DET_SENSOR	Detection Sensor	$600 \times 80$



## 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

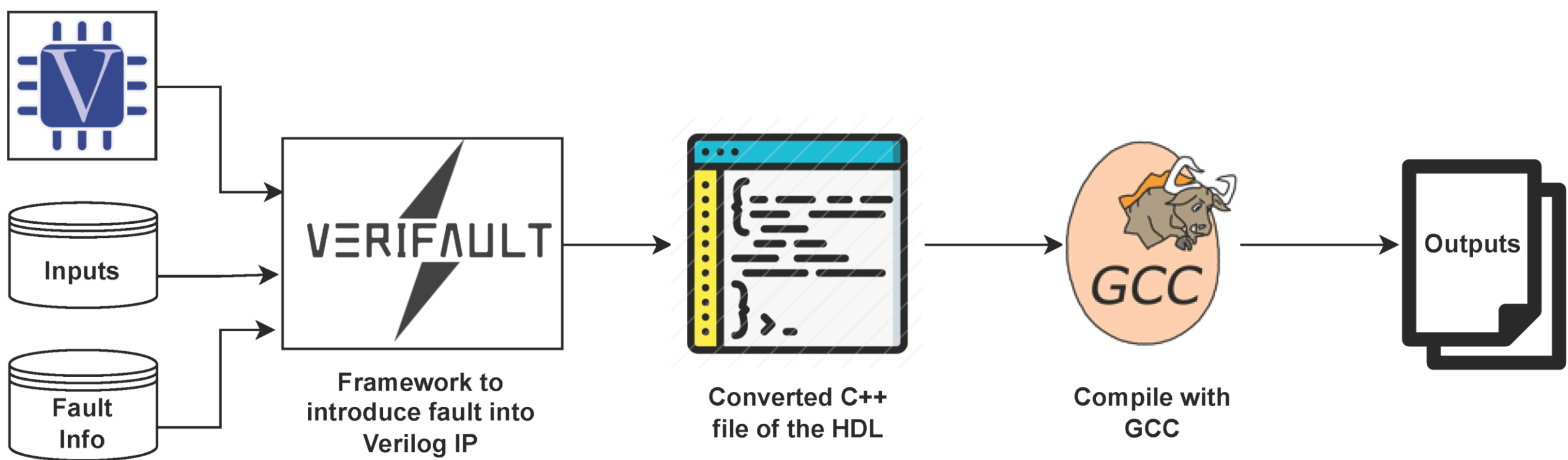
- **Problems with Traditional Method**

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

- **Simulation-Based Method**

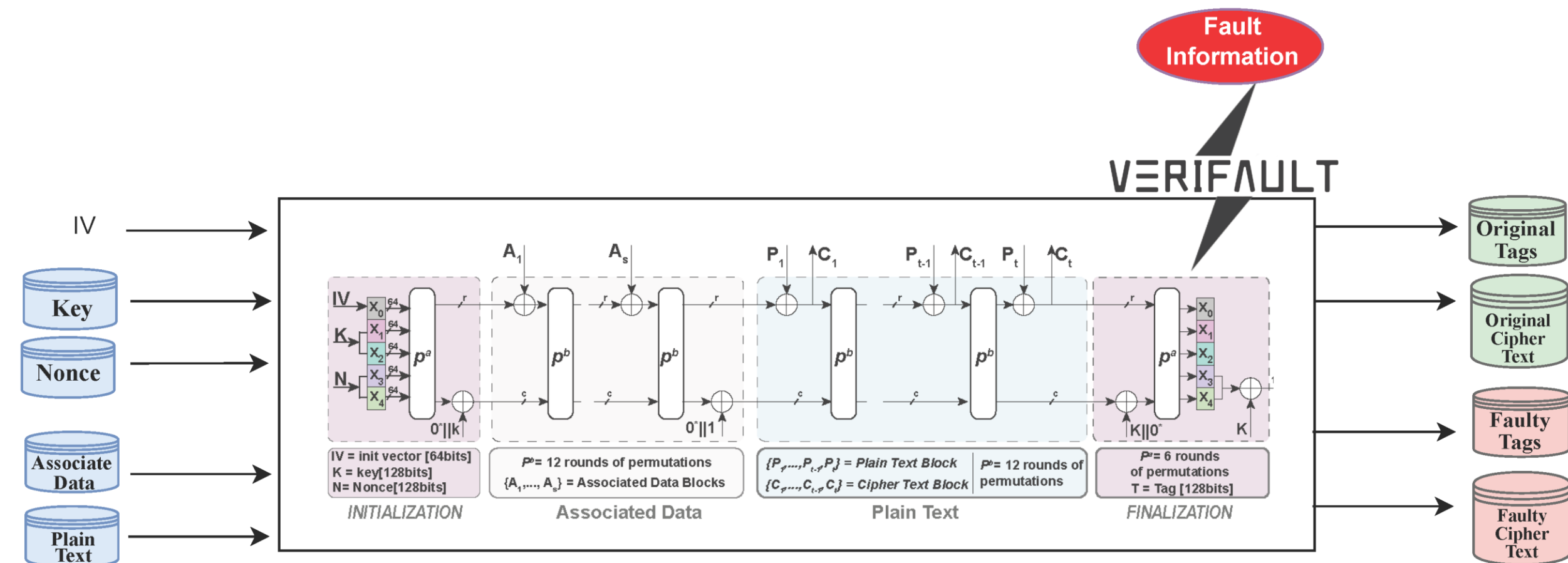
- Software-based solution that directly simulates faults using Verilog hardware designs
- Easy to simulate for any given condition with expected outcome
- A framework is created for inducing fault in ASCON at any stage
- Verilator environment seems most suitable as it can declare internal nets or registers 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

```
100110001111000101101011001010110100010101001110111001110100001000000010010000101000001011000011000000111000001111
1011001000011100010001111101110110010010011101001111100010100000100000001001000010100000101100001100000011000001111
100101110001101010000001100000001110110010100000110011010110000100000001001000010100000101100001100000011000001111
000110111000111010010010001100111110100011100011111101000001000010000000100100001010000010110000110000001100000111
1000011011101011011100000101101001000101101100011110100100001000010000000100100001010000010110000110000001100000111
00000011100101000000101101000011110001100000000110000010000000010000000100100001010000010110000110000001100000111
11110010111010110010110000011001101110011000110001011001001000010000000100100001010000010110000110000001111
11100100000001001000011111010011001101100000000100000100000001001000010100000101100001100000011000001111
011100000010111011100110101000010100010000100110101000010000000100100001010000010110000110000001100000111
0111100100110011001101010000101000100001000101000101010000100000001001000010100000101100001100000111
10110001011100100010001001101100110010101000011111000001000001000000010010000101000001011000011000001111
101100010111001000100000100101110011001010100001000001000000010010000101000001011000011000001111
010100010100111010010000100001010010101111010100100001000000010010000101000001011000011000001111
0000111011001100010110010101001001110100000000100000101000010100000101000001011000011000001111
111001100000010110001011011000001110011010111101000010000100000001001000010100000101100001100001111
10101100000001001100000101111101111100111101010100001000000010010000101000001011000011000001111
011110001000100100100001010100110100111001110011001100110000010010000101000001011000011000001111
```

### Fault Free Tags

```
1001011010100011100001110101110000111101010101111000101100000100000110010110101000110101100110001000001011010011101
00010110111001010011010000010101101001011100111000010101001010000111011000110100000110000100011010100110110000011001100111
1000011010010110101111001000011100000110001000010010101010111000001000011011000111010101111100101100110101010100
1100110100001010101010011101010010011111110111001000101000000010000110111101000001001111110000110101010100101001101
1010101101101100010101100101100100100111100110000010110000010110010100111000111101000110100000101001111010111010011001101
10011111000110101111101001111101101110001011011000100001011010100000101101100110110100101010000011001111010000100
010000101101100011011000011011100011010010101000101001000000001110101001010111110000100001100000001100011010101000
011001001101110001100010011011100011101101100011010001110001011100100111000110100101010010110001001000010110100
011000101011100110001000111010101010010111110000001010101100111111100000100011001110000110010111110100111001100111
111100100011010010100000100000101011101001110011101100100010010011000110110011010101101011010011011001101100
001111101011010000011001101011000110001010000100000101001011101101000110011000010111000011010010101110001111
111111101011010110001100010000001010000100110111100101001011100010111101001101011001100100101011000111000111101
10000000101001100111101011110011010010010111010100010011100100100111001001101100101010110110110001010100100
0111010101000100010101110010000010101010010110100101101010010110111001011110001101000100101101111001001110111
00010011010011111110001001100110001010100101101101100100001111011001010100110011000001011010111010111100000101
110111111011100111110000111011100101101100100000001011100101110010101000011001110010111011011111000110100001010110
```

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**.

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: <a href="https://www.riscure.com/security-highlight-ascon/">https://www.riscure.com/security-highlight-ascon/</a>
[NIST23]	NIST LWC selection announcement, 2023, URL: <a href="https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon">https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon</a> , <a href="https://www.nist.gov/news-events/news/2023/02/nist-selects-lightweight-cryptography-algorithms-protect-small-devices">https://www.nist.gov/news-events/news/2023/02/nist-selects-lightweight-cryptography-algorithms-protect-small-devices</a>
[NIST18]	NIST Call for LWC, 2018, URL: <a href="https://www.nist.gov/news-events/news/2018/04/nist-issues-first-call-lightweight-cryptography-protect-small-electronics">https://www.nist.gov/news-events/news/2018/04/nist-issues-first-call-lightweight-cryptography-protect-small-electronics</a>
[Rambus23]	ASCON IP from Rambus, 2023, URL: <a href="https://www.rambus.com/blogs/rambus-ip-solution-supports-new-nist-lightweight-cryptography-algorithm/">https://www.rambus.com/blogs/rambus-ip-solution-supports-new-nist-lightweight-cryptography-algorithm/</a>
[HN23]	HackerNews, 2023, URL: <a href="https://thehackernews.com/2023/02/nist-standardizes-ascon-cryptographic.html">https://thehackernews.com/2023/02/nist-standardizes-ascon-cryptographic.html</a>
[ASCON21]	ASCON final specification, URL: <a href="https://ascon.iaik.tugraz.at/specification.html">https://ascon.iaik.tugraz.at/specification.html</a> , <a href="https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf">https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf</a>
[B19]	Buchanan B., “One of the greatest advancements in Cybersecurity: The Sponge Function (Keccak, SHAKE and ASCON)”, 2019, URL: <a href="https://medium.com/asecuritysite-when-bob-met-alice/one-of-the-greatest-advancements-in-cybersecurity-the-sponge-function-keccak-and-shake-6e6c8e298682">https://medium.com/asecuritysite-when-bob-met-alice/one-of-the-greatest-advancements-in-cybersecurity-the-sponge-function-keccak-and-shake-6e6c8e298682</a>
[Keccak]	URL: <a href="https://keccak.team/">https://keccak.team/</a>
[T22]	Tezcan C., “ASCON Authenticated Encryption Scheme”, 2022, URL: <a href="https://www.youtube.com/watch?v=YIjiTB9ZxYw">https://www.youtube.com/watch?v=YIjiTB9ZxYw</a>

[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: <a href="https://www.youtube.com/watch?v=rC4lb6T-qm4">https://www.youtube.com/watch?v=rC4lb6T-qm4</a> , <a href="https://github.com/Steinegger/riscv_asconp_accelerator">https://github.com/Steinegger/riscv_asconp_accelerator</a>
[P22]	Primas, R., 2022, URL: <a href="https://github.com/ascon/ascon-hardware">https://github.com/ascon/ascon-hardware</a>
[RD22]	Rezvani, B., Diehl W., 2022, URL: <a href="https://github.com/vtsal/ascon_lwc_aead_hash">https://github.com/vtsal/ascon_lwc_aead_hash</a>
[NGK22]	Nagpal, R., Gaj, K., Kaps, J.P., 2022, URL: <a href="https://github.com/GMUCERG/Ascon">https://github.com/GMUCERG/Ascon</a>
[MGK22]	Mohajerani, K., Gaj, K., Kaps, J.P., 2022, URL: <a href="https://github.com/kammoh/bluelight/tree/api3/Ascon">https://github.com/kammoh/bluelight/tree/api3/Ascon</a>
[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: <a href="https://github.com/IAIK/ascon_hardware/tree/master/caesar_hardware_api_v_1_0_3/ASCON_ASCON">https://github.com/IAIK/ascon_hardware/tree/master/caesar_hardware_api_v_1_0_3/ASCON_ASCON</a>
[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.

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