Team JayHackers Exploiting ASCON

*Tanvir Hossain, *Mahmudul Hasan, †Anupam Golder, † Zachary J Ellis, †Arijit Raychowdhury and *Tamzidul Hoque
*Electrical Engineering and Computer Science, University of Kansas
†Electrical and Computer Engineering, Georgia Institute of Technology
Email: *{tanvir, m.hasan, hoque}@ku.edu
†{anupamgolder, zellis7}@gatech.edu †arijit.raychowdhury@ece.gatech.edu

***Abstract*—In this report, we present our adopted techniques for differential power analysis (DPA) and differential fault analysis (DFA), as well as, the designed countermeasures for an ASCON IP released by HOST Microelectronics Security Competition. We have successfully performed DPA on the first round of *INITIALIZATION* phase of this ASCON IP using measured traces from an Artix-7 FPGA with a key recovery success rate $\geq 96\%$ @$1M$ traces. We have implemented a lightweight SCA countermeasure by round unrolling the permutation, which increases the hypothesis search space for an attacker. We have developed a simulation framework to analyze the effect of fault injection and have been able to inject faults on the S-box outputs of the last round of *FINALIZATION* phase and generate the fault-free and faulty tags. From there, we extracted the sensitive keys, which are used to XOR'ed with the output of the diffusion layer. Finally, we have also implemented an error correction code (ECC)-based fault detection countermeasure and observed 100% detection rate for the generated faults on the S-box outputs.**

***Index Terms*—Lightweight Cryptography, IP Security, Differential Power Analysis, Differential Fault Analysis**

I. Introduction

ASCON [1] is a symmetric-key encryption algorithm that offers both confidentiality and authenticity while operating in its authenticated encryption with associated data (AEAD) mode shown in Fig. 1. The authenticated encryption is completed in 4 phases: (a) keyed *INITIALIZATION*, (b) associated data *ABSORPTION*, (c) plaintext *ENCRYPTION*, and (d) keyed *FINALIZATION*. The algorithm's underlying construction is based on a permutation function, with the encryption and authentication of plaintext messages achieved through the use of a key, a nonce, and optionally associated data. ASCON was developed by a team of scholars from Graz University of Technology and gained recognition as the winner of the CAESAR project's cryptographic competition as well as the winner of the lightweight cryptography (LWC) competition organized by the national institute of standards and technology (NIST). Its robust security features and reliance on well-established encryption techniques make it a trusted and widely applicable cryptographic solution.

IEEE HOST Microelectronics Security Challenge 2023 [2] has released an IP for ASCON to be analyzed and hardened against side-channel information leakage and fault injection attacks. We have made the following contributions in phase-1 of the competition :

1) We thoroughly studied ASCON's specifications and analyzed the provided IP by writing a testbench to verify its functionality.
2) We identified the security vulnerabilities of the ASCON IP against power side-channel attacks and fault injection attacks.
3) We performed simple and differential power side-channel analysis on the *INITIALIZATION* phase of ASCON_AEAD and successfully recovered the key bits.
4) We adopted and developed a fault simulation framework to perform SFA and DFA on the ASCON IP.
5) We implemented simple fault injection attacks on the *FINALIZATION* phase of the ASCON_AEAD to corrupt the tag bits.
6) Finally, we implemented an error correction code (ECC)-based DFA countermeasure that successfully detected all the induced faults.

II. Analysis of ASCON IP

For ASCON, we implemented several layers of testbench to collect the input-output(IO) and intermediate nodes response. An API controls the whole data propagation in the main ASCON RTL design module. We separately generated the testbench for the top module API (i.e. ASCON64) and the ASCON main module (i.e. ASCON_AEAD), as shown in Appendix C. The testbench input contains the control inputs, associated data & its length, plaintext data & its length, key, and nonce. We revised the testbench based on the S-box input and output for simple fault analysis (SFA) and differential fault analysis (DFA). Further, we verified the IO responses against the python implementation of ASCON [3] and ensured the correct functionality. All the testbenches, design modules, as well as the video and presentation, are uploaded in the GitHub link [1].

III. Power Analysis

A. Power Measurement Setup

We have implemented the RTL on an AMD-Xilinx Artix-7 (xc7a100t) FPGA-based side-channel evaluation board from ChipWhisperer, namely, CW305 [4]. The experimental setup is shown in Fig. 2. The setup includes a PC to supply power to

[1]https://gitfront.io/r/user-6977671/cAG4YNMPF4fa/HOST23-IP-Sec-JayHackers/

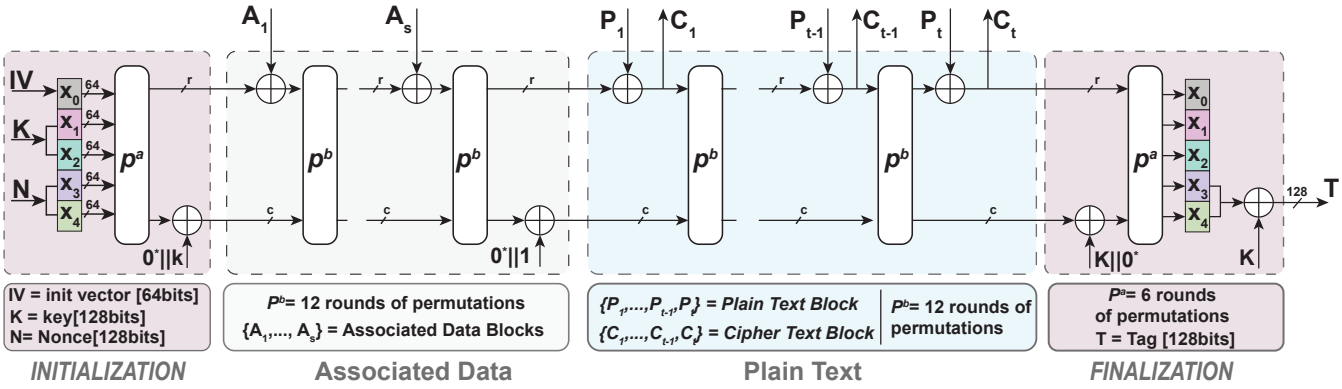

Fig. 1. Operation of the ASCON. A duplex construction-based lightweight cryptographic module.

the capture board and the target board, and send bitstreams and commands, CWLite capture board to send clock and triggering to the target FPGA, and capture power consumption traces, CW305 FPGA board which houses the target FPGA as well as the shunt resistor on the power pin followed by a 20-dB low noise amplifier (LNA). As in this setup, the sampling clock is derived from the device clock, and the power consumption at the clock edge can be reliably measured. Appendix A contains the measurement specifications.

B. Simple Power Analysis (SPA)

Due to the parallel execution on the whole 320-bit state (i.e., algorithmic noise) in a constant-time manner, it is not possible to visually distinguish between two different power consumption traces of the given ASCON implementation. Fig. 3 contains ten power consumption traces for the *INITIALIZATION* phase superimposed on each other. However, it is possible to observe the peaks in the power consumption traces that correspond to the state register updates as those switchings consume the largest amount of dynamic power. These peaks can serve as a proxy for clock ticks for an attacker to precisely identify and isolate the part of the cryptographic operations that they may want to attack. For example, the first peak appears when the state is initialized with the initialization vector IV, the key K, and the nonce N. The subsequent peaks are due to the permutation rounds during the *INITIALIZATION* phase. So, the second peak in Fig. 3 corresponds to the target leakage time sample that an attacker may utilize during differential power analysis (DPA) or correlation power analysis (CPA).

C. Differential Power Analysis (DPA)

The DPA techniques presented in the following are based on the work presented in [5].

1) Threat Model and Attack Point: We note that before the first round of permutation during the *INITIALIZATION* (Fig. 4(a) and (b)), the attacker has knowledge of the largest part of the state compared to any other phase. x_1 and x_2 registers contain the constant key bits before the first round of permutation during the *INITIALIZATION* phase. x_3 and x_4 registers contain the nonce bits, as shown in Fig. 1. As the nonce (i.e., public message number) is publicly known, in this threat model, we assume that the attacker knows the value of the nonce. We also assume that the nonce is a 128-bit uniformly random number. But we do not consider the nonce misuse case (repeated use of the same nonce) against which the proposers of ASCON do not claim security [1]. In the following, we have evaluated the security under this nonce-respecting scenario, with the assumption of the attacker-known or attacker-controllable nonce.

2) Development of DPA technique: We can use the properties of Boolean XOR and AND operations to rewrite the equations for ASCON's S-box operation. Appendix B contains all 5 equations of ASCON's S-box using just XOR and AND operations. If we look at the first output bit y_1, we note that several of its terms (x_0, x_1, x_2) are constant as long as the key is fixed. Only the nonce part changes from one *INITIALIZATION* phase to another. As such, all the bits that add a constant amount to the activity of the register can be removed.

$$y_0 = x_0 \oplus x_4 \oplus x_2 \oplus x_1x_2 \oplus x_3 \oplus x_4 \oplus x_1 \oplus x_0x_1 \oplus x_4x_1 \quad (1)$$

$$y_0 = x_1x_4 \oplus x_3 \quad (2)$$

Note that this intermediate value only depends on one bit from one key register and two bits from two nonce registers. From the ASCON specifications [1], the equation for linear diffusion layer operating on x_0 register is:

$$\Sigma x_0 = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \quad (3)$$

Combining the equation for the linear diffusion layer, we finally get the selection function:

$$S_i(N, K^*) = k_i^*(n_{64+i}) \oplus n_i \oplus k_{i-19+64}^*(n_{64+i-19+64}) \oplus n_{i-19+64} \oplus k_{i-28+64}^*(n_{64+i-28+64} \oplus n_{i-28+64} \quad (4)$$

Since there are 3 bits in the selection function, there are 8 key guesses. This 1-bit selection function can be used to split the measured traces into two groups based on the known values of nonce bits and each of the key hypotheses. To approximate

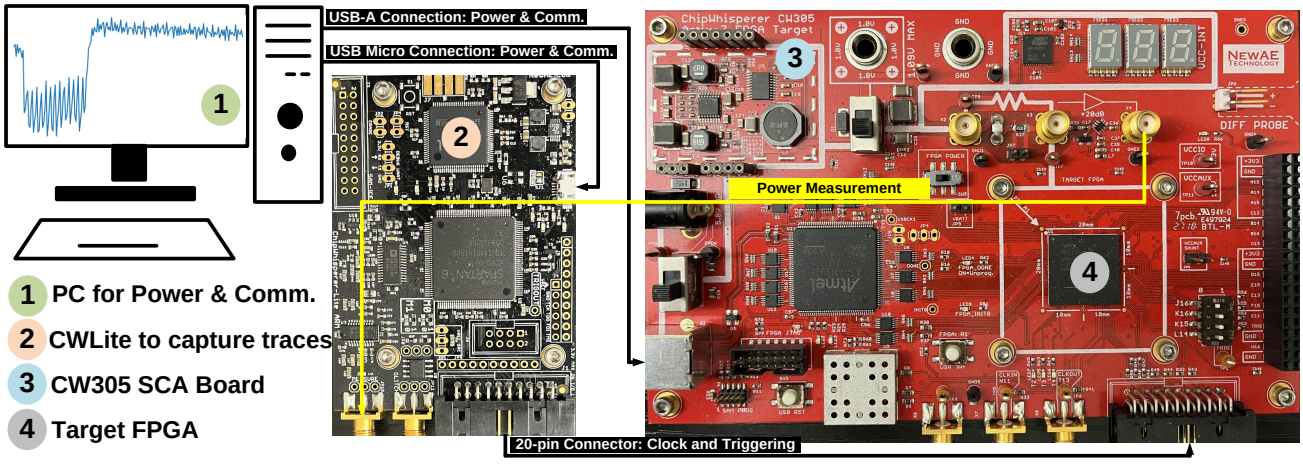

Fig. 2. Power consumption measurement setup

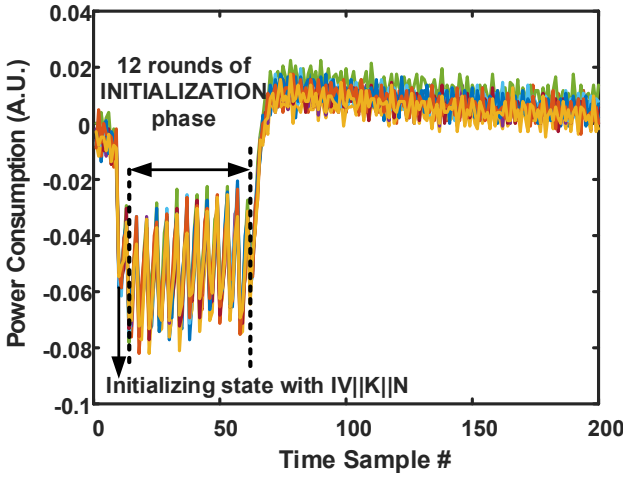

Fig. 3. 10 superimposed power consumption traces for ASCON INITIALIZATION running on Artix-7 FPGA

1-bit of switching activity, we need to use 6 known bits of the nonce.

The remaining number of unknown bits of the key is still too large to enumerate. So, the least significant half of the key must be attacked as well. From Appendix B, note that only y_1 has a result with a quadratic term containing bits from x_2 register which contains the least significant half of the key before the first round. Again, by removing all the bits that add a constant amount of activity to the register, we get:

$$y_1 = x_3(1 \oplus x_1 \oplus x_2) \oplus x_4 \tag{5}$$

As in the attack, it will not be possible to distinguish between x_1 and x_2, their result can be regarded as one term x_{12}.

$$y_1 = x_3(1 \oplus x_{12}) \oplus x_4 \tag{6}$$

The equation for linear diffusion layer for x_1 register is:

$$\Sigma x_1 = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39) \tag{7}$$

Combining these equations, we get:

$$\begin{aligned} S_i(N, K^*) = n_i(1 \oplus k_i^*) \oplus n_{64+i} \\ \oplus n_{i-61+64}(1 \oplus k_{i-61+64}) \oplus n_{64+i-61+64} \\ \oplus n_{i-39+64}(1 \oplus k^*_{i-39+64} \oplus n_{64+i-39+64} \end{aligned} \tag{8}$$

Since there are 3 bits in the selection function, there are 8 key guesses. However, one subtlety that has not been mentioned in [5] is the fact that for x_2 register, there is an effect of the round constant addition step as well. The round constant for ASCON-128 in the first round is $0xF0$, which means for 4 cases of the selection function, we need to take the invert of k_i^*. If we account for that, the rest of the cases are similar to that of attacking the x_1 register. Once the merged bits x_{12} are recovered, the bits of key register x_2 can be recovered by simply XORing x_{12} with previously recovered x_1, i.e., $x_2 = x_{12} \oplus x_1$.

3) Implementation of DPA attack: A *busy* signal was added to the RTL to indicate the completion of the *INITIALIZATION* phase. We recover 1-bit of the key register in each run; hence we need to repeat it 64 times to recover 64 bits of the key register x_1 and another 64 times to recover the bits of the key register x_2. Success rate (SR) is defined as:

$$SR = \frac{\#\ of\ bits\ recovered}{Total\ \#\ of\ bits\ in\ the\ key\ register} \times 100\% \tag{9}$$

The SR for both key registers x_1 and x_2 are shown in Fig. 5. In Fig. 6, the correct hypothesis emerges from the incorrect hypotheses after roughly $150K$ traces. With $1M$ traces, the SR for both registers reaches $\geq 96\%$. We can also see how the hypotheses evolve over a number of traces in Fig. 6 based on the difference of means (DoM) between two groups of traces separated by the selection function. To speed up the analysis, we performed a known key correlation (KKC) analysis with the hypothetical power traces based on hamming distance (HD)-leakage model. This allowed us to perform DPA on the most informative leakage sample in the traces to reduce the

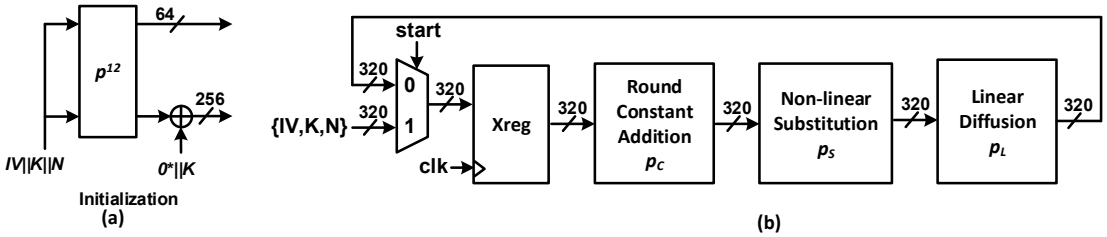

Fig. 4. (a) *INITIALIZATION* phase of ASCON (b) Simplified dataflow for ASCON's round update (round constant generation and phase controls have been left out)

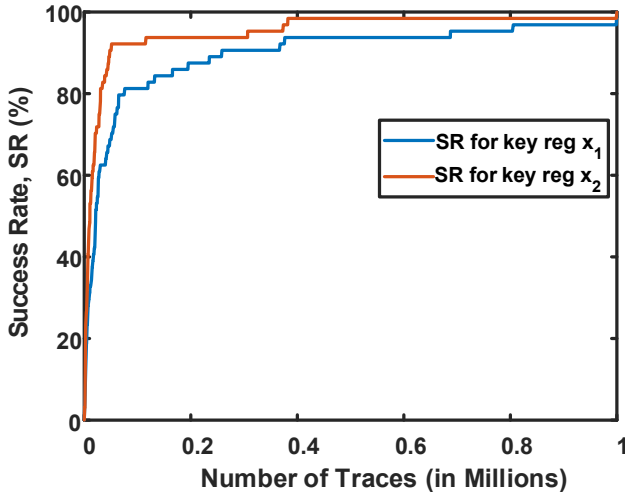

Fig. 5. Success Rate of DPA attack on two key registers x_1 and x_2

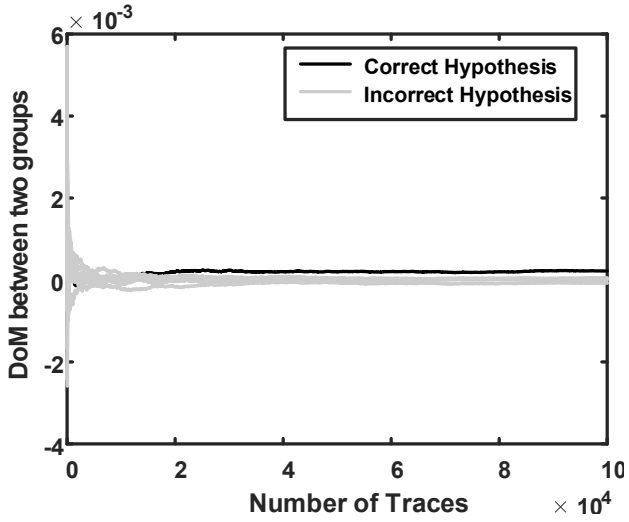

Fig. 6. Hypothesis Evolution of a single-bit attack

run time of the analysis. In the next phase, we plan to improve both the attack model and the attack scripts to perform analysis on larger number of traces.

D. Countermeasures against SPA/DPA

1) Lightweight Countermeasures: Note that the DPA technique illustrated in the previous section was successful because of the limited hypothesis search space (3 bits), and due to the fact that the whole 128-bit nonce was used to initialize the state, which was attacker controllable.

Because of the low-latency datapath of ASCON, it is possible to easily make a round unrolled implementation (i.e., multiple rounds/cycle execution) of ASCON. As traditional SCA attacks target the resisters where the round updates are stored, and the dynamic power consumption of unrolled implementations lacks correlation with HD between two consecutive register updates, they are more robust against the aforementioned threat model. Even for a $2R/C$ implementation (shown in Fig. 7(a)), the hypothesis search space increases by an infeasible amount for an attacker due to deeper diffusion of the key [6]. The downside of this is the increase in the area due to an increase in the combinatorial logic and an increase in latency due to the lengthening of the critical path, but the throughput and energy efficiency are improved [7]. We have implemented the $2R/C$ implementation that is not vulnerable to the DPA technique outlined in the previous section. We note that this countermeasure does not hide the side-channel leakage but rather makes it harder for an attacker to exploit it. Also, hamming weight (HW) of intermediates may be targeted, but the success rate would be much lower for the same number of measurement traces compared to the $1R/C$ implementation [8]. We note that the idea of unrolled cryptographic implementations to provide higher side-channel resistance was first proposed in [6] and later examined in [8] and [9]. This countermeasure has been implemented in this phase and provided in the GitHub repository. We will evaluate its DPA resistance before the next phase of the competition.

We can also choose to limit the data complexity that an attacker has access to. The DPA attack illustrated in the previous section was possible because 128-bit of nonce is absorbed at once, which allows the attacker to know much of the 320-bit state before the first round update. An interesting proposal was made in [10] as an SCA countermeasure for Keccak, which limits the attacker known data before each permutation to one or a few bits of nonce. It has also been proposed for XOODYAK and ISAP, both of which were LWC competition candidates. This idea of trickled nonce absorption is illustrated in Fig. 7(b). In this case, the permutation can be carried out for a reduced number of rounds after absorbing each bit of the state, as no part of the state is sent to the output. But instead of having just 12 rounds, the number

of rounds, in this case, increases to $6 \times 128 + 6 + 12$ in the case only one bit of nonce is absorbed at once. The most promising thing about this countermeasure is its low hardware cost. But the downside is the one-time performance overhead during *INITIALIZATION* phase due to added number of rounds. We are currently implementing this countermeasure and will analyze it before the next phase of the competition.

2) Provably Secure Countermeasures: If larger overhead can be tolerated, provably secure countermeasures against first-order DPA can be implemented, such as domain-oriented masking (DOM) [11]. In this case, the shares (or domains) of the inputs can be created by masking with random bits, and the linear layers of the permutation can simply be duplicated. The non-linear operation p_s mixes data from both domains, and we need to instantiate 320 DOM-protected AND gates, which require one random bit and four registers each gate to enable full pipelining. The circuit diagram for these modified AND gates can be seen in Fig. 8 and shows the three processes achieved. The calculation of the non linear function (in this case AND), the integration of the masking bit, and the resharing of the values. This method will require 320 random bits in each cycle which adds to the power and area overhead, with 5 64 bit RNGs based on the Trivium cipher [12] used for this purpose. In this case the state register was able to be moved to the integration registers inside the DOM AND gates in the S preserving the 1 round/cycle datapath. Only one extra cycle is needed before initialization to create the shares. From synthesis on the TSMC65_lp process, the approximate area for the two cores was calculated, with an area of 400 x 50 um for the unprotected core, and 400 x 300 um for the DOM protected core with 5 Trivium RNGs.

IV. Fault Analysis

A. Fault Simulation Method

Fault simulation is a technique to induce the fault on hardware nets or registers and observe the fault propagation through the output. In hardware, a fault can be induced through various methods, such as introducing random noise, altering the clock frequency, changing temperature, introducing voltage spikes, or focusing a laser beam. As a proxy for generating and observing the effects of fault injection on real devices, fault simulation has emerged to alleviate the difficulty of fault generation.

Verilator is an open-source simulator that converts a hardware description language (HDL) design into a C++ program that monitors the execution cycles of the original design. This allows the design to be simulated in software, revealing the timing of each execution part and the output result of a module. We developed a fault simulation program similar to [13], that manages inputs to the module and clocks it until the output is obtained. Verilator functions similarly to GCC compilation but is designed specifically for hardware simulation. Other tools, like VerFI [14], which can simulate faults in a netlist-based design, were available. But, it requires synthesis of the design to a specific technology beforehand, which is out of the scope of this competition. We were interested in a purely software-based solution that directly simulates faults using verilog hardware designs. The ASCON hardware architecture is preserved during the fault generation to collect the faulty *tag* results to analyze. Moreover, we wanted to create a general framework for fault generation and simulation, and the Verilator environment was the most suitable candidate. In Verilator, we can declare internal nets or registers as public and assign any operations on them, e.g., flipping a bit randomly or in a deterministic fashion.

B. Simple Fault Analysis (SFA)

The *FINALIZATION* phase in ASCON is vulnerable to both SFA and DFA due to the key whitening step at the end. We injected faults on the outputs of two S-boxes during the last round of permutation of the *FINALIZATION* phase. Precisely, we performed the following steps for SFA:

- To simplify our analysis, we simulated the design without the associated data absorption and the plaintext encryption phases. This allowed us to limit our execution to only 24 rounds (*INITIALIZATION+FINALIZATION*) for each execution of ASCON_AEAD.
- A function was defined in Verilator to simulate a random fault injection occurring subsequent to the S-box operation in the 24-th round. A random value was XOR'ed with the last two S-box outputs.
- Due to the fault injection, the tag results changed for the golden key and nonce value. Further, we performed 250 executions of random fault injections on the last three S-box outputs and captured all the tag results for future analysis. We plan to analyze the faulty tag bits due to random fault injections in the next phase of the competition to infer the key bits.

C. Differential Fault Analysis (DFA)

Differential fault analysis (DFA) is widely recognized as an effective technique for analyzing cryptographic systems. Specifically, DFA targets cryptographic hardware by exploiting injected faults with the aim of recovering secret keys. The underlying principle of DFA involves introducing controlled faults into a cryptographic system and observing the differences in behavior between correct and faulty executions. By analyzing these discrepancies, an attacker can infer information about the system's internal state, and ultimately recover the secret key.

In our DFA attack, we have implemented a fault attack approach utilizing the statistically ineffective fault analysis (SIFA) technique, which employs double-fault injection and key dividing as described in [15]. In [15], Ramezanpour et al. proposed a systematic fault analysis methodology by integrating SIFA and fault injection techniques, which specifically target a selected pair of S-boxes. A key dividing strategy is utilized to reduce the key search space, while the intermediate variable is determined by evaluating the output value of the targeted S-boxes. The ultimate goal of the fault injection procedure is to target the S-box operations executed during the final round of the *FINALIZATION* phase.

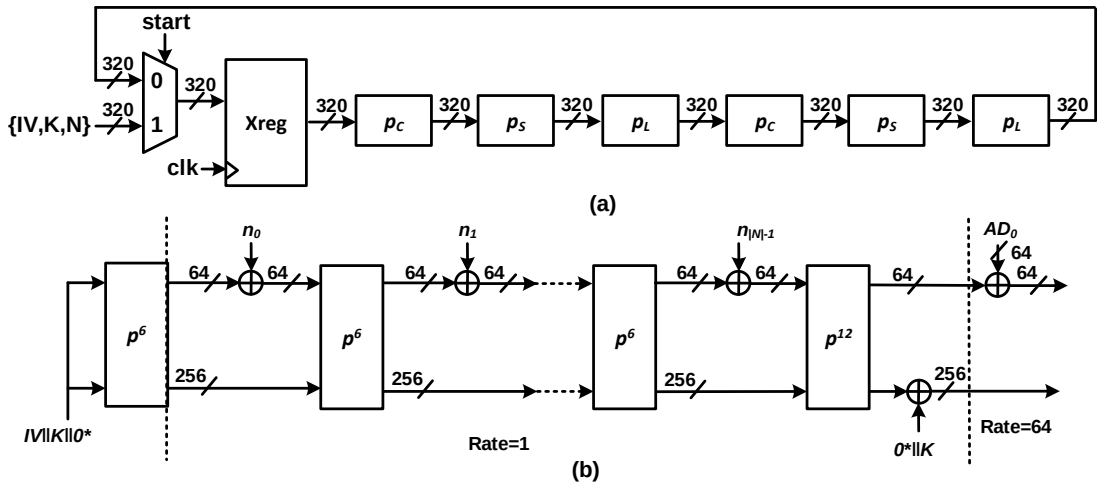

Fig. 7. (a) Simplified dataflow in a 2R/C unrolled implementation, (b) Trickled nonce absorption during *INITIALIZATION*

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

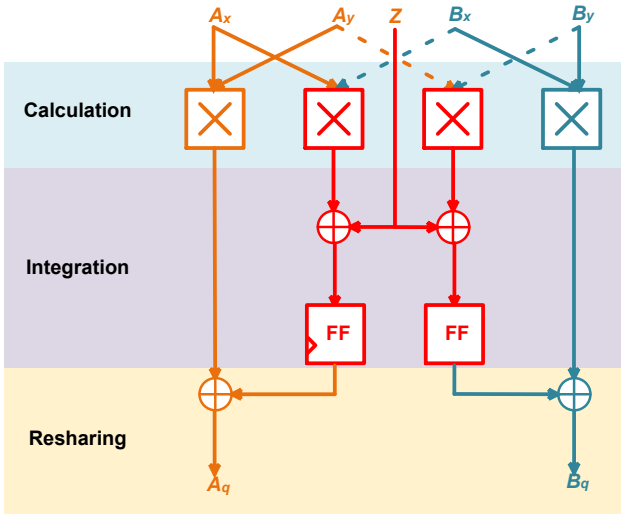

Fig. 8. Circuit diagram of a DOM AND gate

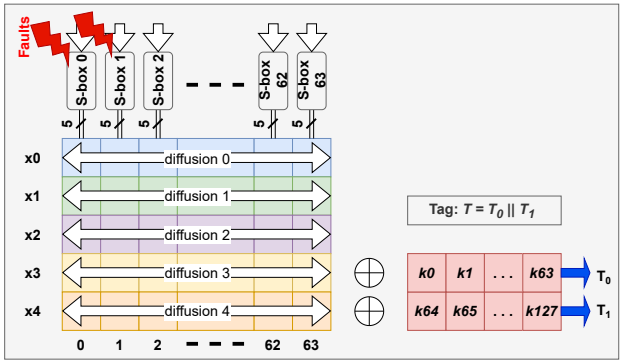

Fig. 9. The location of double fault injection at the last round of the *FINALIZATION* phase of ASCON_AEAD

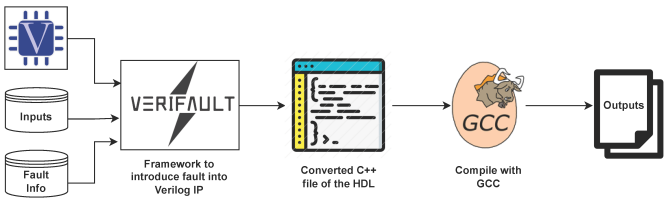

Fig. 10. Execution flow of proposed VeriFault Simulator.

1) Development of DFA technique: The adopted methodology for fault analysis involves the combination of SIFA with fault injections on a selected pair of S-boxes, along with a key dividing strategy to reduce the key search space. The intermediate variable used in the attack is the value at the output of the targeted S-boxes, while the fault injection is aimed at the S-box operations in the last round of the *FINALIZATION* phase. The permutation function of ASCON has unique properties that make it unsuitable for attacking

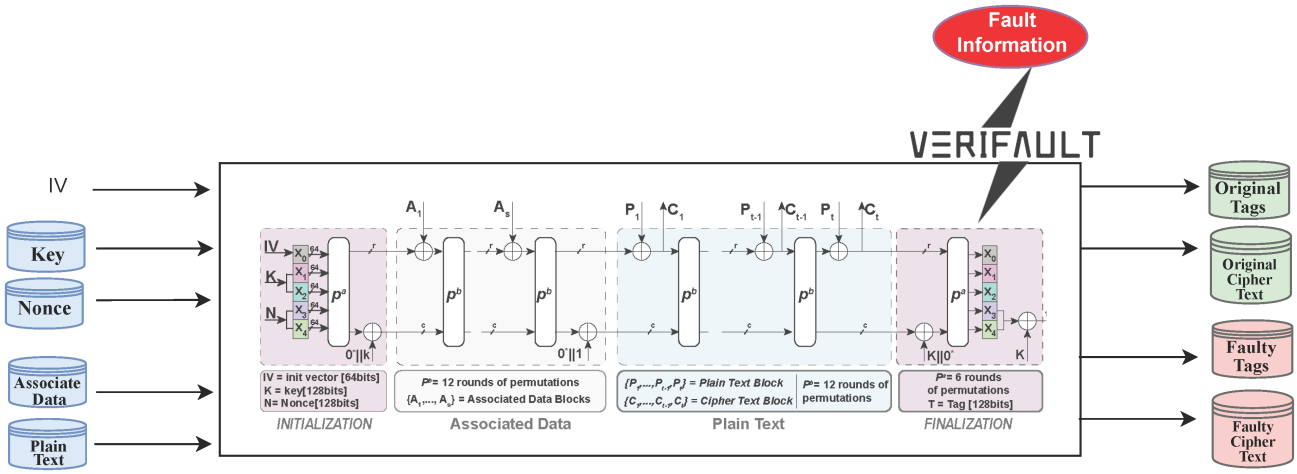

Fig. 11. Attack on the final round of the ASCON encryption using proposed VeriFault Simulator

using a single S-box fault injection. The fault model that has been hypothesized enables the attacker to instigate a fault at a selected pair of S-boxes during the last round of *FINALIZATION* phase. This results in a double-fault injection experiment, where it is possible for more than two S-boxes to suffer a fault. The proposed attack plan revolves around the observation that, for any hypothetical key, the operation that reverses the cipher output to the output of an S-box is a one-to-one linear mapping. Thus, the output data distribution of the S-box follows a non-uniform distribution with a consistent bias for any potential key assumption. Furthermore, to calculate the bits at the input of the diffusion layer, the inverse diffusion functions for state words x_3 and x_4 necessitate a sum total of 68 bits from the key and the output, rendering the search space impractical with a magnitude of 2^{68} for brute-force search. To resolve this challenge, the authors suggest a strategy for dividing the key. The proposed attack methodology offers a compromise between the count of double-fault injection experiments and the range of hypotheses for the encryption key. The key is fragmented into units of w bits in length, and necessitates $(w-1)$ experimental iterations that involve multiple plaintexts and fault injection at an S-box pair during each encryption. The hypothesis search space for each experimental round amounts to $2^{(256/w)}$. The principal challenge for the algorithms utilized in key analysis resides in the magnitude of the hypothesis search space, which must remain manageable for the purpose of conducting an exhaustive search.

D. Results of DFA

In the event of an attacker inducing a stuck-at-zero (bit-reset) fault at the input of an S-box, the subsequent output of the S-box would be parameterized. Considering the diffusion layer, it is important to note that the 0th bit of x3 depends on columns 0, 54, and 47 based on the diffusion layer equation. Setting the inputs to this specific S-box triple to zero guarantees that the 0th bit of x1, which undergoes an XOR operation with the key, will also be zero. Consequently, the 0th tag bit becomes equivalent to the 0th key bit.

In a similar manner, the remaining bits can be recovered. For instance, when examining the 0th bit of x4, the corresponding columns are 0, 57, and 23. This approach allows for the efficient extraction of key information, which can be leveraged for further analysis or exploitation in a cryptographic context.

E. Countermeasures against SFA/DFA

Error detection schemes have been recognized as an effective countermeasure against potential attacks on ASCON, owing to its non-linear and lightweight S-box implementation. In this phase of the competition, we have implemented error detection schemes based on logic gates and look-up tables (LUTs) in the hardware implementations of ASCON-128, as proposed in [16]. The error detection schemes presented, namely signatures and interleaved signatures, have been devised to detect both permanent and transient faults, including single, biased, or burst/adjacent faults. These error detection schemes have been applied to the 5-bit S-Box located in the nonlinear layer of ASCON-128, with the aim of augmenting the overall security of the system. The effectiveness of the implemented error detection schemes has been evaluated through fault simulations, revealing a high degree of fault coverage.

This study employs an error-correcting code (ECC) countermeasure that makes use of the output from the original ASCON S-box to generate interleaved, even and odd parity signatures. The outputs of the S-box, namely $\{y_0, y_1, y_2, y_3, y_4\}$, are utilized as inputs to the parity checker logic. To implement the S-box in ASCON, the input values $\{x_0, x_1, x_2, x_3, x_4\}$ are substituted with the values of the S-box in parallel, where x_0 represents the most significant bit (MSB), and x_4 denotes the least significant bit (LSB). This parallel implementation enables the full S-box to be executed within a single clock cycle, providing greater efficiency and faster processing times.

The parity logic performs a simultaneous check on a total of 64×5 bits in parallel, thereby generating 64-bit signatures

TABLE II
ACCURACY OF ECC BASED COUNTERMEASURE AGAINST FAULT ATTACKS IN ASCON

1-round	2-round	3-round
100%	100%	100%

for the interleave (p_0), even (p_1), and odd (p_2) parity. The parity checker evaluates the parity of each bit in the input data and produces parity bits as output. These parity bits are then used to generate the three signatures. The Boolean logic expressions for the signatures are presented in Equations 10, 11, and 12 respectively. The illustration of the S-box and parity operations are shown in Fig. 13. These expressions are used to calculate the parity bits, and the resulting signatures are appended to the message being sent.

Overall, this ECC countermeasure provides an effective means of enhancing the security of ASCON by introducing redundancy through the use of parity bits. By employing the output of the original S-box to generate interleaved, even, and odd parity signatures, this countermeasure can detect and correct any errors introduced during data transmission or due to fault injection.

$$p_0 = y_0 \oplus y_1 \oplus y_2 \oplus y_3 \oplus y_4 \tag{10}$$

$$p_1 = y_0 \oplus y_2 \oplus y_4 \tag{11}$$

$$p_2 = y_1 \oplus y_3 \tag{12}$$

In this study, the deliberate introduction of single and multiple-bit flip faults was employed to generate erroneous outputs of the S-box. The modification of the S-box output directly affects the parity block signature, which is verified by the parity checking module. Table I presents the input and output patterns of the S-box, along with their corresponding parity signatures. During the testing phase, the parity signature for the specific input bits is examined. To analyze the effectiveness of the implemented ECC countermeasure, 250 experiments for each class with fault induction in every encryption execution were conducted. The implemented ECC countermeasure was able to detect all faults with 100% accuracy.

As we are currently working on implementing DFA for the final phase of the competition, we plan to reassess the accuracy of this countermeasure.

V. CONCLUSION

In this brief, we have reported the completed tasks for the IEEE HOST Microelectronics Challenge 2023 - IP Security track. So far, we have been able to thoroughly analyze the provided IP for ASCON, implement it on an FPGA platform to capture power consumption traces, and perform SPA and DPA. We have also implemented a simple DPA countermeasure to increase SCA resistance. We have also developed a fault simulator based on an existing cycle-accurate simulation tool flow. This enabled us to perform the SFA and verify ECC-based DFA countermeasures. In the next phase of the competition, we plan to implement the DFA technique and evaluate its success rate using our fault simulation framework. We also plan to design DPA and DFA countermeasures which are more resilient to implementation vulnerabilities.

REFERENCES

[1] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1. 2: Lightweight authenticated encryption and hashing," *Journal of Cryptology*, vol. 34, pp. 1–42, 2021.

[2] HOST, "Ip security track," 2023. [Online]. Available: http://www.hostsymposium.org/call-for-challenge.php

[3] M. Eichlseder et al., "pyascon," 2023. [Online]. Available: https://github.com/meichlseder/pyascon

[4] A. Dewar, J.-P. Thibault, and C. O'Flynn, "Naean0010: Power analysis on fpga implementation of aes using cw305 & chipwhisperer," 2020.

[5] N. Samwel and J. Daemen, "Dpa on hardware implementations of ascon and keyak," in *Proceedings of the Computing Frontiers Conference*, 2017, pp. 415–424.

[6] S. Bhasin, S. Guilley, L. Sauvage, and J.-L. 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*. Springer, 2010, pp. 195–207.

[7] H. Gross, E. Wenger, C. Dobraunig, and C. Ehrenhöfer, "Suit up!–made-to-measure hardware implementations of ascon," in *2015 Euromicro Conference on Digital System Design*. IEEE, 2015, pp. 645–652.

[8] A. Singh, N. Chawla, J. H. Ko, M. Kar, and S. Mukhopadhyay, "Energy efficient and side-channel secure cryptographic hardware for iot-edge nodes," *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 421–434, 2018.

[9] T. Moos, "Unrolled cryptography on silicon: a physical security analysis," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 416–442, 2020.

[10] M. Taha and P. Schaumont, "Side-channel countermeasure for sha-3 at almost-zero area overhead," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 2014, pp. 93–96.

[11] H. Groß, S. Mangard, and T. Korak, "Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order," *Cryptology ePrint Archive*, 2016.

[12] C. De Canniere and B. Preneel, "Trivium," *New Stream Cipher Designs: The eSTREAM Finalists*, pp. 244–266, 2008.

[13] S. Pelissier, "Verilaptor: Software fault simulation in hardware designs," 2021. [Online]. Available: https://research.kudelskisecurity.com/2021/09/21/verilaptor-software-fault-simultation-in-hardware-designs/

[14] V. Arribas, F. Wegener, A. Moradi, and S. Nikova, "Cryptographic fault diagnosis using verfi," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 229–240.

[15] K. Ramezanpour, P. Ampadu, and W. Diehl, "A statistical fault analysis methodology for the ascon authenticated cipher," in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2019, pp. 41–50.

[16] J. Kaur, M. Mozaffari Kermani, and R. Azarderakhsh, "Hardware constructions for error detection in lightweight authenticated cipher ascon benchmarked on fpga," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 4, pp. 2276–2280, 2022.

APPENDIX A
EXPERIMENTAL SETUP SPECIFICATIONS

In the following, Table III contains the specifications for the experimental setup.

APPENDIX B
ALTERNATE EQUATIONS FOR ASCON'S S-BOX

Let the input and the output bits of ASCON's S-box be $(x_0, x_1, x_2, x_3, x_4)$ and $(y_0, y_1, y_2, y_3, y_4)$, respectively. Then the expressions for output in terms of XOR and AND operations of the inputs are listed in Table IV.

Type	Name
Target Board	NewAE CW305 Artix-7 FPGA Target
Synthesis Tool	AMD˙Xilinx Vivado 2019.1
ASCON software	pyascon [3]
CryptoCore frequency	10 MHz
Sampling frequency	40 MHz
ADC Resolution	10-bit

TABLE III
EXPERIMENTAL SETUP

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	$x_3 \oplus x_4 \oplus x_1 \oplus x_0x_1 \oplus x_4x_1$

TABLE IV
S-BOX OUTPUT WITH XOR AND AND OF INPUT

APPENDIX C
ASCON IP

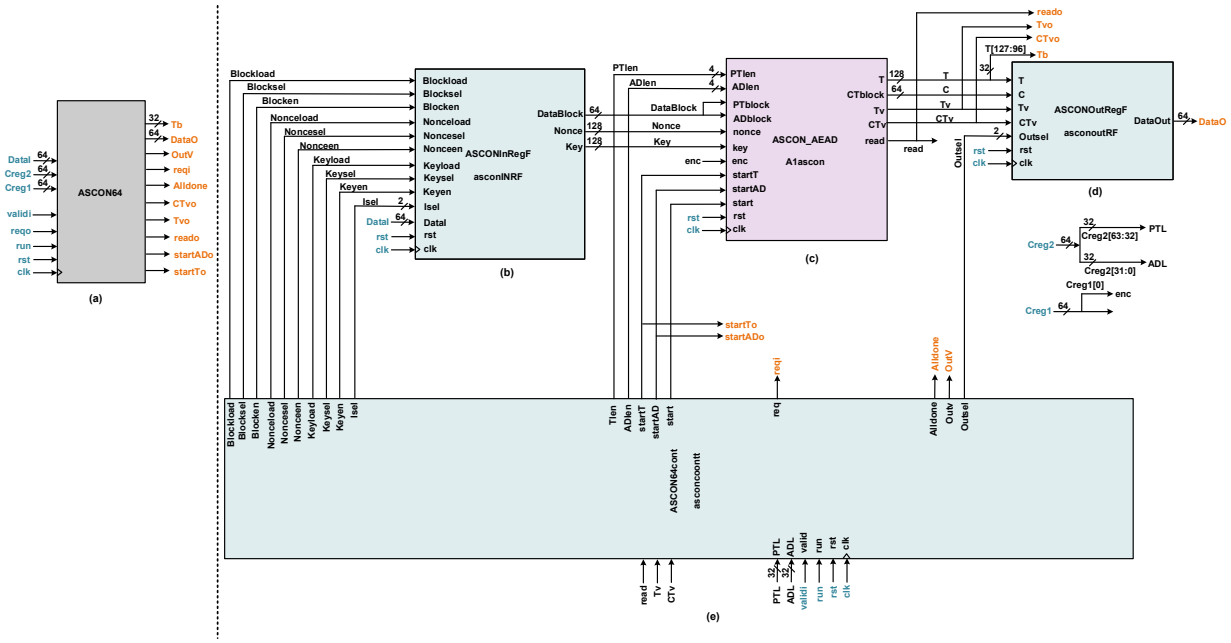

Fig. 12. Top level module view of given RTL of the ASCON and its application programming interface.

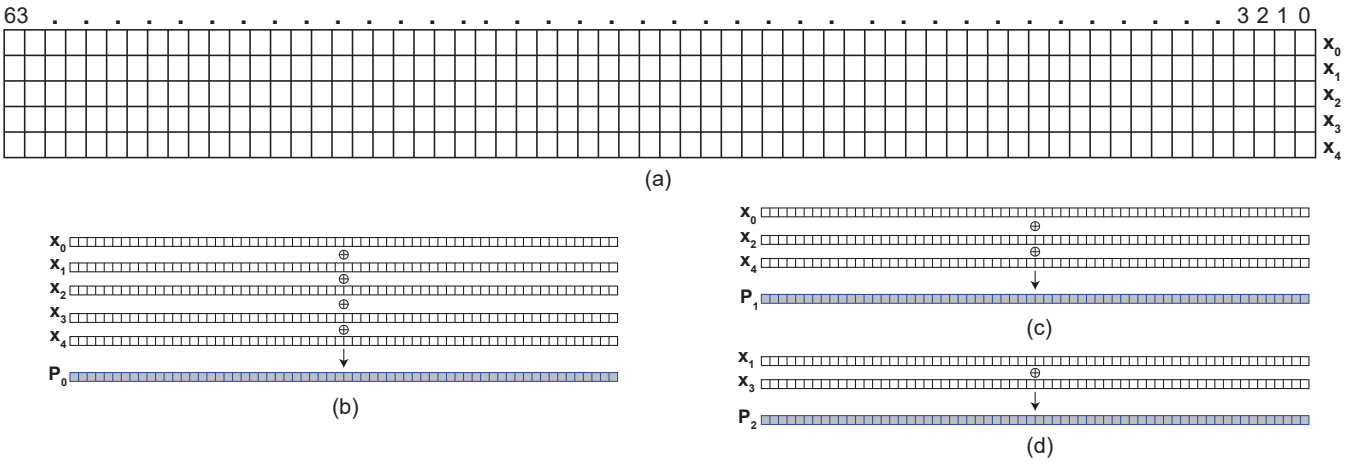

Fig. 13. Generation of ECC signatures