Side-Channel and Fault Resistant ASCON Implementation: A Detailed Hardware Evaluation

(Extended Version)

Aneesh Kandi*, Anubhab Baksi[○†], Peizhou Gan[†], Sylvain Guilley[‡], Tomáš Gerlich[●§], Jakub Breier[◇¶], Anupam Chattopadhyay[†], Ritu Ranjan Shrivastwa[‡], Zdeněk Martinásek[●§] and Shivam Bhasin[†]

*Indian Institute of Technology Madras, India
aneeshkandi@gmail.com
†Nanyang Technological University, Singapore
anubhab.baksi@ntu.edu.sg, peizhou.gan@ntu.edu.sg, anupam@ntu.edu.sg, sbhasin@ntu.edu.sg
‡Télécom Paris, Paris, France; Secure-IC, Cesson-Sévigné, France
sylvain.guilley@telecom-paristech.fr, ritu-ranjan.shrivastwa@secure-ic.com
§Brno University of Technology, Brno, Czechia
xgerli02@vut.cz, martinasek@vut.cz
¶TTControl GmbH, Vienna, Austria
jbreier@jbreier.com

***Abstract*—In this work, we present various hardware implementations for the lightweight cipher ASCON, which was recently selected as the winner of the NIST organized Lightweight Cryptography (LWC) competition. We cover encryption + tag generation and decryption + tag verification for the ASCON AEAD and also the ASCON hash function. On top of the usual (unprotected) implementation, we present side-channel protection (threshold countermeasure) and triplication/majority-based fault protection. To the best of our knowledge, this is the first protected hardware implementation of ASCON with respect to side-channel and fault inject protection. The side-channel and fault protections work orthogonal to each other (i.e., either one can be turned on/off without affecting the other). We present ASIC and FPGA benchmarks for all our implementations (hash and AEAD) with/without countermeasures for varying input sizes.**

***Index Terms*—ASCON, Hardware Implementation, Side-Channel Attack, Threshold Implementation, Fault Attack, Countermeasure**

I. Introduction

In the contemporary era of electronic communication, confidentiality, and integrity/authentication act as the two vital components. To this end, various cryptographic primitives are proposed in the literature. While the realm of cryptography is a few decades old, there is a relatively new trend in designing the so-called 'lightweight' ciphers, which aim at providing strong security despite having a low device (hardware and/or software) footprint. Recently, the US government's National Institute of Standards and Technology (NIST) has organized the 'Lightweight Cryptography'[1] (LWC) project in order to further boost research and ultimately set a standard. From many submissions, ASCON [2] has been selected as the winner. Therefore, in the coming days, one could expect to see a rapid increase in the usage of ASCON in various industry applications as well as it being a prime candidate for academic research[2]. Amid this situation, a natural question that comes to mind is the hardware cost of implementing this cipher. Also, because the cipher is mainly intended to be used in resource-constrained embedded devices (such as Internet-of-Things appliances), one has to consider the impact of the physical attacks. These attacks come in two flavors, namely 'side-channel analysis' [4], [5] and 'fault analysis' [6]. Consequently, one has to further consider the cost of implementing adequate countermeasures. However, being relatively new, there is not many research works attempting hardware implementation/optimization of ASCON, as found during our literature survey (Section I-B). This prompted us to do a thorough, systematic, easy-to-use, and publicly accessible implementation of ASCON in hardware. Our codes are written from scratch and do not have any other dependency.

A. Our Contributions

In this work, we present various hardware (Verilog) implementations of ASCON [2], which is a lightweight hash function and AEAD (authenticated encryption with associated data) family[3]. We consider regular (unprotected)

This is an extended version of the paper with the same title accepted in IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2024. An early version of this paper is available as [1].

○: This project is partially supported by the Wallenberg-NTU Presidential Post-Doctorate Fellowship.

●: This work is partially supported by the Ministry of Interior of Czech Republic under grant VJ02010010.

◇: This research is funded by the European Commission, under the Horizon Europe project aerOS, grant number 101069732.

[1] https://csrc.nist.gov/Projects/lightweight-cryptography.

[2] Notably, it is used in a post-quantum cipher [3].

[3] In summary, a hash function takes an arbitrary length message and returns a fixed-length output. An AEAD consists of two ends: at one end (the sender) encryption + tag generation is done, and on the other end (the recipient) decryption + tag verification is done.

implementations as well as side-channel and fault attack protection.

In summary, we implement and benchmark the following:

(α) Unprotected ASCON (encryption + tag generation, decryption + tag verification; and hashing).
(β) Side-channel attack resistant ASCON using *threshold implementation*.
(γ) Fault attack resistant ASCON using triplication/majority.
(δ) Combined side-channel and fault attack protected ASCON using threshold and triplication/majority.

Our implementations use a simple interface. The side-channel and fault protections can be turned on/off easily depending on the use case (it is not necessary that both the countermeasures have to be used all the time) by making minimal adjustments to the interface. Our source codes are accessible as an open-source project[4].

B. Previous Works

The hardware implementation of ASCON has been explored before, for instance, in [7], [8], [9], [10], [11]. However, as far as we can tell, no side-channel-protected (threshold) implementation exists. Also, the common countermeasure against fault (that relies on triplication/majority) has not been explored before. The following papers [7], [8], [9] implement only the unprotected version of ASCON and in [10], [11], protection for fault injection attacks has not been implemented.

II. ASCON Description

The ASCON family has 2 variants — ASCON-128 (block size = 64 bits) and ASCON-128A (block size = 128 bits). Both of those take a 128-bit key and nonce and have a 320-bit state. The ciphertext and the 128-bit tag are generated after (encryption + tag generation), and the plaintext is recovered from the ciphertext as well as the tag is verified in (decryption + tag verification).

A. Permutation

The main strength of ASCON lies in the permutation process. p^n represents the number of rounds for the permutation. There are two types of permutation of i) p^a consisting of a rounds (used for initialization and finalization) and ii) p^b consisting of b rounds (used for data processing).

The 320-bit state S of the ASCON is divided into 5 registers of 64 bits each. $S = x_0||x_1||x_2||x_3||x_4$. These 5 registers are then sent for further processing. Each permutation round is further divided into three layers – the constant addition layer, the substitution layer, and the linear diffusion layer.

[4]https://github.com/aneeshkandi14/ascon-hw-public. This repository contains multiple threshold implementations of the SBox, any of which can be ported to the actual implementation (some of these are benchmarked in Table II.

1) Constant XOR Layer: In this layer, a constant term is added with the x_2 register word. The constant term added depends on the current round number of the permutation. For p^a, round constant c_r is used and for ρ^b, round constant c_{a-b+r} is used, where r is the number of rounds. The number of rounds and related constants are given in Table I.

TABLE I: Round constants and number of rounds for ASCON

ρ^{12}	ρ^8	ρ^6	Constant	ρ^{12}	ρ^8	ρ^6	Constant
0			0000000000000000f0	6	2	0	0000000000000096
1			0000000000000000e1	7	3	1	0000000000000087
2			0000000000000000d2	8	4	2	0000000000000078
3			0000000000000000c3	9	5	3	0000000000000069
4	0		0000000000000000b4	10	6	4	000000000000005a
5	1		0000000000000000a5	11	7	5	000000000000004b

2) Substitution Layer: This layer implements the 5-bit SBox operation of the ASCON which is the only non-linear operation within the permutation. The ASCON SBox can be given by the following 5-bit look-up table: (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).

3) Linear Diffusion Layer: This layer is used to shuffle the bits of each register internally with the help of right rotation and XOR. It is performed with the operations:

$$x_0 \leftarrow x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28)$$
$$x_1 \leftarrow x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39)$$
$$x_2 \leftarrow x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6)$$
$$x_3 \leftarrow x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17)$$
$$x_4 \leftarrow x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41)$$

B. Authenticated Encryption with Associated Data: (Encryption + Tag Generation) and (Decryption + Tag Verification)

ASCON is a family of authenticated encryption and (verified) decryption scheme which can be parameterized by 4 variables; namely the key (K), rate (r) and number of rounds (a and b) for the permutation computation. The key length is, $k \leq 128$ bits and other parameters vary depending on the type of ASCON. Inputs for the authenticated encryption are plaintext P, associated data A, key K, and nonce N (k-bits); and outputs are the ciphertext C and tag T. Inputs for the verified decryption are key K, nonce N (k-bits), ciphertext C, and tag T; and output is plaintext P if the tag is successfully verified, otherwise, an invalid response is returned (indicated as $\perp$).

The operation of ASCON can be divided into four sub-routines, namely:

1) Initialization
2) Processing associated data
3) Processing plaintext/ciphertext
4) Finalization

Upon initialization, the algorithm is set up by creating a state of 320 bits by concatenating the fixed initialization vector, key, and the nonce, which are then passed through a rounds of permutation and XOR operation of the least significant $(320 - r)$ bits with the key padded with 0's (on the left) before proceeding to the next stage.

In the next stage, associated data is absorbed into the algorithm by dividing it into data sets of r bits each and the last data set is padded with a 1 followed by 0's to make the length equal to r.

The next stage processes the plaintext in a similar way and in addition, it generates ciphertexts in encryption and does the opposite for decryption. Processing of associated data and plaintext both have b rounds of the permutation.

The finalization stage generates a tag in encryption which is used in the finalization stage of decryption to verify if the processed data is correct.

C. Hashing

ASCON Hashing is based on the so-called SPONGE construction and is parameterized by 4 variables — maximal output length (h), rate (r), and internal number of rounds (a and b) for the permutation computation. Based on the value of parameter b, we have two variants of Hashing — ASCON-Hash (b = 6) and ASCON-HashA (b = 8). The input for the hashing algorithm is message data M and the output is the hash data H.

The operations can be divided into three sub-routines, namely:

1) Initialization
2) Absorbing message
3) Squeezing

At the initialization stage, the algorithm is initialized by creating a state of 320 bits by padding the fixed initialization vector with 0's on the LSB side, which is then passed through a rounds of permutation.

In the next stage, the message data is absorbed into the algorithm similar to the plain text processing stage mentioned above. It uses b rounds of permutation.

In the last stage, the ASCON state is first passed through a permutation rounds which generates the first block of hash data. The output is then passed through b rounds of permutations till all the blocks of hash data are generated.

III. (Unprotected) Hardware Implementation

A. Substitution Layer

The substitution layer employs a 5-bit SBox (see Section II). Possibly the most straightforward approach to implementing the SBox is by utilizing a look-up table. However, this method incurs a significant area cost.

An alternative approach involves using the coordinate functions. In general, it can be stated that the coordinate function-based implementation takes much less area than what would be required for a look-up-based implementation. Expressed in the algebraic normal form, the coordinate functions of the ASCON SBox are as given:

$$y_0 = x_4x_1 \oplus x_3 \oplus x_2x_1 \oplus x_2 \oplus x_1x_0 \oplus x_1 \oplus x_0$$
$$y_1 = x_4 \oplus x_2x_3 \oplus x_3 \oplus x_3x_1 \oplus x_2 \oplus x_1x_2 \oplus x_1 \oplus x_0$$
$$y_2 = x_4x_3 \oplus x_4 \oplus x_2 \oplus x_1 \oplus 1$$
$$y_3 = x_4x_0 \oplus x_3x_0 \oplus x_4 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0$$
$$y_4 = x_4x_1 \oplus x_4 \oplus x_3 \oplus x_1x_0 \oplus x_1$$

B. Linear Layer

The linear layer, which is discussed in Section II, can be realized using the right rotation and XOR. In our implementation, we opted to use only the XOR operation. This approach requires 2 XOR operations for each row, resulting in a total of 640 XOR operations for the entire layer[5]. Although both methods need the same area, XOR implementation is more flexible in terms of code. Additionally, it is easier to transform it into the threshold implementation of the linear layer using the latter method.

IV. Protection Against Side-Channel Attacks

Side-channel attacks, particularly those relying on information from power consumption or electromagnetic emanation, are of prominent concern while dealing with the physical security of the ciphers [4], [15], [16]. It has been systematically shown that a cipher with sufficient classical security claims falls short against an adversary equipped with a side-channel attack set-up. Therefore, understanding the attacks and finding low-cost countermeasures are among the top research priorities.

Side-channel attacks are based on the connection between the (a priori or learned) model and any intermediate variable in the implementation that might be leaking. Therefore, the countermeasures attempt to destroy the linkage of the model and the intermediate variables.

A. Theory of Threshold Implementation

The Threshold Implementation (TI) technique is renowned for its application in side-channel protected hardware. In the threshold-protected version of a cipher, the state of the cipher S is described by $d+1$ shares S_0, $S_1, \ldots, S_d$; so that $S \equiv S_0 \oplus S_1 \oplus \cdots \oplus S_d$. Each share is separately randomized, thereby removing the data dependent leakage. All the shares undergo separate processing with individually randomized inputs and are subsequently combined (through XOR of all the shares) at the end to cancel out their individual randomness. For more details, one may refer to, e.g., [15], [17], [18].

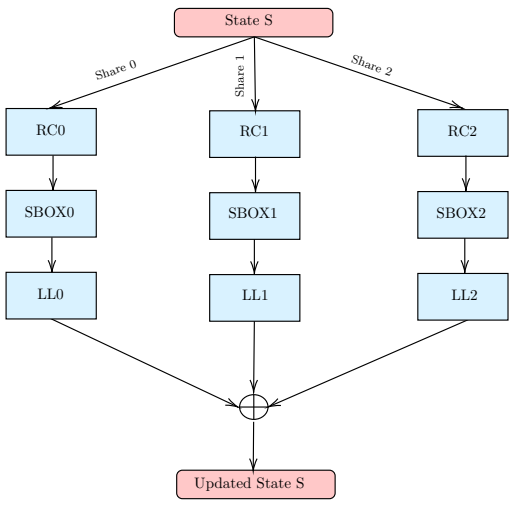


Figure 1: ASCON permutation under 3-share threshold (schematic)

Typically, the TI of an affine function is considered straightforward to implement, while that of a non-linear (in most block ciphers, the only non-linear component

[5]Equivalently, the linear layer can also be implemented using 320 XOR3 operations. The problem of implementation with higher input XOR gates is studied in the literature [12], [13]. Also, note that the same binary matrix format is considered in [14].

is the SBox) function is considered a strenuous task to accomplish. The TI of a given SBox can be realized either through *without decomposition* (the SBox is implemented as a combinational circuit) or *with decomposition* (the SBox is implemented as a sequential circuit) [18].

The ASCON permutation, for instance, consists of three stages as mentioned in Section II-A. Each share has a distinct constant addition layer, substitution layer, and linear layer that are cleverly designed so that the output of all three shares can be merged at the conclusion of the permutation phase to yield the same state value as in unprotected ASCON.

The schematic of the ASCON permutation with threshold is presented in Figure 1, which demonstrates how state S is divided into three shares. Each of the shares is then processed with distinct permutation processes, where RCi represents the round constant layer, SBOXi represents the substitution layer, and LLi represents the linear layer for share i. See Figure 3 for the schematic representation of ASCON permutation as a flow-chart.

B. ASCON SBox Threshold

The SBox, being the only non-linear component, is considered the hardest to implement in threshold. The minimum number of shares needed is 1 more than the algebraic degree of the SBox (thus, we need at least 3 shares). Also note that the total number of monomials in the combined shares is related to the product of the number of shares and the total number of monomials in the coordinate functions.

Our implementation uses a first-order (3 shares) ASCON SBox sharing generated using our in-house algorithm, which is described in [18]. Five sharing options were generated by shuffling the monomials by randomization. The corresponding benchmarks for those are shown in Table II. Subsequently, we chose sbox3 for our implementation because it occupies the least area (marked in Table II). The sharing is given afterward.

TABLE II: ASIC (STM 130nm) benchmarks for ASCON SBox

SBox	Cells	Area (μm^2)	Critical Path (ps)	Dynamic Power (nW)
sbox1	76	1002	641	10369
sbox2	77	1015	641	11670
sbox3	**68**	**897**	656	11720
sbox4	83	1088	641	9420
sbox5	75	989	724	12829

a) Share 0:

$$y_{00} = x_{40}x_{12} \oplus x_{42}x_{10} \oplus x_{42}x_{12} \oplus x_{32} \oplus x_{20}x_{12} \oplus x_{20} \oplus x_{22}x_{10} \oplus x_{22}x_{12} \oplus x_{22} \oplus x_{10}x_{00} \oplus x_{10}x_{02} \oplus x_{10} \oplus x_{12}x_{00} \oplus x_{12}$$
$$y_{10} = x_{42} \oplus x_{31}x_{22} \oplus x_{31}x_{11} \oplus x_{31}x_{12} \oplus x_{32}x_{21} \oplus x_{32}x_{22} \oplus x_{32}x_{11} \oplus x_{32}x_{12} \oplus x_{21}x_{11} \oplus x_{21}x_{12} \oplus x_{22}x_{11} \oplus x_{11}$$
$$y_{20} = x_{40}x_{32} \oplus x_{42}x_{30} \oplus x_{42} \oplus x_{22} \oplus x_{12}$$
$$y_{30} = x_{40}x_{01} \oplus x_{40} \oplus x_{41}x_{00} \oplus x_{30}x_{01} \oplus x_{31}x_{00} \oplus x_{31} \oplus x_{21} \oplus x_{11} \oplus x_{01}$$
$$y_{40} = x_{40}x_{12} \oplus x_{40} \oplus x_{42}x_{10} \oplus x_{42}x_{12} \oplus x_{42} \oplus x_{32} \oplus x_{10}x_{02} \oplus x_{10} \oplus x_{12}x_{00}$$

b) Share 1:

$$y_{01} = x_{40}x_{10} \oplus x_{40}x_{11} \oplus x_{41}x_{10} \oplus x_{41}x_{11} \oplus x_{30} \oplus x_{20}x_{10} \oplus x_{20}x_{11} \oplus x_{21}x_{10} \oplus x_{21} \oplus x_{10}x_{01} \oplus x_{11}x_{00} \oplus x_{11} \oplus x_{00} \oplus x_{01}$$
$$y_{11} = x_{41} \oplus x_{30}x_{21} \oplus x_{30}x_{10} \oplus x_{30}x_{11} \oplus x_{30} \oplus x_{31}x_{20} \oplus x_{31}x_{21} \oplus x_{31}x_{10} \oplus x_{31} \oplus x_{20}x_{11} \oplus x_{21}x_{10} \oplus x_{21} \oplus x_{10} \oplus x_{00} \oplus x_{01}$$
$$y_{21} = x_{41}x_{32} \oplus x_{41} \oplus x_{42}x_{31} \oplus x_{42}x_{32} \oplus x_{21} \oplus x_{11}$$
$$y_{31} = x_{40}x_{00} \oplus x_{40}x_{02} \oplus x_{42}x_{00} \oplus x_{42} \oplus x_{30}x_{00} \oplus x_{30}x_{02} \oplus x_{30} \oplus x_{32}x_{00} \oplus x_{32} \oplus x_{20} \oplus x_{22} \oplus x_{10} \oplus x_{00}$$
$$y_{41} = x_{41}x_{11} \oplus x_{41}x_{12} \oplus x_{41} \oplus x_{42}x_{11} \oplus x_{31} \oplus x_{11}x_{02} \oplus x_{11} \oplus x_{12}x_{01} \oplus x_{12}x_{02} \oplus x_{12}$$

c) Share 2:

$$y_{02} = x_{41}x_{12} \oplus x_{42}x_{11} \oplus x_{31} \oplus x_{21}x_{11} \oplus x_{21}x_{12} \oplus x_{22}x_{11} \oplus x_{11}x_{01} \oplus x_{11}x_{02} \oplus x_{12}x_{01} \oplus x_{12}x_{02} \oplus x_{02}$$
$$y_{12} = x_{40} \oplus x_{30}x_{20} \oplus x_{30}x_{22} \oplus x_{30}x_{12} \oplus x_{32}x_{20} \oplus x_{32}x_{10} \oplus x_{32} \oplus x_{20}x_{10} \oplus x_{20}x_{12} \oplus x_{20} \oplus x_{22}x_{10} \oplus x_{22}x_{12} \oplus x_{22} \oplus x_{12} \oplus x_{02}$$
$$y_{22} = x_{40}x_{30} \oplus x_{40}x_{31} \oplus x_{40} \oplus x_{41}x_{30} \oplus x_{41}x_{31} \oplus x_{20} \oplus x_{10} \oplus 1$$
$$y_{32} = x_{41}x_{01} \oplus x_{41}x_{02} \oplus x_{41} \oplus x_{42}x_{01} \oplus x_{42}x_{02} \oplus x_{31}x_{01} \oplus x_{31}x_{02} \oplus x_{32}x_{01} \oplus x_{32}x_{02} \oplus x_{12} \oplus x_{02}$$
$$y_{42} = x_{40}x_{10} \oplus x_{40}x_{11} \oplus x_{41}x_{10} \oplus x_{30} \oplus x_{10}x_{00} \oplus x_{10}x_{01} \oplus x_{11}x_{00} \oplus x_{11}x_{01}$$

Given the coordinate functions (Section III-A), note that the following conditions are satisfied (refer to [18] for relevant discussion):

- **Sharing of input variables:** $x_i = \bigoplus_{j=0}^{2} x_{ij}$ for $i = 0, 1, 2, 3, 4$.
- **Correctness:** $\bigoplus_{j=0}^{2} y_{ij} = y_i$ for $i = 0, 1, 2, 3, 4$.
- **Non-completeness:** At least one variable from $\{x_{i,0}, x_{i,1}, x_{i,2}\}$ is missing in each of $y_{j,0}, y_{j,1}, y_{j,2}$ for all $i, j \in \{0, 1, 2, 3, 4\}$. For instance, in our case (i.e., sbox3), $x_{01}, x_{11}, x_{21}, x_{30}, x_{31}$ and x_{41} are missing in y_{00}. Similar inferences can be drawn to the other output shares as well.
- **Uniformity:** All non-zero entries in the x_i ($\forall i$) versus $y_{j,k}$ ($\forall j, k$) frequency distribution table are equal. For instance, it can be verified that y_2 contains the following monomials: $\{x_{32}x_{41}, x_{32}x_{42}, x_{31}x_{41}, x_{30}x_{41}, x_{31}x_{42}, x_{20}, x_{42}, x_{41}, x_{30}x_{40}, x_{12}, x_{22}, x_{21}, x_{30}x_{42}, x_{11}, x_{40}, x_{10}, x_{31}x_{40}, 1, x_{32}x_{40}\}$; and each has the frequency of 256 in the distribution table.

There is another way to employ the threshold implementation for a given SBox, that involves decomposing it into two lower degree SBoxes. However, the ASCON SBox is quadratic (similar to that of BAKSHEESH [19]), so it is not possible to employ this method.

V. Protection Against Fault Injection Attacks

Since fault injection attacks [6] rely on some form of error propagation, the idea is to use redundancy. The same circuit is replicated (could be in the temporal domain or in the spatial domain) and depending on the power, we may need to duplicate or triplicate:

- Duplicate and compare works against *Differential Fault Attack* (DFA).
- Triplicate and take majority works against *Statistical Ineffective Fault Attack* (SIFA), although duplication-based SIFA countermeasures do exist (see [20], [21]).

In our implementation, we employed triplication and majority-based countermeasure techniques. Specifically, all the procedures are executed thrice, and the final output is determined by selecting the majority output from the three.

In a case where the three results are distinct, a random output is produced. Figure 2 illustrates the working of the triplication countermeasure. F1, F2, and F3 are the three instances of ASCON whose output is finally combined with the majority operation.

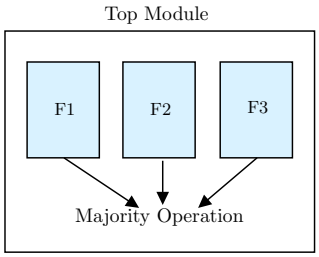


Figure 2: Triplication based countermeasure (schematic)

VI. Architecture and Interface

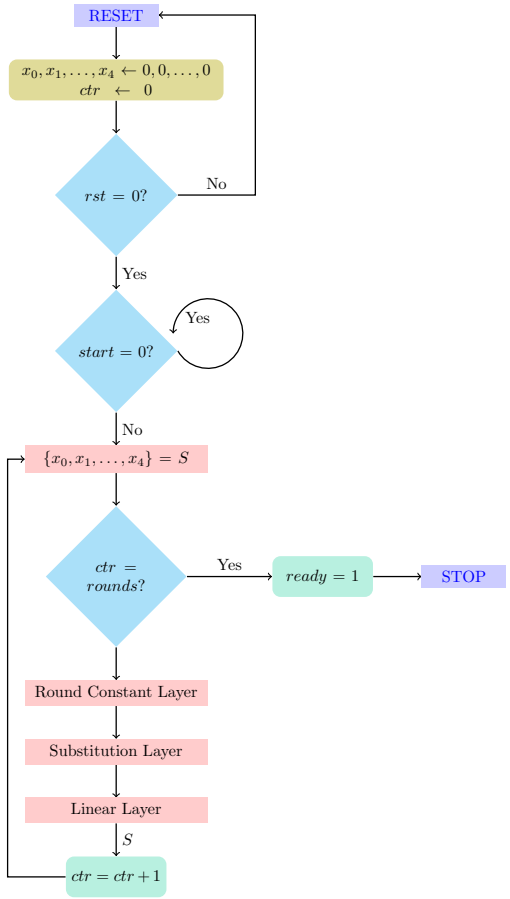


Figure 3: Finite state diagram for ASCON permutation (with round counter)

A. Inputs and Outputs

The input data consist of the key, 128-bit nonce, associated data (AD), plain text (Pt), control signals, and random numbers (generated using an external entropy source, which is not considered within the scope). The output data consists of the cipher text (Ct), 128-bit tag, and ready signals to indicate the end of the processing. Some of the important signals are explained below:

- keyxSI signal is of width 6 bits. The LSB bit carries the key information, and the other 5 bits carry random numbers, which are utilized for threshold implementation.
- noncexSI, plain_textxSI, and associated_dataxSI signals are of width 6 bits, and the distribution of the bits is similar to key.
- encryption_startxSI and decryption_startxSI are 1-bit control pulses that signal the start of encryption/decryption, respectively.
- r_64xSI signals are of 14 bits width, r_128xSI signals are of 3 bits width, and r_ptxSI signals are of 3 bits width carrying random numbers which are utilized for threshold implementation.
- message_authentication is a 1 bit output signal. It is 0 when the tag received from encryption + tag generation process is not matching the tag generated by the decryption + tag verification process.

Figure 4 represents the top-level diagram of the proposed ASCON architecture, which includes all the signals mentioned above.

B. Parameters

The followings parameters can be tuned to any specific configuration:

- k is the key size
- r is the rate (or the block size)
- a and b internal number of rounds which vary based on the ASCON variant
- l is the length of associated data
- y is the length of plain text
- TI is set to 1 for threshold implementation; else 0
- FP is set 1 for fault protection; else 0

Figure 3 shows a finite state diagram representation of the ASCON permutation. The finite state of ASCON permutation process begins with a reset state where the round counter is set to 0. The state then waits for the permutation_start signal to be activated before proceeding to divide the ASCON state S into five registers, which are updated after every round. Each round consists of three stages, namely round constant addition, substitution layer, and linear layer, as described in Section II. At the end of each round, the counter variable ctr is incremented by 1.

Figure 5 depicts the finite state diagram for the ASCON encryption process. The finite state of ASCON encryption process begins with RESET state, where all signals are reset to 0. The process then proceeds to the IDLE state, where the ASCON state is initialized based on the key, nonce, and cipher configuration. The system remains in this state until the encryption_start pulse is activated.

Upon receiving the start pulse, the system enters the INITIALIZE state, where the initialization process occurs. The next state, ASSOCIATED_DATA, is where the associated data is processed. The associated data is processed in multiple blocks, and the permutation process runs on each block one after the other. Once all the blocks are processed,

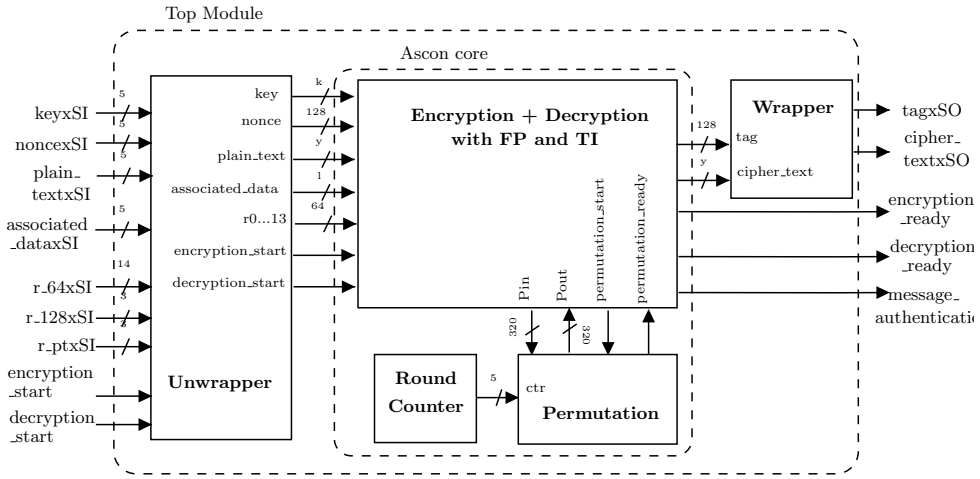


Figure 4: Top level diagram of ASCON hardware module

the system enters the PTCT state, where the plain text is processed, and cipher text is generated. This stage is similar to the ASSOCIATED_DATA stage. The final stage is the FINALIZE state, where the tag is generated. After this, the system enters the DONE state, where it waits for new data and the next start signal.

C. Entropy Requirement

In order to attain the desired security, it is essential that the some randomness is used (which is generated by an external source of entropy). We need randomness here for the following three purposes (see Section VI-B for the notations):

1) **Fault protection.** In cases where all three outputs differ, a random number is produced as the output of. The number of random bits used: $2 \times 128 + 2y$.
2) **Tag mismatch.** If the message_authentication signal is 0, then the output (plain text) of decryption + tag verification is a random number. The number of random bits used: $128 + y$.
3) **Threshold.** The masking algorithm uses random bits at different stages of the ASCON encryption/decryption/permutation processes. The number of random bits used: $4k + 4 \times 128 + 4l + 4y + 14 \times 64$.

Therefore, we count the total number of random bits required as: $4k + 4l + 7y + 1792$.

VII. ASIC and FPGA Benchmarks

In Table III, we show the benchmark results for the following configurations on ASIC (STM 130nm) and FPGA (Kintex-7) with the following settings: (α) unprotected ASCON, (β) ASCON with TI, (γ) ASCON with fault protection, (δ) ASCON with TI and fault protection. The variant used here is ASCON-128 with a length of 32 bits for both plain text (Pt) and associated data (AD). The results indicate that the ASIC implementation of ASCON with threshold requires approximately 4.5× the area of (α), while the implementation with fault protection requires approximately 2.83× the area of (α), and the implementation with both threshold and fault protection requires approximately 12.57× the area (α).

However, in FPGA implementation, we noticed that the area of ASCON with threshold is approximately 3.15× the area of (α), while the implementation with fault protection remains almost same. We speculate that this behavior might be attributed to the tool's optimization process, wherein it could be removing the extra copies during its optimization steps.

TABLE III: ASCON-128 hardware benchmarks (unprotected and protected)

(a) ASIC (STM 130nm)

Configuration	Cells	Area (μm^2)	Critical Path (ps)	Dynamic Power (mW)
Unprotected	12853	184289	20853	1.34
TI	57898	763912	20504	9.00
FP	36389	519621	20840	3.79
FP and TI	161531	2080207	20956	28.8

(b) FPGA (Kintex-7)

Configuration	LUT	FF	Frequency (MHz)
Unprotected	2809	1830	181.82
TI	8841	5419	158.73
FP	2817	1830	181.82
TI and FP	8841	5419	158.73

The ASIC and FPGA benchmarks for (encryption + tag generation) and (decryption + tag verification) are presented in Tables IV and V respectively, showcasing both the protected and unprotected configurations. The data highlights that the area required in the threshold implementation (TI) configuration is approximately 3.52× larger compared to the configuration without threshold implementation in both circuits.

In Table VI, we show the benchmark for ASCON hash on Kintex-7 FPGA and STM 130nm ASIC technologies. We include unprotected hash and protected hash in the benchmark for comparison. The ASCON-Hash variant is used, with the length of the message being 40 bits and the length of the hash output being 256 bits.

Table VII shows the area occupied in number of cells using the TSMC 65nm technology, for varied sizes of (Pt, AD) for both protected and unprotected versions of ASCON-128. Similarly, Table VIII shows the area of protected and unprotected versions of ASCON-Hash with

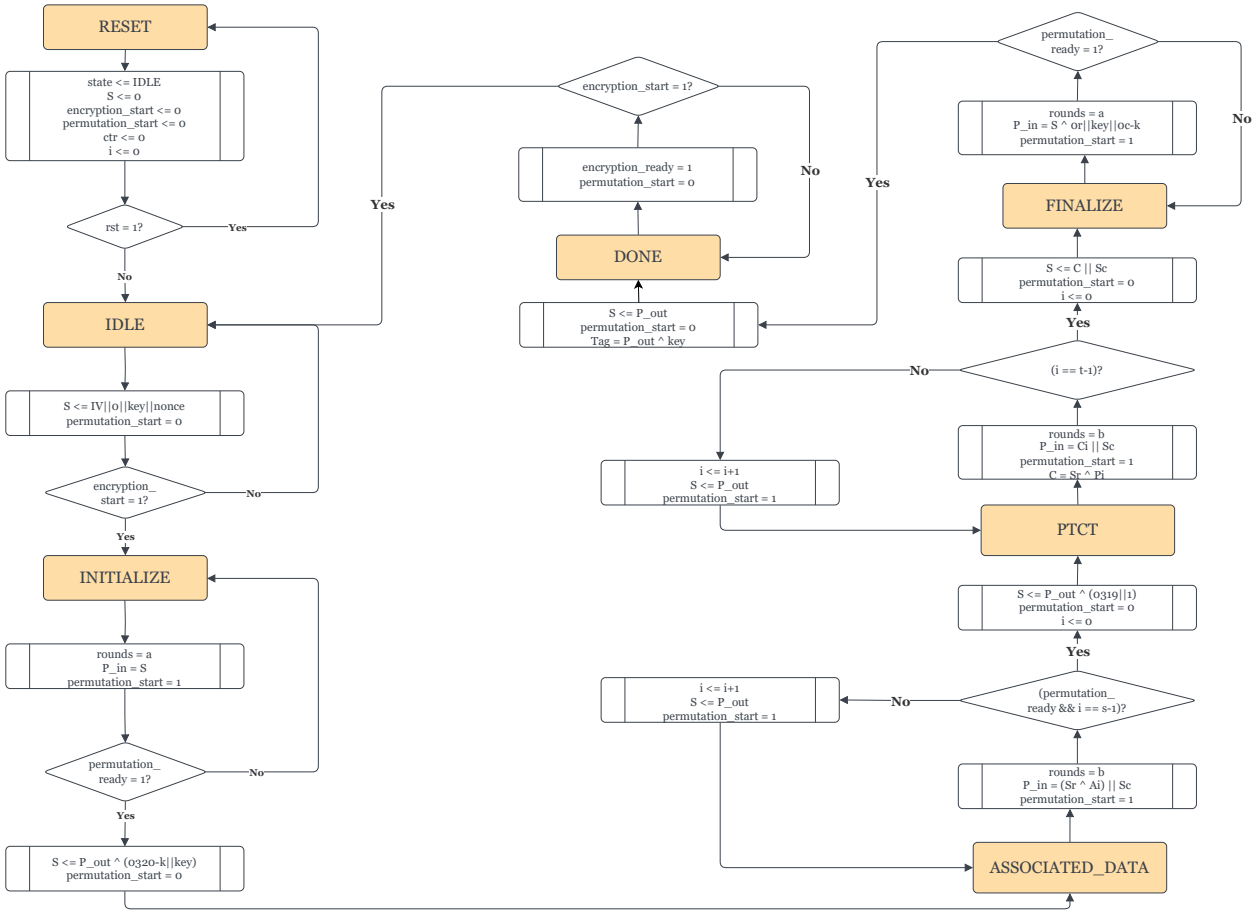


Figure 5: Finite state diagram for ASCON hardware implementation

TABLE IV: Encryption + Tag generation benchmarks for ASCON-128

(a) ASIC

Configuration	Cells	Area (μm^2)	Critical Path (ps)	Dynamic Power (mW)
Unprotected	7585	107317	36523	0.97
TI	27067	393279	36294	4.48
FP	22034	297077	36537	2.68
TI and FP	75647	1053039	36124	12.66

(b) FPGA

Configuration	LUT	FF	Frequency (MHz)
Unprotected	1232	822	200.00
TI	4332	2452	156.74
FP	1232	822	200.00
TI and FP	4623	2448	153.85

TABLE V: Decryption + Tag Verification benchmarks for ASCON-128

(a) ASIC (STM 130nm)

Configuration	Cells	Area (μm^2)	Critical Path (ps)	Dynamic Power (mW)
Unprotected	7609	108864	36975	0.86
TI	28441	399630	37517	4.54
FP	21727	301658	36731	2.48
TI and FP	77376	1067886	36522	12.51

(b) FPGA

Configuration	LUT	FF	Frequency (MHz)
Unprotected	1216	821	188.68
TI	4423	2445	163.93
FP	1216	821	188.68
TI and FP	4423	2445	163.93

varying message size.

In Table IX, we summarise our area benchmarks for the circuits. The numbers indicate the ratio of the cells/LUT of the circuit corresponding with that of its unprotected version.

Figure 6 shows the plot of latency against varying (plaintext, associated data) sizes for ASCON-AEAD (Figure 6a) and ASCON-Hash (Figure 6b). An increase in the lengths of the AD and PT fields is associated with a corresponding increase in latency. This behavior aligns with the design of ASCON, where the number of permutation rounds scales with the input size.

Figure 7 shows the TSMC 65nm ASIC *amoeba* view of the layout diagrams of ASCON-128 with PT and AD of 32 bits each. Comparing the two images, it can be seen that the substitution and the linear layers occupy a large amount of area in the protected implementation.

VIII. Conclusion and Outlook

This work presents a full-stack hardware suite for ASCON hash and AEAD [2]. There seems to be no comprehensive side-channel and fault attack-protected hardware implemen-

TABLE VI: `ASCON-Hash` hardware benchmarks (unprotected and protected)

(a) ASIC (STM 130nm)

Configuration	Cells	Area (μm^2)	Critical Path (ps)	Dynamic Power (mW)
Unprotected	4980	76765	7423	0.53
TI	24058	321471	8568	3.78
FP	15674	239740	8862	1.86
TI and FP	69679	925986	10377	11.79

(b) FPGA (Kintex-7)

Configuration	LUT	FF	Frequency (MHz)
Unprotected	847	911	192.31
TI	4795	2719	172.11
FP	847	911	192.31
TI and FP	4840	2719	172.41

TABLE VII: Area benchmarks for ASIC (TSMC 65nm) with varying (Pt, AD) sizes for `ASCON-128`

(a) Unprotected

AD size	Plaintext size			
	32	64	128	256
32	13611	14677	17423	21951
64	13817	15021	17691	22139
128	14155	15340	17829	22379
256	14615	15728	18363	22916

(b) TI

AD size	Plaintext size			
	32	64	128	256
32	53173	55691	64739	78151
64	54266	57803	66439	79285
128	55225	58334	68232	79821
256	57264	61571	70067	82816

(c) FP

AD size	Plaintext size			
	32	64	128	256
32	38234	41392	49875	63989
64	38643	42228	50531	64233
128	39269	42756	50624	64637
256	39865	43226	51414	65477

(d) TI and FP

AD size	Plaintext size			
	32	64	128	256
32	149078	157240	184447	223151
64	151867	162987	188732	224129
128	152982	162367	191092	223376
256	155365	168424	193454	229068

tation of this cipher, so we expect our work will become useful in the future.

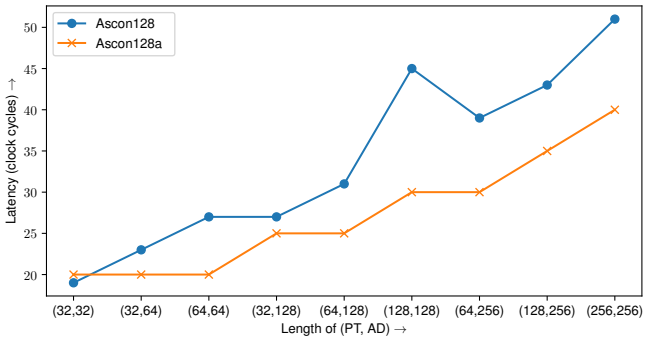
Note that we exclusively consider triplication-based SIFA countermeasure for the interest of simplicity. The overall area can be reduced by using a more complicated duplication-based SIFA countermeasure as explained in [20], [21]. This can be covered in a future scope.

A hardware application programming interface for LWC is proposed by GMU [22]. It is possible to make our code compliant to the API (somewhat comparable to [23]), and this task is left as a future work.

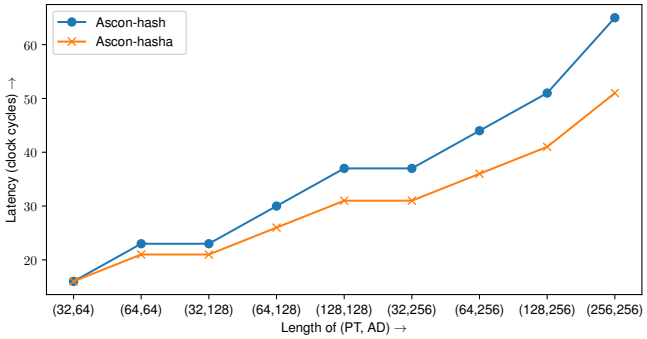
Finally, one may be interested in evaluating the effect of side-channel attacks on unprotected implementation and how the protected implementation resists it.

TABLE VIII: Area benchmarks for ASIC (TSMC 65nm) with varying message size for `ASCON-Hash` (256-bit)

Configuration	Message size			
	32	64	128	256
Unprotected	5983	6239	6344	6815
TI	24168	24888	25433	26938
FP	18495	19075	19007	19656
TI and FP	69969	71455	72047	74255



(a) AEAD



(b) Hash

Figure 6: Throughput benchmarks on FPGA

REFERENCES

[1] A. Kandi, A. Baksi, T. Gerlich, S. Guilley, P. Gan, J. Breier, A. Chattopadhyay, R. R. Shrivastwa, Z. Martinásek, and S. Bhasin, "Hardware implementation of ascon," NIST LWC Workshop, 2023. [Online]. Available: https://csrc.nist.gov/csrc/media/Events/2023/lightweight-cryptography-workshop-2023/documents/accepted-papers/07-hardware-implementation-of-ascon.pdf 1

[2] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2," Submission to NIST, 2019, https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf. 1, 7

[3] V. Srivastava, N. Gupta, A. Jati, A. Baksi, J. Breier, A. Chattopadhyay, S. K. Debnath, and X. Hou, "Ascon-sign," NIST PQC Additional Round 1 Candidates, 2023, https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/Ascon-sign-spec-web.pdf. 1

[4] E. Peeters, *Advanced DPA Theory and Practice: Towards the Security Limits of Secure Embedded Circuits*, 1st ed. Springer-Verlag New York, 2013. [Online]. Available: https://www.springer.com/gp/book/9781461467823 1, 3

[5] A. Baksi, *Side Channel Attack*. Singapore: Springer Singapore, 2022, pp. 99–108. [Online]. Available: https://doi.org/10.1007/978-981-16-6522-6_4 1

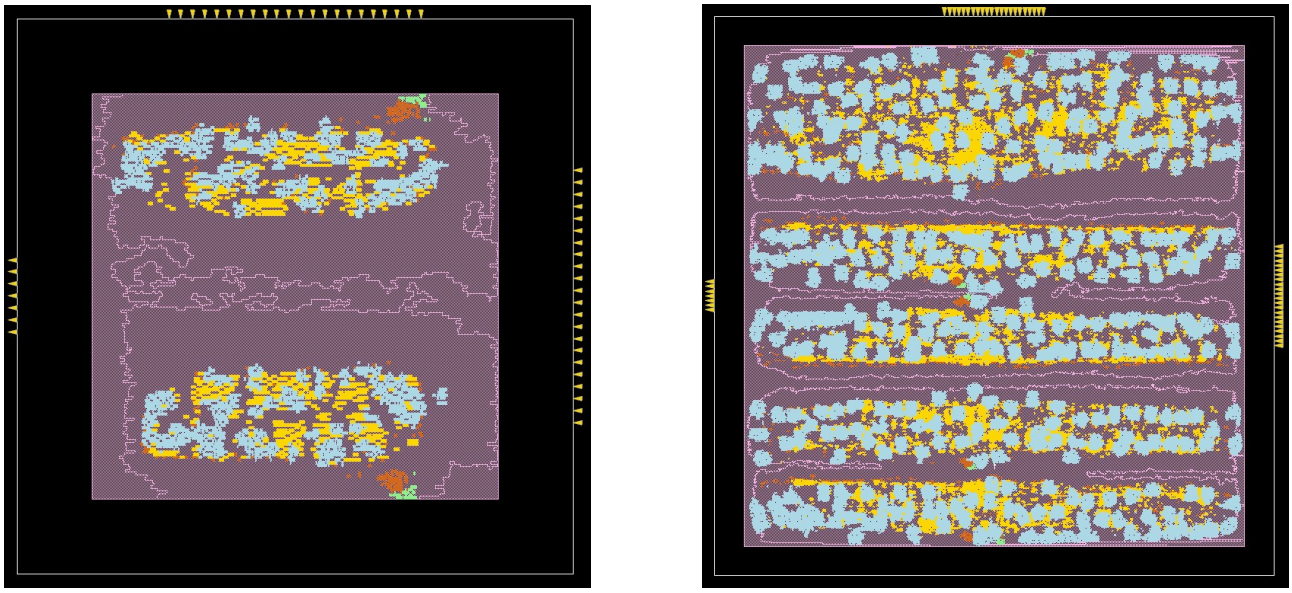
[6] A. Baksi, S. Bhasin, J. Breier, D. Jap, and D. Saha, "A Survey on Fault Attacks on Symmetric Key Cryptosystems," *ACM Comput. Surv.*, vol. 55, no. 4, pp. 86:1–86:34, 2023. [Online]. Available: https://doi.org/10.1145/3530054 1, 4

[7] X. Wei, M. El-Hadedy, S. Mosanu, Z. Zhu, W.-M. Hwu, and X. Guo, "RECO-HCON: A High-Throughput Reconfigurable Compact ASCON Processor for Trusted IoT," in *2022 IEEE 35th International System-on-Chip Conference (SOCC)*, 2022, pp. 1–6. 2

[8] S. Khan, W.-K. Lee, and S. O. Hwang, "Evaluating the Performance of Ascon Lightweight Authenticated Encryption for AI-Enabled IoT Devices," in *2022 TRON Symposium (TRONSHOW)*, Dec 2022, pp.

TABLE IX: Summary of relative area requirement for ASCON hardware benchmarks

Configuration (TI, FP)	ASIC (STM 130 nm)				FPGA (Kintex-7)			
	(✗, ✗)	(✗, ✓)	(✓, ✗)	(✓, ✓)	(✗, ✗)	(✗, ✓)	(✓, ✗)	(✓, ✓)
Enc. + T. gen.	1×	3.57×	2.90×	9.97×	1×	3.51×	1×	3.51×
Dec. + T. ver.	1×	3.73×	2.85×	10.17×	1×	3.63×	1×	3.63×
ASCON-128	1×	4.50×	2.83×	12.57×	1×	3.15×	1×	3.15×
ASCON-Hash	1×	4.83×	3.15×	14.00×	1×	5.66×	1×	5.66×



(a) Unprotected **(b)** Protected (TI + FP)

Substituition Layer ◇ Round Constant ◇ Linear Layer ◇ Round Counter

Figure 7: ASIC (TSMC 65nm) layout for ASCON implementations

1–6. 2

[9] K. Raj and S. Bodapati, "FPGA Based Light Weight Encryption of Medical Data for IoMT Devices using ASCON Cipher," in *2022 IEEE International Symposium on Smart Electronic Systems (iSES)*, 2022, pp. 196–201. 2

[10] H. Gross, E. Wenger, C. Dobraunig, and C. Ehrenhöfer, "Ascon Hardware Implementations and Side-Channel Evaluation," *Microprocessors and Microsystems*, vol. 52, 11 2016. 2

[11] S. H. Prasad, F. Mendel, M. Schläffer, and R. Nagpal, "Efficient Low-Latency Masking of Ascon without Fresh Randomness," Cryptology ePrint Archive, Paper 2023/1914, 2023. [Online]. Available: https://eprint.iacr.org/2023/1914 2

[12] A. Baksi, V. A. Dasu, B. Karmakar, A. Chattopadhyay, and T. Isobe, "Three Input Exclusive-OR Gate Support for Boyar-Peralta's Algorithm," in *INDOCRYPT*, vol. 13143. Springer, 2021, pp. 141–158. [Online]. Available: https://doi.org/10.1007/978-3-030-92518-5_7 3

[13] Q. Liu, W. Wang, L. Sun, Y. Fan, L. Wu, and M. Wang, "More inputs makes difference: Implementations of linear layers using gates with more than two inputs," *IACR Cryptol. ePrint Arch.*, p. 747, 2022. [Online]. Available: https://eprint.iacr.org/2022/747 3

[14] S. Roy, A. Baksi, and A. Chattopadhyay, "Quantum Implementation of ASCON Linear Layer," Cryptology ePrint Archive, Paper 2023/617, 2023, https://eprint.iacr.org/2023/617. 3

[15] B. Bilgin, "Threshold Implementations As Countermeasure Against Higher-Order Differential Power Analysis," Ph.D. dissertation, Katholieke Universiteit Leuven and University of Twente, 2015, https://www.esat.kuleuven.be/cosic/publications/thesis-256.pdf. 3

[16] V. Lomné, "Power and electro-magnetic side-channel attacks: Threats and countermeasures," Ph.D. dissertation, Docteur de l'Université Montpellier II, 2010, https://sites.google.com/site/victorlomne/research. 3

[17] A. Jati, N. Gupta, A. Chattopadhyay, S. K. Sanadhya, and D. Chang, "Threshold implementations of gift: A trade-off analysis," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 2110–2120, 2020. [Online]. Available: https://doi.org/10.1109/TIFS.2019.2957974 3

[18] A. Baksi, S. Guilley, R.-R. Shrivastwa, and S. Takarabt, "From substitution box to threshold," Cryptology ePrint Archive, Paper 2023/633, 2023, https://eprint.iacr.org/2023/633. 3, 4

[19] A. Baksi, J. Breier, A. Chattopadhyay, T. Gerlich, S. Guilley, N. Gupta, T. Isobe, A. Jati, P. Jedlicka, H. Kim, F. Liu, Z. Martinásek, K. Sakamoto, H. Seo, R. Shiba, and R. R. Shrivastwa, "Baksheesh: Similar yet different from gift," Cryptology ePrint Archive, Paper 2023/750, 2023, https://eprint.iacr.org/2023/750. 4

[20] A. Baksi, V. B. Y. Kumar, B. Karmakar, S. Bhasin, D. Saha, and A. Chattopadhyay, "A Novel Duplication Based Countermeasure To Statistical Ineffective Fault Analysis," Cryptology ePrint Archive, Report 2020/1268, 2020, https://eprint.iacr.org/2020/1268. 4, 8

[21] A. Baksi, S. Bhasin, J. Breier, A. Chattopadhyay, and V. B. Y. Kumar, "Feeding three birds with one scone: A generic duplication based countermeasure to fault attacks (extended version)," Cryptology ePrint Archive, Paper 2020/1542, 2020, https://eprint.iacr.org/2020/1542. 4, 8

[22] J.-P. Kaps, W. Diehl, M. Tempelmeier, E. Homsirikamol, , and K. Gaj, "Hardware api for lightweight cryptography v1.1 (with support for sca-protected implementations)," GMU ATHENa website, 2022, https://cryptography.gmu.edu/athena/LWC/LWC_HW_API_v1_1.pdf. 8

[23] A. Sathvik, T. Rahul, A. Baksi, and V. Pudi, "Hardware implementation of spoc-128," Cryptology ePrint Archive, Paper 2022/119, 2022, https://eprint.iacr.org/2022/119. 8