SCA Resistance Analysis on FPGA Implementations of Sponge based MAC-PHOTON

N. Nalla Anandakumar

Hardware Security Research Group, Society for Electronic Transactions and Security, India.

nallananth@gmail.com

Abstract. PHOTON is a lightweight hash function which was proposed by Guo et al. in CRYPTO 2011. This is used in low-resource ubiquitous computing devices such as RFID tags, wireless sensor nodes, smart cards and mobile devices. PHOTON is built using sponge construction and it provides a new MAC function called MAC-PHOTON. This paper deals with FPGA implementations of MAC-PHOTON and their side-channel attack (SCA) resistance. First, we describe three architectures of the MAC-PHOTON based on the concepts of iterative, folding and unrolling, and we provide their performance results on the Xilinx Virtex-5 FPGAs. Second, we analyse security of the MAC-PHOTON against side-channel attack using a SASEBO-GII development board. Finally, we present an analysis of its Threshold Implementation (TI) and discuss its resistance against first-order power analysis attacks.

Keywords: SCA, Lightweight Cryptography, Sponge functions, MAC, PHOTON, Threshold Implementation.

1 Introduction

Hash functions are one of the most important and invaluable primitives in modern cryptography. Recently, Bertoni et al. [6] proposed a new way of building hash functions from a fixed permutation which is called sponge function. A sponge function H is a one-way function that converts arbitrary-length message M into variable-length hash code H(M) (or digest). In practice, sponge based hash functions are very useful for constructing Message Authentication Codes (MACs) [5]. A MAC algorithm accepts as input a secret key K and a message M of arbitrary-length and produces a short-tag as output. The purpose of a MAC is to provide integrity and authenticity assurances on the message.

Recently, a sponge based hash function called PHOTON [14] has been proposed, especially for usage in lightweight security devices. The design structure of PHOTON has an AES like internal permutation. In this study, we present the iterative, folding and unrolling architectures of the MAC-PHOTON on FPGA (Field-Programmable Gate Array). The proposed constructions are suited for the lightweight cryptographic applications such as FPGA-based RFID tags [13], FPGA-based wireless sensor nodes [12,25]. Moreover, the side-channel security

resistance of these non-serialised implementations of MAC-PHOTON has not been evaluated quantitatively.

In 2013, Susana et al. [11] presented an analysis of side channel resistance of HMAC [3] based on fully serialized implementation of PHOTON [14] hash functions. They make strong assumptions on the target implementation to discover the state information, and they use same key variant for HMAC prefix-suffix construction. They also mention that their implementation is not suitable for high-speed resource constrained devices. Our goal in this work is to present implementations suitable for high-speed resource constrained devices.

Side-channel attacks on a non-serialised hardware implementation of MAC-PHOTON would be much more challenging to implement. Up until now, there has not been much prior work along this direction. In a side-channel attack, an adversary exploits the secret information which is leaking from a physical implementation of the algorithm. In MAC-PHOTON construction, obtaining the full secret information or even partial disclosure of secret information can lead to a forgery of the MAC for arbitrary messages. This work deals with security of three FPGA implementations against side-channel analysis such as correlation power analysis (CPA) [10]. We also provide Threshold Implementation (TI) of MAC-PHOTON and discuss its resistance against first-order power analysis attacks. To the best of our knowledge, this is the first security analysis of the unprotected and protected of MAC-PHOTON against first-order CPA attacks.

Our contributions. The primary goal of this work is to provide an analysis of the SCA resistance of the sponge based MAC construction that uses either iterative or folding or unrolling based architecture of PHOTON hash function. We also analyse security of threshold implementation of the MAC-PHOTON against first-order CPA attacks. Our contributions are summarized as follows:

- 1. Our first contribution is to present the iterative, folding and unrolling architectures of the MAC-PHOTON, and to provide their performance results on the Xilinx Virtex-5 FPGAs. Our three implementations yield better throughput per area ratio when compared with existing FPGA implementation of PHOTON-80/20/16 [2,11] and HMAC-PHOTON-80/20/16 [11].
- 2. Our second contribution is to present the side channel security analysis of the iterative, folding and unrolling architectures of the MAC-PHOTON against first-order CPA attack. As a result, the iterative, folding and unrolling architectures have resistance against side channel attack up to 10000, 8000, 50000 messages, respectively. Moreover, our MAC-PHOTON implementations provide better security compared to Susana et al. [11].
- 3. Our third contribution is to present the iterative, folding based threshold implementations of MAC-PHOTON, and to analyse their security against first-order CPA attack. As a result, our implementations yield better throughput per area ratio when compared with existing FPGA implementations of PHOTON-80/20/16 [2,11] and HMAC-PHOTON-80/20/16 [11]. Moreover, our implementations are resistant against first-order CPA attacks even if an attacker is capable of measuring 100,000 power traces.

The rest of this paper is organised as follows. First we provide the several preliminaries on PHOTON, SCA and MAC calculation in Section 2. In Section 3 we present the hardware architecture of the MAC-PHOTON structure and implementation results for Xilinx FPGAs. In Section 4 we describe a CPA attack strategy to analyze its resistance against side-channel attacks. We then furnish its experimental results. In Section 5 we present the threshold implementation of the MAC-PHOTON-80/20/16 and to evaluate their security against first-order CPA attacks. The paper concludes in Section 6.

2 Technical Background

In this section, we give a brief description of the PHOTON hashing algorithm, followed by an overview of the MAC-PHOTON constructions and also give an overview of the side channel analysis.

2.1 PHOTON Description

PHOTON is a cryptographic hash function based on the sponge construction with arbitrary-length input and variable-length output. Each PHOTON hash function is denoted by PHOTON-n/r/r', where its input bitrate r, its output bitrate r', and its hash output size n. There are five hash function in the PHOTON family: PHOTON-80/20/16, PHOTON-128/16/16, PHOTON-160/36/36, PHOTON-224/32/32, and PHOTON-256/32/32. The size of the internal state (t bits, t=c+r; r input bitrate and c capacity) depends on the hash output size.

PHOTON has three phases: 1) initialization, 2) absorbing and 3) squeezing. In the initialization phase, the input message is padded and cut blocks of r bits. During the absorption phase, the r-bit input message blocks are XORed into the first r bits of the state and then interleaved with the t-bit permutation function P. Once all message blocks have been handled the squeezing phase starts. During this phase, the extracting r' bits from the bitrate part of the internal state and then applying the permutation P on it. The squeezing process continues until the proper digest size n is reached.

The PHOTON internal permutation P is also AES-like permutations. It also consists of 12 rounds, each round is composed as the application of the following four operations:

- AddConstants (AC): first column of the internal state is bitwise XORed with round and internal constants;
- SubCells (SC): the PRESENT S-box [8] is applied to the internal state;
- ShiftRows (SR): cell row i of the internal state is cyclically shifted by i positions to the left;
- MixColumnsSerial (MCS): each cell column of the internal state is transformed by multiplying it once with MDS matrix $(A)^d$ (or d times with matrix A).

We focus on PHOTON-80/20/16 in our analysis, because it is the lightest and the simplest version of the family. It presents an internal state of (5×5) cells and each cell represents a 4-bit nibble. The PHOTON-80/20/16 MDS matrix $(A)^5$ is defined as follows:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 2 & 9 & 9 & 2 \end{pmatrix}; \qquad (A)^5 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 2 & 9 & 9 & 2 \end{pmatrix}^5 = \begin{pmatrix} 1 & 2 & 9 & 9 & 2 \\ 2 & 5 & 3 & 8 & 13 \\ 13 & 11 & 10 & 12 & 1 \\ 1 & 15 & 2 & 3 & 14 \\ 14 & 14 & 8 & 5 & 12 \end{pmatrix}$$

2.2 The MAC Construction

For sponge construction, the output is only a small part of the squeezing phase and hence it is protected from length extension weakness which is mentioned in [5,7,14]. Thus, the HMAC nested construction does not require for sponge based constructions [4,5,7,14,26]. Indeed, we simply prepend the key to the message and then we apply the sponge construction to generate a MAC as recommended by PHOTON [14] designers.

$$MAC(M, K) = H(K||M) \tag{1}$$

We will denote the MAC algorithm that uses PHOTON-80/20/16 to instantiate H by the term "MAC-PHOTON-80/20/16". We give in Figure 1 the construction of the sponge based MAC-PHOTON-80/20/16. In the first step, the t-bit internal state A_i is initialized to initial vector $A_0 = IV$. Then, the secret key and the input message is split into blocks of r-bits each, which are denoted by key $K = (k_0, k_1, ..., k_{n-1})$ and message $M = (m_0, m_1, ..., m_{n-1})$ respectively. The absorbing phase, the r-bit input blocks are XORed with r leftmost bits of the state, then interleaved with the permutation function P. During this phase, the key blocks are processed first and then the message blocks are processed. Once all key and message blocks have been absorbed, the squeezing phase begins.

In the squeezing phase, the first r'-bits of the state are returned as output blocks z_i from the internal state, and then interleaved with the permutation function P. The squeezing process continues until the proper MAC $(z_0||...||z_{n-1})$ size is reached. In the above MAC construction, obtaining the actual secret key (K), or recovering the internal state A_i would be enough to forge the MAC for arbitrary messages.

2.3 Side Channel Analysis

Side channel attacks have become an important field of cryptographic research. It is a class of attack that exploits information leaking from physical implementation of cryptosystems. Differential Power Analysis (DPA) [19] and Correlation Power Analysis (CPA) [10] are most common forms of the side channel analysis. DPA exploits the relationship between power consumptions and data generated

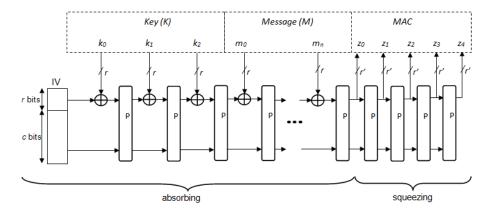


Fig. 1. The block diagram of the sponge based MAC-PHOTON-80/20/16 construction

during execution. In a CPA attack, the secret key can be derived by using the Pearson's correlation coefficient to correlate the recorded power consumption (so often power trace) with the hypothetical power consumption model. The hypothetical power consumption model is computed by using a Hamming Distance (HD) model [10]. The HD represents the number of bit-flips between two clock cycles. Side channel attack on MAC based on several hash functions was studied in [24], [9] and [27]. In this paper, we demonstrate CPA attack on MAC-PHOTON-80/20/16.

3 FPGA implementation of the MAC-PHOTON-80/20/16

In this section, we present three FPGA implementations of the MAC-PHOTON based on the concepts of iterative, folding and unrolling, and to provide their performance results on the Xilinx Virtex-5 FPGAs.

In order to demonstrate the security of the MAC-PHOTON-80/20/16 construction against CPA attacks, we implemented the MAC-PHOTON-80/20/16 in VerilogHDL and targeted Xilinx Virtex-5 FPGA (XC5VLX50-1FFG324). We used Mentor Graphics ModelSimPE for simulation purposes and Xilinx ISE v13.4 for synthesizing and implementation purposes. For MAC-PHOTON-80/20/16 analysis, we have selected 256 bits (260 bits with required padding) message length and 60 bits key length. A 60-bit key provides security for up to 30,000 messages per key [14]. For higher key length, the higher versions of the PHOTON hash core must be replaced as recommended by PHOTON [14]. We give in Table 1 the detailed synthesis results of the iterative, folding and unrolling based implementations of the MAC-PHOTON. The iterative architecture computes one round per clock cycle, while the folding architecture computes one round per 2 clock cycles. In the unrolling architecture computes 12 rounds per clock cycle.

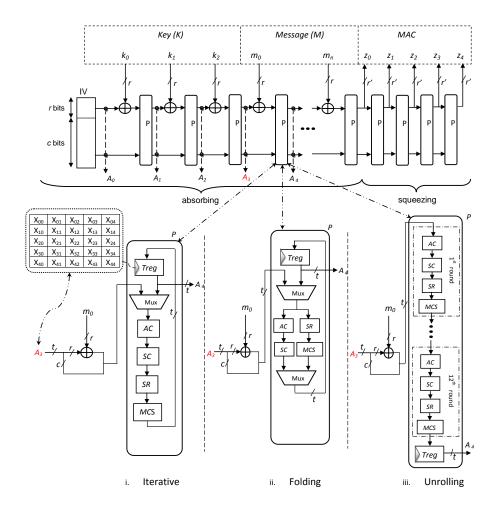


Fig. 2. The block diagram of the iterative, folding, unrolling implementations of the MAC-PHOTON-80/20/16

Iterative: The main goal of the design is moderate throughput and area requirements. We give in Figure 2 the block diagram of the basic iterative (denoted (i) in Figure 2) FPGA implementation of MAC-PHOTON-80/20/16. Initially, the key value and input message value split into blocks of r-bits (20-bit). In absorbing phase, first 3 key blocks are processed, after that 13 message blocks are processed, where each block consists of 12 rounds. The data register Treg is updated every round after processing AC, SC, SR, and MCS operations in one clock cycle. Hence, it requires 192 clock cycles to process 16 blocks (where, 36 clock cycles for 3 key blocks and 156 clock cycles for 13 message blocks). In squeezing phase, r'-bit (16-bit) of 5 output blocks are extracted from the internal state which requires 48 clock cycles (i.e. only 4 permutations are executed). Therefore, 240

clock cycles are required in order to complete both phases. We obtain 302 slices, while the throughput reaches 287.83 Mbps. As can be seen from the Table 1, our work seems to require much less area than most ciphers [17,16,18,15] and also yields a better throughput per area ratio compared to MD5 [17], SHA-I [16], SHA-256 [18], PHOTON-80/20/16 [2,11] and HMAC-PHOTON-80/20/16 [11].

Folding: The main goal of the design is reasonable throughput and better area requirements. In Figure 2, horizontal folding by a factor of two is demonstrated (denoted (ii) in Figure 2). In this architecture, a half of a round is implemented as combinational logic, and the entire round is executed using 2 clock cycles. The data register Treq is updated every half of a round (either after processing AC, and SC operations or after processing SR, and MCS operations in one clock cycle). The datapath width and state size are stays the same as in the basic iterative architecture. Hence, 384 clock cycles are required to process 16 blocks in absorbing phase and 96 clock cycles (i.e. only 4 permutations are executed) are required to process 5 output blocks in squeezing phase. Therefore, 480 clock cycles are required in order to complete both the phases. We obtain 251 slices, while the throughput reaches 171.42 Mbps. As seen from the Table 1, our folding based MAC-PHOTON implementation seems to require much less area than most ciphers [17,16,18,15] and also yields a better throughput per area ratio compared to MD5 [17], SHA-l [16], SHA-256 [18], PHOTON-80/20/16 [2,11] and HMAC-PHOTON-80/20/16 [11].

Unrolling: The main goal of the design is on high throughput and not on low area requirements. We give in Figure 2 the block diagram of the unrolling (denoted (iii) in Figure 2) FPGA implementation of MAC-PHOTON-80/20/16. The combinational logic of a round is replicated, so now 12 rounds of internal permutation P are executed in one clock cycle. Thus, the data register *Treg* is updated after every permutation P. Hence, it requires 16 clock cycles to process 16 blocks in absorbing phase and 4 clock cycles (i.e. only 4 permutations are executed) are required to process 5 output blocks in squeezing phase. Therefore, 20 clock cycles are required in order to complete both the phases. We obtain 1066 slices, while the throughput reaches 508.6 Mbps. As seen from the Table 1, our unrolling based MAC-PHOTON implementation seems to require much less area than KECCAK-256 [18,15] and also yields a better throughput per area ratio compared to MD5 [17], SHA-1 [16], SHA-256 [18], PHOTON-80/20/16 [2,11] and HMAC-PHOTON-80/20/16 [11].

4 Side channel attack Resistance of MAC-PHOTON-80/20/16

In this section, we present a CPA attack strategy to analyze the security of MAC-PHOTON against side-channel attack using our communication interface (see Appendix A) on a SASEBO-GII development board, especially CPA with Hamming Distance model and we furnish experimental results of it.

Table 1. Performance Results of the MAC-PHOTON-80/20/16 and TI implementation of MAC-PHOTON-80/20/16 on Virtex-5-xc5vlx50.

				Max.	Total Nun	nber of		
Design	Area	LUTs	FFs	freq	Clock Cycle	es (cycles)	T.put	T.put/Area
	(slices)			(MHz)	internal	whole hash	(Mbps)	(Mbps/slices)
					permutation P	function H		
iterative	302	508	415	172.7	12	240	287.83	0.95
folding	251	515	414	205.7	24	480	171.42	0.68
unrolling	1066	3065	411	25.43	1	20	508.6	0.48
PHOTON-80 [2]	82	188	135	302.68	54	648	9.34	0.11
PHOTON-80 [2]	69	159	89	285.2	30	360	15.84	0.22
PHOTON-80 [11]	149	_	_	250	59	_	7	0.05
HMAC-PHOTON-80 [11]	199	_	_	114	59	17,700	38.64	0.19
MD5 [17]	613	_	_	96	_	_	77.4	0.12
SHA-l [16]	518	_	_	82	_	_	51.8	0.10
SHA-256 [18]	609	_	_	260	_	_	198	0.32
KECCAK-256 [18]	1433	_	_	205	_	_	8397	5.86
KECCAK-256 [15]	1395	_	_	_	_	_	12777	9.16
KECCAK-256 [15]	1980	_	_	_			15362	7.76
KECCAK-256 [15]	3849	—	_	_	_	_	12652	3.29

TI implementation of MAC-PHOTON- $80/20/16$								
TI-iterative	739	1626	819	172.7	12	240	238.3	0.32
TI-folding	687	1738	814	194.3	24	480	162	0.24

4.1 Attacking MAC-PHOTON-80/20/16

The attacker needs either to recover the actual secret key K (see Table 2) or the internal state A_i (t=100 bits; r=20 bits and c=80 bits) to forge MAC for arbitrary messages. In the MAC-PHOTON-80/20/16 construction (see Figure 2), K only affects the internal state values A_1, A_2, A_3 before the message is inserted and also these internal state values are fixed and unknown. In order to perform a CPA attack, we require fixed unknown data to be combined with variable known data. This criterion is fulfilled, when the known and variable m is combined with the secret internal state A_3 (combined nibbles are represented in gray cells in Figure 3). This internal state value A_3 (see Table 2) does not change if K is fixed for any message m. In summary, the goal of our attack is to recover the secret internal state A_3 (marked in red in Figure 2) before the message digesting phase.

Table 2. Secret values

Secret Key (K)	FA4B7 5A4BC 9AB8C
Secret internal state value (A_3)	8F4D6 0112A ABADC D0FF7 14971

One can see that the incoming message block M is processed through the P permutation. First, the permutation P takes r-bit leftmost of the incoming internal state A_3 is XORed with r-bit known incoming first message block and storing the result in the first row (denoted m_{0i} in Figure 3) of the matrix rep-

resenting the internal state, while the four other rows (denoted x_{ij} in Figure 3) are filled with the remaining c-bits of the incoming internal state A_3 . Second, AddConstants (denoted c_i in Figure 3) are XORed to the first column of the internal state, then the SC and SR operations are performed (denoted s_{ij} in Figure 3). Finally, the MCS operation is performed (denoted s_{ij} in Figure 3).

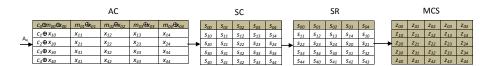


Fig. 3. One round of the internal permutation P of MAC-PHOTON-80/20/16.

Iterative: In the iterative architecture, we recover the incoming internal secret data (A_3) by correlating the power traces with a hypothetical model at a point of first round MCS state output during the A_4 permutation. In Figure 3, we can see that known and internal secret data (2-5 rows) are mixed after MCS operation is performed, where each column will depend on one known value and five unknown secret values. Overall, at the end of the first round, the first column (z_{i0}) on the output can be written as in the following matrix

$$\begin{pmatrix} z_{00} \\ z_{10} \\ z_{20} \\ z_{30} \\ z_{40} \end{pmatrix} = \begin{pmatrix} 1 & 2 & 9 & 9 & 2 \\ 2 & 5 & 3 & 8 & 13 \\ 13 & 11 & 10 & 12 & 1 \\ 1 & 15 & 2 & 3 & 14 \\ 14 & 14 & 8 & 5 & 12 \end{pmatrix} \begin{pmatrix} s_{00} \\ s_{11} \\ s_{22} \\ s_{33} \\ s_{44} \end{pmatrix}$$

If we look at the first output nibble z_{00} , it is given by

$$z_{00} = 01 \cdot s_{00} \oplus 02 \cdot s_{11} \oplus 09 \cdot s_{22} \oplus 09 \cdot s_{33} \oplus 02 \cdot s_{44}$$

If we focus on the first round, we can substitute s_{00} , s_{11} , s_{22} , s_{33} and s_{44} with $SC(x_{00} \oplus m_{00} \oplus c_0)$, $SC(x_{11})$, $SC(x_{22})$, $SC(x_{33})$ and $SC(x_{44})$. The output nibble z_{00} can then be written as

$$z_{00} = 01 \cdot SC(x_{00} \oplus m_{00} \oplus c_0) + q_{00}; q_{00} \in [0, ..., 15]$$
(2)

where, known constant c_0 is 1; unknown constant q_{00} can write as follows: $q_{00} = 02 \cdot SC(x_{11}) + 09 \cdot SC(x_{22}) + 09 \cdot SC(x_{33}) + 02 \cdot SC(x_{44})$

From equation 2, we observe that m_{00} is variable and known, whereas x_{00} is fixed and unknown secret. q_{00} is also fixed and unknown constant. Therefore, a CPA attack can be launched by making hypotheses about x_{00} and q_{00} , and computing the corresponding values of z_{00} . First, we recover the value of x_{00} ,

whereas hypotheses for q_{00} is initially ignored because it is not related to m_{00} . Hence, 2^4 hypotheses for x_{00} are required. Using the Hamming Distance (HD) model, the 2^4 possibilities for the previous state x_{00} (A_3), must also be taken into account. In our case same 2^4 hypotheses for the x_{00} are used in both the states. Therefore, the attacker correlates the power traces with the 2^4 hypotheses for HD(x_{00} , z_{00}). This allows the attacker to recover the secret value of x_{00} . Once recovering the secret value of x_{00} , the attacker can now make the 2^4 hypotheses on the q_{00} for HD(x_{00} , z_{00}). Hence, the fixed value of q_{00} is revealed. Furthermore, with knowledge of both x_{00} and q_{00} , the attacker can now accurately predict z_{00} for any message m. By following the above strategy, the attacker can recover the remaining internal state secrets. This attack model can decrease the complexity of internal state (A_3) from 2^{100} to 25×2^8 for MAC-PHOTON-80/20/16.

Folding: For folding architecture, we divide the attack in two phases. In the first one, we recover the bitrates part (first row in Figure 3) of the incoming internal secret data (A_3) by correlating the power traces with a hypothetical model at a point of first round SC state output during the A_4 permutation. Once recovering the bitrates part, we recover the left part of the incoming internal secret data by correlating the power traces with a hypothetical model at a point in output of the second round SC state operation during the A_4 permutation. The SC state is denoted by s_{ij} for first round and by s_{ij} for second round, respectively.

$$s_{ij} = SC(x_{ij} \oplus m_{ij} \oplus 1) \tag{3}$$

$$s_{ij}^{\dots} = SC(z_{ij} \oplus 3) \tag{4}$$

where z_{ij} value is obtained from equation 2

Focusing on equation 3, the attacker correlates the power traces with the 2^4 hypotheses $\mathrm{HD}(x_{ij}, s_{ij})$ for each nibble to recover the bitrates part. Using equation 4, the attacker can launch a CPA attack on s_{ij} by forming hypotheses $\mathrm{HD}(z_{ij}, s_{ij})$ to recover the remaining state values of A_3 . This attack model can efficiently decrease the complexity of internal state (A_3) from 2^{100} to 25×2^4 for MAC-PHOTON-80/20/16.

Unrolling: In the unrolling architecture, the data register Treg is updated only after processing every internal permutation P. Thus, the attacker can launch a CPA attack at a point of last round MCS state output during the A_4 permutation by forming hypotheses $HD(A_3, A_4)$ to recover the state values of A_3 . In this way, hypothesis test involves too many hypothesis for A_4 state which is derived from A_3 state. Therefore, we correlating the power traces with the following two hypothetical model approaches to recovers the internal state values of A_3 . First one is computed similar to iterative architecture, while second is computed similar to folding architecture.

4.2 Experimental Results

The SASEBO-GII hosts two FPGAs, i.e., one control FPGA (Xilinx XC3S400A-4FTG256, Spartan-3A series) and one cryptographic FPGA (Xilinx XC5VLX50-1FFG324, Virtex-5 series). In order to obtain CPA power traces from the design, the cryptographic FPGA was configured with the MAC-PHOTON-80/20/16 circuit through Parallel JTAG Cable. A USB cable to supply power to the SASEBO-GII board and to act as an interface between the board and the host PC. In all the experiments the clock signal is provided by a 24MHz oscillator which is divided by 3 using a frequency divider, i.e., the cryptographic FPGA is clocked at a frequency of 8MHz. Measurements are performed using an Agilent MSO7104B 1GHz oscilloscope at a sampling rate of 4GS/s and by means of a SMA-BNC cable which captures the voltage drop over an 1Ω shunt resistor inserted into the 1V VCORE (J2) line of the targeted FPGA. Therefore, the traces recorded on the oscilloscope were proportional to the power consumption of the FPGA during the execution of MAC-PHOTON-80/20/16 algorithm.

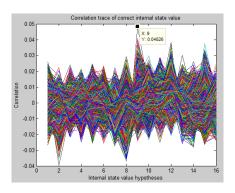


Fig. 4. Correlation Co-efficient plot for Side-channel attack (number of measurements = 10,000) on iterative based MAC-PHOTON implementation

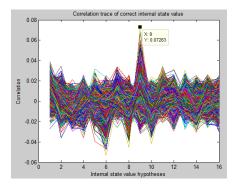


Fig. 5. Correlation Co-efficient plot for Side-channel attack (number of measurements = 8,000) on folding based MAC-PHOTON implementation

Iterative: In the iterative architecture, using the previously defined set-up and hypothetical model approaches, a total of 10,000 input random messages and 10,000 points per trace were required to obtain a successful CPA attack, which recovers that conform the secret internal state A_3 of the MAC-PHOTON. Figure 4 shows the result of iterative MAC-PHOTON-80/20/16 against CPA analysis. The correct first nibble of intermediate state A_3 value is 8 (Matlab array index value minus one) shows up clearly after around 10,000 traces.

Folding: In the folding architecture, using the previously defined set-up and hypothetical model approaches, a total of 8,000 input random messages and

10,000 points per trace were required to obtain a successful CPA attack, which recovers that conform the secret internal state A_3 of the MAC-PHOTON. Figure 5 shows the result of folding based MAC-PHOTON-80/20/16 against CPA analysis. The correct first nibble of intermediate state A_3 value is 8 (Matlab array index value minus one) shows up clearly after around 8,000 traces.

Unrolling: Using the previously defined set-up and hypothetical model approaches, we performed CPA attacks on the unrolling implementation of MAC-PHOTON with 50,000 power traces. In the unrolling MAC-PHOTON-80/20/16 analysis, without any surprise, we could not reveal correct value of the intermediate state A_3 for our two hypothetical approaches. Hence, our unrolling MAC-PHOTON-80/20/16 design resist against correlation power analysis on Hamming distance model.

5 Threshold implementation of the MAC-PHOTON-80/20/16

The preceding sections have analysed security of the MAC-PHOTON algorithm against first-order CPA attacks. We now examine security of threshold implementation (TI) of MAC-PHOTON against first-order CPA attacks. In 2006, Nikova et al. [21] introduced the concept of a threshold implementation scheme that is based on secret sharing techniques and is provable resistant against first order DPA even in the presence of glitches. The sharing can have three properties: Correctness, Non-completeness and Uniformity. Correctness means that combining the output of the different shares retrieves the original output in a correct way. Non-completeness means that each output share of a function is independent of at least one input share. Uniformity means that if the input shares are uniformly distributed, the output shares must also be uniformly distributed.

In order to design a threshold implementation for MAC-PHOTON there are two choices, iterative and folding. In both cases, we use three shares throughout the entire implementations. Hence, we need three times the registers compared to the unprotected iterative and folding implementations. Since the S-box used in PHOTON is same as that used in PRESENT, the decomposing and sharing techniques are borrowed from [23]. Figure 6 shows how to apply the threshold countermeasure to a 4-bit S-box: first it is decomposed into two stages G and F, then each stage is split into 3 shares. Figure 6 also shows that in [23] the authors implemented F and G using six Boolean functions $F_1, F_2,, G_3$ which can be calculated by the formulas in [23, Appendix A], but in this article the S-box decomposition is implemented without using a Y register [23] in between the G-function and the F-function. In our proposed two architectures where 25 instances of the TI S-box are implemented. We use the above analysis and provide a complete threshold implementation of MAC-PHOTON.

Iterative: A iterative based architecture computes one round per clock cycle. This architecture is depicted in Figure 7. For this architecture, we need two

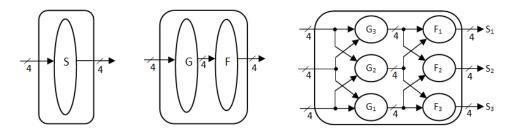


Fig. 6. Decomposition of an S-box

randomly generated masks (mask1 and mask2), which are XORed with the data (= key||message) chunk during input data block absorbing. The unmasking step is performed by simply XORing all three shares (h1, h2, h3) yielding the output MAC. Furthermore, the SC is now replaced by a decomposed and shared SC module similar to [23]. The data register Treg is updated every round after processing AC, shared SC, SR, and MCS operations in one clock cycle. We give in Table 1 the detailed results of the iterative based TI implementation of the MAC-PHOTON. As seen from the Table 1, our threshold implementation of the iterative architecture seems to require much less area than KECCAK-256 [18,15] and also yields a better throughput per area ratio compared to MD5 [17], SHA-1 [16], SHA-256 [18], PHOTON-80/20/16 [2,11] and HMAC-PHOTON-80/20/16 [11].

Folding: A folding based architecture computes one round per two clock cycles. The threshold implementation of the folding architecture is depicted in Figure 8. For this architecture, we need two randomly generated masks (mask1 and mask2), which are XORed with the data (= key||message) chunk during input data block absorbing. Furthermore, the SC is now replaced by a decomposed and shared SC module similar to [23]. The data register Treg is updated on every half of a round operations in one clock cycle (similarly as unprotected folding based implementation). We give in Table 1 the detailed results of the folding based TI implementation of the MAC-PHOTON. As seen from the Table 1, our threshold implementation of the folding architecture seems to require much less area than KECCAK-256 [18,15] and also yields a better throughput per area ratio compared to MD5 [17], SHA-l [16], PHOTON-80/20/16 [2,11] and HMAC-PHOTON-80/20/16 [11].

We repeat the experiments described in section 4.1 on our threshold implementations of the iterative and folding architectures. We verify that our implementations are resistant against first order CPA attacks even if collects up to 100,000 power traces.

6 Conclusion

In this paper, we have presented an analysis of SCA resistance of implementation of PHOTON hash algorithm in MAC construction. The implemented MAC-PHOTON-

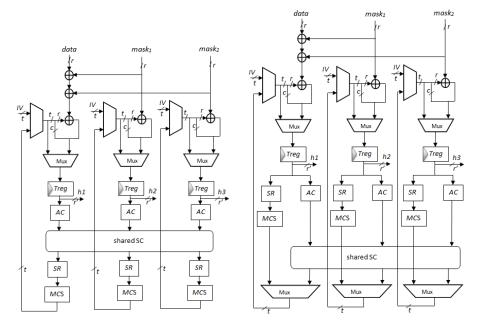


Fig. 7. Architecture of the round-based TI of MAC-PHOTON

Fig. 8. Architecture of the folding based TI of MAC-PHOTON

80/20/16 features are more efficient for processing short messages when compared to HMAC construction. Without compromising the system security, our results show that without any protection and key refreshment, it is possible to interchange up to 10000, 8000, 50000 messages for iterative, folding and unrolling implementations, respectively. Resistance of TI implementations of MAC-PHOTON against first order CPA attacks has been tested. As we noted, our implementations are resistant against first order CPA attacks even if an attacker is capable of measuring 100,000 power traces. Our results showed that both protected and unprotected MAC-PHOTON constructions seems to be very well suited for lightweight applications (even high-speed) when compared to construction of HMAC design based protocols.

References

- 1. FT2232D DUAL USB TO SERIAL UART/FIFO IC Datasheet, 2010, Available at:. 2nd ed., Future Technology Devices International Ltd.
- 2. N. Nalla Anandakumar, Thomas Peyrin, and Axel Poschmann. A Very Compact FPGA Implementation of LED and PHOTON. In Willi Meier and Debdeep Mukhopadhyay, editors, Progress in Cryptology INDOCRYPT 2014 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings, volume 8885 of Lecture Notes in Computer Science, pages 304–321. Springer, 2014.

- Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In Neal Koblitz, editor, CRYPTO, volume 1109 of Lecture Notes in Computer Science, pages 1–15. Springer, 1996.
- 4. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In Ali Miri and Serge Vaudenay, editors, Selected Areas in Cryptography, volume 7118 of Lecture Notes in Computer Science, pages 320–337. Springer, 2011.
- Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the security of the keyed sponge construction. In G. Leander and S.S. Thomsen editors, SKEW, 2011.
- Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Cryptographic sponge functions, 2011, Available at:. http://sponge.noekeon.org/CSF-0.1.pdf.
- Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. The Keccak sponge function family, 2011, Available at:. http://keccak.noekeon.org/.
- 8. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, CHES, volume 4727 of Lecture Notes in Computer Science, pages 450–466. Springer, 2007.
- Christina Boura, Sylvain Lévêque, and David Vigilant. Side-Channel Analysis of Grøstl and Skein. In *IEEE Symposium on Security and Privacy Workshops*, pages 16–26. IEEE Computer Society, 2012.
- Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Marc Joye and Jean-Jacques Quisquater, editors, CHES, volume 3156 of Lecture Notes in Computer Science, pages 16–29. Springer, 2004.
- 11. Susana Eiroa and Iluminada Baturone. FPGA implementation and DPA resistance analysis of a lightweight HMAC construction based on photon hash family. In *FPL*, pages 1–4. IEEE, 2013.
- 12. Andreas Engel, Björn Liebig, and Andreas Koch. Feasibility Analysis of Reconfigurable Computing in Low-Power Wireless Sensor Applications. In Andreas Koch, Ram Krishnamurthy, John McAllister, Roger Woods, and Tarek A. El-Ghazawi, editors, ARC, volume 6578 of Lecture Notes in Computer Science, pages 261–268. Springer, 2011.
- 13. Martin Feldhofer, Manfred Josef Aigner, Thomas Baier, Michael Hutter, Thomas Plos, and Erich Wenger. Semi-passive RFID development platform for implementing and attacking security tags. In *ICITST*, pages 1–6. IEEE, 2010.
- 14. Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON Family of Lightweight Hash Functions. In *Advances in Cryptology–CRYPTO 2011*, pages 222–239. Springer, 2011.
- 15. Ekawat Homsirikamol, Marcin Rogawski, and Kris Gaj. Throughput vs. Area Trade-offs in High-Speed Architectures of Five Round 3 SHA-3 Candidates Implemented Using Xilinx and Altera FPGAs. In Bart Preneel and Tsuyoshi Takagi, editors, CHES, volume 6917 of Lecture Notes in Computer Science, pages 491–506. Springer, 2011.
- 16. Kimmo Järvinen. Design and implementation of a SHA-1 hash module on FPGAs. In *Helsinki University of Technology Signal Processing Laboratory*, 2004.
- 17. Kimmo U. Järvinen, Matti Tommiska, and Jorma Skyttä. Hardware Implementation Analysis of the MD5 Hash Algorithm. In 38th Hawaii International Conference on System Sciences (HICSS-38 2005), CD-ROM / Abstracts Proceedings, 3-6 January 2005, Big Island, HI, USA. IEEE Computer Society, 2005.

- 18. Kazuyuki Kobayashi, Jun Ikegami, Kazuo Sakiyama, Kazuo Ohta, Miroslav Knezevic, Ünal Koçabas, Junfeng Fan, Ingrid Verbauwhede, Eric Xu Guo, Shin'ichiro Matsuo, Sinan Huang, Leyla Nazhandali, and Akashi Satoh. Prototyping Platform for Performance Evaluation of SHA-3 Candidates. In Jim Plusquellic and Ken Mai, editors, HOST, pages 60–63. IEEE Computer Society, 2010.
- Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, CRYPTO, volume 1666 of Lecture Notes in Computer Science, pages 388–397. Springer, 1999.
- 20. Future Technology Devices International Ltd. CodeExamples, Available at:. http://www.ftdichip.com/Support/SoftwareExamples/CodeExamples/CSharp.htm.
- 21. Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings, volume 4307 of Lecture Notes in Computer Science, pages 529–545. Springer, 2006.
- 22. National Institute of Advanced Industrial Science Technology (AIST). Side-channel Attack Standard Evaluation Board SASEBO-GII specification, 2009, Available at:. http://www.rcis.aist.go.jp/special/SASEBO/SASEBO-GII-ja.html.
- 23. Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-Channel Resistant Crypto for Less than 2,300 GE. *Journal of Cryptology*, 24(2):322–345, 2011.
- Mostafa M. I. Taha and Patrick Schaumont. Side-Channel Analysis of MAC-Keccak. In HOST, pages 125–130. IEEE, 2013.
- 25. Tim Tuan, Arif Rahman, Satyaki Das, Steven Trimberger, and Sean Kao. A 90-nm Low-Power FPGA for Battery-Powered Applications. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 26(2):296–300, 2007.
- 26. Tolga Yalçin and Elif Bilge Kavun. On the Implementation Aspects of Sponge-Based Authenticated Encryption for Pervasive Devices. In Stefan Mangard, editor, CARDIS, volume 7771 of Lecture Notes in Computer Science, pages 141–157. Springer, 2012.
- Michael Zohner, Michael Kasper, Marc Stöttinger, and Sorin A. Huss. Side channel analysis of the SHA-3 finalists. In Wolfgang Rosenstiel and Lothar Thiele, editors, DATE, pages 1012–1017. IEEE, 2012.

A Our Communication Interface for SASEBO-GII

Our communication interface for SASEBO-GII [22] is derived from the work proposed in [18] with slight modifications which is suitable and customisable for cryptographic primitives. Our entire interface control logic was implemented based on a finite-state machine and also provides the MATLAB solutions instead of SASEBO-Checker [18] to work with the FTDI chip. This choice is made for accessibility and ease of maintenance. Figure 9 shows the overview of the SASEBO-GII communication interface. This interface is used to communicate with the PC and two FPGAs of SASEBO-GII board. They are a cryptographic FPGA (Virtex-5) and control FPGA (Spartan-3A), a cryptographic FPGA usually implements the cryptographic algorithm and a control FPGA which communicates the data between the PC and the cryptographic FPGA. In our case, the MAC-PHOTON-80/20/16 module was ported into the cryptographic

FPGA whereas the control FPGA acted as a bridge between the PC and the MAC-PHOTON-80/20/16 module.

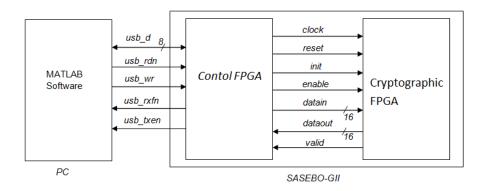


Fig. 9. SASEBO-GII communication Interface

A.1 The Interface Between the Control and Cryptographic FPGAs

The control FPGA module consists of the following 5 states: initial, receiveusb, ControlFPGAsend, ControlFPGAreceive and sendusb. During initial state, the USB module in the control FPGA is initialized through the FT2232D USB chip [1]. In receiveusb state, the input data is received 8-bits at a time from the PC (MATLAB) through the USB chip and then the values are stored in the data registers. During ControlFPGAsend state, a MAC-PHOTON-80/20/16 module in the cryptographic FPGA via init signal is initialized first. Then, the control FPGA sends the input data 16-bits wide via datain signal from the input data registers to the cryptographic FPGA. Once the data is processed the ControlFPGAreceive state receives the output data 16-bits-wide via dataout signal from the cryptographic FPGA and stores the data into the output data registers. During sendusb state, the output data (MAC) is sent back (8-bits wide) to the PC (MATLAB) from output data registers through the FT2232D USB chip. Hence, it requires 30 clock cycles to process the interface between the Control and Cryptographic FPGAs.

The cryptographic FPGA module consists of the following 3 states: process, CryptoFPGAreceive and CryptoFPGAsend. In CryptoFPGAreceive state, the cryptographic FPGA start to receives the input data from the control FPGA when the init signal is reached and then the values are stored in the data registers. The process state, is to execute the MAC-PHOTON-80/20/16 module. The CryptoFPGAsend state, once the MAC-PHOTON-80/20/16 module is processed, sends the output data (MAC) 16-bits wide via dataout signal to the control FPGA.

A.2 The Interface Between the PC and Control FPGA

The FT2232D USB chip was permanently mounted with the contol FPGA of the SASEBO-GII board. This chip acts as the communication interface between the MATLAB software and the control FPGA. This MATLAB software is run on the host PC and it is the control center of the whole system. In this work, the MATLAB is used for 2 purposes: one is to record the traces from the oscilloscope and the other is to send or receive the data from the PC to the control FPGA via FT2232D USB chip from FTDI inc. Although MATLAB provides support to call shared library functions, there is no readily available MATLAB solutions [20] to work with the FTDI chip. In this work, we translate from working .Net wrapper [20] to MATLAB with call shared library functions.

The translation program is divided into 4 parts: initialization, transfer, receive and closing. During initialization, the data length is defined, the library functions are loaded and also handle is defined to specify that the device (USB port) is opened. Once initialization is complete, the program tells the user that it is ready to receive data and asks the user to trigger the FPGA. During the transfer stage, the program continuously write the input data to the control FPGA until the expected number of data length. During the receive stage, the program read the output data from the control FPGA. Once receive stage is complete, handle device (USB port) is closed. Hence, it requires 216 clock cycles to process the interface between the PC and Control FPGA.