SCA Resistance Analysis of Sponge based
MAC-PHOTON

N. Nalla Anandakumar

Hardware Security Research Group,
Society for Electronic Transations and Secuity, India

`nallananth@gmail.com`

Abstract. `PHOTON` is a lightweight hash function which was proposed by Guo et al. in CRYPTO 2011 for low-resource ubiquitous computing devices such as RFID tags, wireless sensor nodes and smart cards. In this paper, we analyze Side-Channel Attack (SCA) resistance of FPGA (Field-Programmable Gate Array) implementations of the `PHOTON`, when it is used with a secret key to generate a Message Authentication Code (MAC). 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 analysed security of the `MAC-PHOTON` against side-channel attack using a SASEBO-GII development board.

Keywords: SCA, Lightweight Cryptography, Sponge functions, `MAC`, `PHOTON`.

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 functions. A sponge function H is a one-way function that converts arbitrary-length message M into variable-length hash code $H(M)$ (or digest). They can be used in many applications such as hashing, pseudo-random sequence generation, key derivation and stream cipher encryption. In practice, cryptographic 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 authenticate both the source of a message and its integrity without the use of any additional mechanisms.

More recently, a sponge based hash function called `PHOTON` [14] has been proposed, especially for lightweight security devices as it requires few resources. The design structure of `PHOTON` has an AES like internal permutation which is especially derived for hardware. In this study, we have implemented the MAC construction based on PHOTON algorithm on FPGA. As it is attractive for implementing cryptographic algorithms in terms of cost, time-to-market and their

flexibility when compared with ASIC. The proposed construction is suited for the lightweight cryptographic applications such as FPGA-based RFID tags [13], FPGA-based wireless sensor nodes [12, 22].

This work deals with security of the FPGA implementation which targets the `MAC` construction based on `PHOTON` hash function against side-channel analysis such as correlation power analysis (CPA) [10]. In a side-channel attack, an adversary may attempt to exploit the secret information which is leaking from a physical implementation, rather than brute force or theoretical weaknesses in the algorithms. In the `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. To the best of our knowledge, this is the first security analysis of the `MAC-PHOTON` security against first-order CPA attacks.

Recently, Susana et al. [11] presented an analysis of side channel resistance of HMAC [2] 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 used same key variant for HMAC prefix-suffix construction. They also mentioned that their implementation is not suitable for high-speed resource constrained devices. In order to cover this lack, all our proposed implementations of MAC based on the `PHOTON` hash function are given in this work which are suited for high-speed, resource constrained devices.

In this paper, we also presents an analysis of side channel resistance of the sponge based `MAC` construction for three architectures (iterative, folding and unrolling) of `PHOTON` functions. To our knowledge, these are the first non-serialised implementations of `MAC-PHOTON`. Moreover, our `MAC-PHOTON` implementations achieve better efficiency and provide better security compared to Susana et al. [11].

Our contributions. The primary goal of this work is to provide a deeper 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. 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 the best throughput per area ratio when compared with existing FPGA implementation of HMAC-`PHOTON` [11].

2. Our second contribution is to present the 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, 30000 messages, respectively.

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. The paper concludes in Section 5.

2 Technical Background

In this section, we introduce 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.

The `PHOTON` algorithm essentially consists of three phases: initialization phase, absorption phase and squeezing phase. `PHOTON` starts with the initialization phase, where the message is padded and split the message into r-bit chunks. During the absorption phase, iteratively processes all the r-bit message chunks by XORing them to the bitrate part of the internal state and then applying the t-bit permutation P. Once all message chunks 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:

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

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, 23]. 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.

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

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.
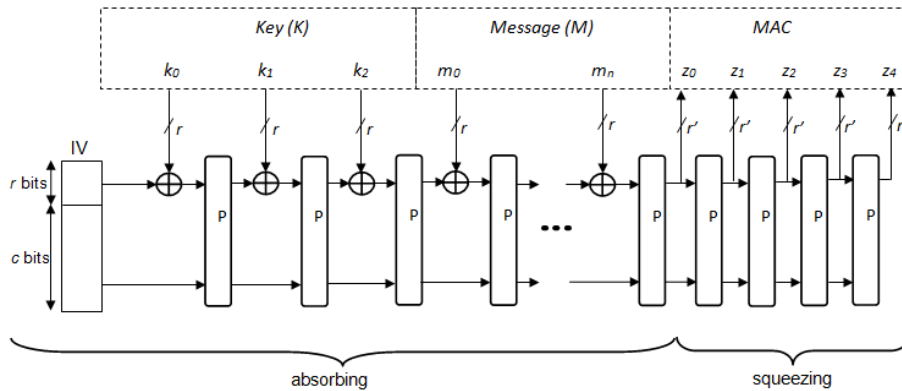

Fig. 1. The block diagram of the sponge based MAC-PHOTON-80/20/16 construction

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) [16] 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 during execution. CPA is a more advanced DPA technique. In this type of 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 [21], [9] and [24]. In this paper, we demonstrate CPA 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 on a SASEBO-GII Board [19], which contains the Xilinx Virtex-5 (xc5vlx50-1ffg324) cryptographic FPGA. We used Mentor Graphics ModelSimPE for simulation purposes and Xilinx ISE v13.4 for synthesizing and implementation purposes. In addition, we describe the communication flow of the modified SASEBO-GII interface in Appendix A. 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 2 the detailed results of the iterative, folding and unrolling based implementations of the MAC-PHOTON. The iterative architecture computes one round of internal permutation P per clock cycle, while the folding architecture computes one round of internal permutation P per 2 clock cycles. In the unrolling architecture performing operations on all rounds (12 rounds) of internal permutation P per clock cycle.
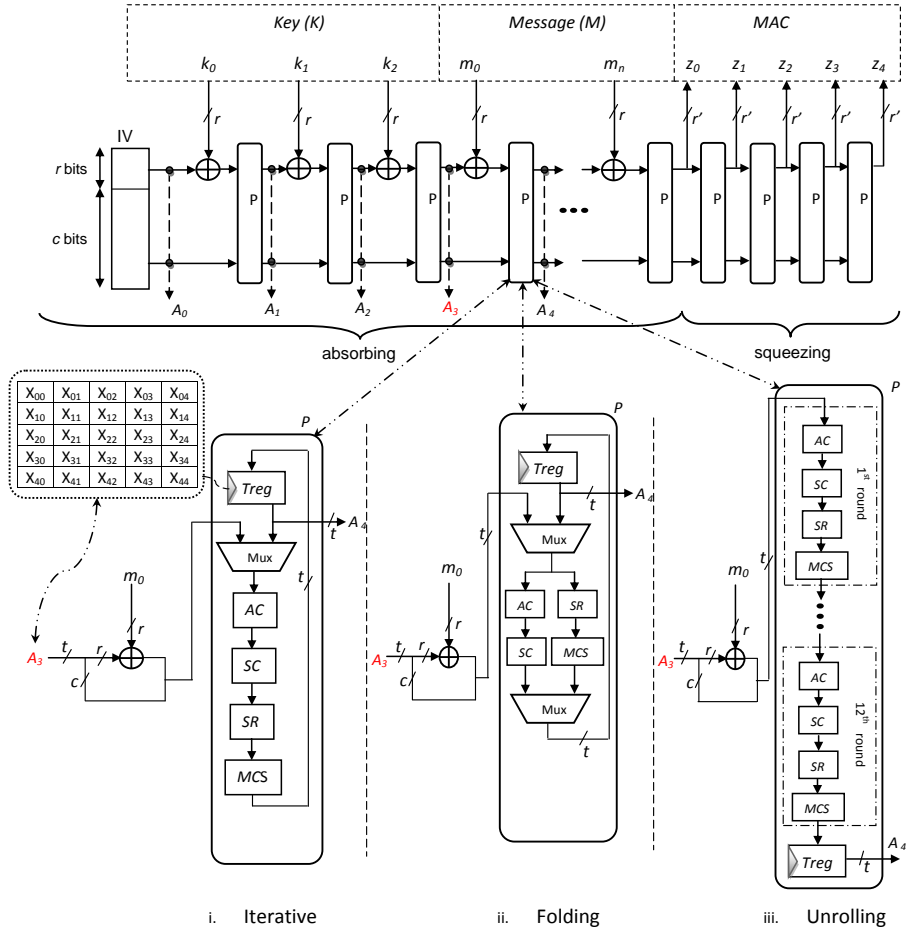
Fig. 2. The block diagram of the iterative, folding, unrolling implementations of the `MAC-PHOTON`-80/20/16

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

Design	Area (slices)	LUTs	FFs	Max. freq (MHz)	Total Number of Clock Cycles (cycles)		T.put (Mbps)	T.put/Area (Mbps/slices)
					internal permutation P	whole hash function H		
iterative	302	508	415	172.7	12	240	287.83	.95
folding	251	505	416	205.7	24	480	171.42	.68
unrolling	1066	3065	411	25.43	1	20	508.6	0.48
HMAC-`PHOTON`-80 [11]	199	—	—	114	59	17,700	38.64	0.19

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 on 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. Therefore, 240 clock cycles are required in order to complete both phases. We obtain 302 slices, while the throughput reaches 287.83 Mbps. In Table 1, One can see that our proposed iterative MAC-PHOTON implementation outperform the fully serialized design HMAC-PHOTON-80/20/16 [11] in terms of throughput per area ratio.

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 *Treg* is updated on 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 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 yield the better throughput per area ratio than 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 are performed per clock cycle. Thus, the data register *Treg* is updated when once the each internal permutation P is computed. Hence, it requires 16 clock cycles to process 16 blocks in absorbing phase and 4 clock cycles 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 1,066 slices, while the throughput reaches 508.6 Mbps. As seen from the Table 1, our work yield the better throughput per area ratio than HMAC-PHOTON-80/20/16 [11].

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

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

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 MACs 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 as 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 as 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 as 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 representing 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 z_{ij} in Figure 3).
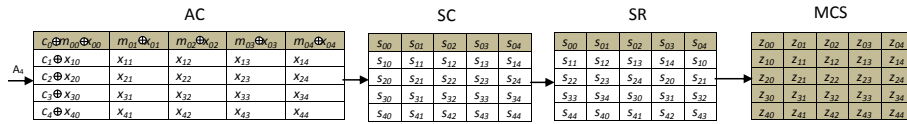
A_4

AC

$c_0 \oplus m_{00} \oplus x_{00}$	$m_{01} \oplus x_{01}$	$m_{02} \oplus x_{02}$	$m_{03} \oplus x_{03}$	$m_{04} \oplus x_{04}$
$c_1 \oplus x_{10}$	x_{11}	x_{12}	x_{13}	x_{14}
$c_2 \oplus x_{20}$	x_{21}	x_{22}	x_{23}	x_{24}
$c_3 \oplus x_{30}$	x_{31}	x_{32}	x_{33}	x_{34}
$c_4 \oplus x_{40}$	x_{41}	x_{42}	x_{43}	x_{44}

SC

s_{00}	s_{01}	s_{02}	s_{03}	s_{04}
s_{10}	s_{11}	s_{12}	s_{13}	s_{14}
s_{20}	s_{21}	s_{22}	s_{23}	s_{24}
s_{30}	s_{31}	s_{32}	s_{33}	s_{34}
s_{40}	s_{41}	s_{42}	s_{43}	s_{44}

SR

s_{00}	s_{01}	s_{02}	s_{03}	s_{04}
s_{11}	s_{12}	s_{13}	s_{14}	s_{10}
s_{22}	s_{23}	s_{24}	s_{20}	s_{21}
s_{33}	s_{34}	s_{30}	s_{31}	s_{32}
s_{44}	s_{40}	s_{41}	s_{42}	s_{43}

MCS

z_{00}	z_{01}	z_{02}	z_{03}	z_{04}
z_{10}	z_{11}	z_{12}	z_{13}	z_{14}
z_{20}	z_{21}	z_{22}	z_{23}	z_{24}
z_{30}	z_{31}	z_{32}	z_{33}	z_{34}
z_{40}	z_{41}	z_{42}	z_{43}	z_{44}

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

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

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. q_{00} is also fixed and unknown constant. Therefore, a CPA attack can be launched by making hypotheses about the unknown values x_{00}, and computing the corresponding values of the current state z_{00} (where, hypotheses for q_{00} is 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 x_{00}, and then calculate z_{00} for any message m. By following the above strategy, the attacker can recover the remaining bitrates part of the internal state.

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.

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

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

where z_{ij} value is obtained from equation 2

Focusing on equation 3, the attacker correlates the power traces with the 2^4 hypotheses HD(x_{ij}, s_{ij}) for each nibble to recover the bitrates part. Using equation 4, the attacker can launch a CPA attack on $s_{ij}^{\cdot\cdot}$ by forming hypotheses HD(z_{ij}, $s_{ij}^{\cdot\cdot}$) to recover the remaining state values of A_3.

Unrolling: In the unrolling architecture, the data register Treg is updated when only after processing every internal permutation P and 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, an attacker correlating the power traces with the following two hypothetical model approaches to recovers the internal state values of A_3. First one is computed similarly as iterative architecture, while second is computed similarly as folding architecture.

4.2   Experimental Results

In order to obtain CPA power traces from the design, the targeted 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 targeted 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 the `MAC-PHOTON-80/20/16` algorithm.

Iterative: In the iterative architecture, using the previously defined set-up and hypothetical model approaches, a total of 10,000 input random messages and 4,000 points per trace were required to obtain a successful DPA 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 4,000 points per trace were required to obtain a successful DPA 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.
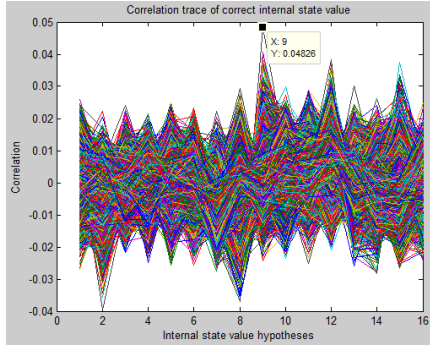
Fig. 4. Correlation Co-efficient plot for Side-channel attack (number of measurements = 10,000) on iterative based `MAC-PHOTON` implementation
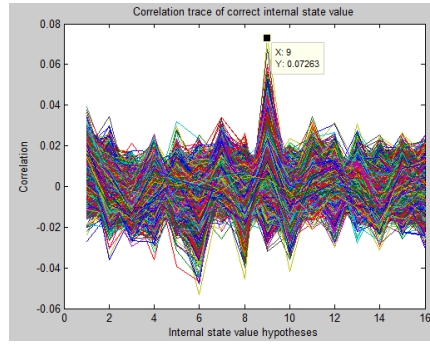

Fig. 5. Correlation Co-efficient plot for Side-channel attack (number of measurements = 8,000) on folding based `MAC-PHOTON` implementation

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 30,000 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 Conclusion

In this paper, we presented an analysis of SCA resistance of `PHOTON` hash algorithm in MAC construction. The implemented `MAC-PHOTON`-80/20/16 features are more efficient for processing short messages when compared to HMAC construction. Our results show that `MAC-PHOTON` construction seems to be very well suited for lightweight applications (even high-speed) when compared to construction of adhoc designed protocols and HMAC designed based protocols. MAC security resistance against first-order CPA attacks has been tested. Without compromising the system security, our results show that without any protection and key refreshment, it is possible to interchange up to 10000, 8000, 30000 messages for iterative, folding and unrolling implementations, respectively. Future work we will improve the security of the iterative, folding based implementations of `MAC-PHOTON` with effective countermeasures [3, 18, 20], and test their security thoroughly.

References

1. FT2232D DUAL USB TO SERIAL UART/FIFO IC Datasheet, 2010, Available at:. 2nd ed., Future Technology Devices International Ltd.
2. 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.
3. Guido Bertoni, Joan Daemen, Nicolas Debande, Thanh-Ha Le, Michael Peeters, and Gilles Van Assche. Power analysis of hardware implementations protected with secret sharing. In *45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2012, Workshops Proceedings, Vancouver, BC, Canada, December 1-5, 2012*, pages 9–16. IEEE Computer Society, 2012.
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.
5. 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.
6. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Cryptographic sponge functions, 2011, Available at:. `http://sponge.noekeon.org/CSF-0.1.pdf`.
7. 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.
9. 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.
10. 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. Kazuyuki Kobayashi, Jun Ikegami, Kazuo Sakiyama, Kazuo Ohta, Miroslav Kneze-vic, Ünal Koçabas, Junfeng Fan, Ingrid Verbauwhede, Eric Xu Guo, Shin'ichiro Matsuo, Sinan Huang, Leyla Nazhandali, and Akashi Satoh. Prototyping Plat-form for Performance Evaluation of SHA-3 Candidates. In Jim Plusquellic and Ken Mai, editors, *HOST*, pages 60–63. IEEE Computer Society, 2010.

16. 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.

17. Future Technology Devices International Ltd. CodeExamples, Available at:. `http://www.ftdichip.com/Support/SoftwareExamples/CodeExamples/CSharp.htm`.

18. Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementa-tion of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.

19. 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`.

20. 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.

21. Mostafa M. I. Taha and Patrick Schaumont. Side-Channel Analysis of MAC-Keccak. In *HOST*, pages 125–130. IEEE, 2013.

22. 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.

23. Tolga Yalçin and Elif Bilge Kavun. On the Implementation Aspects of Sponge-Based Authenticated Encryption for Pervasive Devices. In Stefan Mangard, edi-tor, *CARDIS*, volume 7771 of *Lecture Notes in Computer Science*, pages 141–157. Springer, 2012.

24. 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 [19] is derived from the work proposed in [15] 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 in-stead of SASEBO-Checker [15] to work with the FTDI chip. This choice is made for accessibility and ease of maintenance. Figure 6 shows the overview of the SASEBO-GII communication interface. This interface is used to commu-nicate with the PC and two FPGAs of SASEBO-GII board. They are a cryp-tographic 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.
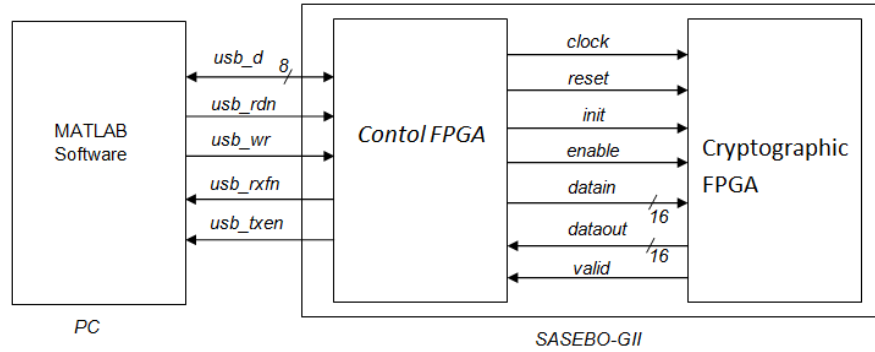
Fig. 6. SASEBO-GII communication Interface

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 [17] to work with the FTDI chip. In this work, we translate from working .Net wrapper [17] 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.