FPGA IMPLEMENTATION AND DPA RESISTANCE ANALYSIS OF A LIGHTWEIGHT HMAC CONSTRUCTION BASED ON PHOTON HASH FAMILY

Susana Eiroa and Iluminada Baturone

Microelectronics Institute of Seville (IMSE-CNM-CSIC)
University of Seville (Dept. of Electronics and Electromagnetism)
email:{eiroa,lumi}@imse-cnm.csic.es

ABSTRACT

Lightweight security is currently a challenge in the field of cryptography. Most of applications designed for embedded scenarios often focus on authentication or on providing some form of anonymity and/or privacy. A well-known cryptographic element employed to provide such security is the HMAC construction. However, reported solutions are not suitable for constrained-resource scenarios due to their heavy approaches optimized for high-speed operations. In order to cover this lack, a lightweight implementation of HMAC based on the Photon family of hash functions is given in this work. Security of the construction against differential power attacks (DPA) is analyzed using a SASEBO-II development board. Implementation and performance results for Xilinx Virtex-5 FPGAs of the HMAC structure is provided.

1. INTRODUCTION

The use of constrained resource devices is widely extended in several fields of everyday life. The most important requirements to be satisfied in secure communications are data integrity and data origin authentication. A very common practice is the use of Message Authentication Codes (MACs) [1]. MACs are widely employed since they can provide security without additional mechanisms. The National Institute of Standards and Technology (NIST) recommends the use of the Keyed-Hash Message Authentication Code (HMAC) for the primitives previously mentioned [2]. Hence, this is the construction studied over this work.

Most of reported HMACs are usually targeted at high speed and throughput applications that require high costs in area and power consumption. This makes them not suitable for constrained-resource scenarios. Therefore, the necessity of new compact HMAC constructions has became relevant.

This work has been partially supported by P08-TIC-03674 project from the Andalusian Regional Government and by IPT-2012-0695-390000 and TEC2011-24319 projects from Ministerio de Economía y Competitividad of the Spanish Government (with support from the PO FEDER-FSE).

This work presents a HMAC construction based on Photon family of lightweight hash functions, suitable for constrained resource devices. Its resistance against DPA side-channel attacks is analyzed. The paper is structured as follows. Section II briefly describes the HMAC standard algorithm and Photon basic characteristics. Section III presents the hardware architecture of the HMAC structure and implementation results for Xilinx FPGAs. A power analysis attack has been developed to analyze its resistance against side-channel attacks. Finally, conclusions are given in Section V.

2. PRELIMINARIES: HMAC AND PHOTON CONSTRUCTIONS

The Keyed-Hash Message Authentication Code (HMAC) represents a type of message authentication code (MAC) based on hash functions. The input of the HMAC is a secret key, K, and the data, $text$. Current HMAC standard, which is described in [2], is essentially a two pass HMAC described as follows:

$$HMAC(K, text) = Hash[(K_o \oplus opad) \| Hash((K_o \oplus ipad) \| text)]$$

where K_o is the key K after any necessary pre-processing to form a multiple of the hash input block size; $ipad$ and $opad$ are, respectively, the inner and outer pads with values x'36' and x'5c'.

The security provided by HMAC constructions comes defined by its underlying hash function. Well-known hashes are MD5 and SHA-1. However, several attacks have been reported since 1993 for MD5 and 2005 for SHA-1 [3]. To solve this, NIST opened a public competition to develop a new standard hash algorithm (SHA-3), pointing Keccak [4] as winner. It is based on the new paradigm of sponge constructions [5]. Specific lightweight hash functions based on sponge constructions for tiny devices have been also proposed, as is the case of Photon family [6]. They work as follows: each time a new input data is processed, it must be

divided into blocks (m_0, m_1, .., m_n) of r bits before entering the function. The sponge function then proceeds in two steps: absorbing and squeezing. During first step, blocks are inserted into the internal state and processed in groups of r bits. When all the input data is absorbed, r bits are provided as output until the required length is achieved. Sponge functions allow minimizing the amount of hardware memory registers because only the bits of the internal state should be stored. Hence, they can offer good speed-area-security trade-off [6].

Each Photon function denoted by Photon-n/r/r' comes defined by its output and input rates, r' and r, and the output hash size, n, which ranges from 64 to 256 [6]. Photon-80/20/16 using a 60 bits key has been selected herein since it is the lightest and simplest version of the family. Minimum values of Photon flavor and key (60 bits) have been intentionally chosen in order to establish a lower bound for security. In case that higher key length and/or pre-image or collision values are required, bigger implementations can be chosen without substantial difference either in the implementation (only the implementation of the Photon hash core must be replaced), or in the model attacks.

3. HMAC IMPLEMENTATION

The system architecture of the HMAC implemented is shown in Figure 1. It is based on Photon-80/20/16 with a key of 60 bits and considers (without any lost of generality) an input text value of 256 bits (260 with required padding). It is composed of three different modules: *Registers Module*, where OPAD and IPAD are stored; *In_data Module*, which provides correct data to the core of the hash, and finally the *Hashing Module***,** which is the heart of the HMAC.

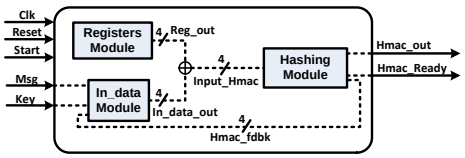

Fig. 1: Architecture of the HMAC implemented.

The *Hashing Module* shown in Figure 2 contains four elements: the *Enable HMAC_Out* block provides the output of the HMAC to the exterior of the module. The controller, *Controller Hashing,* synchronizes the input values for the *Hash Core.* The *Hash Core* itself, which is the kernel of the hashing, implements the four types of operations performed over the internal state (the serial implementation proposed by the authors in [6] has been selected to achieve a low area utilization). Finally, an internal memory, *HMAC Internal Memory*, formed by a set of registers is needed to store the first and second hash values. The memory units have 4 bits. Since the Photon provides only a part of the complete hash digest in each squeezing step, the final value is obtained by the concatenation of the consecutive outputs. Hence, it is necessary to store fragments of the final hash value. The 80 bits memory can be read in serial mode, if it is the feedback for the second hashing, or in parallel, if it is the final value.

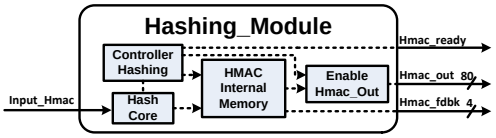

Fig. 2: Internal structure of HMAC Hashing_Module.

3.1. Operation mode of the HMAC structure

The HMAC has to perform two passes of the hash algorithm, each with different input data sizes.

First hash generation: the input data is the concatenation of M bits of the text plus K bits of the key. Hence, it requires to process (M+K)/20 blocks. In order to absorb and squeeze a block, a round must be run 12 times. According to [6], a round requires $(2 \times d(d+1) - 1)$ clock cycles, where d is related to the size of the internal state (five bits in Photon-80/20/16). Total time required to process all the input data is:

$$(N_{absblock} + N_{sqzblock}) \times 12 \times (2 \times d(d+1) - 1)$$

Where $N_{absblock}$ is the number of blocks in the absorbing phase and $N_{squblock}$ is the number of blocks in the squeezing phase. The selected Photon-80/20/16 for a message of 260 bits concatenated with a key of 60 bits takes 14,160 clock cycles for the first hash.

Second hash generation: the input data is now the output value of the previous hash concatenated to the key. The hash output is obtained, so, after 7,788 clock cycles because $N_{absblock}$ and $N_{squblock}$ are now 7 and 4, respectively.

As it is noticeable, the amount of clock cycles to carry out the HMAC is dominated by the latency of absorbing and squeezing processes. For the current example the total latency is around 21,948 clock cycles. A considerably amount of clock cycles is devoted to absorbing the blocks related to the key: $(2 \times d(d+1) - 1)$*K/20 clock cycles in each hash calculation. In the case of a 60-bit key, these are 2,124 clock cycles. If several messages are to be authenticated with the same key, the intermediate results of the hash function on the blocks related to the key can be precomputed once (at the time of generating the key or before its first use). This way, the total latency could be reduced to above 17,700 cycles. The saving in the total latency is more significant when authenticating short streams of data.

3.2. Implementation results

Table 1 compares implementation results of Photon-80/20/16 hashing module (see Figure 2) with MD5, SHA-1, SHA-2

Table 1: Hash functions implementation in FPGAs

	Throughput [Gbps]	Power [mW]	Area [Slices]	Max. Freq. [MHz]
MD5 [7]	0.744	-	613	96
SHA-1 [8]	0.518	-	518	82
SHA-256 [9]	1.98	210@24MHz	609	260
Keccak [9]	8.397	290@24MHz	1,433	205
Keccak [10]	0.077	-	188	285
Photon (this work)	0.007	0.35@25MHz	149	250

Table 2: Area occupation of HMAC IP cores

	$HMAC_{SHA\text{-}1}$ [11]	$HMAC_{SHA\text{-}256}$ [12]	This work
Area [Slices]	6,011	3,463	199
Device	Virtex-E	Virtex-E	Virtex-5

and SHA-3 winner (Keccak) [1]. From results shown in Table 1 it can be concluded that Photon-80/20/16 implementation is competitive in area and power consumption. The cost to pay is a low operation speed in terms of throughput and latency. Hence, it can be a suitable option for non high-speed constrained-resource scenarios.

Table 2 shows area results in FPGAs of some HMAC IP cores based on SHA-1, SHA-2 and Photon-80/20/16 implemented in this work [2]. It is apparent that the occupied area decreases substantially by using HMAC structure described above. Of course, this comparison, should not be strictly restricted to area due to the different security level provided.

Table 3 shows a summary of HMAC-Photon-80/20/16 implementation features in a Virtex-5 FPGA of Xilinx. Values obtained include the padding (carried out by the *In_data* module) and the text and key input fragmentation. Therefore, the structure behaves as an autonomous entity without additional hardware requirements. The maximum working frequency is 114 MHz. This means that for 320-bit message (60-bit key and 256-bit text), the total latency of the HMAC is 192,5 μs without reusing the key (155,3 μs if the key is reused). Power consumption values were obtained using the Xpower Analyzer tool from Xilinx ISE 12.4. The total HMAC process would require 440μW@25MHz and 870μW@50MHz. More than 70% of those values correspond to the hash core process.

Table 3: HMAC-Photon-80/20/16 features for Virtex-5

Power [mW]	Area [Slices]	Max. Freq. [MHz]	Latency[μs]
0.44@25MHz	199	114	192.5@114MHz

[1]Results of MD5 and SHA-1 correspond to Virtex 2 and Virtex 1 whereas SHA-256, Keccak and Photon are provided for Virtex 5.

[2]To the best of our knowledge, there are not results for HMAC implementations based on SHA-3 candidates.

4. DPA RESISTANCE OF THE IMPLEMENTATION

Theoretical security analysis about sponge functions and Photon structure has been performed in [5] and [6]. From data presented in those works, it is immediate to calculate that the HMAC based on Photon-80/20/16 with 60-bit key has 60 bits of security with collision and pre-image resistance of 2^{40}. However, no tests have been performed to define their resistance against the so called hardware attacks. In order to define the resistance of the HMAC structure to power analysis, a DPA has been performed over the construction. Typically, DPA attacks follow a well-defined strategy consisting in the next steps [13]: a) choose an intermediate value of HMAC, obtain the function that models the relationship among the value, the input message, and the key, and map it into power consumption values (usually with Hamming Distance); 2) obtain real power consumption and; 3) compare hypothetical model with real power traces (usually, by calculating the correlation coefficient between both models).

In the HMAC construction considered, the secret key K only affects the internal state value of the sponge function before the text is inserted. The internal state value does not change if K is fixed. From an attacker point of view, there is no difference between revealing the internal state and the key itself. The basic idea of the attack described herein is to find the internal state with differential power analysis.

The complete hash procedure carried out by any Photon-80/20/16 presents an internal state of 5x5 cells of 4 bits per cell. The data is absorbed in groups of 20 bits and a permutation, 'P', is applied to the 25 cells presented in the internal state. A permutation consists in applying 12 times the following four operations: AddConstant, SubCell, ShiftRows and MixColumnsSerial [6]. As a consequence, if the key has 60 bits, it is necessary to wait until 3 key-related blocks are processed before the text enters the Photon internal state. The intermediate results selected to be analyzed during the attack are the differences in the values of the flip-flops associated to the cells in the internal state. Since the text is absorbed in the state and the cells are displaced as shown in Figure 3, the intermediate results are measured after the ShiftRows operation of the first round.

According to this, the three models (HD1, HD2, and HD3) proposed for obtaining each of the five first secrets appear described in Equations 1-3 (where 'Sb' states for Sbox operation, and 'HW' for Hamming Weight).

$$HD1 = HW[Sb(M00 \oplus S00_I \oplus 1) \oplus Sb(M01 \oplus S01_I)] \quad (1)$$

$$HD2 = HW[Sb(M02 \oplus S02_I) \oplus Sb(M03 \oplus S03_I)] \quad (2)$$

$$HD3 = HW[Sb(M03 \oplus S03_I) \oplus Sb(M04 \oplus S04_I)] \quad (3)$$

Each value allows to obtain two secrets simultaneously with each hypothetical value. Since there is no direct relation between the input text and the secrets for the rest of the internal state during first round, it is necessary to wait

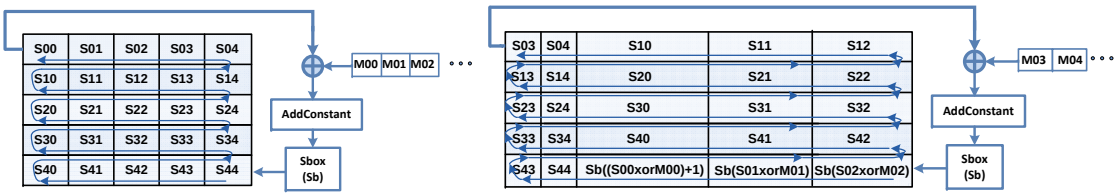

Fig. 3: Absorbing process of the message: (a) initial state with 5x5 cells, (b) the fourth block is absorbed.

until the second round to discover the remaining 20 secrets. Models used are similar to those presented in Equations 1-3, assuming that the initial state is given by the values of the cells presented at the end of the first round.

In order to obtain the power consumption traces experimentally, the HMAC construction was implemented in a SASEBO-II board which contains two FPGAs [14]. One of them (Virtex-5) usually implements the cryptographic target elements, and the other (Spartan-3A) acts as interface to communicate with the external world and to control the measurement modules, such as the oscilloscopes, needed to obtain the power values. In this work, the HMAC module was inserted into the Virtex-5 FPGA whereas the Spartan-3A FPGA stored the secret key and acted as a bridge between the HMAC and a PC. A C program was used to record the traces to generate the plaintexts, and to activate the oscilloscope that measures the power traces when processing each text (the used oscilloscope had 1-GHz bandwidth and a resolution of 8 bits).

Using the previously defined set-up and models, a total of 13,000 input plain texts and 10,000 points per trace were required to obtain a successful DPA attack, which recovers that conform the secret internal state of the Photon hash.

5. CONCLUSIONS

The implemented HMAC based on Photon-80/20/16 features low power and hardware costs together with a suitable speed when processing short messages.Using this proposal, well-known authentication protocols can be used in lightweight applications, avoiding the construction of ad-hoc designed protocols. HMAC security resistance against differential power attacks has been tested. Results show that even for such a constrained implementation without any protection and without key refreshment, it is possible to interchange up to 13,000 messages without compromising the system security.

6. ACKNOWLEDGMENTS

Authors would like to thank the EMSEC of Ruhr-University of Bochum, in particular, A. Moradi and C. Paar.

7. REFERENCES

[1] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *CRYPTO'96*, 1996, pp. 1–15.

[2] J. M. Turner, "The keyed-hash message authentication code (hmac)," *Federal Information Processing Standards Publication*, 2008.

[3] I. Mironov, "Hash functions: Theory, attacks, and applications," *Microsoft Research, Silicon Valley Campus*, 2005.

[4] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Keccak sponge function family main document," *National Institute of Standards and Technology (NIST)*, 2009.

[5] ——, "Sponge functions," in *ECRYPT Workshop on Hash Functions 2007*.

[6] J. Guo, T. Peyrin, and A. Poschmann, "The Photon family of lightweight hash functions," in *CRYPTO'11*. Springer, 2011, pp. 222–239.

[7] K. Jarvinen, M. Tommiska, and J. Skytta, "Hardware implementation analysis of the MD5 hash algorithm," in *HICSS'05*, 2005, pp. 298a–298a.

[8] K. Jarvinen, "Design and implementation of a SHA-1 hash module on FPGAs," *Helsinki University of Technology Signal Processing Laboratory*, 2004.

[9] K. Kobayashi, J. Ikegami, M. Knezevic, E. X. Guo, S. Matsuo, S. Huang, L. Nazhandali, U. Kocabas, J. Fan, and A. Satoh, "Prototyping platform for performance evaluation of SHA-3 candidates," in *HOST'10*, 2010, pp. 60–63.

[10] S. Kerckhof, F. Durvaux, N. Veyrat-Charvillon, F. Regazzoni, G. de Dormale, and F.-X. Standaert, "Compact FPGA implementations of the five SHA-3 finalists," *in CARDIS'11*, pp. 217–233, 2011.

[11] H. Michail, A. Kakarountas, A. Milidonis, and C. Goutis, "Efficient implementation of the keyed-hash message authentication code (HMAC) using the SHA-1 hash function," in *ICECS'04*, 2004, pp. 567–570.

[12] M. Juliato and C. Gebotys, "FPGA implementation of an HMAC processor based on the SHA-2 family of hash functions," *University of Waterloo, Tech. Rep*, 2011.

[13] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer-Verlag New York Inc, 2007, vol. 31.

[14] "Side-channel attack standard evaluation board SASEBO-GII specification." [Online]. Available: http://www.toptdc.com/en/product/sasebo/