Toward Secured IoT Devices: a Shuffled 8-Bit AES Hardware Implementation

Ghita Harcha, Vianney Lapotre, Cyrille Chavet, Philippe Coussy

Toward secured IoT devices: a shuffled 8-bit AES hardware implementation

Ghita Harcha, Vianney Lapôtre, Cyrille Chavet and Philippe Coussy
Univ. Bretagne-Sud, UMR 6285, Lab-STICC
firstname.lastname@univ-ubs.fr

Abstract*— In this paper, we present a lightweight secured AES hardware implementation designed to further resist to Side Channel Attacks relying on Power Analysis. The proposed architecture is based on an 8-bit data-path, and the protection is provided by shuffling computations and memory locations. Our shuffling module is based on a permutation network controlled by a Random Number Generator and leads to the best compromise between security, area, and performances compared to state-of-the-art. Implementation results on a spartan-6 FPGA show that the proposed protection mechanisms impact the area and the timing performance of the unprotected design by factors of 1.58 and 0.35 respectively. Security evaluation based on simulation results shows that the proposed secure architecture resists to a regular CPA by revealing a unique key byte when attacking with up to 1 million traces while state-of-the-art shuffled designs requires only 50000 traces to retrieve the entire secret key. Considering an integrated CPA (also called windowing attack), the proposed architecture allows increasing up to ×300 the required number of traces (Measurements to Disclosure) to retrieve 40% of the key bytes and reveals no more than 9 key bytes when attacking with up to 1 million traces.

***Keywords*—security, SCA, shuffling, HW design, AES**

I. INTRODUCTION

In the era of Internet-of-Things (IoT), devices manipulate more and more sensitive information. Consequently, they have to efficiently protect data through cryptographic primitives. These primitives must be carefully designed to consider stringent energy and area constraints; and they must also resist to various kind of attacks. Side Channel Attacks (SCA) are one of the major threats against cypher designs. The goal of SCAs is to retrieve data unintentionally leaked by cypher implementations. Among SCAs, one can find electromagnetic attacks [11], timing attacks [16] or power analysis attacks [15]. Power analysis attacks can mainly be divided in two categories: Simple Power Analysis (SPA) and Differential Power Analysis (DPA). SPA aims at extracting secret information by analyzing few power traces usually searching for known patterns which are dependent of the target algorithm (i.e. cryptosystem) and its implementation. Differential Power Analysis (DPA) attacks was first proposed by Kocher et al [4], for both hardware and software implementations. It takes advantage of the fact that the power consumption of a given cryptographic device depends on both the data it processes and the related operation it performs. Contrary to SPA, DPA exploits a large number of power traces using statistical methods to extract secret information. In particular, the Correlation Power Analysis (CPA) [23] uses correlation coefficients.

In this paper, we propose an AES-128 implementation based on a small 8-bit architecture for IoT devices and secured against power analysis attacks. The design is based on a shuffling module controlled by an embedded Random Number Generator (RNG). Randomization is applied to computation schedule and storage locations, while ensuring the correctness of the algorithm.

This paper is organized as follows: section II reminds the main concepts of the AES algorithm and introduces state of the art shuffling-based countermeasures against SCA for hardware implementations. Section III presents the proposed shuffled-AES architecture. Section IV details implementation results and provides a security evaluation of the proposed approach. Finally, section V concludes this paper.

II. AES ALGORITHM AND RELATED PROTECTION WORKS

A. Advanced Encryption Standard (AES)

AES is a block symmetric cryptographic algorithm [1]. The plaintext is divided in 128-bits blocks. The internal state is seen as a 4x4 bytes matrix. The blocks are cyphered through a fixed number of loop iterations named rounds depending on the key size. Three standard key sizes are defined: 128, 192 and 256 bits (K_S), which requires 10, 12 or 14 rounds (N_r) respectively. To output a 128-bit *cyphertext,* AES algorithm performs N_r rounds based on four operations:

- S*ubbytes:* input bytes are substituted through a multiplicative inverse in $GF(2^8)$ using a given polynomial followed by an affine transformation.
- S*hiftrows*: a cyclic byte-wise left rotation of each row of the state matrix. The i^{th} row is shifted i positions to the left.
- *Mixcolumns*: The columns of the state matrix are considered as a four-term polynomial over $GF(2^8)$ and are multiplied modulo $(x^4 + 1)$ with a specific matrix.
- *Addroundkey:* the state bytes are *XORed* with the round key.

It is worth noting that there is no data dependency between the state bytes during *addroundkey*, *shiftrows* and *subbytes* operations; however, in the *mixcolumns*, a complete column of the state matrix (i.e. 4 bytes) is needed to compute one output byte. Finally, the round keys can be updated in parallel of each computation round from the initial key. These characteristics are used to design optimized HW implementations as proposed for example in [5].

B. AES protection against power analysis attacks

To resist to SCAs, secured AES designs have been proposed in the literature for both hardware and (most of it) software implementations.

In order to protect cryptosystem implementations against these attacks, main countermeasures can be classified in two groups: hiding and masking [25]. For that purpose, different

approaches have been proposed in the literature including techniques leading to uniform power consumption [13], moving the leakage points in time or space [12], adding delay [19] or dummy operations [20], building non-deterministic processor [7], providing code polymorphism [8], shuffling the execution order of operations [3] and dynamic reconfiguration [21]. It is worth noting that hiding and masking techniques can be combined as proposed in [17] and [18].

Shuffling is a lightweight countermeasure which randomly permutes sensitive computations and storage of a cryptosystem. Shuffling approaches have been well studied in the literature for software implementations. In particular, an important evaluation of shuffling efficiency and cost has been provided in [2]. However, for hardware implementations, very few works describing and evaluating complete shuffled-based architectures have been proposed. In [6], the authors propose a lightweight 8-bit shuffled AES implemented on a SPARTAN-6 FPGA. In order to shuffle independent operations, permutations are generated following the approach proposed in [2]. First, two initial permutations are stored in a 64 x 4-bit memory. Then, a 4-bit register is used as a temporary storage during a swapping process which is driven from a random value. In order to evaluate the proposed implementation in terms of security, a CPA is performed on both the unprotected and the protected designs. The authors conclude that the shuffling approach leads to factor of 250 in terms of number of power traces required to recover the entire secret key compared to the unprotected design. In [9], the authors propose an 8-bit AES architecture implemented on a Virtex-5 which shuffles two rounds. In order to shuffle independent operations, the authors propose an algorithm for permutation generation among a set of operations. However, the algorithm implementation and its impact on the performances are not presented. Regarding security, the authors use Test Vector Leakage Assessment (TVLA) [22] and conclude that the proposed approach allows reducing leakage compared to one round shuffling.

While the existing approaches [6][9] relies on limited number of processing order sequences. In our work the idea is to shuffle both the computation orders and storage locations by using randomized sequences of bytes to process. Contrary to [6] and [9], a sequence is generated in one clock cycle by a dedicated permutation network. This approach limits the impact on the latency of secured architectures.

III. PROPOSED ARCHITECTURE

In this work, we propose a lightweight hardware design of a shuffled AES-128. Our architecture is based on an 8-bit data-path (DP), a shuffling module (SM) and associated controller as described in Fig. 1. A Random Number Generator (RNG) is also used to drive the SM.

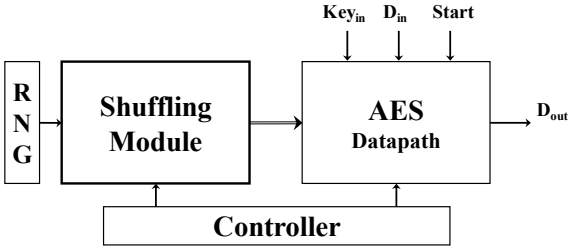

Fig. 1. Proposed shuffled AES Architecture

As mentioned in the previous section, each byte of the 4x4 state matrix can be processed independently from the other one except during the *mixcolumns* operation. Hence, 16! processing orders can be considered for cyphering. The SM must thus be able to generate a large number of permutations: the larger the number of permutations, the better the countermeasure against SCA since it decreases the signal-to-noise existing in side-channel measurements [2]. A network is used to permute a set of 16 constant values ranging from 0 to 15. The random order generated by this network is stored in a memory next used by the controller to drive the data-path. As shown in section IV, different permutation networks such as Benes network [10] or Omega network [24] can be used. Control bits of the network are generated by a RNG. Because the order of computations cannot be predicted and since in the *mixcolumns* each output byte of a column depends on an entire input column, the temporary results must be accumulated into different storage elements (see MEM_i in Fig. 2). Instead of registers classically used in AES implementations like [5], these memories are designed (and controlled) in order to provide correct output for our shuffled AES. In other words, they are designed to support and ensure data dependency during randomized out-of-order AES computations. Moreover, our architecture also shuffles the addressing of the memories. To handle this randomness, we use look-up tables in order to ensure the correct translation between the address of a byte and its "shuffled address" (in the table).

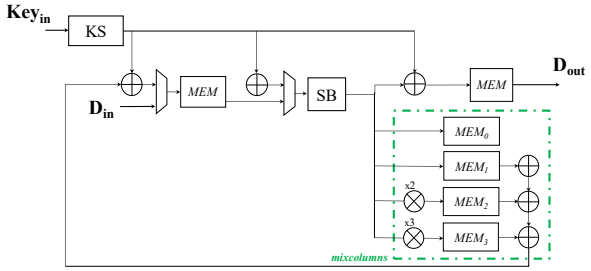

Fig. 2. Architecture AES Datapath

During the *Key Scheduling* (*KS*) the round key is updated (see Fig. 2) and the key bytes are provided to the *addroundkey*. The subbytes (SB) operation is performed through tables stored in memory elements. The addressing of this memory is performed thanks to the results of the previous *addroundkey*.

Because of data dependency, the latency of the *mixcolumns* is increased compared to a non-shuffled implementation [5]. The initial *addroundkey* (on the first round), the *subbytes* and the multiplication of the *mixcolumns* are computed and stored in 16 clock cycles. After these 16 clock cycles, the *mixcolumns* has received all the required data, then it can be completed. The *addroundkey* is then computed. In the last round, 16 additional clock cycles are required. The computation of each round requires 32 clock cycles, so finally the latency of the full encryption is 340 cycles.

IV. EXPERIMENTAL RESULTS

A. Implementation results

The shuffled designs we propose have been implemented targeting a Spartan 6 FPGA, under Xilinx ISE 14.7. In order to fairly analyze the overheads due to the proposed protection

mechanisms, the reference design presented in [5] has been re-implemented with our experimental setup. Results of the original design and our own implementation of [5] can be found in Table I. Area has been kept unchanged and the clock frequency has been increased thanks to the more recent version of the target FPGA and its associated software tool set we used.

Two versions of the shuffling module have been implemented. The first one uses a Benes Network (BN) and the second one uses an Omega Network (ON), later referred as BN-based Cypher (BNC) and ON-based Cypher (ONC) respectively. BN allows the permutation of a set N inputs into any output order leading to *N!* possible permutations [11]. BN was designed with a set of switches, each one composed of two multiplexers controlled by a single bit. To perform *N!* permutations, $\frac{N}{2} * [2 * \log_2(N) - 1]$ switches are necessary. Hence, to shuffle 16 constants values and support *16!* permutations, 56 switches were needed. *ON* has also been considered to shuffle the computation orders of the bytes and memory locations. This alternative to BN brings lower area overhead, since only 32 switches are needed. However, the number of different permutation generated is 2^{32}.

Shift-Register based on *Look-up table* (SRL) of the target FPGA is intensively used to implement efficiently the memories of the data-path (see *MEM* in Fig. 2) and the shuffling module of architectures on a Spartan 6 FPGA. This design option offers the best compromise between complexity (*i.e.*, number of slices) and performances (*i.e.*, clock freq.) with respect to our shuffled computation.

A third shuffled architecture, later referred as Custom BN-based Cypher (CuBNC), which benefits from the parallelism of the AES algorithm to optimize the performances, has also been designed. In this implementation, shuffling limits to the processing order of the columns and to the processing order of the bytes in a column. This optimization decreases the complexity of the data-path and the shuffling module and also reduces the latency overhead. In return, the number of permutations is limited and the permutations are less diversified. The permutations are generated using a two 4x4 BN each composed of 6 switches and providing 4! permutations each, for a total of $(4!)^2$ different permutations. SRL are once again used to implement the memory blocks.

Results from Table I show that regarding area, the proposed shuffling mechanisms lead to factor overheads 1.58, 1.47 and 1.35 for BNC, ONC and CuBNC respectively, compared to the unprotected optimized design. Timing performances are also impacted as explained at the end of the section III. This latency overhead is reduced to 244 cycles with CuBNC since parts of computation are partially ordered in this architecture. Due to a more complex controller for shuffled architectures, the maximal reachable frequency is reduced. This leads to throughputs between 35.58 Mbps to 54.22 Mbps for the protected designs while our unprotected implementation of [5] reaches 100Mbps.

Compared to the shuffled designs of the literature, BNC, ONC and CuBNC offer good area-performance tradeoffs. Indeed, objective of the work proposed in [6] was to minimize the total area. This strategy led to a very low throughput of 7.82 Mbps which is a limitation for computational intensive applications. In contrast, the work presented in [9] leads to an important impact on the area, from 3.2 up to 4.65 in terms of slices depending on the version of [9] (2.3 up to 4.3 for slice LUT, and 3.9 up to 6.75 in terms of slice register).

TABLE I. AREA/PERFOMANCE RESULT AND COMPARAISON

	Unprotected implementation		Protected implementations			This work		
	[5]		[9]		[6]	BNC	ONC	CuBNC
	Original paper	Our design	one round shuffling	two rounds shuffling				
Clock frequency (Mhz)	73	125	82.44	70.02	90	93.41	97.42	106.37
Latency	160	160	n/a	n/a	1471	340	340	244
Throughput ($Mb.s^{-1}$)	58.13	100	n/a	n/a	7.82	35.58	37.11	54.22
Slice	80	80	403	503	24	127	118	108
Slice LUT	n/a	217	842	1188	94	357	301	274
Slice register	n/a	128	464	689	29	102	102	117

B. Security evaluation

1) Experimental setup

In order to evaluate the proposed architecture, execution traces of full AES-128 encryption are extracted from functional post-synthesis simulations. A set of random values are used as inputs of the design for both the plaintext and the permutation control signals. Following the simulation, resulting 'vcd' files are extracted. Then, for each signal transition, the hamming weights of all signals are summed to generated activity traces. Finally, these traces are used for security evaluation through both regular and integrated CPA. For both attacks, the S-Box output of the first round is targeted considering the Hamming weight power model as distinguisher.

2) Regular CPA

In a first step, we performed a regular CPA on the three protected implementations BNC, ONC, CuBNC. Our reference design for the security evaluation is obtained by using an unprotected BNC wherein the shuffling module is disabled.

Fig. 3 shows the number of key bytes revealed for each set of traces. With the non-secure reference design, the key is fully recovered with around 1000 traces. With CuBNC, 16 key bytes are recovered with less than 100000 traces. This result shows that the low permutation diversity of CuBNC allows to optimize both area overhead and performances but does not provide a satisfying level of security against CPA. In contrast, BNC and ONC provide a good resistance to a regular CPA. Indeed, a unique key byte is retrieved when performing a regular CPA considering up to 1 million traces while in [6], the key is recovered with 50 000 traces only.

3) Integrated CPA Attack

When shuffling is used, the leakage signals are randomly spread along the traces which is not taken into account by a regular CPA. For this purpose, an integrated CPA, as described in [14], was implemented. It consists in performing a CPA on an integrated version of the traces. The correlation is then computed between the prediction and the integrated traces. For this advanced attack, only BNC and ONC have been considered since CuBNC implementation is defeated by the regular CPA.

Results presented in Fig. 4 show that no more than 9 key bytes are revealed when attacking with up to 1 million traces. Thus, the entire key was not fully recovered. However, we can consider the implementation as defeated by recovering 40% of the secret key bytes (i.e. 7 bytes) since the remaining bytes can be recovered using a brute force attack. In this context, with the reference design only 100 traces are required to retrieve 40% of

the key bytes while 30000 and 15000 traces are necessary when considering BNC and ONC respectively (see Fig.4). Consequently, the proposed approach provides a security improvement up to a factor ×300 compared to an unprotected solution.

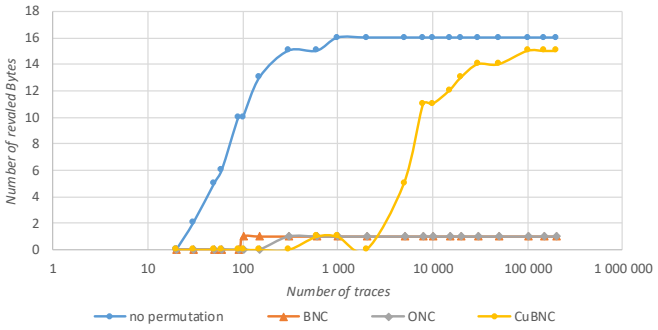

Fig. 3. CPA Attack

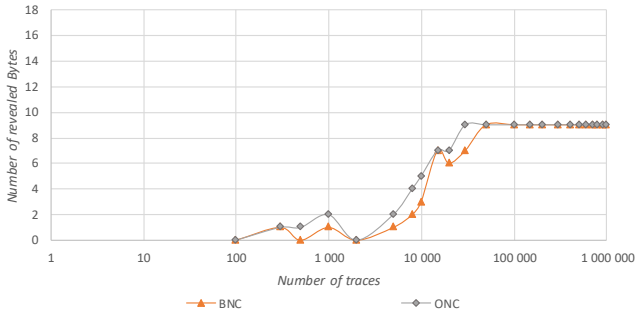

Fig. 4. Integrated CPA Attack

V. CONCLUSION

In this paper, a hardware implementation of an optimized SCA-protected hardware architecture for an 8-bit AES-128 has been presented. For this purpose, the design is based on an original hardware-shuffling module based on a permutation network controlled by an embedded Random Number Generator (RNG). Results show that the proposed approach resists to a regular CPA and provides a security improvement by a factor 300 considering an integrated CPA. Future works will target the extension of the proposed architecture to 16- and 32-bit AES implementations on ASIC and FPGA. Security evaluations based on measured power traces will be performed on FPGA. Coupling our approach with masking techniques will be also studied.

REFERENCES

[1] J. Daemen, V. Rijmen, *The Design of Rijndael AES — The Advanced Encryption Standard*, Springer-Verlag, 2002.

[2] N. Veyrat-Charvillon et al. "Shuffling against side-channel attacks: A comprehensive study with cautionary note." *International Conference on the Theory and Application of Cryptology and Information Security. Springer, Heidelberg, 2012.*

[3] C. Herbst, E. Oswald, S. Mangard, "An AES Smart Card Implementation Resistant to Power Analysis Attacks", in *Proc of the 4th international conference on Applied Cryptography and Network Security* (ACNS'06). Springer-Verlag, Berlin, Heidelberg, *2006.*

[4] P. Kocher, J. Jaffe, B. Jun, "Differential power analysis", *In Annual International Cryptology Conference 1999 Aug 15 (pp. 388-397). Springer, Berlin, Heidelberg.*

[5] J. Chu, J. and M. Benaissa, "Low area memory-free FPGA implementation of the AES algorithm", *In 22nd International Conference on Field Programmable Logic and Applications (FPL), 2012.*

[6] P. Sasdrich, T. Güneysu, "A grain in the silicon: SCA-protected AES in less than 30 slices". *In IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP) 2016*

[7] D. May, H-L. Muller, "Non-deterministic processors". *In Australasian Conference on Information Security and Privacy 2001 Jul 11 (pp. 115-129). Springer, Berlin, Heidelberg.*

[8] D. Couroussé, T. Barry, et al. "Runtime code polymorphism as a protection against side channel attacks". *In IFIP International Conference on Information Security Theory and Practice 2016 Sep 26 (pp. 136-152). Springer, Cham.*

[9] S. Patranabis, et al. "Shuffling across rounds: A lightweight strategy to counter side-channel attacks". *In IEEE 34th International Conference on Computer Design (ICCD) 2016 Oct 2 (pp. 440-443).*

[10] KY. Lee. "A new Benes network control algorithm". *IEEE Transactions on Computers. 1987 Jun;100(6):768-72.*

[11] J.-J. Quisquater and D. Samyde, "Electromagnetic analysis (EMA): Measures and counter-measures for smard cards" *E-smart'2001, vol. 2140 of LNCS, pp. 200-210, Springer-Verlag, 2001*

[12] T. Güneysu and A. Moradi, "Generic side-channel countermeasures for reconfigurable devices". In International Workshop on Cryptographic Hardware and Embedded Systems. Springer, 2011

[13] K. Tiri, I. Verbauwhede "A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation". In Proceedings of Design, Automation and Test in Europe Conference 2004 Feb 16 (Vol. 1, pp. 246-251). IEEE.

[14] M. Rivain, E. Prouff and J. Doget, "Higher-order masking and shuffling for software implementations of block cyphers". *In International Workshop on Cryptographic Hardware and Embedded Systems. Springer, Berlin, Heidelberg, 2009. p.171-188.*

[15] S. Ors, et al, "Power-Analysis Attack on an ASIC AES implementation". In International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. IEEE, 2004. p. 546-552.

[16] PC. Kocher "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems". In Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 1996. p. 104-113.

[17] C. Herbst et al, "An bit AES smart card implementation resistant to power analysis attacks". In International conference on applied cryptography and network security. Springer, Berlin, Heidelberg, 2006. p. 239-252.

[18] A. Kamal et al. An area-optimized implementation for AES with hybrid countermeasures against power analysis, in International Symposium on Signals, Circuits and Systems. IEEE, 2009.

[19] J-S. Coron, I. Kizhvatov, "An efficient method for random delay generation in embedded software". *In International Workshop on Cryptographic Hardware and Embedded Systems. Springer, Berlin, Heidelberg, 2009. p. 156-170.*

[20] J.A. Ambrose, S. Parameswaran, A. Ignjatovic, "MUTE-AES: A multiprocessor architecture to prevent power analysis based side channel attack of the AES algorithm". *In Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design. IEEE Press, 2008. p. 678-684.*

[21] B. Hettwer et al. "Securing Cryptographic Circuits by Exploiting Implementation Diversity and Partial Reconfiguration on FPGAs". *In Design, Automation & Test in Europe Conference (DATE). IEEE, 2019*

[22] G. Becker et al. "Test vector leakage assessment (TVLA) methodology in practice" *In International Cryptographic Module Conference. 2013. p.13.*

[23] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model". *In International Workshop on Cryptographic Hardware and Embedded Systems. Springer 2004.*

[24] J. Lenfant, S. Tahé, "Permuting data with the Omega network" *Acta Informatica, 1985, vol. 21, no 6, p. 629-641.*

[25] S. Mangard, E. Oswald, T. Popp, "Power Analysis Attacks , revealing the secret of smart cards" *in Springer, Heidelberg (2007)*