International Journal of Research
Available at https://edupediapublications.org/journals

p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 13
September 2016

A More Efficient AES Threshold Implementation

*V.NEELIMA **M.SANJAY

*M.TECH ,Dept of ECE, VAAGDEVI ENGINEERING COLLEGE

Warangal

**Assistant. Prof Dept of ECE, VAAGDEVI ENGINEERING COLLEGE

Warangal

Abstract:-

Our contribution is two fold: first we describe a very compact hardware implementation of AES-128, which requires only 2400 GE. This is to the best of our knowledge the smallest implementation reported so far. Then we apply the threshold countermeasure by Nikova et al. to the AES S-box and yield an implementation of the AES improving the level of resistance against first-order side-channel attacks. Our experimental results on real-world power traces show that although our implementation provides additional security, it is still susceptible to some sophisticated attacks having enough number of measurements.

Keywords: side-channel attacks, countermeasures, secret sharing, lightweight, ASIC

1.INTRODUCTION

The mass deployment of pervasive devices promises many benefits such as lower logistic costs, higher process granularity, optimized supply-chains, or location based services among others. Besides these benefits, there are also many risks inherent in pervasive computing: many foreseen applications are security sensitive, such as wireless sensor networks for military, financial or automotive applications. With the widespread presence of embedded computers in such scenarios, security is a striving issue, because the potential damage of malicious attacks also increases. An aggravating factor is that pervasive devices are usually not deployed in a controlled but rather in a hostile environment, i.e., an adversary has physical access to or control over the devices. This adds the whole field of physical attacks to the potential attack scenarios. Most notably are here so called side-channel attacks, especially Simple, Differential and Correlation Power Analysis [6, 18].

1.1 Related Work

Low-power low-area implementations of the AES have been reported in [15] requiring 3100 GE and 160 clock cycles and in [13] requiring 3400 GE and 1032 clock cycles. Both implementations use an 8-bit serialized data path and implement only a quarter of the MixColums operations. The first design, [15], implements two S-boxes and performs the data path and the key schedule operations in parallel, while the latter implementation is fully serial and uses a RAM-like architecture. Canright has investigated very thoroughly how to implement the AES S-box in hardware with minimal area requirements [8]. On the other hand, several masking schemes have been proposed to create a masked AES S-box using either multiplicative or additive approaches. A common approach is to use the tower field representation for an additive masking scheme because of the linearity of the inversion in GF(22). The examples are [4] and [26] which are provably secure, but in practice obvious first-order leakages have been observed [20]. Later, Canright et al. [9] applied the idea of [26] to his very compact S-box resulting in the most compact masked S-box to date. However, as expected its hardware implementation still has first-order leakage.

1.2 Our Work

Our first contribution is a description of the smallest hardware implementation of AES known to date. Our design goal was solely low area, and thus we were able to set the time-area and the power-area tradeoffs differently, and in favour for a more compact hardware realization. To pursue our goal, we have taken a holistic approach that optimizes the total design, not every component individually. In total we achieved an implementation that requires only 2400 GE and needs 226 clock cycles, which is to the best of our knowledge 23% smaller than any previously published implementations.

As a second contribution, we investigate side-channel countermeasures for this lightweight AES implementation. It turns out that when using Canright's representation, the only non-linear function is the multiplication in GF(22). An example for how to share this function using only three shares has been presented by Nikova et al. in [24]. Building on these findings, we applied the countermeasure to our unprotected AES implementation. For this architecture we conducted a complete side-channel evaluation based on real-world power traces that we obtain from SASEBO. We use a variety of different power analysis attacks to investigate the achieved level of resistance of our implementation against first order DPA attacks even if an attacker is capable of measuring 100 million power traces.

1.3 Outline

We first give a brief introduction to Differential Power Analysis and countermeasures in the following Section. A general overview follows a more detailed description of the masking scheme presented in [23, 24], which we use for our experimental evaluation. Subsequently in Section 3 AES and Canright's optimized S-box are briefly recalled, before we describe a shared AES S-box. Based on these findings, in Section 4 we propose two hardware architectures – unprotected and protected – of AES-128 and mount DPA attacks on its real-world power traces
in Section 5.

2.INTRODUCTION TO DPA

Smart cards and other types of pervasive devices performing cryptographic operations are seriously challenged by side-channel cryptanalysis. Several publications, e.g., [12] have stressed that such physical attacks are an extremely practical and powerful tool for recovering the secrets of unprotected cryptographic devices. In fact, these attacks exploit the information leaking through physical side channels and involved in sensitive computations to reveal the key materials. Amongst the known sources of side channels and the corresponding attacks most notable are power analysis attacks [18]. Many different kinds of power analysis attacks, e.g., simple and differential power analysis (SPA and DPA) [18], template-based attacks [2], and mutual information analysis [14], have been introduced while each one has its own advantages and is suitable in its special conditions. However, correlation power analysis (CPA) [6], which is a general form of DPA, got more attention since it is able to efficiently reveal the secrets by comparing the measurements to the estimations obtained by means of a theoretical power model which fits to the characteristics of the target implementation.

2.1 Countermeasures

Generally speaking, the goal of a DPA countermeasure is to prevent a dependency between the power consumption of a cryptographic device and characteristics of the executed algorithm, e.g., intermediate values, executed operations, and taken branches [19]. Amongst the countermeasures proposed at different levels of design and abstraction Masking methods, which rely on randomizing key-dependent intermediate values processed during the execution of the cipher, are widely applied on either the algorithmic level [26] or the cell level [27]. An n-order masking

technique is in fact an (n + 1, n + 1) secret sharing scheme [3, 31], where all shares of the secret are required to proceed. When an algorithmic masking scheme is applied on a microprocessor-based platform, it is often combined by shuffling [16] which randomizes the order of operations. Applying a masking scheme in a software implementation (microprocessor) can be defeated by higher order attacks [11, 34]. However, practical experiences like [20] showed that still there is a first-order leakage when hardware (ASIC or FPGA) is protected by a masking scheme at algorithm level. This leakage can be exploited by sophisticated power models, e.g., toggle-count
model, or by a template-based DPA attack.

In short, currently there exists no perfect protection against DPA attacks. However, applying appropriate countermeasures makes the attacker's task more difficult and expensive. Chari et al. have shown in [10] that up to n-th order DPA attacks can be prevented by using n masks. Following this direction, Nikova et al. extended the idea of masking with more than two shares in [23] to prevent those attacks which use sophisticated power models, e.g., counting the glitches
occuring when the inputs of a complex combinational circuit change. They showed that non-linear functions implemented in such a way, achieve provable security against first-order DPA attacks and also resist higher-order attacks that are based on a comparison of mean power consumption. Estimations of a hardware implementation of these ideas are presented in [24] where an S-box of the Noekeon cipher [17] is considered as a case study without practical evaluation of its resistance to DPA attacks. Afterwards, the same approach is applied on the S-box of the PRESENT cipher [5], and its resistance against first-order attacks is verified in [28]. Since it seems to be a promising candidate for a lightweight and side-channel resistant implementation, we have chosen this scheme to implement the AES S-box and have a comparison (on its first-order leakage) to the masked AES S-boxes proposed so far, e.g., [9] and [26].

3.SHARED COMPUTATION OF THE AES S-BOX USING COMPOSITE FIELDS

In this section first an algorithmic description of AES is given, before the AES S-box as described by Canright is expressed. Finally, the threshold countermeasure of Nikova et al. is applied to the Canright AES S-box that will be used in the next section for a protected implementation of the AES.

3.1 Algorithmic Description of AES

In November 2001 the Rijndael algorithm was chosen as the Advanced Encryption Standard (AES) by the National Institute of Standards and Technology (NIST) [22]. AES is a symmetric block cipher, that processes data blocks of 128 bits. Three different key lengths are specified: 128, 192, and 256 bits, resulting in 10, 12 or 14 rounds, respectively. AES is, depending on the key length, also referred to as AES-128, AES-192, and AES-256 and in the remainder of this
article we focus on the encryption process of AES-128. At the beginning of the algorithm, the input is copied into the State array, which consists of 16 bytes, arranged in four rows and four columns (4 _ 4 - Matrix). At the end, the State array is copied to the output. The bytes of the State are interpreted as coefficients of a polynomial representation of finite field elements in GF(28). All byte values in the remainder of this article will be written in hexadecimal notation in the form {ab}. In encryption mode, the initial key is added to the input value at the very beginning, which is called an initial round. This is followed by 9 iterations of a normal round and ends with a slightly modified final round. During one normal round the following operations are performed in the following order: SubBytes, ShiftRows, MixColumns, and AddRoundkey. The final round is a normal round without the MixColumns stage.

SubBytes is a non-linear, invertible byte substitution and consists of two transformations that are performed on each of the bytes independently: First each byte is substituted by its multiplicative inverse in GF(28) (if existent),
element {00} is mapped to itself. Then the following affine transformation over GF(2) is applied: b 0 i = bi_b(i+5)mod8_b(i+6)mod8_b(i+7)mod8_ci for 0 _ i _ 8, where bi(ci) is the i-th bit of the byte b(c), c = f63g = 011000112.

ShiftRows cyclically shifts each row of the State by a certain offset. The first row is not shifted at all, the second row is shifted by one, the third row by two, and the fourth row by three bytes to the left. MixColumns processes one column of the State at a time. The bytes are interpreted as coefficients of a four-term polynomial over GF(24). Each column is multiplied modulo x4 + 1 with a fixed polynomial a(x) = f03gx3 + f01gx2 + f01gx+f02g. This can be written as the following matrix multiplication, where s0(x) = a(x) s(x):

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \text{for } 0 \le c \le 3.$$

International Journal of Research
IJR
International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 13
September 2016

AddRoundKey adds the 128-bit round key generated from KeyExpansion to the 128-bit State. It is a simple XOR-addition of the round key and the State. KeyExpansion derives 10 round keys from the initial key iteratively. The key is grouped into four words w0, w1, w2, and w3, that consist of four bytes each. w3 is cyclically shifted to the left by one byte. The result is bytewise substituted by the S-box and then a round constant RCon is XOR-added. Finally the result is XOR added to w0 yielding w00 . w01 is obtained by XOR adding w00 with w1, w02 = w01 _ w2 and w03 = w02 _ w3. The new key state or round key RKi is then formed by RKi = w00 jw01 jw02 jw03 . The round constants RConi are derived by the following equation: RConi = xi mod m(x), where i denotes the round number, 0 _ i _ 9 and the irreducible polynomial m(x)= x8+x4+x3+x+1. For further details on AES, the interested reader is referred to [29].

3.2 Canright's Representation of the AES S-box

Canright investigated the hardware requirements of the AES S-box very thoroughly in [8]. He proposed a very compact S-box that is composed of smaller fields. As one can see from Fig. 1 the input to the S-box is transformed by a linear mapping that changes the basis from GF(28) to GF(28)/GF(24)/GF(22) (please ignore pipelining and register remarks in this step, these issues are addressed in Section 3.3 and in Section 5). The output is transformed by a linear mapping that combines the basis change back to GF(28) and the inverse mapping of the AES S-box. Beside two 4-bit XORs, a GF(24) inverter (center module), a GF(24)

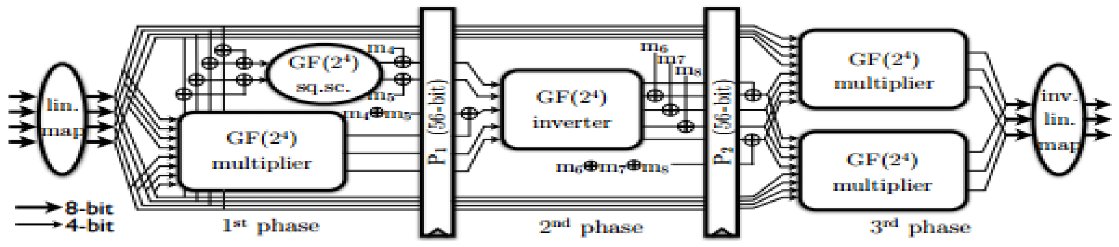


Fig. 2: The Sbox of our implementation.

square-scaler (top left module) and three instances of a GF(24) multiplier (right and bottom left) are required. The GF(24) square-scaler uses a normal basis (□4; □) and only consists of wiring and three XOR gates. The GF(24) inverter uses a normal basis (□4; □) and consists of 5 XOR gates, some wiring and three instances of a GF(22) multiplier (thick lined rectangles)3. The GF(24) multiplier consists of nine XOR gates, some wiring and three parallel instances of a GF(22) multiplier.

3.3 A Shared AES S-box

To apply the threshold countermeasure of Nikova et al. [24] we need to share the non-linear functions of the algorithms, while the linear functions are simply implemented s times in parallel, where s denotes the amount of shares. Particularly interesting are realizations with minimal amount of shares, i.e., s = 3, because they require the fewest hardware resources. Having a closer look on the representation of Canright, it turns out that the only non-linear parts of the AES S-box are the multipliers in GF(22). In [24] an exemplary realization of this multiplier using only three shares has been presented. It is noteworthy to point out that the threshold countermeasure requires registers between different

stages of shared functions. As can be seen from Fig. 1, Canright's S-box representation requires in total five pipelining stages. Note that not only the output of the shared functions, but all signals have to be pipelined. This implies that in total we need to store 174 bits, which as we will see in Section 4 will increase the area requirements even further (please ignore remasked register remarks in this step, this issue is discussed in Section 5).

4. HARDWARE ARCHITECTURES

This section is dedicated to the description of the different hardware profiles that we will attack in the next section. For this purpose we first introduce the design flow used before we detail the hardware architectures, and finally summarize the implementation results.

4.1 Design flow

We used Mentor Graphics ModelSimXE 6.4b and Synopsys Design Compiler version A-2007.12-SP1 for functional simulation and synthesis of the designs to the Virtual Silicon (VST) standard cell library UMCL18G212T3 [33], which is based on the UMC L180 0.18_m 1P6M logic process with a typical voltage of 1:8V. We used Synopsys Power Compiler version A-2007.12-SP1 to estimate the power consumption of our ASIC implementations. For synthesis and for power estimation we advised the compiler to keep the hierarchy and use a clock frequency of 100 KHz, which is a widely used operating frequency for RFID applications.

Note that the wire-load model used, though it is the smallest available for this library, still simulates the typical wire-load of a circuit with a size of around 10 000 GE.

To substantiate our claims on the efficacy of the proposed countermeasures, we implemented the ASIC cores on SASEBO to obtain and evaluate real-world power traces. For design synthesis, implementation and configuration of SASEBO we used Xilinx ISE v10.1.03 WebPACK. In a typical application scenario the cryptographic core would be part of an integrated ASIC, hence for the power measurements on SASEBO we embedded the cryptographic core in a framework that handles the communication between the two FPGAs.

4.2 A Very Compact Implementation of AES

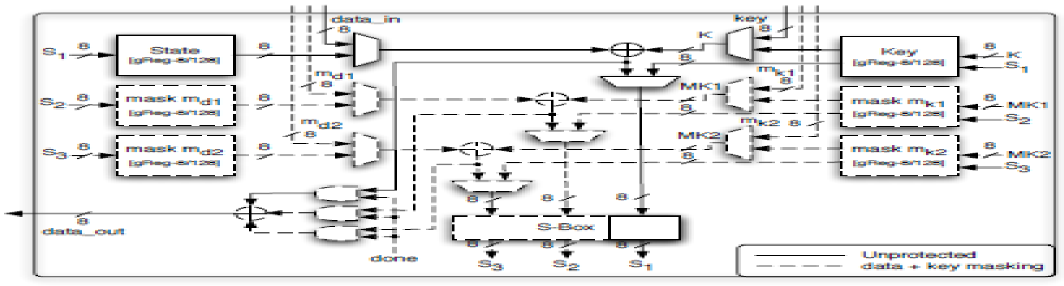
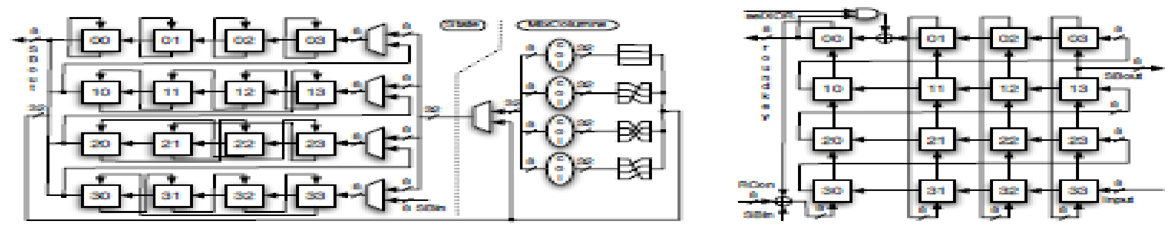


Fig. 2. Hardware architectures of both implementations of a serialized AES-128 encryption-only core.

The most area consumption typically occurs for storing the intermediate state, because typically flip-flops are used, which have high area requirements. In the technology we used, a single-input, positive edge triggered D flip-flop requires 5 GE and can store 1 bit. If you have more than one input, e.g. the output from SubBytes, the output from ShiftRows and the output from MixColumns, you need multiplexers. A Multiplexer for a selection from two inputs to one output (2-to-1 MUX) costs 2.33 GE per bit. Scan flip-flops combine a D flip-flop and a 2-to-1 MUX for 6 GE per bit. That is a saving of 1.33 GE per bit of storage. For the AES this sums up to 340 GE. Scan flip flops have been used before, e.g. in implementations of PRESENT [30] and KATAN/KTANTAN [7].

Based on the properties of scan flip-flops (2 inputs "for free"), we designed the architecture for our tiny AES implementation. As can be seen in Fig. 3, both the State array and the Key array each consist of a 16 stage 8-bit width shift register. Each of the stages comprises 8 scan flip-flops (cells 00 to 33) with two inputs. One input receives the output of the previous stage, while the other one contains the result of ShiftRows, which comes for free in our design, since shifting is done by wiring. Instead of adding one 2-to-1 MUX for every cell of the State array, we designed our architecture in a way that we only need one additional MUX for every row. These are the 4 2-to-1 MUXes (each 8-bit width) on the right hand side of the cells (03) to (33), accounting for 75 GE instead of 300 GE. This choice is strongly related to the choice of parallelism of the MixColumns operation. Both [13] and [15] implemented MixColumns in a serialized way, that is, it takes 4 clock cycles to calculate one column. We opted to implement MixColumns not in a serialized way, because, as we are going to show below, the hidden overhead is larger than the potential savings.



(a) State array with *ShiftRows* and *MixColumns* (b) Key array

Fig. 3. Architectures of storage modules for the *State* and the *Key* arrays.

The Key array consists of a similar 128 flip-flop array as the State array, but the wiring between the registers is different. There are two shifting directions: horizontal and vertical. The current 8-bit chunk of the round key is output during the horizontal shifting, while the S-box look-up for the key schedule is performed during vertical shifting. Note that the RotWord operation is implemented by taking the output of the (13) cell instead of the (03) cell as the input for the S-box look-up. The S-box output is XORed to the round constant RCon and the output of the (00) cell. Once all four S-box look ups have been performed the first column of the key state contains already the new roundkey, but the other three columns do not. The remaining steps of the key update is performed during the output of the round key chunk by XORing the output of cell (00)
to the output of cell (01) as the new input of cell (00). Once the whole row is output, i.e., every fourth clock cycle, the feedback XOR is not required, and thus the output of cell (00) is gated with an AND gate. Note that on top of the cost for storage (768 GE) and the calculation and storage of the round constant (89 GE), in our implementation the whole key schedule requires only one 8-bit AND gate (11 GE), an 8-bit XOR gate with two inputs (19 GE) and an 8-bit XOR
gate with three inputs (35 GE). We believe that our results are very close to a theoretical optimum. This is reflected in the area savings compared to previous results: 924 GE4 vs. 1076 in [15]. [13] uses a RAM-like storage, which includes both, the State and the Key arrays. Thus for a fair comparison

we have to add both modules together: 1678 GE vs. 2040 GE in [13].

In our architecture, MixColumns is realized by four instances of a module called col, which outputs the result of the first row of the MixColumns matrix. Since the matrix used is circulant, one can use the same module and just rotate the input accordingly. Note that in hardware rotation can be realized by simple wiring and comes nearly for free. By serializing MixColumns, one can save 75% of the area (280 GE). Also, 3 of the 4 MUXes on the right hand side of every row can be discarded, and the 32-bit width 2-to-1 MUX (75 GE) at the right hand side of the dashed line in Fig. 3 could be shrinked to an 8-bit width 2- to-1 MUX (19 GE), leading to savings of 112 GE. So in total, the potential savings for the whole design (not only MixColumns) are 392 GE. However, one needs to temporarily store at least 3 of the output bytes, because we cannot over-write the input bytes, before all four output bytes are calculated. That is a storage overhead of 5_24 = 120 GE. Since the MixColumns matrix is circulant, we need to rotate the input to the col module with a different offset for every output byte. This can be implemented by simple wiring (see the right hand side of col in Fig. 3), followed by a 32-bit width 4-to-1 MUX (192 GE) to select the correct input. In summary, the potential savings are in this case reduced to 80 GE, while at the same time one needs far more complex control logic to orchestrate the control signals for the MUXes and the additional temporary storage flip-flops (see below).

Instead of using a Finite State Machine (FSM), we rather spent considerable amount of time and effort to decrease the area requirements for the control logic for the unprotected version (Profile 1). The control signals are derived from a 5-bit LFSR with taps at bit position 1 and 5 that has a cycle length of 21. This is exactly the amount of cycles required to perform one round of AES and the key schedule: 16 cycles for AddRoundKey, 1 for ShiftRows (during which the Key state is not clocked) and 4 for the parallel execution of MixColumns and SubWord. Every time a cycle is completed a pulse is generated that is used to control the MUXes and the clock gating logic. Simple Boolean logic is used to derive all control signals from this pulse, such that in total only 73 GE are required for the control logic. In [15] no details about the control logic are given, and 220 GE are required for both control logic and “others”. Thus a fairer comparison is 80 GE vs. 220 GE. As a consequence of a very serialized implementation, a RAM-like storage, and usage of an FSM, [13] requires 400 GE for control logic (including the round constant generation) compared to 162 GE for our implementation. Similar to [15], we used Canright's description of the AES S-box [8], which is the smallest known.

Our envisioned target application is a very constrained device, e.g. a low-cost passive RFID-tag or similar. By re-ordering the input and output bytes, it is possible to reduce the area significantly, to be precise by 13.5%. As a consequence, our implementation requires an input and output ordering that is row-wise, i.e., S00jS01jS02jS03jS10 : : : S32jS33 and not column-wise (S00jS10jS20jS30jS01 : : : S23jS33), where Sij denotes one byte of the input/output with 0 _ i; j _ 3. If column-wise ordering is needed, 20 additional 8-bit wide 2-to-1 MUXes are required
(373 GE). I n fact with our approach we forward the effort of re-ordering the bytes to the other communication party. In an RFID scenario this will most likely be a reader or a database server, which is by far not as constrained as a passive RFID tag. Hence, the costs for the byte re-ordering are marginal. Furthermore, when two devices with our AES implementation communicate, no byte re-ordering is needed at all. We believe that this re-ordering does not pose a severe problem in practice, while at the same time results in an attractive area saving.

4.3 A Threshold Implementation of AES

If we share both the data path and the key schedule we obtain the threshold version (profile 2). The additional hardware requirements for this profile are depicted in Fig. 2 by the dashed lines. For this profile we need four randomly generated masks (md1, md2, mk1, mk2), which are XORed to the data chunk and the key chunk. The unmasking step is performed by simply XORing all three shares yielding the output (data_out). The state of the masks also needs to be
maintained, which leads to two more instantiations of both the State and the Key module (mask md1, mask md2, mask mk1 and mask mk2). Furthermore, the S-box is now replaced by a shared S-box module that contains five pipelining stages (see Fig. 1). This delays the computation of the round keys and, as a consequence, the pipeline needs to be emptied in every encryption round. Thus profile 2 needs 25 clock cycles for one round and uses a small FSM to derive the control signal (77 GE).

4.4 Performance Figures

Table 1 summarizes the implementation figures of both profiles. The upper part gives a detailed breakdown of the area requirements both in absolute and relative values. The lower part lists the smallest achievable area requirements, power estimations, clock cycles, and throughput at 100 KHz. Profile 1 (unprotected) has an area footprint of 2400 GE of which 70% are required to store the key and the data state. MixColumns and S-box are the other two main contributors to the area requirements. Profile 2 (threshold version) increases the area demands more than four-fold to 10793 GE. The main reason for

Table 1. Breakdown of the post-synthesis implementation results for both architectures of a serialized AES-128 encryption-only core.

Goal	AES-128		*Profile 1* (unprotected) %	*Profile 1* (unprotected) GE	*Profile 2* (threshold version) %	*Profile 2* (threshold version) GE
Area	sequential:	Round constant	3	89	0.5	89
		State array	32	843	8	843
		Key array	32	835	8	842
		m_{d1} array			8	843
		m_{d2} array			8	843
		m_{k1} array			8	842
		m_{k2} array			8	842
	combinational:	MUXes	5	128	3	376
		KeyAdd	1	21	0.5	64
		S-box	9	233	35	4071/4244*
		MixCol	14	373	10	1120
		control	3	72	0.5	77
		other	1	7	0.5	89
	compile simple	sum	100	2601	100	10941/11114*
	compile ultra	sum	2400 GE		10793/11031* GE	
	cycles		226 clk		266 clk	
	power @100 KHz		3.7 μA		13.4 μA	
	throughput @100 KHz		57 Kbps.		48 Kbps.	
Area and Speed	*compile ultra*	sum	2421 GE			
	cycles		210 clk			
	power @100 KHz		3.7 μA			
	throughput @100 KHz		61 Kbps.			

* using remasked registers excluding PRNGs (explained in Section 5)

this is the S-box, which increases more than 10 fold and now occupies a whopping 35% of the area. This increment mainly comes from the 13-fold increment of the GF(22) multiplier (13 GE vs. 173 GE) and the four pipelining stages that need to store an additional 174 bits (870 GE).

Profile 1 requires 21 clock cycles per round and 16 clock cycles to output the result (226 clock cycles in total). Profile 2 needs 4 additional clock cycles per round, due to the pipelining stages in the S-box, which leads to a total of 266 clock cycles (18% increment). Please note that the time required can be reduced by 16 clock cycles for additional 21 GE for profile 1 and 64 GE for profile 2 by adding another XOR gate for the final KeyAdd allowing to interleave consecutive message blocks. The power consumption was estimated at 100 KHz and a supply voltage of 1:8V. The unprotected implementation (profile 1) requires 3:7 _A and thus is suitable for passive RFID-tags. For profile 2, however, this figure increases more than threefold to 13:4 _A, which might already decrease the reading range of a passive RFID tag. If required, power saving techniques might be applied to reduce the power consumption at the cost of additional area. Please note that power figures for different standard-cell libraries cannot be compared in a fair manner. Furthermore, power estimates vary greatly depending on the simulation method used and effort spent. Therefore we did compare our power figures with previous works.

5. EXPERIMENTAL RESULTS

In addition to the performance and area consumption features of our threshold implementation, we have implemented the whole AES encryption design on an FPGA-based platform and analyzed the actual power consumption traces to practically investigate its resistance to first-order DPA attacks. Later in this section the platform used and the measurement setup are introduced, then practical results are shown to validate the desired security levels.

5.1 Measurement Setup

A SASEBO (Side-channel Attack Standard Evaluation Board) which is particularly designed for side-channel attack experiments [1] has been used as the measurement platform. It contains an xc2vp7 Virtex-II Pro FPGA [35] as the crypto FPGA, clocked at a frequency of 3MHz5, to implement the design. A LeCroy WP715Zi 1.5GHz oscilloscope at a sampling rate of 1GS/s and a differential probe which captures voltage drop of a 1 resistor at VDD (1:8V) path are used as the measurement equipments to collect the power traces.

5.2 Side-Channel Resistance

In order to find the leakage points and have a reference to fairly judge about the power analysis resistance of our implementation, we have switched off the mask generators and kept all masks as zero to prevent randomization by masking. 100 000 traces are collected from this implementation while encrypting random plaintexts. As expected and also observed in [20], CPA attacks which use a HW model predicting the S-box input or output are not able to recover the secrets of hardware implementations. What should directly lead to a successful attack is a CPA using HD model which predicts bit flips on a part of the state register when S-box outputs are overwritten to each other. Therefore, two consecutive key bytes, i.e., 216 hypotheses, should be guessed. The results of such an attack, which shows the amount of information leakage related to register updates, is depicted by Fig. 4(a). Note that to reduce the attack complexity we have given a favor to the attacker by knowing a key byte and reducing the key hypotheses to 28. As shown in Fig. 4(b), around 30 000 traces are sufficient to perform asuccessful attack. Bcause of the pipeline architecture of the S-box the correct key guess appears at more than one clock cycle in the attack results. Also, a

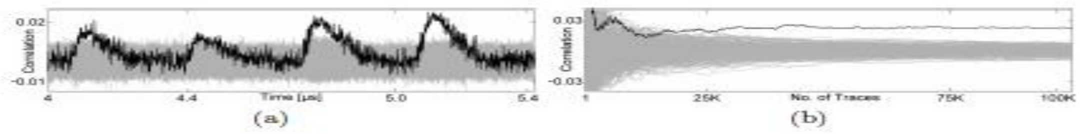


Fig. 4. CPA attack results when the mask generators are off by means of a HD model (a) using 100K measurements and (b) at point 5.1µs over the number of traces.

mutual information analysis attack using the same distinguisher, i.e., HD of the register updates, is efficiently capable of recovering the secret. The results of this attack are shown in Fig. 5(a) and Fig. 5(b). It is noteworthy to mention that those four clock cycles in which the secret leaks clearly in both Fig. 4 and Fig. 5 are when the intermediate results of the target S-box computation are consecutively stored in the pipeline registers of the shared S-box.

In order to observe the combinational circuit leakage a correlation-enhanced collision attack, presented in [21], is mounted by getting average over the acquired traces based on the plaintext bytes, and correlating the mean traces after alignment based on the clock cycles when the target S-boxes are computed. In fact, this attack is very similar to a template-based DPA attack using only the mean vectors of the templates and avoiding the profiling step. The result of this attack presented in Fig. 6 shows that the leakage of the combinational circuit, i.e., the S-box instance, also leads to successfully revealing the linear difference between two key bytes.

In the second step we have measured 5 million traces while the random number generators are turned on and work normally. The plaintext bytes are randomly selected, and the masks are shared neither between the plaintext and key bytes nor between computation rounds of encryptions. In short, there is no mask reuse in our target design. All attacks, mounted on the first step when the random number generators were off, are repeated on the new measurements. The CPA attack using HD, whose result is shown in Fig. 7(a), is expectedly not successful since registers are masked by means of three shares and the predicted HD does not fit to the register updates. However, the registers which contain the shares are updated at the same time, and their information leakages through

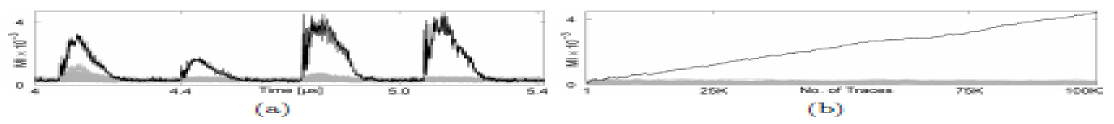


Fig. 5. MIA attack results when the mask generators are off by means of a HD model (a) using 100K measurements and (b) at point 5.1μs over the number of traces.

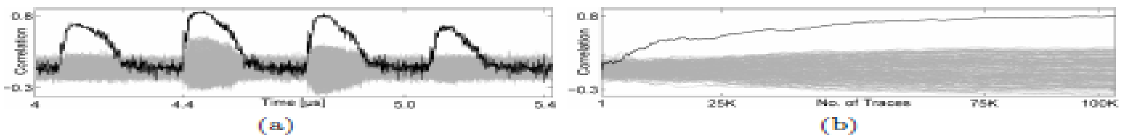


Fig. 6. Correlation collision attack results when the mask generators are off (a) using 100K measurements and (b) at point 4.8μs over the number of traces.

power consumption are inherently summed up. As observed in [32] the sum of shared registers leakages is not independent of the actual (unshared) value, and a mutual information analysis is expected to recover the secret. We have repeated the last mutual information analysis attack by means of a HD model as the distinguisher. The corresponding attack result is shown in Fig. 7(b), but it still cannot distinguish the correct hypothesis. This might be related to the number of traces; in other words, 5 million traces seem to be not enough due to the amount of switching and electronic noise in our platform. However, the same issue has been addressed in [25], where it is argued that the combinational functions following the registers change the distribution of shared register leakages leading to failed mutual information analysis attacks.

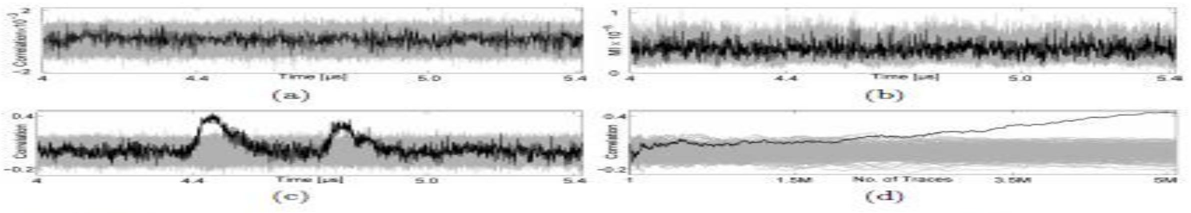


Fig. 7. Attack results when the mask generators are working using 5 million traces (a) CPA using a HD model, (b) MIA using a HD model, (c) correlation collision attack, and (d) correlation collision attack at point 4.45μs over the number of traces.

On the other hand, repeating the last correlation collision attack, whose results are given in Fig. 7(c) and Fig. 7(d), led to revealing the secret using around 3:5 million traces. Since this attack recovers the first-order leakage of combinational circuits, it shows that our shared S-box still has first-order leakage. During the investigation of this issue (as also addressed in [25]) we have realized that the values which are saved in the intermediate registers of our shared S box are not uniformly distributed. This means, property 3 illustrated in [23] and [25] does not hold although we have used the shared multiplication in GF(22) proposed by the original authors. The problem arises when the output of the shared multiplication modules which have some shared inputs are leakage of the combinational circuit caused by the glitches leads to a recoverable first-order leakage. Since searching through all possible correction terms and their combination to check whether they lead to a uniform distribution in our design was a very time consuming task, we could neither check all possible cases nor could we find a suitable case. Instead, (as also addressed in

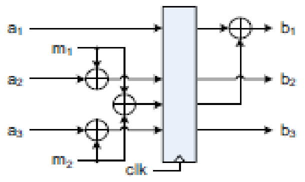


Fig. 8. Remasking scheme for a 3-share case

Finally 100 million traces have been acquired from the last design when all random number generators worked normally and the plaintext bytes were randomly selected. It should be noted that the fresh masks for the remasked registers are provided by means of LSFRs which have enough period considering 100 million measurements. All the attacks illustrated have been repeated here on all measured traces. A CPA and an MIA using a HD model on S-box outputs are still not mixed by means of the linear functions. In fact, the correction terms which have been added to the shared multiplications to provide uniformity are canceled out. It is actually a practical evidence showing that if the uniformity property does not hold, the [25]) we have tried to use random fresh masks inside each pipeline stage when required. The scheme we have used to add fresh masks, so-called remasking, is shown by Fig. 8. We have simulated our shared S-box and tried to find the minimum cases where remasking is required, and finally yielded the design shown in Fig. 1; the remasked registers are marked by O. applicable; their results are depicted in Fig. 9(a) and Fig. 9(b) respectively. Also, we have performed a third-order CPA attack by cubing the power traces and correlating the results to predictions of a HD model in order to recover the leakage of the inherently summed shared register updates. The result of this attack shown in Fig. 9(c) indicates that 100 million traces are still not enough for such a higher-order attack. The correlation collision attack is also not applicable. Its results are shown in Fig. 9(d). This means that our target design could prevent the first-order leakage under Gaussian assumption since correlation collision attack applies only the mean traces6. This confirms the statement given in

[25] that the average power leakage of a threshold implementation should be independent of the processed values. We examined several models and performed a couple of mutual information attacks, and finally could make the secret distinguishable using HD of the S-box input. Using this model, similar to correlation collision attacks, the linear difference between two key bytes can be recovered. The result of this attack is shown by Fig. 9(e) and Fig. 9(f), and indicates that the secret gets distinguishable using more than 80 million traces.

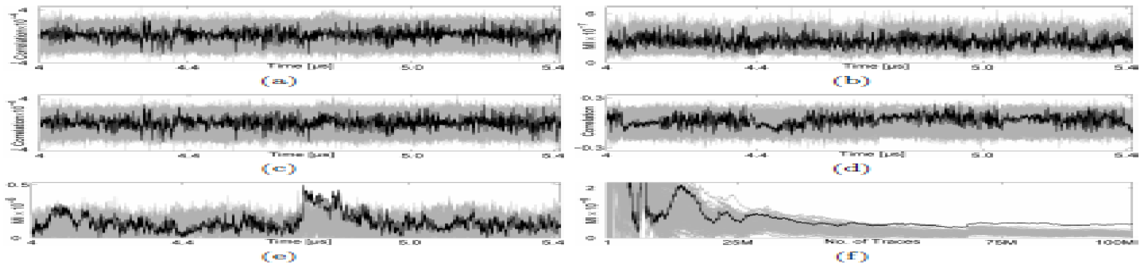


Fig. 9. Attack results when the mask generators are working and the remasked registers are applied using 100 million traces (a) CPA and (b) MIA and (c) third-order CPA using a HD model on S-box outputs, (d) correlation collision attack, (e) MIA using a HD model on S-box input, and (f) MIA at point 4.7μs over the number of traces.

6. CONCLUSION

While implementations of cryptographic algorithms in pervasive devices seriously face area and power constraints, their resistance against physical attacks has to be taken into account. Unfortunately, nearly all side-channel countermeasures introduce power and area overheads which are proportional to the values of the unprotected implementation.

Therefore, this fact prohibits the implementation of a wide range of proposed countermeasures and also limits possible cipher candidates for ubiquitous computing applications. Most of the countermeasures proposed for implementing a side-channel resistant

AES in hardware remained unfortunately with a first-order leakage. In this article we have applied a recently proposed secret sharing-based masking scheme to the AES S-box in order to improve the first-order resistance. Decomposition of the AES S-box into a series of S-boxes of algebraic degree two and splitting them into (at least) three shares is a challenging task. However, we have used the architecture of the smallest AES S-box and have shared the non-linear operation which is a GF(22) multiplier. To separate the glitches of different parts of the circuit we have designed the S-box in five pipeline stages by adding four sets of intermediate registers and applying a remasking scheme on some selected registers. Our proposed hardware architecture for the AES reduces the area requirements to only 2400 GE, which is

23% smaller than the smallest previously published. After the secret sharing based countermeasure has been applied, the area requirements are 11031 GE, while the timing overhead compared to our unprotected implementation with a similar architecture is only 18%. According to practical side-channel investigations, masking the state and the key registers by means of two shares each could improve the resistance against the considered (most well-known) first-order DPA attacks. Our protected implementation offers 128-bit standardized security with improved side-channel resistance for around 11 000 GE.

REFERENCES

1. Side-channel attack standard evaluation board (sasebo). Further information are available via http://www.rcis.aist.go.jp/special/SASEBO/index-en.html.

2. D. Agrawal, J. R. Rao, and P. Rohatgi. Multi-channel Attacks. In CHES 2003, volume 2779 of LNCS, pages 2–16. Springer, 2003.

3. G. R. Blakley. Safeguarding Cryptographic Keys. In National Computer Conference, pages 313–317, 1979.

4. J. Blömer, J. Guajardo, and V. Krummel. Provably Secure Masking of AES. In SAC 2004, volume 3357 of LNCS, pages 69–83. Springer, 2004.

5. A. Bogdanov, G. Leander, L. Knudsen, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT - An Ultra-Lightweight Block Cipher. In CHES 2007, volume 4727 of LNCS, pages 450–466. Springer, 2007.

6. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In CHES 2004, volume 3156 of LNCS, pages 16–29. Springer, 2004.

7. C. D. Cannière, O. Dunkelman, and M. Knezevic. KATAN & KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In CHES 2009, volume 5747 of LNCS, pages 272–288. Springer, 2009.

8. D. Canright. A Very Compact S-Box

AUTHOR 1:-

* **V.NEELIMA** completed her B tech in BALAJI INSTITUTE OF TECHNOLOGY & SCIENCE in 2013 and completed M-Tech in VAAGDEVI ENGINEERING COLLEGE.

AUTHOR 2:-

****Mr. M.SANJAY** is working as Asstistant. Prof in Dept of ECE VAAGDEVI ENGINEERING COLLEGE.