ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

A High-Performance VLSI Architecture for Threshold Implementations Illustrated on AES

K.Anusha
M.Tech,
VLSI System Design,
GNIT JNT University, Hyd.
anushashankar93@gmail.com

M.Suman Kumar
Associate Professor,
GNIT JNT University, Hyd.
Sumankumar.gnit@gmail.com

B.Kedarnath
HOD,
Dept of ECE,
GNIT JNT University, Hyd.
bkedarnath@gmail.com

Dr.S.Sreenatha Reddy
Principal,
GNIT JNT University, Hyd.
Sreenath_sakkamm@yahoo.com

Abstract:
Embedded cryptographic devices are vulnerable to power analysis attacks. Threshold Implementations provide provable security against first-order power analysis attacks for hardware and software implementations. Like masking, the approach relies on secret sharing but it differs in the implementation of logic functions. While masking can fail to provide protection due to glitches in the circuit, Threshold Implementations rely on few assumptions about the hardware and are fully compatible with standard design flows. We investigate two important properties of Threshold Implementations in detail and point out interesting trade-offs between circuit area and randomness requirements. We propose two new Threshold Implementations of AES that, starting from a common previously published implementation, illustrate possible trade-offs. We provide concrete ASIC implementation results for all three designs using the same library, and we evaluate the practical security of all three designs on the same FPGA platform. Our analysis allows us to directly compare the security provided by the different trade-offs, and to quantify the associated hardware cost.

Keywords:
Threshold Implementation, First-order DPA, Higher order DPA, Glitches, Sharing, AES, S-box.

I. INTRODUCTION:

An increasing number of embedded devices implement some security functionality, for instance smart cards (banking, SIM, public transport, access control, passports), car keys, set-top boxes (pay TV), media players, mobile phones, tablets, medical implants, etc. These devices use cryptographic algorithms that are secure against mathematical cryptanalysis. This means that a system's security relies on the secrecy of a so-called cryptographic key, and that there are no mathematical shortcuts that allow breaking the system. However, in the late 90s the security of such devices has been shown to depend also on the algorithms' implementation [1]. During the computation of an algorithm the device leaks information, for instance through its power consumption, electromagnetic emanations, etc. Side channel attacks (SCA) can reveal the key from these leakages and are often inexpensive; hence they are among the most relevant threats for the security of implementations of cryptographic algorithms.

Certain countermeasures against SCA aim to introduce noise in the side channel, e.g. random delays, random order execution, dummy operations, etc., while masking conceals all sensitive intermediate values of a computation with random data. Different masking schemes, like additive [2], [3] and multiplicative [4], have been proposed in order to provide security against differential power analysis (DPA) attacks. 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

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

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 socalled side-channel attacks, especially Simple, Differential and Correlation Power Analysis.

II. EXISTING SYSTEM:

A. RAW IMPLEMENTATION:

This TI of the S-box (details will be given in the following section) requires four input shares; therefore we initially share the plaintext in four shares. We share the key in two shares and XOR them with two of the plaintext shares before the S box operation. More details about the key scheduling will be given later in this section.

B. ADJUSTED IMPLEMENTATION:

Each of the existing three shares is XORed with a random byte and the sum of these random bytes is taken as the fourth share. This also ensures uniformity of the S-box input. Together with the state, the number of shares for Mix Columns and Key XOR increases to three.

III. EXISTING SYSTEM DRAWBACKS:

- The longest critical path
- The maximum area of occupancy
- Low speed

Proposed 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 favor for a more compact hardware realization, compared to [13] and [15]. 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 (2 2). 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.

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.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

A Threshold Implementation of AES:

If we share both the data path and the key schedule we obtain the threshold version. 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 . 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.

IV. IMPLEMENTATION:

In this section we will discuss three different TIs of AES which we refer to as raw, adjusted and nimble implementations. All implementations share the same data flow and timing. The implementations differ mostly in the S-box calculation and/or the number of shares that are used in different blocks of the algorithm. The raw implementation is from our paper at Africa Crypt 2014 [17] and forms the basis of the other two implementations. Hence, we will mainly describe the raw implementation and point out the differences with the other two. The main feature of the raw implementation is that it uses the smallest possible number of shares for each function, except the linear transformations in the S-box, provided that the shared functions are uniform. In other words, all nonlinear operations are performed with $n > 2$ shares such that the circuits are uniform and n is as small as possible. The linear operations outside the S-box are performed with two shares, whereas the linear operations in the S-box use two, three or four shares. The adjusted implementation on the other hand ensures that at least three shares are used in every operation, including the linear ones. With this implementation we intend to observe the effect of moving from at least two shares to at least three shares in linear operations on the resistance against higher order DPA, and to quantify the associated cost. In the nimble implementation the number of shares is always minimal, i.e. $n = d+1$ where d is the degree of the unshared function, even if the resulting shared function is not uniform. The uniformity of the circuit is satisfied by re-masking.

General Data Flow:

We use a serial implementation for round operations and key schedule as proposed in [16], [17] which requires only one Sbox instance and loads the plaintext and key byte-wise in row wise order. We also use one Mix Columns instance that operates on the whole column and provides an output in one clock cycle. Due to this extreme serialization, one round requires at least 21 clock cycles even for the unprotected implementation [16]. All our TIs execute one round in 23 clock cycles. In the first 16 clock cycles, the plaintext is XORed with the key and sent to the S-box. Its output will be taken from the 3rd to the 18th clock cycles and stored in the state registers, i.e. the S-box is executed in three clock cycles. The Shift Rows operation is performed in the 19th clock cycle followed by four cycles of Mix Columns calculation. The S-box takes its input from the key schedule for four cycles starting from the 18th cycle. In the 17th, 22nd and 23rd clock cycles, the S-box inputs and unused random bits are set to 0. Therefore, the calculation of AES takes $23 \times 10 + 16 = 246$ clock cycles, including 16 cycles to output the cipher text.

1) Raw implementation:

We use two sets of state registers, each consisting of sixteen 16-bit registers, corresponding to the two shares of the state. The MixColumns and the Key XOR operations are also performed with two shares. This can be seen in Fig. 1, as the key and the state registers are 256 bits implying the two shares.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

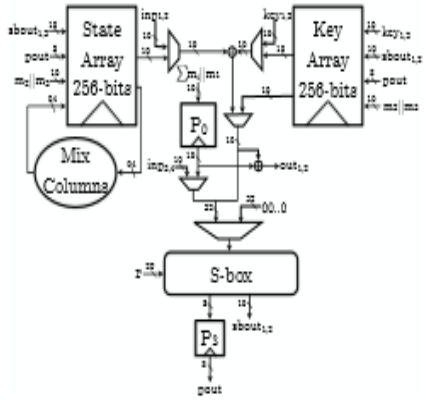


Figure 1: Architecture of the serialized TI of raw AES-128.

This TI of the S-box (details will be given in the following section) requires four input shares, therefore we initially share the plaintext in four shares. We share the key in two shares and XOR them with two of the plaintext shares before the Sbox operation. More details about the key scheduling will be given later in this section. Besides the shared input, the S-box needs 20-bits of randomness r.

The first two output shares sbout1,2 are written to the state register S33 (Fig. 2) whereas the remaining share sbout3 is written to register P3. The data in the state registers are shifted to the left for the following 16 cycles so that the next output of the S-box can be stored in the same registers. During this shift, the data in P3 (pout in Fig. 1) is XORed with the second share of the S-box output, which is in the state register S33, to reduce the number of shares from three to two. To achieve this signal sig2 is active from the 4th to the 19th clock cycle.

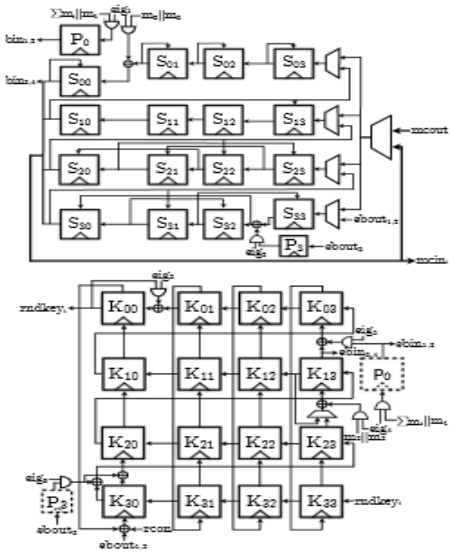


Figure 2: Architecture of the state (top) and key (bottom) arrays for our raw implementation where Si, Ki and P0 hold two shares and P3 holds one share. The registers P0 and P3 are used by the state and the key array. The XOR of the value in P3 and S33 (resp. K30) is on one share of the value in register S33 (resp. K30) whereas all the other combinational operations are on two shares.

2) Adjusted implementation:

This version works on three shares for both the state and the key schedule which increases the area significantly. The S-box still requires four input shares and outputs three shares, hence the register P0 is reduced to 8-bits (one share) and the register P3 is not required. Similar to the raw implementation, we use 24-bits of randomness to increase the number of shares from three to four one cycle before the S-box, i.e. each of the existing three shares is XORed with a random byte and the sum of these random bytes is taken as the fourth share. This also ensures uniformity of the S-box input. Together with the state, the number of shares for MixColumns and Key XOR increases to three.

3) Nimble implementation:

Similar to the raw implementation, this one also uses two shares for the state and key arrays. The main difference is that the S-box needs three input shares instead of four.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Hence the size of the register P0 is reduced to 8-bits (one share). As a result, we need only 16bits of randomness to increase the number of shares from two to three before the S-box operation, i.e. each share is XORed with one byte of randomness and the XOR of the random bytes is taken as the third share. The S-box requires 16-bits of extra randomness per iteration and outputs three shares. Hence the logic of the register P3 to reduce the number of shares back to two stays the same.

TI of the AES S-box:

The S-box implementations in [16] use the tower field approach up to GF(22) for a small implementation. Therefore, the only nonlinear operation is GF(22) multiplication which must be followed by registers and re-masking to avoid first order leakages. We also chose to use the tower field approach, however, we decided to go until GF(24) instead of GF(22). With this approach, the GF(24) inverter (algebraic normal form provided in Appendix B) can be seen as a four bit permutation and the GF(24) multiplier (algebraic normal form provided in Appendix A) as a four bit multiplication both of which are well studied in [22]. Therefore, we can find uniform TIs for each of these nonlinear functions. This might allow us to reduce the number of fresh random bits needed since we will have fewer nonlinear blocks compared to [22] hence possibly require less re-masking in order to use their outputs. Moreover, with this approach the S-box calculation takes three clock cycles instead of five.

1) Raw implementation:

The uniformity of each function is individually satisfied. The uniform sharing with four input and three output shares that is used to share each term in the multiplication is provided in Appendix C. For the inversion, which belongs to class C4282 [14], we consider two options. Either using four shares, which is the minimum number of shares necessary for a uniform implementation in that class, and decomposing the function into three uniform sub-functions as Inv(x) = F(G(H(x))), or using five shares without any decomposition.

Our experiments show that both versions have similar area requirements but need a different number of clock cycles. To reduce the number of cycles, we chose the version with five shares, generated by applying the formula in Appendix F to each term of the inversion. This sharing is found by using the method described in [9] which is slightly different from the direct sharing [14]. We chose this sharing since it can be implemented in hardware with less logic gates compared to the direct sharing.

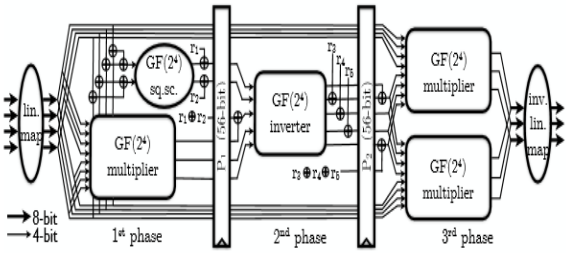


Figure 3: The S-box of the raw implementation.

Even though it is enough to use only two shares for linear operations, we sometimes chose to work on more than two shares to avoid the need of extra random bits. The linear map of the tower-field S-box operates on four shares since the multiplication needs four input shares. The inverter requires five input shares and the multiplication outputs only three shares, therefore we use two shares for the square scalar to have five shares in the beginning of the 2nd phase. We use three shares for the inverse linear map of the tower-field S-box since the multiplication outputs three shares.

For all the linear operations, the shared functions are created as instantiations of the unshared function for the first share and as unshared function without the constant term for the other shares. During the combination of these uniform circuits, we face the challenges described in Section II-E to keep the uniformity in the pipeline registers. We apply re-masking on the first pipeline register where we combine the two output shares of the square scaler and the three output shares of the multiplier to generate five shares.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Note that this combination also acts as the XOR of the outputs of the square scaler and the multiplier. By Theorem 4, it is enough to re-mask only the output shares of one of the functions to achieve uniformity. We choose to re-mask the output of the square scaler since it operates on less shares, hence requires less random bits. The correction mask,i.e. the XOR of the masks, is XORed to one of the output shares of the multiplier to achieve correctness.

2) Adjusted implementation:

As mentioned in the earlier sections, the only difference between the raw and the adjusted implementation is that the adjusted implementation requires at least three shares for all the blocks including the linear operations in the S-box. For that reason, the shared square scaler circuit is instantiated with three shares. This Sbox also requires 44-bits of randomness per iteration.

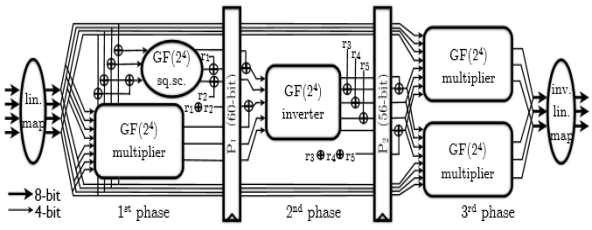


Figure 4: The S-box of the adjusted implementation.

3) Nimble implementation:

As can be observed in Figs. 3 and 4, we use fresh randomness at the end of the 1st phase to satisfy uniformity during the combination of the square scaler's and the multiplier's outputs, and after the inverter to break the dependency between the inputs of the multipliers in the 3rd phase. Since these re-masking steps conserve the uniform it y property and the security of each block is achieved only by the correctness and non-completeness properties (Observation 1), we can discard the uniformity property and implement these nonlinear functions with the smallest number of shares ns.t. n > d, i.e. n = d + 1, where d is the degree of the unshared functions.

We use the sharing with three input and output shares provided in Appendix D for each term of the multiplier and the sharing with four input and output shares provided in Appendix E for each term of the inverter. With this new construction, it is enough to have three input shares to the S-box since the multiplier block requires only three shares. We need to reduce the number of shares from five to four at the end of the 1st phase for the inverter and from four to three at the end of the 2nd phase for the following multipliers. This construction requires only 32-bits of extra randomness per S-box calculation, including increasing the number of shares for the S-box input.

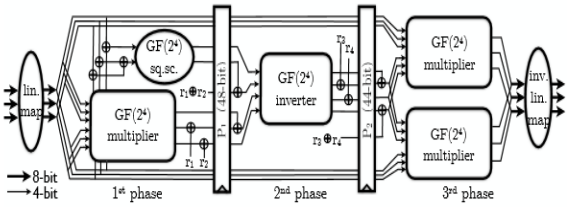


Figure 5: The S-box of the nimble implementation.

Performance:

Like any other DPA countermeasure, TI also allows tradeoffs between area, randomness and the resistance against DPA. In Table III, we provide the area costs (GE) and randomness requirements (bits) for the different S-box implementations. For all the implementations, we performed two different compilation methods. The first one is are gular compilation with the compile command,that does not optimize or merge modules, performed on the whole implementation. The second method on the other hand uses the compile ultra command for each module to let the tool optimize each of them individually and combine the result. It is very important that the modules are not merged for area optimization in this step, to not violate the non-completeness property.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

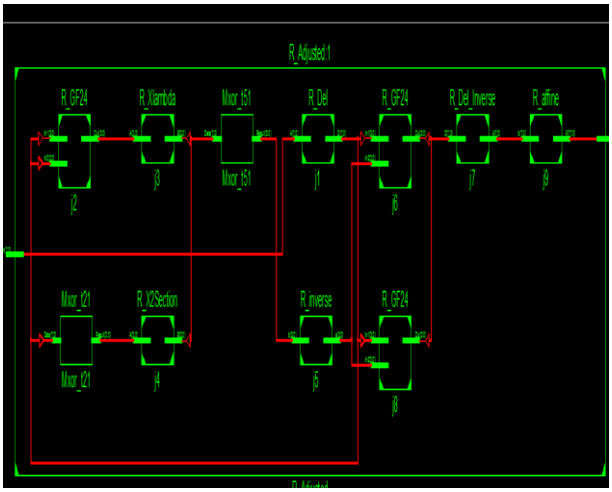
S-box	Raw	Adjusted	Nimble
Lin.Map.	168 / 120	168 / 120	126 / 90
Sq.Sc.	18	27	18
Multiplier	625 / 458	625 / 458	418 / 308
Inverter	618 / 490	618 / 490	594 / 375
Inv.Lin.Map	99 / 72	99 / 72	99 / 72
1st Ph.	919 / 704	916 / 701	646 / 500
2nd Ph.	690 / 562	702 / 574	654 / 435
3rd Ph.	1374 / 1013	1374 / 1013	959 / 713
Registers*	725	661	576
Total	3708 / 3004	3653 / 2949	2835 / 2224
Random.	44	44	32

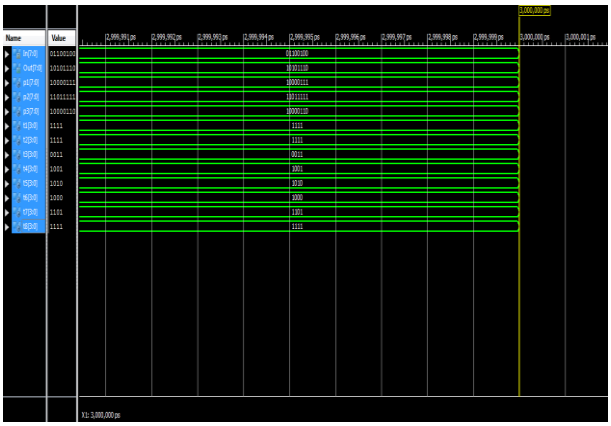
*: including the registers P_0 and P_3

Table: Synthesis results for different versions of S-box TI with compile / compile ultra commands.

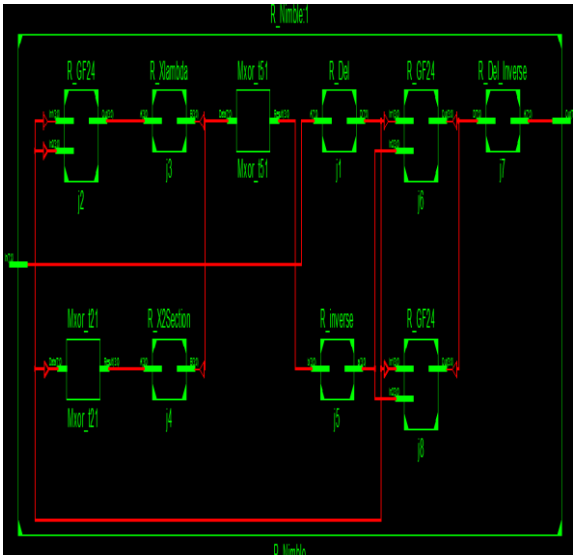
In our implementations, the S-box occupies 30% to 40% of the total area. Compared to the implementation in [16] our S-boxes with uniform blocks are 13% smaller and our Sbox with non-uniform blocks is 33% smaller. These results show a significant area and randomness improvement for the nimble implementation, indicating that using non uniform shared functions can be advantageous if the uniformity of the circuit is satisfied by re-masking.

V. RESULTS:

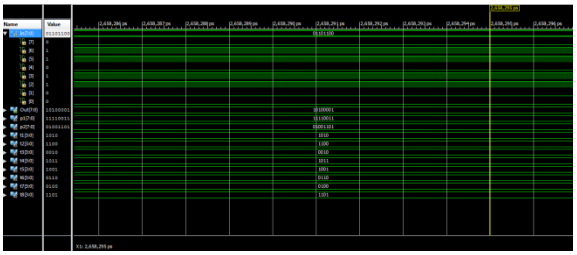




Adjusted simulation

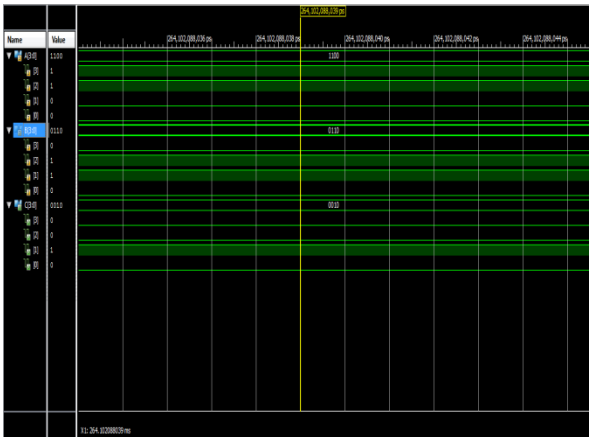


Nibble implementation synthesis



ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Nibble implementation simulation



Raw implementation simulation

Raw implementation synthesis

VI. APPLICATIONS:

- ATM machines
- Wireless communication
- Mobile Phones
- Image processing and Network security

VII. HARDWARE REQUIREMENT:

- FPGA Spartan 6

VIII. SOFTWARE REQUIREMENT:

- ModelSim 6.4c
- Xilinx 9.1/13.2

IX. FUTURE ENHANCEMENT:

The advanced encryption standard (AES) is a specification for the encryption of electronic data. The AES algorithm is a symmetric block cipher that can encrypt (encipher) and decrypt (decipher) information. Encryption converts data to an unintelligible form called cipher text; decrypting the cipher text converts the data back into its original form, called plaintext.We will implement the nimble implementation in AES Encryption.

X. CONCLUSION:

We discuss three different versions of TIs of AES. We show that it is possible to achieve first-order DPA resistance with non-uniform shared functions ifre-masking is applied properly. In the case of AES, our "non-uniform" nimble implementation requires less randomness than our "uniform" raw implementation, due to the decreased number of shares. However, for other algorithms and other S-boxes, re-masking may increase the amount of randomness required. This idea can be used to trade-off between the randomness and area requirements. Moreover, we empirically confirm that increasing the number of shares has a significant impact on the performance of higher-order attacks, which provides another trade-off between area and DPA resistance. Our most efficient implementation is approximately 8k GE small and requires only 32 bits of fresh randomness per S-box calculation, which is a significant improvement over all previous works.

REFERENCES:

[1] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in CRYPTO, ser. LNCS, vol. 1666. Springer, 1999, pp. 388–397.

[2] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in CRYPTO, ser. LNCS, vol. 1666. Springer, 1999, pp. 398–412.

[3] L. Goubin and J. Patarin, "DES and differential power analysis the "duplication" method," in CHES, ser. LNCS, vol. 1717. Springer, 1999, pp. 158–172.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

[4] T. S. Messerges, "Securing the AES finalists against power analysis attacks," in FSE, ser. LNCS, vol. 1978. Springer, 2000, pp. 150–164.

[5] S. Mangard, T. Popp, and B. M. Gammel, "Side-channel leakage of masked CMOS gates," in CT-RSA, ser. LNCS, vol. 3376. Springer, 2005, pp. 351–365.

[6] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully attacking masked AES hardware implementations," in CHES, ser. LNCS, vol. 3659. Springer, 2005, pp. 157–171.

[7] A. Moradi, O. Mischke, and T. Eisenbarth, "Correlation-enhanced power analysis collision attack," in CHES, ser. LNCS, vol. 6225. Springer, 2010, pp. 125–139.

[8] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold implementations against side-channel attacks and glitches," in ICICS, ser. LNCS, vol. 4307. Springer, 2006, pp. 529–545.

[9] S. Nikova, V. Rijmen, and M. Schl¨affer, "Secure hardware implementation of nonlinear functions in the presence of glitches," J. Cryptology, vol. 24, no. 2, pp. 292–321, 2011.

[10] E. Prouff and T. Roche, "Higher-order glitches free implementation of the AES using secure multi-party computation protocols," in CHES, ser. LNCS, vol. 6917. Springer, 2011, pp. 63–78.