The ISC Int'l Journal of Information Security

November 2022, Volume 14, Number 3 (pp. 93–99)

http://www.isecure-journal.org

Presented at the ISCISC'2022 in Rasht, Iran.

Secure and Low-Area Implementation of the AES Using FPGA ☆

Muhamadali Hajisoltani [1], Raziyeh Salarifard [2,*], and Hadi Soleimany [1]

[1] *Cyberspace Research Institue, Shahid Beheshti University, Tehran, Iran.*

[2] *Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran.*

ARTICLE INFO.

Keywords:
Side-Channel Attacks, FPGA, Threshold Implementation, AES

Type:
Research Article

doi:
10.22042/isecure.2022.14.3.10

dor:
20.1001.1.20082045.2022.14.3.10.4

ABSTRACT

Masking techniques are used to protect the hardware implementation of cryptographic algorithms against side-channel attacks. Reconfigurable hardware, such as FPGA, is an ideal target for the secure implementation of cryptographic algorithms. Due to the restricted resources available to the reconfigurable hardware, efficient secure implementation is crucial in an FPGA. In this paper, a two-share threshold technique for the implementation of AES is proposed. In continuation of the work presented by Shahmirzadi *et al.* at CHES 2021, we employ built-in Block RAMs (BRAMs) to store component functions. Storing several component functions in a single BRAM may jeopardize the security of the implementation. In this paper, we describe a sophisticated method for storing two separate component functions on a single BRAM to reduce area complexity while retaining security. Out design is well suited for FPGAs, which support both encryption and decryption. Our synthesis results demonstrate that the number of BRAMs used is reduced by 50% without affecting the time or area complexities.

1 Introduction

One of the most challenging aspects of cryptographic algorithm implementation is protecting them from side-channel attacks. The attacker exploits information leaked from the hardware on which the cryptographic primitive is running to extract the secret value (the secret key) via a side-channel attack. Data leakage can occur in a variety of ways, including increased power consumption, processing time, and electromagnetic radiation. Differential power analysis is one of the most effective side-channel attacks [1]. Several software and hardware techniques have been proposed to defend against this type of attack [2, 3]. Masking techniques are one of the most popular countermeasures against side-channel attacks. Providing secure and efficient masking techniques are more subtle for hardware implementation due to the presence of glitch and their effect on information leakage [4]. Various solutions have been proposed to mitigate these challenges. Threshold Implementation (TI) is one of the secure and efficient masking methods for the hardware implementation of cryptography algorithms, which can provide provable security against power analysis attacks [5].

Suitable hardware masking against power analysis attacks with the desired level of security and perfor-

* Corresponding author.

☆ The ISCISC'2022 program committee effort is highly acknowledged for reviewing this paper.

Email addresses: mu.hajisoltani@Mail.sbu.ac.ir, r_salarifard@sbu.ac.ir, h_soleimany@sbu.ac.ir

ISSN: 2008-2045

mance is indispensable. There is usually a trade-off between different parameters like "area", "latency", and "random bits". Numerous masking techniques have been devised to enable the masked implementation of various cryptographic primitives, including AES [6–10]. The majority of proposed masking approaches for AES implementation use either Application-Specific Integrated Circuits (ASICs) or microcontrollers as the platform for implementation. While the proposed ASIC-based techniques can also be implemented on FPGA as well, the FPGA resources are not always efficiently utilized. To fill this gap, the use of BRAM units on the FPGA chip has been introduced recently as an effective method for decreasing the area in [11]. Shahmirzadi *et. al.* proposed a two-share threshold technique for the implementation of AES which is secure in the Glitch Extended attack model. The proposed method in the [11] has the minimum component functions for a secure implementation of the AES algorithm. In this paper, we aim to revisit the previous technique proposed in [11], to minimize the number of BRAMs as much as possible. Our technique does not increase the latency or the number of random bits. More precisely, the properties of our technique are as follow:

- Choosing separate component functions that are stored on a single BRAM maintains the security of the implementation.
- Individual component operations can be implemented on a single BRAM without adding additional area or time complexity.

In comparison to the previous work, our method leads to a 50% reduction in the utilization of the number of BRAMs.

The rest of this paper is organized as follows. Section 2 gives a short introduction to threshold implementation and the attack model. The threshold implementation of the AES S-box is presented in Section 3. Section 4 discusses a method for reducing the implementation area. Section 5 presents the experimental results. Finally, we conclude the paper in Section 6.

2 Preliminaries

A sensitive value x is divided to $d+1$ shares $(x_0, \ldots, x_d)$ in the masking schemes. This sharing must be done in such a way that the value x can be recovered only when all $d+1$ shares are available. Such masking provides d-order security against side-channel attacks. In Boolean masking, x is split into different shares such that $x = \oplus_{\forall i} x_i$.

2.1 Threshold Implementation

As previously stated, "glitch" acts as an undesirable event in electronic circuits. The threshold implementation [5] is the initial solution proposed for glitch handling. Correctness, non-completeness, and uniformity are three properties of a threshold implementation that can maintain the security of hardware implementation. Assume we seek to implement a function $y = f(x)$ in such a way that the input and output are calculated using the $d+1$ shares. In other words, the function f accepts the values $x_0, \ldots, x_d$ and returns the values $y_0, \ldots, y_d$.

- **Correctness** indicates that the masking function is correct for all input shares and produces the expected output as it is demonstrated in Equation 1.

$$\forall x_i, f(\oplus_{\forall i} x_i) = \oplus_{\forall i} y_i \tag{1}$$

- **Non-completeness** ensures that the calculation of each output share reveals no information about the secret value. For example, if the function f is a collection of functions f_i (also known as component functions), each of these f_i will calculate the appropriate y_i value by processing a set of input shares that lack the i_th input share $(x_0, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d)$. As a reason, each output share is independent of at least one input share, and processing it reveals no information about x.
- **Uniformity** implies that if the input x is uniformly distributed, the function's output must be uniformly distributed as well. Otherwise, the function in the next phase will be given inputs with a non-uniform distribution. Re-masking at the output is one of the most well-known strategies for persuading this characteristic [6].

2.2 Attack Model

A glitch is a transient state created at the output of a logic circuit as a result of a delay discrepancy in the logic circuit's inputs or internal wires and gates. Due to the nature of glitches, forecasting their accurate effect is a challenging task. Glitch Extended Probing [12] is an attack model that has been developed to investigate and consider the impact of glitches on the security of the cryptographic primitive's hardware implementation. The model assumes that by probing any point of the circuit, the attacker will have complete knowledge of not only that specific point but also all the values involved in the path of the logic circuit from that point back to the last synchronization point. The number of points probed by the attacker concurrently defines the attack's order (or security order).

2.3 Number of Shares

The required number of input shares can be computed using Equation 2 in the threshold implementation:

$$t.d + 1 \tag{2}$$

where t is the degree of the function, and d represents the implementation security order. This number can be lowered without compromising the security of the implementation by adding extra random bits [7, 13]. The required number of shares, in this case, is shown in Equation 2.

$$d + 1 \tag{3}$$

Equation 4 illustrates a secure implementation of an AND gate ($y = f(a, b) = ab$) proposed in [7] for the security order $d = 1$.

$$\begin{aligned} &f_0(a_0, b_0) = a_0b_0 \rightarrow y_0^{'} \\ &f_1(a_0, b_1, r) = a_0b_1 + r \rightarrow y_1^{'} \quad y_0^{'} + y_1^{'} = y_0 \\ &f_2(a_1, b_0, r) = a_1b_0 + r \rightarrow y_2^{'} \quad y_2^{'} + y_3^{'} = y_1 \\ &f_3(a_1, b_1) = a_1b_1 \rightarrow y_3^{'} \end{aligned} \tag{4}$$

where a_0, a_1, b_0, and b_1 represent the input shares, y_0 and y_1 represent the output shares, r represents a random bit for re-masking to preserve uniformity, and f_i represents the component functions of the function f. The outcomes of the component functions (i.e. $y_i^{'}$) must be stored in a register and then compressed into two shares before proceeding to the next level function.

3 Threshold Implementation of AES with Two Shares

This section describes the threshold implementation of AES presented in [11]. To compute the S-box of AES, an inversion over $GF(2^8)$ is first performed, followed by an Affine transform over $GF(2)$. To provide first-order security ($d = 1$), the inputs are split into two shares. To minimize the amount of component functions assigned to the S-box, the following process is suggested in [11]: A matrix of component functions is defined as f_{ij}, where the i_{th} row contains the indices that should be given as input to the i_{th} component function. As an illustration, $(0, 1, 1)$ means the i-th component function is $f_i(a_0, b_1, c_1)$. Besides, the indices associated with the j_{th} input bit are represented in the j_{th} column of the matrix. As an illustration, consider the following example:

$$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} b_0 \\ b_1 \\ b_0 \end{pmatrix}$$

In other words, columns correspond to the input bits, whereas rows correspond to the component functions. As a result, we can construct a sharing table in this manner. Equation 5 demonstrates how to implement the function $y = ab + c$ by utilizing four component functions.

$$\begin{pmatrix} 0\ 0\ 0 \\ 0\ 1\ 0 \\ 1\ 0\ 1 \\ 1\ 1\ 1 \end{pmatrix} \begin{matrix} f_0(a_0, b_0, c_0) = a_0b_0 + c_0 \\ f_1(a_0, b_1, c_0) = a_0b_1 \\ f_2(a_1, b_0, c_1) = a_1b_0 \\ f_3(a_1, b_1, c_1) = a_1b_1 + c_1 \end{matrix} \tag{5}$$

The objective is to reduce the size of the S-box implementation by reducing the number of component functions required to realize a 2-share masked form of 8-to-1 cubic functions. Each sharing table has a maximum of eight rows and eight columns (number of input bits). For example, the elements $(0, 1, 1, 0, 0, 1, 1, 0)$ in the i_{th} row indicate that the component function f_i, receives values $(a_0, b_1, c_1, d_0, e_0, f_1, g_1, h_0)$, and may contain terms such as $a_0b_1d_0$. According to [11], AES S-box contains 12 component functions, as illustrated below:

$$\begin{pmatrix} 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0 \\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1 \\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1 \\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0 \\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0 \\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 1 \\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1 \\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0 \\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0 \\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 1 \end{pmatrix} \begin{matrix} f_0(a_0, b_0, c_0, d_0, e_1, f_1, g_0, h_0) \\ f_1(a_0, b_0, c_0, d_1, e_1, f_0, g_1, h_1) \\ f_2(a_0, b_0, c_1, d_0, e_0, f_0, g_0, h_1) \\ f_3(a_0, b_0, c_1, d_1, e_0, f_0, g_1, h_0) \\ f_4(a_0, b_1, c_0, d_1, e_0, f_1, g_0, h_1) \\ f_5(a_0, b_1, c_1, d_0, e_1, f_0, g_1, h_0) \\ f_6(a_1, b_0, c_0, d_0, e_0, f_0, g_1, h_0) \\ f_7(a_1, b_0, c_1, d_1, e_1, f_1, g_0, h_1) \\ f_8(a_1, b_1, c_0, d_0, e_1, f_1, g_1, h_1) \\ f_9(a_1, b_1, c_0, d_1, e_1, f_0, g_0, h_0) \\ f_{10}(a_1, b_1, c_1, d_0, e_0, f_1, g_0, h_0) \\ f_{11}(a_1, b_1, c_1, d_1, e_0, f_0, g_1, h_1) \end{matrix} \tag{6}$$

The authors of [11] proposed the following technique for designing a masked variant of AES S-box that has low latency (fewer register stages are required) and reduced fresh randomness. The inversion over $GF(2^8)$ can be represented as $X^{-1} = x^{254}$. X^{254} can be computed by decomposition into cubic permutations, as illustrated in Equation 7 [14].

$$\begin{aligned} X^{254} = (X^n)^m = (X^m)^n, \\ HW(n) = HW(m) = 3, \end{aligned} \tag{7}$$

where $HW(\alpha)$ denotes the Hamming Weight of α.

The authors of [11] chose the parameters as $m = 49$, $n = 26$ due to the lack of preference for any pair of m and n over the other. Based on the decomposition demonstrated in Equation 7, the cubic functions

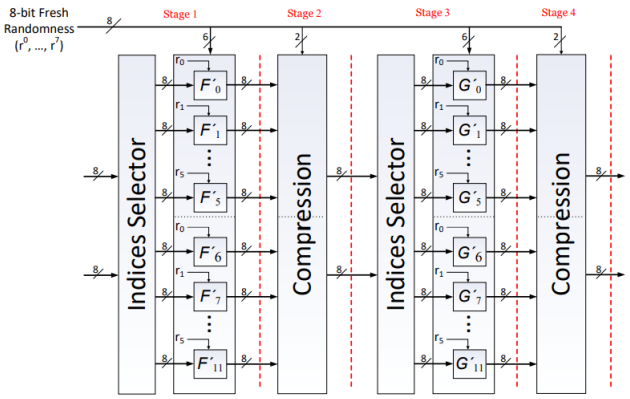

Figure 1. The structure of the two-share S-box in [11]

$F(X) = X^n$ and $H(X) = X^m$ can be utilized for the implementation of AES S-box and its inverse can be computed as demonstrated in Equation 8 and Equation 9, respectively.

$$S(X) = A \circ H \circ F(X) = G \circ F(X) \quad (8)$$

$$S^{-1}(X) = F \circ H \circ A^{-1}(X) = F \circ W(X) \quad (9)$$

Each of the F, G, and W functions in Equation 8 and Equation 9 can be implemented as 12 component functions, each of which is implemented in a BRAM. Figure 1 demonstrates the overall architecture, where r_i signifies random bits.

Each BRAM contains nine input bits (eight main bits and one random mask bit), which are located on the BRAM address port and eight output bits. The indices selection layer allocates the shares among the component functions, and the compression layer is required to maintain the uniformity property in the subsequent step.

To implement both encryption and decryption in the same circuit, an additional bit is added to the location of the address port's most important bit that determines the operation. Since BRAMs have two separate ports, the relevant encryption and decryption component functions can be written on a single BRAM. Figure 2 illustrates the final implementation approach. 24 BRAMs are used for the AES S-box and its inverse. In the presented scheme in [11], 8 S-boxes are used for encryption, 8 S-boxes for decryption, and 4 S-boxes for the key expansion operations. As a result, 240 BRAMs are employed throughout the implementation.

4 Proposed Secure and Low-Area Implementation of the AES

In this section, we propose a secure and low-area implementation of AES leveraging threshold implementation with two shares. The component functions are implemented based on the look-up table implementation and are stored in BRAMs. The proposed approach and its security against glitch-extended probing attacks are explained further below.

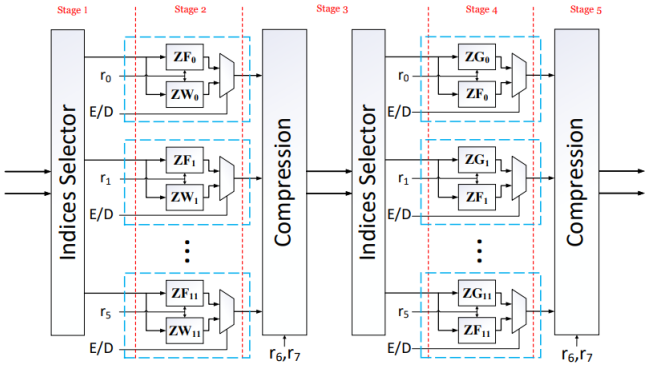

Figure 2. The structure of the S-box and its inverse in [11]

4.1 Overview of Our Approach

In the threshold implementation of AES proposed in [11], the F and G functions (used for encryption) and the W and F functions (used for decryption) are implemented utilizing up to 24 BRAMs. Furthermore, the information stored on each BRAM is accessed using just 10 bits of the address, implying that only $2^{10} = 1024 = 1Kbit$ of each BRAM is used.

Our goal is to limit the amount of required BRAMs to reduce the area. We want to utilize this approach by storing more component functions in a single BRAM unit. Two factors must be taken into account:

- The modified implementation should be secure in the glitch-extended probing attack. Simple use of this approach can weaken the implementation's security.
- We should consider the trade-off between area, delay, and the number of new random bits. In most applications, decreasing the area by increasing the amount of new randomness or delay is undesirable.

In summary, increasing the number of component functions in a single BRAM should be done in such a way that the implementation's security is not compromised and the delay and quantity of fresh random bits are not increased.

4.2 The Proposed Architecture

BRAM's address port is ten bits in length, where the most valuable bit (MSB) determines the encryption and decryption operations. We adopt the architecture proposed in [11]. The uniformity condition of threshold implementation must be applied to the address

port due to the BRAM design. The random mask r is utilized to achieve uniformity. r is allocated in the 9-th bit of the address port. The remaining eight bits represent a combination of input shares. Additionally, BRAMs contain two ports, A and B, that are fully independent of one another and simply share the data they carry. Port A is used to implement the component operations of the encryption operation, whereas port B is used to execute the component functions of the decryption process.

To reduce the implementation area, we will implement the functions of the corresponding F_i and G_i using a single BRAM unit. To make the best use of the remaining BRAM capacity in this way, the length of the data on the BRAM address port should be modified. More precisely, we increase the address length by one bit. As a result, the BRAM address is increased to 11 bits. In our design, we consider twelve BRAMs each of which contains the component functions G_i and F_i which will be used for encryption and decryption, respectively. Similarly, we consider twelve BRAMs each of which contains the component functions F_i and W_i which will be used for encryption and decryption, respectively. In the encryption (res. decryption) process, first, F_i (res. W_i) should be accessed and then G_i (res. F_i) should be accessed. We refer to these two steps as the first step and second step, respectively. To perform the first step of the encryption (res. decryption), the data of F_i (res. W_i) will be read from the corresponding BRAM when the MSB of the address is 0. To perform the second step of the encryption (res. decryption), the data of the G_i function from the encryption operation and the F_i function from the decryption operation will be read from the BRAM, when the most valued bit of the address is 1.

Now we need to control the input address dependent on whether the data is for the first or second step of the computation. We add a multiplexer to the input of the indices selector layer for this purpose.

It's worth noting that the multiplexer's selector bit is the same as the bit we added to the MSB of the address bit. Because when this bit is 0, it indicates that we want to perform the first step operation and will obtain the corresponding address from the multiplexer; and when it is 1, it indicates that we want to perform the second step operation and will obtain its address from the multiplexer using the output of the first step operation. As a result, we do not require a separate circuit to control this multiplexer. The new implementation is illustrated in Figure 3.

Figure 4 provides too much detail about each of the blocks depicted in Figure 3.

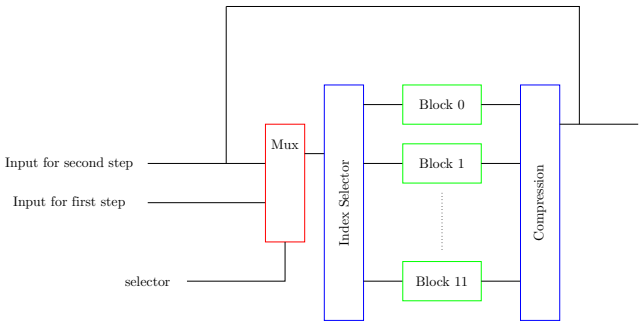

Figure 3. New structure

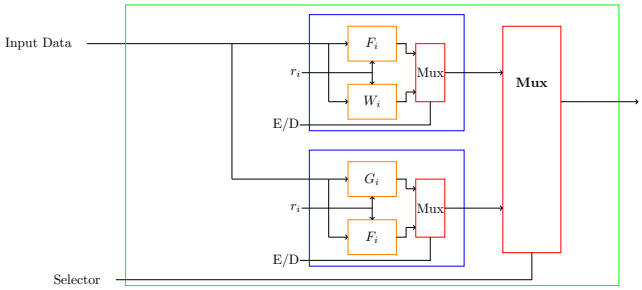

Figure 4. Detailed structure of each block

4.3 Evaluation of Proposed Design

In our design, we selected the component functions that are located in each BRAM wisely. Each BRAM includes only F_i, W_i and G_i. In other words, the BRAM does not include two different component functions with different indices. For instance, we could have considered F_i and F_j in one BRAM where $i \neq j$. However, this strategy was not suitable as it could lead to violating the non-completeness property. Since the component functions F_0 to F_{11} (as well as the functions G_0 to G_{11}) are allocated in different BRAMs, in our design, the security maintains in the Glitch Extended Probing. Even if we assume that the attacker knows all the values involved in the BRAM, he is not able to retrieve any information about processed data.

A BRAM unit reads the address, determines its data, and places it in the output register in around two cycles; that is, the process of reading the data takes one clock pulse, and the operation of setting it on the output register takes one clock pulse. The output register is an internal register of this block that can also be disabled, but it is required in this implementation technique to maintain uniformity in the output of BRAMs.

Because the compression layers and the index selector do not require a clock pulse, the result of the first step operation is instantly sent to input, and the BRAM begins reading the data of the second step operation at this moment. This two-stage register (two clock pulses of processing time) allows us to reduce

the area while maintaining the same implementation latency. It's worth mentioning that the quantity of fresh random bits has remained constant throughout this process. Because the structure [11] uses 8 random bits for each S-box and its inverse, for a total of 160 random bits, there is no change in the number of S-boxes and thus the number of fresh random bits in our proposed method.

5 Results

In this section, the FPGA implementation results of the proposed (Threshold Implementation of) AES architecture are presented. The target hardware platform for implementations is mainly Spartan-6 of the Xilinx family.

We choose a number of Threshold Implementation of AES published in [11, 15, 16] for comparison. To the best of our knowledge, they appear to be the most efficient in comparison to the other previous work. Furthermore, we chose the area, time and throughput metrics as comparison points.

The architecture has been implemented on spartan-6 and the results of this experiment have been presented in Table 1. The BRAM of the proposed architecture implemented on Spartan-6 is 50% lower than the best previous work [11]. Since our architecture supports both encryption and decryption mode, we have compared our experimental with the E/D architecture presented in [11]. Moreover, the required time for these implementations is, respectively, 10.8% and 2.9% less than the reported in [11, 16]. Furthermore, the throughput of the proposed architecture is 3% higher than the best previous work [11]. It is worth noting that, as compared to the best recent work, the exploited BRAM is half while all other parameters remain the same. Furthermore, as compared to designs that do not use BRAM, the number of required registers is reduced.

6 Conclusion

A method for first-order secure implementation of the AES encryption algorithm on an FPGA chip is provided in this work, as well as a way for decreasing its implementation area and hence the utilization of FPGA chip resources. This area optimization was accomplished without affecting the implementation latency.

So even though higher-level secure implementations necessitate the use of significantly more resources from the FPGA chip (e.g., order 2 security, where the attacker can probe two gate outputs simultaneously, increasing the number of required component functions to establish security), this method of reducing the area will enable even more secure implementations on most FPGA chips.

References

[1] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[2] Elena Trichina. Combinational logic design for aes subbyte transformation on masked data. *IACR Cryptol. EPrint Arch.*, 2003:236, 2003.

[3] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Annual International Cryptology Conference*, pages 463–481. Springer, 2003.

[4] Stefan Mangard, Thomas Popp, and Berndt M Gammel. Side-channel leakage of masked cmos gates. In *Cryptographers' Track at the RSA Conference*, pages 351–365. Springer, 2005.

[5] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *International conference on information and communications security*, pages 529–545. Springer, 2006.

[6] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of aes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 69–88. Springer, 2011.

[7] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *TIS@ CCS*, page 3, 2016.

[8] Takeshi Sugawara. 3-share threshold implementation of aes s-box without fresh randomness. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 123–145, 2019.

[9] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. A more efficient aes threshold implementation. In *International Conference on Cryptology in Africa*, pages 267–284. Springer, 2014.

[10] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Trade-offs for threshold implementations illustrated on aes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(7):1188–1200, 2015.

[11] Aein Rezaei Shahmirzadi, Dušan Božilov, and Amir Moradi. New first-order secure aes performance records. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 304–327, 2021.

[12] Sebastian Faust, Vincent Grosso, SMD Pozo, Clara Paglialonga, and F-X Standaert. Compos-

Table 1. Comparison of implementations

Design	Reg.	LUT	BRAM	Lat.	Rand. bits	Clk_{MHZ}	$\text{Thr.put}_{Gbit/s}$	$\text{time}_{\mu s}$	device
[15]	1566	2888	16	512	51.2	100	0.025	5.12	Virtex-II pro
[16]	5328	7783	0	70	0	131	1.676	0.534	Spartan-6
[11] E	527	2068	240	50	160	116	1.484	0.431	Spartan-6
[11] E/D	527	3778	240	50	160	102	1.305	0.490	Spartan-6
Our Method	526	3728	120	50	160	105	1.344	0.476	Spartan-6

able masking schemes in the presence of physical defaults & the robust probing model. 2018.

[13] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In *Annual Cryptology Conference*, pages 764–783. Springer, 2015.

[14] Felix Wegener and Amir Moradi. A first-order sca resistant aes without fresh randomness. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 245–262. Springer, 2018.

[15] Tim Güneysu and Amir Moradi. Generic side-channel countermeasures for reconfigurable devices. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 33–48. Springer, 2011.

[16] Aein Rezaei Shahmirzadi and Amir Moradi. Re-consolidating first-order masking schemes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 305–342, 2021.

Muhamadali Hajisoltani received his B.Sc. degree in Telecommunication Engineering from Tabriz University in Tabriz, Iran in 2020, and he worked as the chief editor of Tabriz University's telecommunication magazine from 2019 to 2022. He is now an M.Sc. student in Cryptography and Secure Telecommunication at Shahid Beheshti University in Tehran, Iran.

Raziyeh Salarifard received the B.Sc. degree from the Sharif University of Technology, Tehran, Iran, in 2012, the M.Sc. and Ph.D. degrees from the same university, in 2014 and 2019 respectively, all in computer engineering (hardware). She is now working as an Assistant Professor at Shahid Beheshti University. Her research interests include hardware security, cryptographic engineering, and secure, efficient computing and architectures.

Hadi Soleimany has been working as an Assistant Professor at Cyberspace Research Institute at Shahid Beheshti University, Iran since 2015. He received his Ph.D. in Theoretical Computer Science from Aalto University, Finland in 2015. He was also a postdoctoral researcher at the Technical University of Denmark (DTU), Denmark in the Summer of 2016 and 2017. His main research interests are practical aspects of cryptography.