Comparison of Advanced Encryption Standard Variants Targeted at FPGA Architectures

Nithin Shyam Soundararajan[1(✉)] and K. Paldurai[2]

[1] Global Security Lab, Schneider Electric, Bangalore, Karnataka, India
nithin.shyams@se.com

[2] Department of Electronics and Communication Engineering, PSG Institute of Technology and Applied Research, Coimbatore, Tamil Nadu, India
paldurai.k@psgitech.ac.in

Abstract. Digital communication of any form must provide data confidentiality as the threats are increasing in today's rapid world. Data privacy and security are crucial factors as data is considered gold in the modern era. The 128-bit Advanced Encryption Standard algorithm, commonly known as AES, has been implemented in several designs, focusing on specific purposes and is used widely. The 256-bit variant uses the same fundamental cipher blocks as the 128-bit version but differs in key size, the key expansion function and the number of cipher rounds. This paper investigates the 256-bit AES algorithm targeted at FPGA-Field Programmable Gate Arrays architectures and compares it with the 128-bit implementation, reporting performance and resource utilization. Also, the security offered is discussed. The security is determined by the complexity of recovering the key using cryptanalytic attacks. Both encryption and decryption processes are handled by this implementation and are tested in Verilog language using the Xilinx Vivado software on the Xilinx Zynq-7000 (xc7z020-clg484-1) FPGA.

Keywords: AES · Encryption · FPGA · Hardware Implementation

1 Introduction

With the development of technologies like 5G and IoT, huge amounts of data need to be exchanged through public communication networks which are considered insecure. Cryptography plays a major role in securing our data. It conceals information from unauthorized parties and can be used to digitally sign information. Encryption algorithms have been classified into two groups: secret key encryption or symmetric and public key encryption or asymmetric. Symmetric key encryption is the fastest and is used to conceal data, while asymmetric encryption is used primarily to share symmetric keys between the communicating parties.

In 2001, The National Institute of Standards and Technology (NIST) chose Joan Daemen and Vincent Rijmen's Rijndael algorithm as the new Advanced Encryption Standard (AES) and thus replaced the aging Data Encryption Standard (DES) as the

S. Nandan Mohanty et al. (Eds.): ICISML 2022, LNICST 470, pp. 346–355, 2023.
https://doi.org/10.1007/978-3-031-35078-8_29

Federal Information Processing Standard (FIPS). In the year 2002, due to proven insecurity, the DES was deprecated. The 56-bit small key size was the major cause of this issue. Deep Crack, developed by the Electronic Frontier Foundation (EFF) can be used for breaking the DES keys in 56 h. Despite the theoretical attacks, the practical security of the algorithm is considered to be higher in the form of Triple DES (3DES). However, AES has superseded over recent years.

Rijndael [4] is a family of symmetric key ciphers containing different block and key sizes. For AES, three members of the Rijndael family were selected by NIST. The block size of each member is 128 bits while the key lengths can be 128, 192 or 256 bits. It has resisted many cryptanalytic attacks and has proved its reliance and has been deemed secure at least until the dawn of high-power quantum computers.

2 Galois Field

In the finite field $gf(2^8)$, bytes are represented in terms of the polynomials. The following polynomial is used for interpreting bytes as finite field elements, where β_n represents a bit

$$\beta_7\chi^7 + \beta_6\chi^6 + \beta_5\chi^5 + \beta_4\chi^4 + \beta_3\chi^3 + \beta_2\chi^2 + \beta_1\chi + \beta_0$$

2.1 Addition in $gf(2^8)$

The corresponding bits of two bytes are subjected to modulo 2 addition. For example,

$$\{0 \times 57\} + \{0 \times 83\} = \{01010111\} + \{10000011\} = \{11010100\} \text{ or } \{0 \times D4\}$$

2.2 Multiplication in $gf(2^8)$

Normal polynomial multiplication is applied with a modulo of an irreducible polynomial of degree 8 (shown in Eq. 1).

$$m(\chi) = \chi^8 + \chi^4 + \chi^3 + \chi + 1 \tag{1}$$

Example,

$$\begin{aligned}\{0 \times 57\} \bullet \{0 \times 83\} &= (\chi^6 + \chi^4 + \chi^2 + \chi + 1)\ (\chi^7 + \chi + 1)\%(\chi^8 + \chi^4 + \chi^3 + \chi + 1)\\ &= \chi^7 + \chi^6 + 1\\ &= \{0 \times C1\}\end{aligned}$$

3 Cipher

3.1 The State

Data is broken into a matrix of size 4×4 where each entry is one byte or eight bits, for a total of 128 bits (Fig. 1).

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Fig. 1. State Matrix [1]

3.2 Algorithm

The input is copied to the State array during the initialization of the Cipher. Based on the key length, a round function is executed 14, 12 or 10 times for transforming the State array after the initial Round Key addition. Each encryption round consists of SubBytes, ShiftRows, MixColumns (not performed for the last round) and AddRoundKey. For decryption, the operations are reversed (Fig. 2).

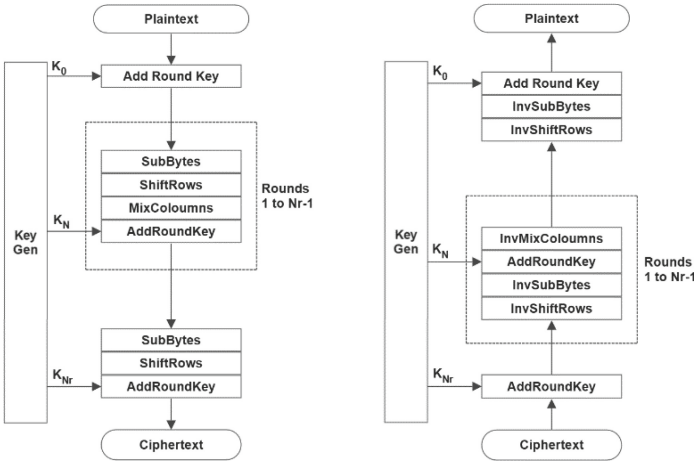

Fig. 2. Encryption & Decryption Block Diagram

3.3 SubBytes and InvSubBytes

Each byte is substituted with the corresponding value in the Substitution Box (S-Box). This non-linear transformation is composed of two processes: first, multiplicative inverse in Rijndael's finite field and then affine transformation. For inverting this operation, the process is reversed: first, affine transformation and then multiplicative inverse in Rijndael's finite field (Fig. 3).

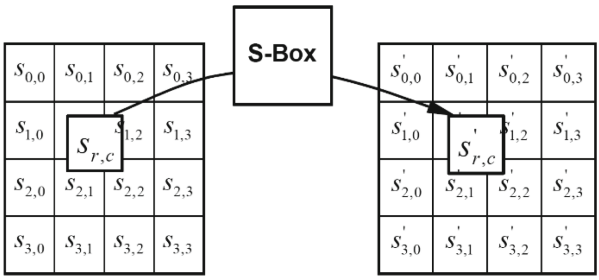

Fig. 3. SBox Transformation [1]

3.4 ShiftRows and InvShiftRows

Each row is shifted to the left (n − 1) times during encryption and shifted right (n − 1) times during decryption i.e. the first row is unchanged while the fourth (last) row is shifted thrice (Fig. 4).

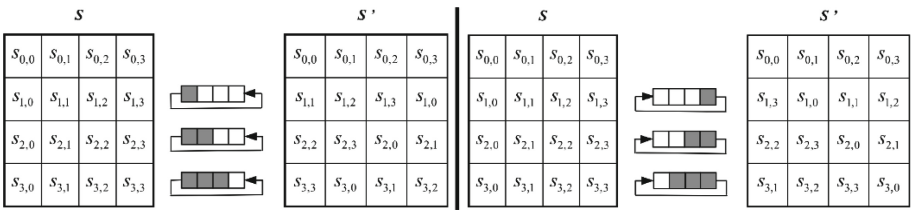

Fig. 4. (left) Shift Rows and (right) Inverse Shift Rows Transformations [1]

3.5 MixColumns and InvMixColumns

An invertible linear transformation, each column of the state matrix is multiplied with a fixed 4 × 4 matrix. The matrix used in InvMixColumns (M′) is the inverse of the matrix used in MixColumns (M).

$$M = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \qquad \mathrm{M}' = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

3.6 AddRoundKey

The state matrix and round key matrix are subjected to the XOR operation in this step. A subkey is derived from the initial key for every round (Fig. 5).

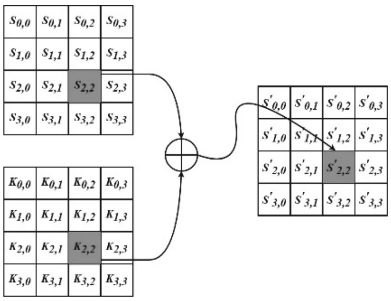

Fig. 5. AddRoundKey Transformation

3.7 Key Generation

The user-given key is used for generating the required number of round keys. Figures 6 and 7 show the Key Expansion for AES-128 and AES-256 respectively. RC or round constant is calculation is shown in the following equation. i starts from 1 and incremented by 1 every time RC is calculated (shown in Eq. 2). Initial RC value, RC_1, is 1.

$$RC_i = (RC_{i-1} << 1) \wedge (0 \times 11b \,\&\, -(RC_{i-1} >> 7)) \tag{2}$$

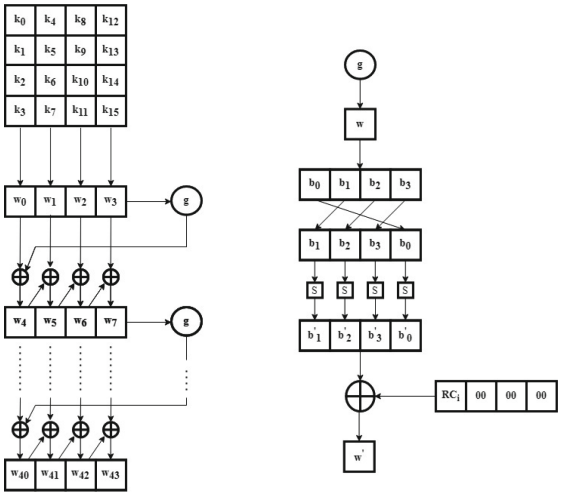

Fig. 6. Key Expansion for AES-128

4 Implementation and Results

Both encryption and decryption activities are handled by this implementation and are tested in Verilog language using the Xilinx Vivado software on the Xilinx xc7z020-clg484-1 FPGA.

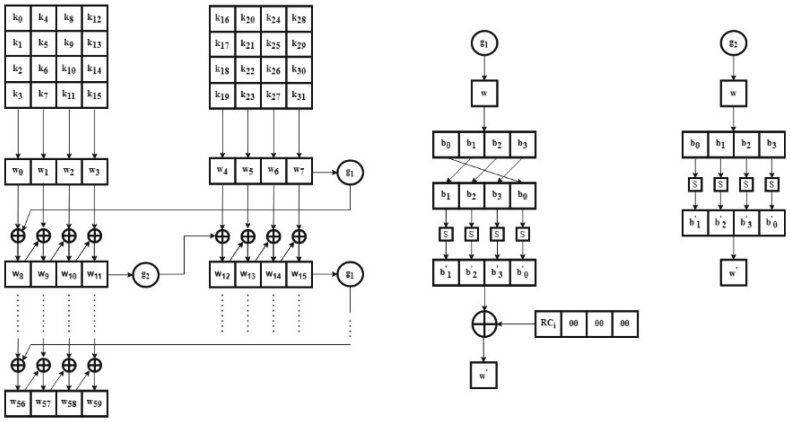

Fig. 7. Key Expansion for AES-256

The design utilizes the basic AES structure and balances resource utilization and performance. In order to improve the performance, SubBytes and ShiftRows transformations were combined together to form a single transformation. Similarly, InvSubBytes and InvShiftRows were also combined. Loop unrolling optimization was utilized. All round keys were generated on the fly. AES Known Answer Test (KAT) Vectors were used for testing.

4.1 AES-128 Results

Table 1 along with Fig. 8 show the performance of the AES-128 implementation.

Plaintext = 6BC1BEE22E409F96E93D7E117393172A
Key = 2B7E151628AED2A6ABF7158809CF4F3C
Ciphertext = 3AD77BB40D7A3660A89ECAF32466EF97

10 keys need to be generated, each of which takes 1 clock cycle or CC for a total of 10 CCs. 1 CC is needed for SubBytes, ShiftRows and AddRoundKey and another CC for MixColumns. So, rounds 1 to Nr-1 will consume 2 CCs and the final round will take 1 CC.

When compared to encryption, the number of clock cycles taken by decryption is higher. This is caused due to the need of the last round key for the process to begin.

The 128-bit AES algorithm implementation's resource utilization, can be seen in Table 2. LUT stands for Look Up Table, FF stands for Flip Flop, and BRAM stands for Block RAM.

4.2 AES-256 Results

Table 3 along with Fig. 9 show the performance of the AES-256 implementation.

Plaintext = 6BC1BEE22E409F96E93D7E117393172A

Table 1. AES-128 Design Performance.

Process Name	Clock Cycles
Key Expansion	10
One Cipher Round (1 to 9)	2
Final Cipher Round	1
Full Encryption Process	19
Full Decryption Process	29

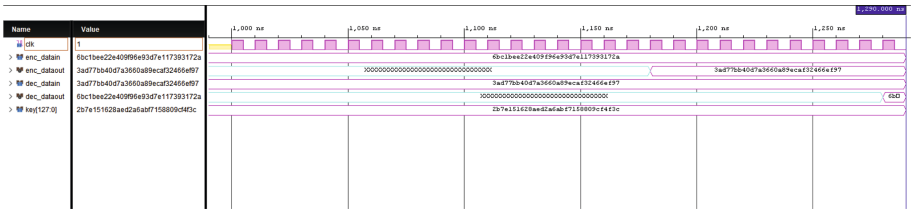

Fig. 8. AES-128 Output Waveform.

Table 2. Resource Utilization of AES-128 design.

Resource	Utilization
LUT	3854
FF	1691
BRAM	70.50

Key = 603DEB1015CA71BE2B73AEF0857D77811F352C073B6108D72D9 810A30914DFF4
Ciphertext = F3EED1BDB5D2A03C064B5A7E3DB181F8

A total of 15 keys are required out of which the first two are taken care by the user given key. So, only 13 keys need to be generated.

The resource utilization of the 256-bit AES algorithm design, can be seen in Table 4.

5 Cryptanalysis and Conclusion

5.1 Cryptanalysis

The Rijndael AES ciphers have been subject to a wide range of cryptanalytic attacks since their introduction. Though the cipher can be vulnerable to side channel attacks, no practical mathematical attack has been invented yet.

Table 3. AES-256 Design Performance.

Process Name	Clock Cycles
Key Expansion	13
One Cipher Round (1 to 9)	2
Final Round	1
Full Encryption Process	27
Full Decryption Process	40

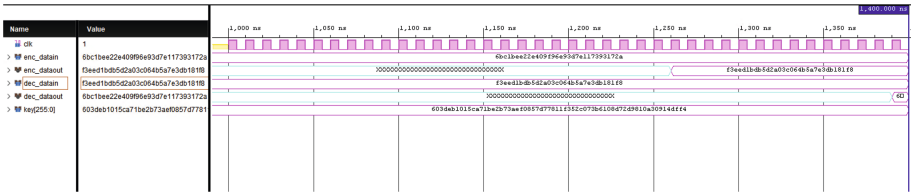

Fig. 9. AES-256 Output Waveform.

Table 4. Resource Utilization of AES-256 design.

Resource	Utilization
LUT	4544
FF	1595
BRAM	70.50

Using Biclique Cryptanalysis, the time complexities of cracking an AES-128 and AES-256 key [7] are close to $2^{126.0}$ and $2^{254.3}$ respectively. The space complexities are in the order of petabytes and terabytes.

Related key attacks, that are practical, have been demonstrated on AES with reduced rounds [8]. But those attacks do not pose a threat to full-round implementations.

5.2 Conclusion and Future Scope

AES-128 and AES-256 have been successfully implemented in hardware and their metrics have been presented. AES-256 requires 4 extra rounds than AES-128. Considering the 2 clock cycles required per round, the total clock cycles needed for the complete encryption/decryption process has scaled in a linear manner. As far as resource utilization is concerned, for both variants, the BRAM usage stayed the same. When compared to AES-128, implementation of AES-256 required nearly 5.6% fewer Flip Flops and 18% more LUTs (Fig. 10). According to the cryptanalytic attack results, an exponentially higher encryption security is offered by AES-256. Considering the advantage, the increase in the necessary resources appears insignificant.

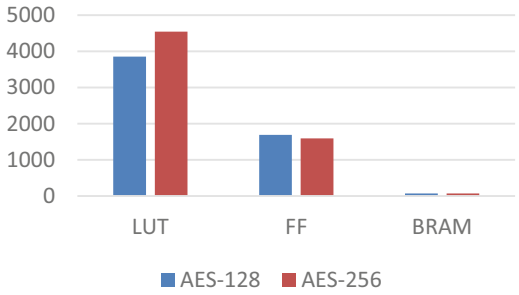

Fig. 10. AES Resource Utilization

In this paper, AES variants designed to run at 2 clock cycles per round were compared. This can be extended to other designs and hardware focusing performance or compaction to verify the scaling of resources.

Acknowledgement. We are thankful to the staff members of the department of Electronics and Communication Engineering at PSG Institute of Technology and Applied Research for their support.

References

1. FIPS 197: Advanced Encryption Standard (AES), 26 November (2001)
2. Chodowiec, P., Gaj, K.: Very compact FPGA implementation of the AES algorithm. In: Walter, C.D., Koc, C.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 319–333. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45238-6_26
3. Chen, S., Hu, W., Li, Z.: High performance data encryption with AES implementation on FPGA. In: 2019 IEEE 5th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS) (2019)
4. Daemen, J., Rijmen, V.: AES Proposal: Rijndael. AES Algorithm Submission, 3 September (1999)
5. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES. https://eprint.iacr.org/2011/449.pdf
6. Biryukov, A., Khovratovich, D.: Related-key Cryptanalysis of the Full AES-192 and AES-256. https://eprint.iacr.org/2009/317.pdf
7. Tao, B., Wu, H.: Improving the Biclique cryptanalysis of AES. In: Foo, E., Stebila, D. (eds.) ACISP 2015. LNCS, vol. 9144, pp. 39–56. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19962-7_3
8. Biryukov, A., Dunkelman, O., Keller, N., Khovratovich, D., Shamir, A.: Key Recovery Attacks of Practical Complexity on AES Variants With Up To 10 Rounds. Cryptology ePrint Archive, Paper 2009/374 (2009). https://eprint.iacr.org/2009/374
9. Borkar, A.M., Kshirsagar, R.V., Vyawahare, M.V.: FPGA implementation of AES algorithm. In: 2011 3rd International Conference on Electronics Computer Technology (2011)
10. Kaur, S., Vig, R.: Efficient implementation of AES algorithm in FPGA device. In: International Conference on Computational Intelligence and Multimedia Applications, December 2007 (2007)

11. Shyamala, C.K., Padmanabhan, T.R., Harini, N.: Cryptography and Security. Wiley India
12. Stallings, W.: Cryptography and Network Security, 6th edn. Pearson
13. Benvenuto, C.J.: Galois Field in Cryptography, 31 May 2012. https://sites.math.washington.edu/~morrow/336_12/papers/juan.pdf