ISSN: 2182-2069 / E-ISSN: 2182-2077

AES Cipher's Candidates: Design and FPGA Implementation

M. F. Al-Gailani[1*]

[1*]Cybersecurity Engineering Department, College of Information Engineering, Al-Nahrain University, Baghdad, Iraq, m.falih@nahrainuniv.edu.iq, https://orcid.org/0000-0003-4307-941X

Received: November 08, 2024; Revised: December 10, 2024; Accepted: January 06, 2025; Published: February 28, 2025

Abstract

In this paper, efforts are devoted to designing hardware for important security algorithms that have a high level of security due to their internal architecture and are designed to be secure against current and future attacks. These algorithms, called MARS, Twofish, and RC6, are among the algorithms competing with the current standard AES and have been shortlisted among the top five. These algorithms are block ciphers with a block size of 128 bits, and their key length is variable. MARS and Twofish are both based on the generalized Feistel structure, while RC6 is an improvement of RC5 that essentially uses data-dependent rotations. These algorithms are used in many applications due to their high security margin. The proposed hardware of the algorithms is designed using System Generator and Xilinx ISE 14.7, and implemented on a Xilinx Virtex-6 xc6vlx195t-3 FPGA board. The architecture of the proposed designs emphasizes area utilization, thus following an iterative looping architecture that seeks to minimize resource and energy usage, making it suitable for applications that have limited resources. The implementation result was promising, as the maximum frequency, throughput, number of occupied slices, and power consumption were calculated and compared to the current standard, AES, and its candidate Serpent algorithm.

Keywords: AES, FPGA, MARS, RC6, System Generator, Throughput, Twofish.

1 Introduction

The previous standard, the Data Encryption Standard (DES) (National Bureau of Standards, 1977) despite its popularity and widespread use in applications, has been replaced by a new standard, the Advanced Encryption Standard (AES) (Daemen & Rijmen, 2020; NIST, 2001) in 2000 due to its smaller cipher key length and block size, which is no longer secure with the current technologies. Although triple DES (Barker & Mouha, 2017) was emerged and used after the replacement decision taken by the National Institute of Standards and Technology in 1997 (NIST, 1997), it is too slow to be used in real-time applications, as it requires repeating the encryption or decryption process three times with two or three different keys to achieve the necessary security. Although it achieves security, it does so at the expense of speed of completion.

MARS (Burwick et al., 1998, 1999), Towfish (Schneier et al., 1998; Schneier, 1999), and RC6 (Rivest et al., 1998) algorithms are strong competitors vying to win the world-standard cryptographic algorithm. They meet all NIST criteria required for the Advanced Encryption Standard (NIST, 1997b) regarding key lengths, block size, efficiency, simplicity, speed, portability, and flexibility(Kramer, 1996).

Journal of Internet Services and Information Security (JISIS), volume: 15, number: 1 (February), pp. 51-66.
DOI: *10.58346/JISIS.2025.I1.004*

*Corresponding author: Cybersecurity Engineering Department, College of Information Engineering, Al-Nahrain University, Baghdad, Iraq.

The process of selecting the current standard took a long time, and eventually, Rijndael was chosen as AES. Their developer believes it probably has won due to its compact and efficient implementation, as it can be processed with just four table lookups and four XOR operations per column per round (William Stallings, 2023).

This research aims to design efficient hardware for the three algorithms, MARS, Twofish, and RC6, due to their importance in security and to provide a secure alternative in case of violation of the current standard.

The structure of the paper is organized as follows: The section following the introduction discusses the work related to the hardware design of the algorithms. The following section describes the MARS, Twofish, and RC6 algorithms in detail, including their structure, layers, and key expansion. Next, the techniques used in applying the hardware design are described. Then, the results are presented and discussed. Finally, the conclusions are outlined.

2 Related Work

In this section, other research papers covering the hardware implementation of the practical part are listed below.

A VHDL code for the Twofish algorithm was created by the authors of (Chodowiec & Gaj, 1999) using Xilinx Foundation Series v. 1.5 and mapped into the Xilinx FPGA XC4028 device. The design of the circuit was focused on achieving high speeds. Accordingly, a throughput of 85.6 Mbit/sec is achieved with a maximum clock frequency of 10.7 MHz as determined by the timing simulation for combinational implementation. In addition, 911 CLBs were required to implement the entire circuit. However, all subkeys are pre-computed and stored in memory, which speeds up throughput at the expense of resources.

The authors in (Lai et al., 2002) presented a VLSI architecture chip for the Twofish algorithm based on the loop-folding technique combined with hardware mapping. It has been implemented using 0.35 µm CMOS technology (Yaremko et al., 2024). The chip operates at 66 MHz, can achieve a throughput of 200 Mbit/sec, and consumes 44 mW power dissipation (Leema et al., 2024). However, the algorithm was designed and implemented on an ASIC, not an FPGA, which may give different results when implemented on the required technology.

In (Gehlot et al., 2013)The authors explained the Twofish algorithm in all its modules and aimed to hardware implement the algorithm efficiently with minimal delay. The algorithm was implemented on Xilinx software - 6.1 xst using VHDL language. The delay was 105.794 ns, and the maximum frequency was 9.40 MHz. The authors added an additional layer to further increase security by transforming the input data stream using an Endian function that acts as an interface between the input data and the cipher. However, it also increases complexity (Sujatha, 2024).

An FPGA implementation of RC6 and CAST-256 encryption algorithms is proposed in (Riaz & Heys, 1999). The proposed hardware design of the RC6 was implemented on a Xilinx XC40200XV-9-BG560 device (Salman & Alomari, 2023). The encryption rate was about 37 Mbit/Sec and the total FPGA resources were about 6450 CLBs. The authors noted that the hardware is characterized by high complexity and low speed, requiring a higher-spec FPGA device. They also pointed out that multiplication and addition operations are significant obstacles to speed. Their suggestion was to design the cipher in a simpler way, using simpler operations, which would give the system good cryptographic properties and be efficient in hardware implementation.

All the above limitations were overcome by using different devices, removing additional suggested layers, computing round keys on the fly to save resource usage, and designing the operations in various ways using the Xilinx system generator. The efficiency of the design was verified by simulation and implementation results.

3 Methodology

This section begins with a detailed explanation of the three algorithms that the hardware is designed for. It starts with the MARS algorithm, then the Twofish algorithm, and finally the RC6 algorithm. This is followed by the hardware design of the algorithmic operations. These operations are listed first by the operations common to the three algorithms, then by the operations common to two algorithms, and finally by the individual operations in each algorithm. The hardware architecture of the algorithms is designed and implemented on the target device FPGA Virtex-6 (xc6vlx195t-3ff1156) using Xilinx System Generator.

1) MARS Encryption Algorithm

MARS is a 128-bit block cipher with a variable key length from 128 to 448 bits. It is word-oriented and has a word length of 32 bits. The structure of MARS is based on a Type-3 Feistel network, in which a single word is used to modify the other three words, thus increasing diffusion properties compared to Type-1 in which half of the data is used to modify the other half in each block. The cipher is designed using a mixed structure as shown in Figure 1, which increases its resilience against current and future attacks. The outer rounds (wrapper layers) are characterized by rapid mixing and key avalanche and have a structure different from the middle rounds (cryptographic core) of keyed transformation.

The wrapper layer involves forward and backward mixing. Forward mixing (top rounds) begins with the key addition followed by eight unkeyed S-box based forward mixing rounds. It consists of a number of operations, as shown in Figure 2, including key addition, exclusive-or, right rotation with a fixed offset, and word substitution using a single (512 × 32) S-box. The backward mixing (bottom rounds) is designed as a mirror image of the top rounds. In addition, the subtraction operation is used instead of the addition operation.

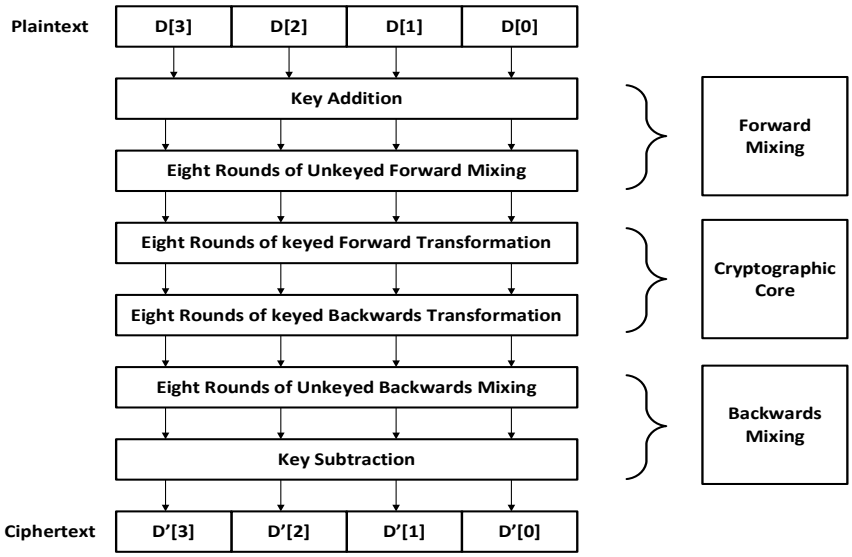


Figure 1: High-level Structure of the Cipher (Adapted from (Burwick et al., 1998))

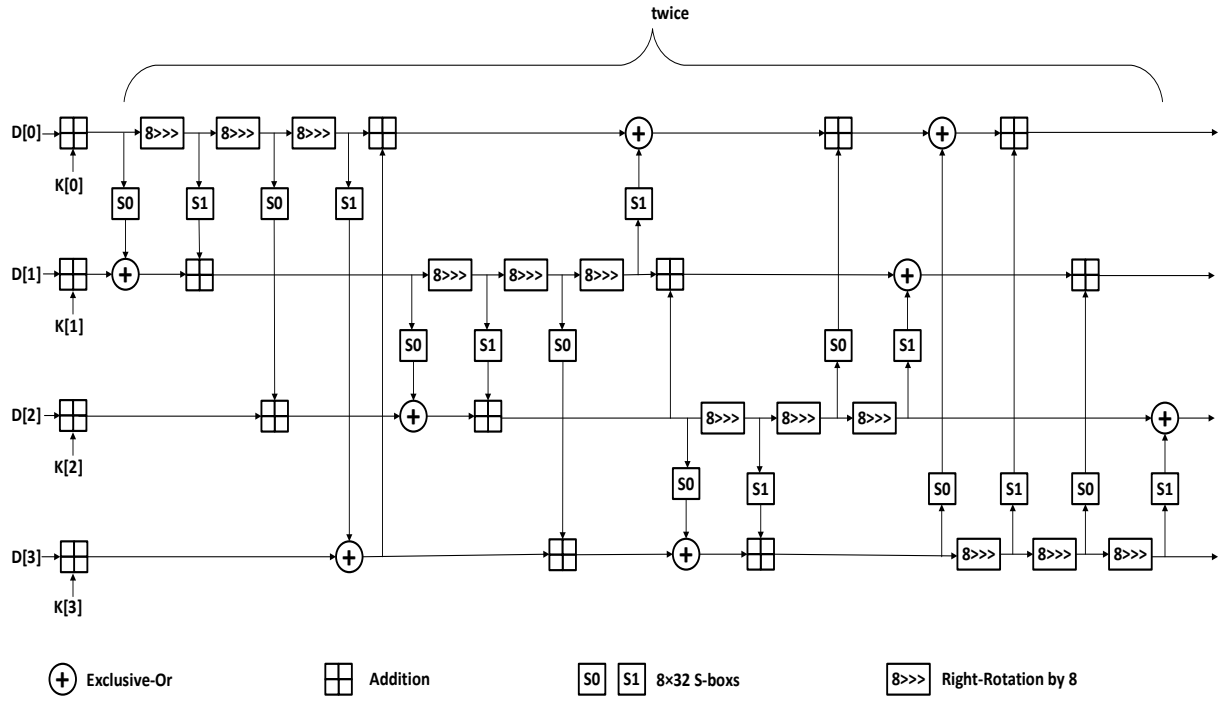


Figure 2: Structure of the Forward Mixing Phase (Adapted from (Burwick et al., 1998))

The cryptographic core, as shown in Figure 3, consists of 16 rounds, half of which are keyed forward transformations that involve a combination of addition, exclusive-or, left rotation with fixed offset, and E-function.

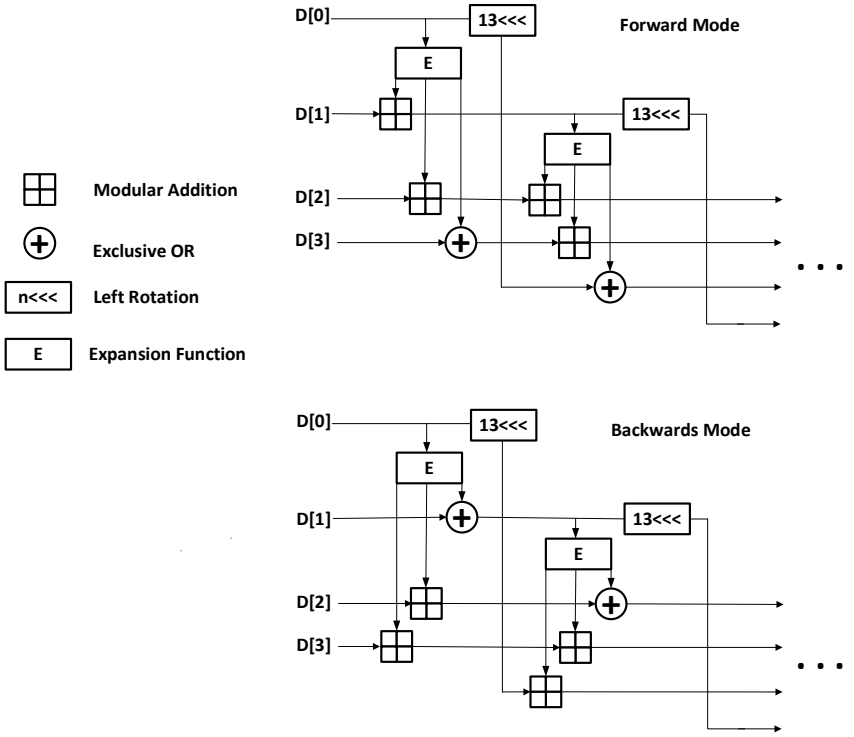


Figure 3: The Type-3 Feistel Network of the Main Keyed Transformation (Adapted from (Burwick et al., 1999))

The E-function, as shown in Figure 4, accepts one data word and outputs three words that XORed with the other words in a different order to further increase the cipher resistance to attacks. In addition to the mentioned, it includes S-box lookups, multiplications, and data-dependent rotations. The cipher effectively protects against linear and differential cryptanalysis by combining data-dependent rotations layer and multiplications operations. The rotation amounts are specified by the highest order bits of the product. The other half are keyed backward transformations, thus ensuring cipher strength in both directions.

The key expansion procedure expands the user-provided key into forty 32-bit words. The procedure also ensures that the key words generated for multiplication follow a specific structure, such that there are no ten consecutive 0's or 1's, and the lowest two bits are set to 1. The operations used in key expansion procedures consist of XOR, fixed left rotation, addition, and word substitution, in addition to modifying the sixteen words used in multiplication to adapt them to the required structure.

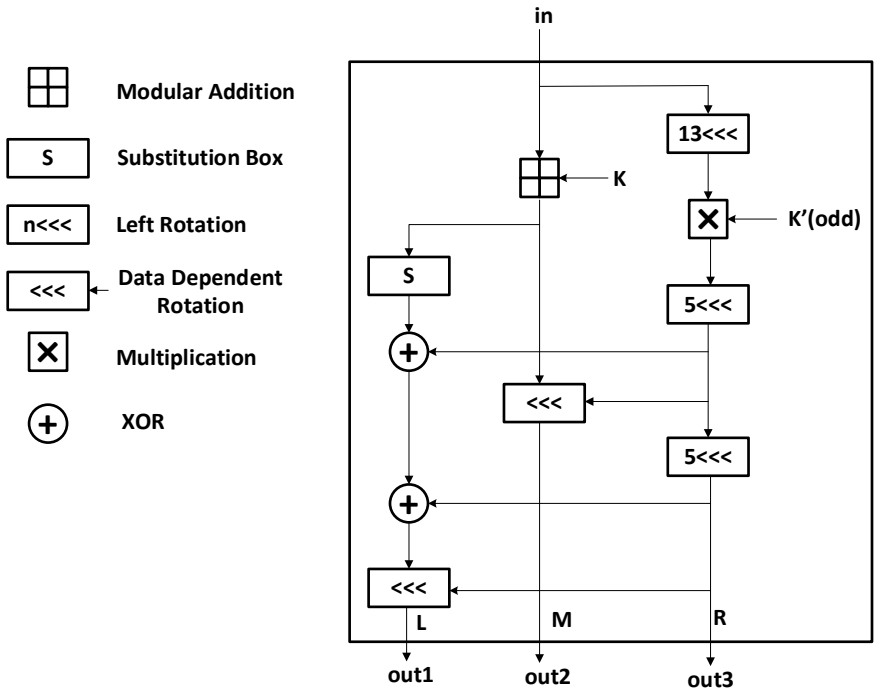


Figure 4: The E-function of the Main Keyed Transformation (Adapted from (Burwick et al., 1999))

2) Twofish Encryption Algorithm

Twofish is a 128-bit block cipher that can handle keys up to 256 bits in length. The cipher is based on a Feistel structure, only by adding 1-bit rotates, with additional input and output whitening.

As shown in Figure 5, the input is first subjected to input whitening by splitting it into four 32-bit words and XORed with the input key. The output is then processed by the round function F, followed by a 1-bit fixed rotation and interchange of the two halves, which is iterated 16 times. The final step is achieved by processing the output whitening.

The round function F includes the following layers:

- **Fixed Rotations:** Rotating the 32-bit input signal 8-bit to the left.
- **Additions Modulo 2^{32}:** Modular addition operation between the 32-bit input signal and the round key.

- **Four Key-dependent S-boxes**: As illustrated in Figure 6, the first step is to convert the 32-bit input signal into four signals with byte word, and each word is handled separately. The bytes are passed through the q-permutations layer three times and XORed twice, alternating with the subkeys S0 and S1. These two parameters are precomputed from the global key, and their values remain unchanged throughout the entire process.

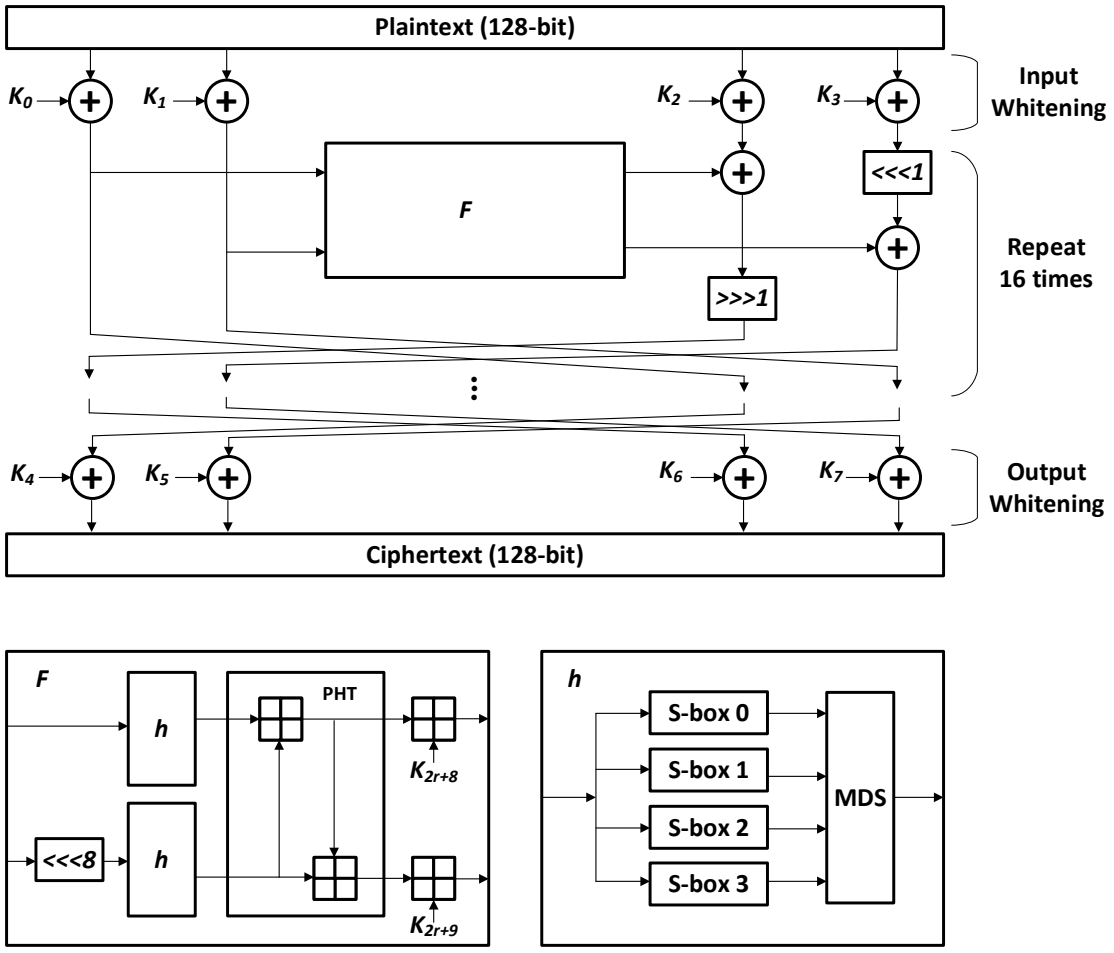


Figure 5: Twofish Overview (Encryption) (Adapted from (Schneier et al., 1998))

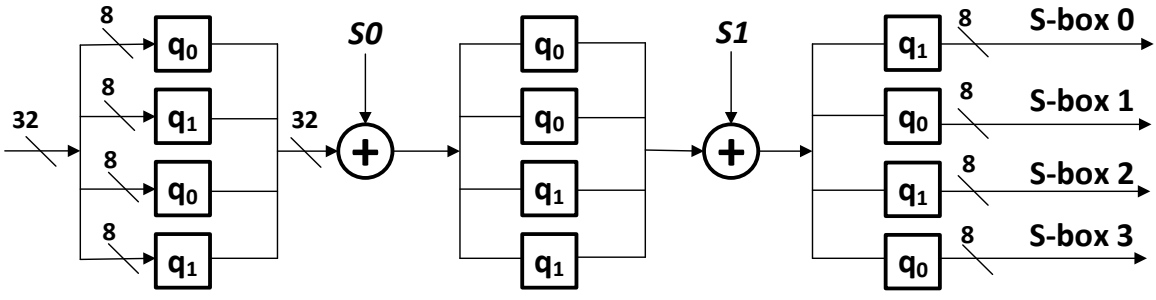
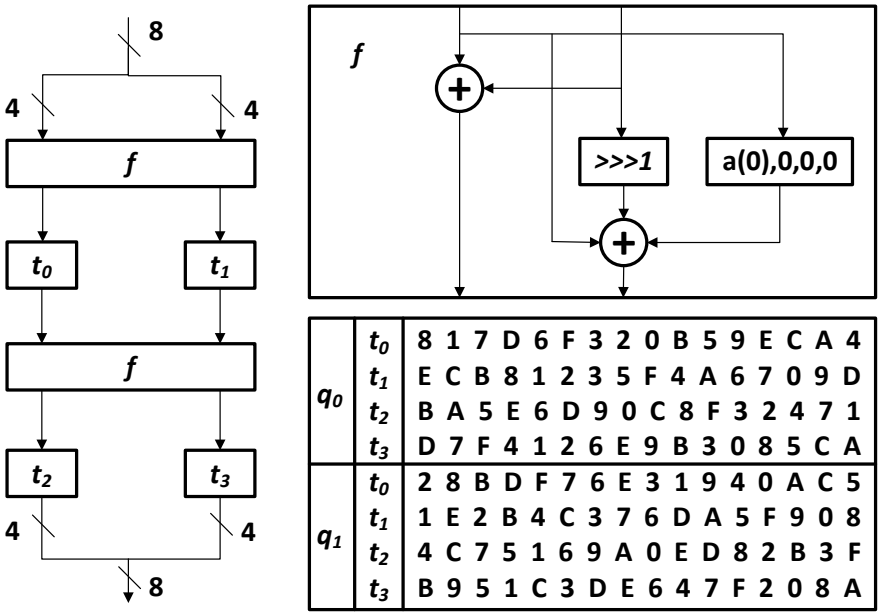


Figure 6: S-boxes (Schneier, 1999)

The structure of the q-permutation is illustrated in Figure 7. It mainly consists of an XOR operation, a fixed rotation of one bit to the right, and finally, most importantly, the use of two sets of q, each made up of four different 4-bit lookup tables.



q_0	t_0	8	1	7	D	6	F	3	2	0	B	5	9	E	C	A	4
	t_1	E	C	B	8	1	2	3	5	F	4	A	6	7	0	9	D
	t_2	B	A	5	E	6	D	9	0	C	8	F	3	2	4	7	1
	t_3	D	7	F	4	1	2	6	E	9	B	3	0	8	5	C	A
q_1	t_0	2	8	B	D	F	7	6	E	3	1	9	4	0	A	C	5
	t_1	1	E	2	B	4	C	3	7	6	D	A	5	F	9	0	8
	t_2	4	C	7	5	1	6	9	A	0	E	D	8	2	B	3	F
	t_3	B	9	5	1	C	3	D	E	6	4	7	F	2	0	8	A

Figure 7: Permutation q (adapted from (Schneier, 1999))

- **4-by-4 Maximum Distance Separable (MDS) Matrices over GF (2^8):** In this layer, the 32-bit input (4 bytes) is multiplied according to Equation 1 by the constant matrix in the GF (2^8) with the primitive polynomial $v(x) = x^8 + x^6 + x^5 + x^3 + 1$ of degree 8 over GF (2).

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \quad (1)$$

- **Pseudo-Hadamard Transform (PHT):** Performs additions modulo 2^{32}.

The round keys are generated by processing the key schedule algorithm. First, two 32-bit keys, S0 and S1, are generated according to Equation 2. Where the output is the results of the polynomial multiplication in Galois Field GF (2^8) between the RS matrix and the cipher key, with the irreducible polynomial $w(x) = x^8 + x^6 + x^3 + x^2 + 1$ of degree 8 over GF (2), their values remain constant during the entire process. They are used to initialize the key-dependent s-boxes. The m key is the cipher key.

A total of forty 32-bit subkeys (K0 – K39) are then generated, two for each round, in addition to 4 keys for each input and output whitening. The round keys are generated as shown in Figure 8; in each iteration, only two 32-bit keys are generated. It is generated in a structure analogous to that used in encryption to facilitate hardware implementation except that the round keys layer is not added and a fixed rotation is added. The global keys M3 to M0 in the figure represent the user-provided key (128-bit cipher key), and their values remain unchanged, where M3 is 32-bit MSB, and M0 is 32-bit LSB.

$$\begin{pmatrix} S_{i,0} \\ S_{i,1} \\ S_{i,2} \\ S_{i,3} \end{pmatrix} = \overbrace{\begin{pmatrix} 01 & A4 & 55 & 87 & 5A & 58 & DB & 9E \\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5 \\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19 \\ A4 & 55 & 87 & 5A & 58 & DB & 9E & 03 \end{pmatrix}}^{RS} \cdot \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix} \quad (2)$$

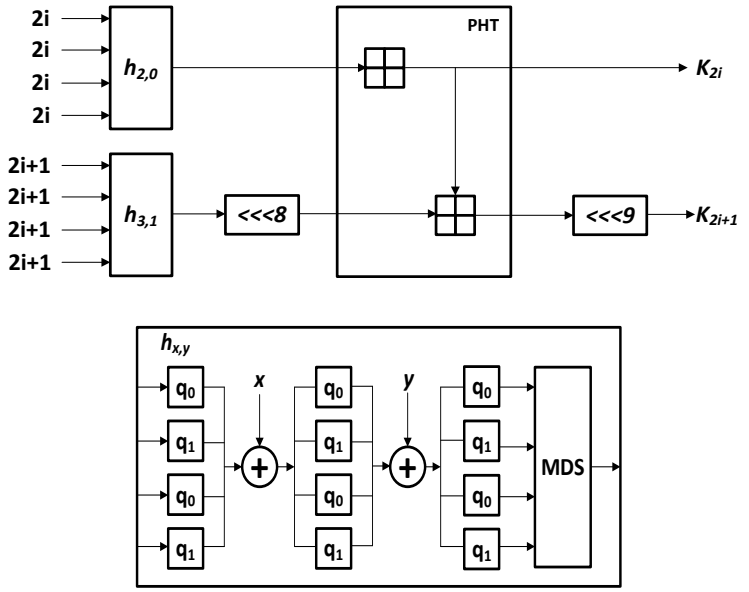


Figure 8: Generation of Subkeys K (Adapted from (Schneier, 1999))

3) RC6 Encryption Algorithm

The RC6 is an improvement over the RC5 (Rivest, 1994) in terms of security and performance. It is a fully parameterized symmetric block cipher designed by Rivest, Robshaw, Sidney, and Yin in 1998 (Rivest et al., 1998). However, to meet the requirements of AES, the parameters are chosen as follows: block size is 128 bits, word size is 32 bits, key length supports 128, 192, and 256 bits, and number of rounds is 20. The operations involved in the rounds are modular addition and subtraction, bitwise exclusive-or, modular multiplication, and essential use of data-dependent rotations in different directions. The cipher encryption overview is shown in Figure 9, and the key schedule pseudocode is shown in below algorithm.

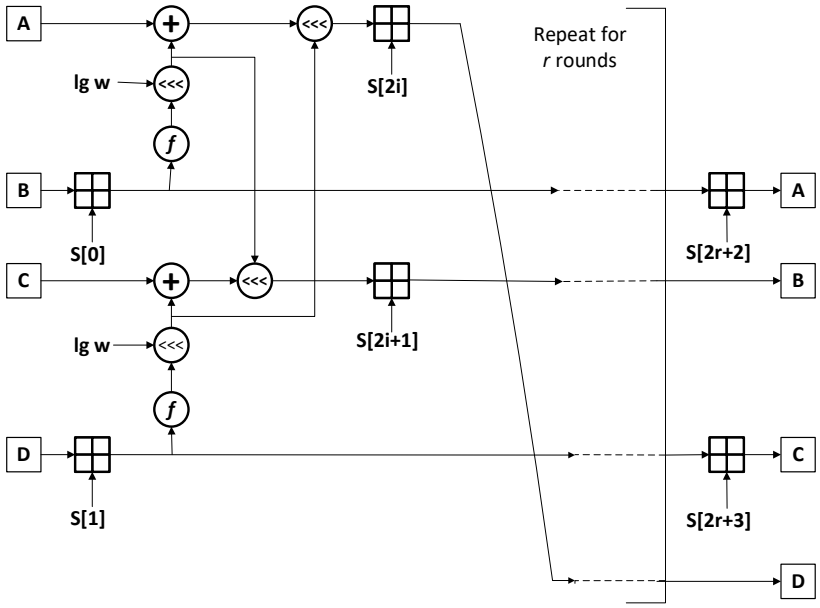


Figure 9: RC6 Encryption Overview ($f(x) = x \times (2x + 1)$) (Adapted from (Rivest et al., 1998))

Algorithm: RC6 key schedule algorithm (Rivest et al., 1998)

$\boldsymbol{Input}$: $User - supplied\ b - byte\ key\ preloaded\ into\ the\ c - word\ array\ L[0, \dots, c\ -\ 1]$
$Number\ r\ of\ rounds$
$\boldsymbol{Output}$: $w - bit\ round\ keys\ S[0, \dots, 2r\ +\ 3]$
$\boldsymbol{Procedure}$: $S[0]\ =\ Pw$
$For\ i\ =\ 1\ to\ 2r\ +\ 3\ do$
$S[i]\ =\ S[i\ -\ 1]\ +\ Qw$
$End\ for$
$A\ =\ B\ =\ i\ =\ j\ =\ 0$
$V\ =\ 3\ \times\ max\{c, 2r\ +\ 4\}$
$For\ s\ =\ 1\ to\ v\ do$
$A\ =\ S[i]\ =\ (S[i]\ +\ A\ +\ B)\ <<<\ 3$
$B\ =\ L[j]\ =\ (L[j]\ +\ A\ +\ B)\ <<<\ (A\ +\ B)$
$i\ =\ (i\ +\ 1)\ mod\ (2r\ +\ 4)$
$j\ =\ (j\ +\ 1)\ mod\ c$
End for

4) FPGA and System Generator

FPGA, an abbreviation for Field Programmable Gate Array, is a general-purpose integrated circuit that allows a designer to program it to perform a specific design without returning it to the device manufacturer. An FPGA can be reprogrammed even after its implementation in the system, unlike an application-specific integrated circuit (ASIC) (Taraate, 2021), which can carry out a comparable function in an electronic system. An FPGA enables the designer to access a two-dimensional array of programmable resources that can be used to carry out a variety of logic and arithmetic operations (Kilts, 2007; Wilson, 2015).

The device is programmed by downloading a bitstream file (configuration program) into the on-chip static random-access memory (SRAM) (Hori et al., 2012). This bitstream results from compilation tools that convert the high-level abstractions created by a designer into a low-level, executable form.

Xilinx System Generator first introduced the concept of building an FPGA program using a high-level Simulink model. System Generator is a system-level modeling tool that facilitates FPGA hardware design. The tool offers high-level abstractions that are automatically compiled into the FPGA. In addition, the tool offers low-level abstractions that enable access to underlying FPGA resources, facilitating the creation of highly efficient FPGA designs (Xilinx, 2012).

5) Hardware Design and Implementation

The hardware architecture of the MARS, Twofish, and RC6 algorithms is designed and implemented on an FPGA using the Xilinx System Generator (XSG). After the XSG generates the required netlist file, Xilinx ISE Foundation 14.7 and ModelSim synthesize and simulate it.

The target machine is xc6vlx195t-3ff1156, which is chosen for two reasons. First, it meets the design requirements, and second, it has been used with AES and Serpent (Biham et al., 1998) before, so a comparison can be made.

The MARS algorithm has a mixed structure that starts with eight rounds of the forward mixing layer, followed by 16 rounds of the cryptographic core layer, and ends with eight rounds of the backward

mixing layer, in addition to the key expansion algorithm. Each round of the forward and backward mixing layer is processed in 1 clock cycle, and each round of the core layer is processed in 4 clock cycles. In total, 80 cycles are needed to process one block of data.

The Twofish algorithm consists of 16 identical rounds, each processed in 1 clock cycle; thus, each data block is processed with 16 clock cycles in total.

The RC6 algorithm consists of 20 identical rounds, each processed in 4 clock cycles; thus, each data block is processed with 80 clock cycles in total.

The hardware design of the operations using Xilinx System Generator is explained as follows:

a. Common Operations between MARS, Twofish, and RC6

- **Additions Modulo 2 (bitwise exclusive-or)**: An XOR operation between two 32-bit input signals. The result of this operation is a 32-bit output signal. It is implemented using the XOR block. However, changing the output bit number to 32 bits is essential when configuring this block.
- **Additions Modulo 2^{32}**: It is a modular addition mod 2^{32}. This layer can be designed using different techniques. For example in (Chodowiec & Gaj, 1999) suggested computing the sum position by position, and propagating intermediate results using the carry chain from the position with the least significance to the position with the most significance. The other technique we follow is to add the two signals using the AddSub block and then subtract the modulus value from the result if its value exceeds or equals the value of the modulus.

b. Common Operations between MARS and Twofish

- **Fixed Rotation:** The hardware design of this layer is achieved by reordering the output signals. It can be realized using a BitBasher block. For left rotation, it is configured as a={b[31-n:0], b[31:31-n+1]}. For right rotation, it is configured as a={b[n-1:0], b[31:n]}. Where n represents the number of rotations.

c. Common Operations between MARS and RC6

- **Subtraction Modulo 2^{32}:** It is a modular subtraction mod 2^{32}. The AddSub block is used, and the result is compared to the value of zero after subtraction. The mode value will be added to the result if it is less than zero.
- **Multiplication:** It is implemented directly using a Multiplication block (Mult). However, processing the data takes three latency periods, which reduces the system's overall performance. All other operations cost 0 latency.
- **Data Dependent Rotation:** A word's rotation is determined from another word. It uses a multiplexer and (n-1) Bitbasher blocks, where n is the word length, 32 in this case, to cover all possible rotations. The input to the selector is the 5 LSB bits deducted from another word using the Slice block to determine the right offset amount.

d. Operation Specific to the MARS Algorithm

- **Substitution:** Two S_boxes are used, each containing 256 32-bit words. Their values are stored in ROM.

- **Forward mixing:** In this layer the four words D[0], D[1], D[2], and D[3] are updated according to the following equations, which are repeated eight times (Burwick et al., 1999).

$$D[1] = (D[1] \oplus S0[D[0]]) + S1[D[0] >>> 8] \quad (3)$$
$$D[2] = D[2] + S0[D[0] >>> 16] \quad (4)$$
$$D[3] = D[3] \oplus S1[D[0] >>> 24] \quad (5)$$
$$D[0] = (D[0] >>> 24) + D[1] \ (if\ i = 1,5) + D[3] \ (if\ i = 0,4) \quad (6)$$
$$(D[0], D[1], D[2], D[3]) = (D[1], D[2], D[3], D[0]) \quad (7)$$

  - The Word D [1] is updated, as shown in Equation 3, by first XORing its value to the word D [0] after substituting the value of D [0] by the S-box *S*0. The result is then added the word D [0] after rotating D [0] to the right by a constant value and substituting its value by the S-box *S*1.
  - The Word D [2] is updated, as shown in Equation 4, by adding its value to the word D [0] after right-rotation D[0] with a constant value and substituting its value by the S-box *S*0.
  - The Word D [3] is updated, as shown in Equation 5, by XORing its value to the word D [0] after right-rotation D[0] with a constant value and substituting its value by the S-box *S*1. These three words, D [1], D [2], and D [3] are implemented in hardware using additions modulo 2 (XOR), additions modulo 2^{32}, fixed rotation, and substitution, which are all explained above.
  - The Word D [0] is updated, as shown in Equation 6, by applying a fixed right rotation, and then its value is added to word D [1] only if the counter value is 1 or 5, or to the word D [3] only if the counter value is 0 or 4, otherwise there will be no addition operation. This condition can be applied in hardware using a multiplexer, whose selector input is the counter value, and its inputs to ports 0 and 4 are the word D, ports 1 and 5 the word B, and a constant zero for the other ports.
  - Finally, the word positions are permuted according to Equation 7.

- **Backward mixing:** In this layer the four words D[0], D[1], D[2], and D[3] are updated according to the following equations 8 to 12, which are repeated eight times (Burwick et al., 1999).

$$D[0] = D[0] - D[1] \ (if\ i = 3,7) - D[3] \ (if\ i = 2,6) \quad (8)$$
$$D[1] = D[1] \oplus S0[D[0]] \quad (9)$$
$$D[2] = D[2] - S0[D[0] <<< 8] \quad (10)$$
$$D[3] = D[3] - S1[D[0] <<< 16] \oplus S0[D[0] <<< 24] \quad (11)$$
$$(D[0], D[1], D[2], D[3]) = (D[1], D[2], D[3], D[0] <<< 24) \quad (12)$$

  - The same procedures for forward mixing are used to design the hardware by applying the above equations and using subtraction instead.

- **Cryptographic core:** In this layer the four words D[0], D[1], D[2], and D[3] are updated according to the following equations, which are repeated sixteen times (Burwick et al., 1999).

$$R = ((D[0] <<< 13) \times K[2i + 5]) <<< 10 \quad (13)$$
$$M = (D[0] + K[2i + 4]) <<< (Low\ 5 - bit\ of\ (R >>> 5)) \quad (14)$$
$$L = S[M] \oplus (R >>> 5) \oplus R) <<< (Low\ 5 - bit\ of\ R) \quad (15)$$
$$D[1] = D[1] + L \ (if\ i < 8) \oplus R \ (if\ i >= 8) \quad (16)$$
$$D[2] = D[2] + M \quad (17)$$
$$D[3] = D[3] \oplus R \ (if\ i < 8) + L \ (if\ i >= 8) \quad (18)$$
$$(D[0], D[1], D[2], D[3]) = (D[1], D[2], D[3], D[0] <<< 13) \quad (19)$$

- Word D [1] is updated, as shown in Equation 16, by adding its value to L, which is calculated according to Equation 15 if the counter value is less than 8. Otherwise, it is updated by XORing its value with R, which is computed by applying Equation 13.
- Word D [2], shown in Equation 17, is updated by adding its value to M, which is calculated according to Equation 14.
- Word D [3], Equation 18, is updated by XORing its value with R if the counter value is less than eight or by adding its value to L.
- So, there are two operations, addition and XORing, in the equation; however, only one is processed depending on the counter value.
- This is achieved in hardware using a multiplexer, where a counter value is connected to the selector input. Depending on the counter value, the multiplexer directs the result of one of the two operations.
- The last step is to permute the locations of the words according to Equation 19.

e. Operation Specific to the Twofish Algorithm

- **Key-dependent S-boxes**: The 32-bit input signal is first split into four 8-bit words by using and configuring four slice blocks. It is then followed by a number of operations, including XOR, Rotating, and mapping. For each q-permutation, four 16-byte RAM (4×2^4) is used.
- **MDS Matrices**: This layer is implemented by repeatedly multiplying by two until the required value is reached. Note that it is a polynomial multiplication, and multiplying by two is realized by shifting the input one bit to the left, followed by a conditional XOR with the primitive polynomial if the MSB value before the shift is high.
- **PHT transforms**: As explained earlier, the PHT transform is an operation that performs two additions modulo 2^{32}. Its implementation is the same as the addition operation explained above.

4 Results and Discussion

The hardware implementation results are shown in Table 1 and Figures 10-13. The algorithms are implemented on a Xilinx Virtex-6 xc6vlx195t-3 FPGA board. The current standard AES block cipher and one of its competitive algorithms, 'Serpent,' were previously implemented on the same hardware. Thus, a comparison between them can be made. The throughput is computed according to Equation 20. By analyzing the results shown in Table 1, it appears clear that the current standard outperforms its competitors in throughput and area usage. However, the Twofish algorithm precedes its counterpart in throughput. The power consumption rate is almost the same for AES, Serpent, and Twofish, but it is lower than the other algorithms. It is also evident from the results that the MARS algorithm occupies more resources than the other algorithms, and this is due to its mixing structure that includes three separate parts. In addition, MARS and RC6 have a lower throughput mainly due to the multiplication layer, which alone consumes three latencies to process. However, if an XOR operation replaces the multiplication layer in the MARS algorithm, the number of clock cycles will drop from 79 to 31, and the throughput will be 2.6 times higher than it was. However, this change certainly requires a comprehensive study and accurate analysis to ensure the safety of the algorithm from any vulnerability and its resistance to any attacks (Edition, 2023).

$$Throughput = \frac{Number\ of\ Processed\ Bits\ \times Maximum\ Frequency}{Number\ of\ Clock\ Cycles} \quad (20)$$

Table 1: Implementation Results and Comparison

Reference	Algorithm	Slice Registers	Slice LUTs	Occupied Slices	DSP48E1s	Maximum Frequency MHz	Throughput (Mbit/Sec)	Supply Power (W)
(Al-Khafaji et al., 2019)	AES	272	1354	423	0	191.975	2457	3.684
(Al-Gailani, 2018)	Serpent	392	1552	512	0	111.757	447	3.746
Proposed design	MARS	458	69992	17796	3	100.170	162	5.058
	Twofish	136	3684	1355	0	82.932	663	3.716
	RC6	259	29608	8330	6	128.617	208	4.076

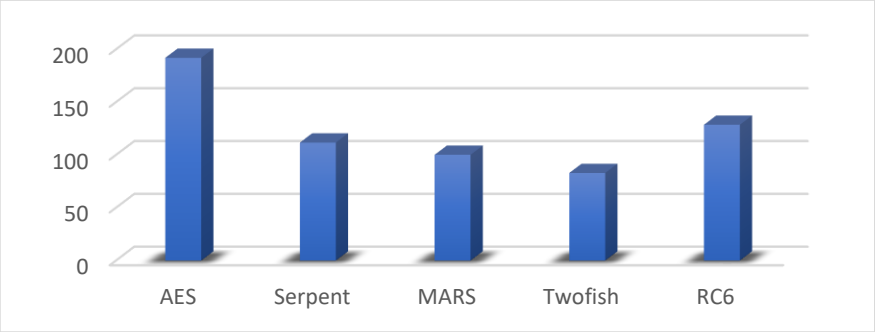


Figure 10: Maximum Frequency (MHz)

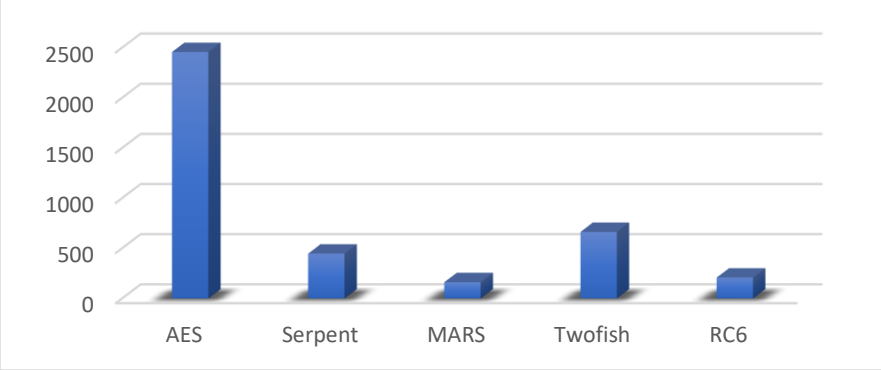


Figure 11: Throughput (Mbit/Sec)

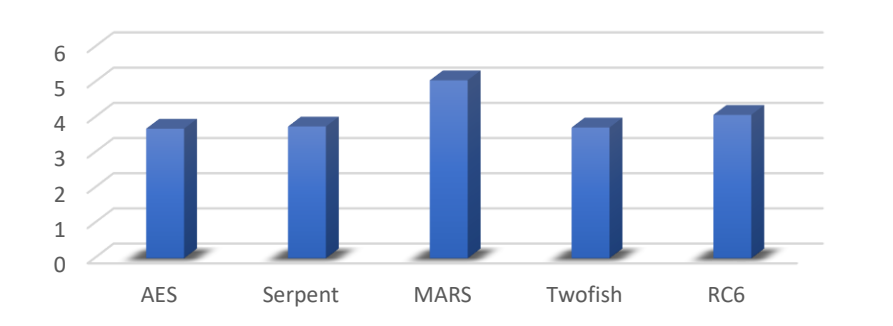


Figure 12: Supply Power (W)

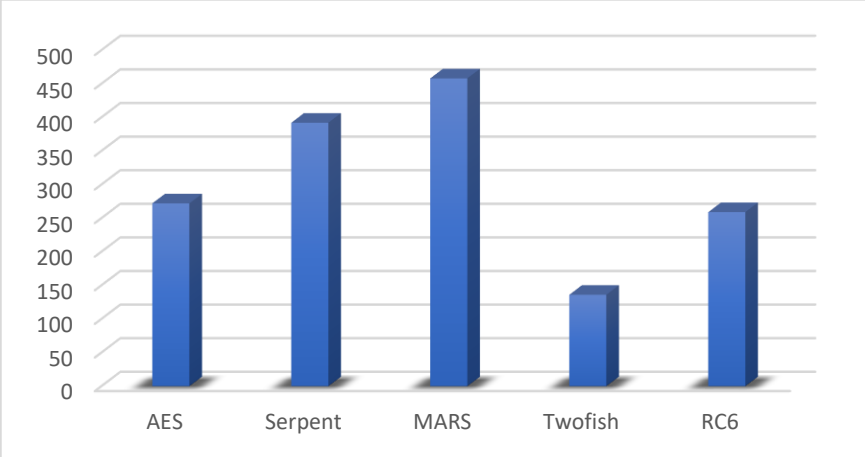


Figure 13: Slice Registers

5 Conclusion

In this paper, hardware for AES candidates, namely MARS, Twofish, and RC6 algorithms, are designed and implemented using a system generator and Xilinx ISE 14.7. The structure of these algorithms was carefully prepared to be secure against current and future attacks. For this reason, they have been analyzed and designed for potential use in applications requiring high security. The designs focus on applications that have limited area and consume less power. Thus, iterative looping architecture is considered, in which the hardware is designed for layers of one round. The whole process is accomplished by iterating this piece of hardware with the number of rounds. This minimizes the resources used and, consequently, reduces power consumption. However, higher throughput can be achieved by modifying the design at the expense of resources, as this requires duplicating the resources by the number of iterations, as in (Al-Gailani & Al-Khafaji, 2019)Round keys are computed on the fly to increase flexibility and reduce memory requirements. However, they can be precomputed to improve processing time and throughput. In general, these algorithms perform well compared to the levels of security they provide.

References

[1] Al-Gailani, M. F., & Al-Khafaji, A. Q. (2019). Loop Unrolling Implementation of an AES Algorithm using Xilinx System Generator. *Iraqi Journal of Information & Communications Technology*, *2*(3), 38–45. https://doi.org/10.31987/ijict.2.3.85

[2] Al-Khafaji, A. Q., Al-Gailani, M. F., & Abdullah, H. N. (2019). FPGA Design and Implementation of an AES Algorithm based on Iterative Looping Architecture. *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, 1–5. https://doi.org/10.1109/ICCE-Berlin47944.2019.8966137

[3] Anny Leema, A., Balakrishnan, P., & Jothiaruna, N. (2024). Harnessing the Power of Web Scraping and Machine Learning to Uncover Customer Empathy from Online Reviews. *Indian Journal of Information Sources and Services*, *14*(3), 52–63. https://doi.org/10.51983/ijiss-2024.14.3.08

[4] Barker, E., & Mouha, N. (2017). *Recommendation for the triple data encryption algorithm (TDEA) block cipher* (No. NIST Special Publication (SP) 800-67 Rev. 2 (Draft)). National Institute of Standards and Technology.

[5] Biham, E., Anderson, R., & Knudsen, L. (1998, March). Serpent: A new block cipher proposal. In *International workshop on fast software encryption* (pp. 222-238). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-69710-1_15

[6] Burwick, C., Coppersmith, D., D'Avignon, E., Gennaro, R., Halevi, S., Jutla, C., Matyas, S. M., O'Connor, L., Peyravian, M., Safford, D., & Zunic, N. (1998, July 17). MARS—a candidate cipher for AES. *Proceedings of the First AES Candidate Conference*. IBM Corporation.

[7] Burwick, C., Coppersmith, D., D'Avignon, E., Gennaro, R., Halevi, S., Jutla, C., Matyas, S. M., O'Connor, L., Peyravian, M., Safford, D., & Zunic, N. (1999, August 27). The MARS encryption algorithm. *Proceedings of the 2nd AES Candidate Conference*. IBM Corporation. https://shaih.github.io/pubs/mars/mars-short.pdf

[8] Chodowiec, P., & Gaj, K. (1999). Implementation of the twofish cipher using FPGA devices. *Electrical and Computer Engineering, George Mason University*. https://www.schneier.com/wp-content/uploads/2015/12/paper-twofish-fpga.pdf

[9] Daemen, J., & Rijmen, V. (2020). Te Design of Rijndael: Te Advanced Encryption Standard (AES). https://doi.org/10.1007/978-3-662-60769-5

[10] Edition, W. S. F. (2023). Cryptography And Network Security.

[11] Hori, Y., Claycomb, W., & Yim, K. (2012). Guest editorial: frontiers in insider threats and data leakage prevention. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, *3*(1/2).

[12] Kilts, S. (2007). *Advanced FPGA design: architecture, implementation, and optimization*. John Wiley & Sons. https://doi.org/10.1002/9780470127896

[13] Kramer, S. (1996). Announcing development of a federal information processing standard for advanced encryption standard. *Federal Register*, *62*, 93-94.

[14] Lai, Y. K., Chen, L. G., Lai, J. Y., & Parng, T. M. (2002, May). VLSI architecture design and implementation for twofish block cipher. In *2002 IEEE International Symposium on Circuits and Systems (ISCAS)* (Vol. 2, pp. II-II). IEEE. https://doi.org/10.1109/ISCAS.2002.1010998

[15] M. F. Al-Gailani. (2018). An FPGA Implementation of the Serpent Algorithm using Xilinx System Generator. *Journal of Engineering and Applied Sciences*, *13*(8), 1974–1979. https://doi.org/doi: 10.3923/jeasci.2018.1974.1979

[16] National Bureau of Standards. (1977, January 15). *FIPS PUB 46, Data Encryption Standard (DES)*. https://csrc.nist.gov/files/pubs/fips/46/final/docs/nbs.fips.46.pdf

[17] NIST. (1997b). Announcing request for candidate algorithm nominations for the Advanced Encryption Standard (AES). *Federal Register*, 62(117), 48051–48058.

[18] NIST. (2001, November 26). *FIPS 197, Advanced Encryption Standard (AES)*. National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf

[19] Purnima Gehlot, Richa Sharma, & S. R. Biradar. (2013). Twofish Algorithm and its Implementation on FPGA. *International Conference on Recent Trends in Computing and Communication Engineering RTCCE 2013*, 177–180. https://doi.org/10.15224/978-981-07-6184-4-38

[20] Riaz, M., & Heys, H. M. (1999). The FPGA implementation of the RC6 and CAST-256 encryption algorithms. *Engineering Solutions for the Next Millennium. 1999 IEEE Canadian Conference on Electrical and Computer Engineering (Cat. No.99TH8411)*, *1*, 367–372. https://doi.org/10.1109/CCECE.1999.807226

[21] Rivest, R. L. (1994, December). The RC5 encryption algorithm. In *International Workshop on Fast Software Encryption* (pp. 86-96). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-60590-8_7

[22] Rivest, R. L., Robshaw, M. J., Sidney, R., & Yin, Y. L. (1998, August). The RC6TM block cipher. In *First advanced encryption standard (AES) conference* (p. 16). https://people.csail.mit.edu/rivest/pubs/RRSY98.pdf

[23] Salman, R. H., & Alomari, E. S. Survey: Homomorphic Encryption-based Deep Learning that Preserves Privacy. https://doi.org/10.9756/IAJSE/V10I2/IAJSE1019

[24] Schneier, B. (Ed.). (1999). *The Twofish encryption algorithm: A 128-bit block cipher*. Wiley.
[25] Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., & Ferguson, N. (1998). Twofish: A 128-bit block cipher. *NIST AES Proposal*, *15*(1), 23-91.
[26] Sujatha, S. (2024). Strategic Management of Digital Transformation: A Case Study of Successful Implementation. *Global Perspectives in Management*, *2*(1), 1-11.
[27] Taraate, V. (2021). ASIC Design and Synthesis. *Springer Nature*.
[28] Wilson, P. (2015). *Design recipes for FPGAs: using Verilog and VHDL*. Newnes.
[29] Xilinx. (2012, October 16). *System Generator for DSP user guide*. Xilinx. https://www.origin.xilinx.com/support/documents/sw_manuals/xilinx14_7/sysgen_user.pdf
[30] Yaremko, H., Stoliarchuk, L., Huk, L., Zapotichna, M., & Drapaliuk, H. (2024). Transforming Economic Development through VLSI Technology in the Era of Digitalization. *Journal of VLSI Circuits and Systems*, *6*(2), 65-74. https://doi.org/10.31838/jvcs/06.02.07

Author Biography

M.F. Al-Gailani is an Associate Professor at Al-Nahrain University, specializing in Information Security. He has bachelor's and Master's degrees in Computer Engineering from the University of Technology, and a Ph.D. degree in Computer Engineering from Newcastle University, UK. Dr. Al-Gailani has been working in the academic and research field for more than twenty years and has published more than twenty research articles in his field of specialization. He has also held many administrative positions during his academic career and is currently the Head of the Department of Cybersecurity Engineering.