

A new multi-level key block cypher based on the Blowfish algorithm

Suhad Muhajer Kareem¹, Abdul Monem S. Rahma²

¹University of Basrah, Collage of Computer Science and Information Technology, Iraq

²University of technology, Department of Computer Science, Iraq

Article Info

Article history:

Received Jul 10, 2019

Revised Dec 26, 2019

Accepted Feb 8, 2020

Keywords:

Blowfish algorithm

Encryption security

Multi-level keys

Symmetric block cipher

ABSTRACT

Blowfish is a block cypher algorithm used in many applications to enhance security, but it includes several drawbacks. For example, the mix between the key and data is limited. This paper presents a new modification to the Blowfish algorithm to overcome such problems realised through a multi-state operation instead of an XOR. Our proposed algorithm uses three keys in the encryption and decryption processes instead of one for controlling the variable block bits sizes (1, 2, 4, and 8) bits and for determining the state table numbers. These tables are formed from the addition in a Galois field $GF(2^n)$ based on block bit size to increase the complexity of the proposed algorithm. Results are evaluated based on the criteria of complexity, time encryption, throughout, and histogram, and show that the original Blowfish, those modified by other scholars, and our proposed algorithm are similar in time computation. Our algorithm is demonstrated to be the most complex compared with other well-known and modified algorithms. This increased complexity score for our proposed Blowfish makes it more resistant against attempts to break the keys.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Suhad Muhajer Kareem,

University of Basrah,

Collage of computer science and information technology, Basrah, Iraq.

Email: suhad_althaher@yahoo.com

1. INTRODUCTION

Rapid development in information technology has led to a reliance on the transmission of electronic information via networks. As it is necessary to provide secure information environments, many researchers address this security challenge [1, 2]. One method for protecting information is the use of encryption algorithms between two parties involved in communication by converting the message into a human-unrecognisable form [3]. Algorithmic encryption is classified into symmetric key and asymmetric key encryption. Symmetric key encryption uses the same key to implement encryption and decryption, while asymmetric-key encryption incorporates different public and private keys. Symmetric algorithms include block and stream cyphers [4, 5], and Blowfish is an example of a block cypher [6].

Most modern encryption algorithms are based on the Feistel network, invented by Horst Feistel, which is defined as a common method for converting a function F into other permutation. Feistel networks comprise multiple rounds of repeated operations, such as a permutation process using P-boxes, substitution using S-boxes, and a logical XOR operation [7, 8]. An encryption algorithm depends on a key as the significant element that can be numeric, alphanumeric text or special symbols [9, 10]. However, modern cryptographic algorithms depend on functions with two states (0, 1) for encryption and decryption. As one of

the block algorithms, Blowfish uses the classic XOR logical operation that depends on two states (0, 1) that includes several weak points. For instance, this level of simplicity can be deciphered easily by attackers. Studies previously attempted to replace these two states with four (0, 1, 2, 3) for increasing key space, which is not perfect works for making algorithm more strong against attacks [11], as described in the following section. Therefore, in this study, we focus on the weak points of XOR by replacing it with a new # operation with variable block bit sizes (n): (1 or 2 or 4 or 8) instead of one block size. Each block will generate different states tables based on addition in $GF(2^n)$. The overall new # operation is managed by using additional two keys. This work is repeated in each round of Blowfish to increase the security level of the algorithm. Our results show that this new modification on Blowfish algorithm will increase the security level of the encryption by increasing the complexity in each round and thus the protection of encrypted messages will be guaranteed against attacks.

The remainder of this paper is organised as follows. Section 2 overviews the principle work of the Blowfish algorithm. Section 3 reviews the relevant literature about modifications of Blowfish, and Section 4 introduces our proposed work and evaluation of the Blowfish algorithm. Section 5 evaluates the complexity of our proposed algorithm, followed by conclusions in Section 6.

2. BLOWFISH OVERVIEW

Blowfish is a symmetric block cypher invented in 1993 by Bruce Schneier as a general purpose, rapid, and free alternative to the DES (Data Encryption System) algorithm. Blowfish includes many features, such as easy, rapid, and compact use in the Secure Socket Layer as well as other applications, and can encrypt multimedia files with different speeds [8, 12]. The Blowfish algorithm is based on a Feistel network comprised of a 16-round iteration with plaintext represented in a 64-bit block and the key length changes from 32 to 448 bits. Blowfish separates the plaintext into left and right halves for applying an XOR operation and an F-function. All operations in each round are additions and XORs using 32-bit words performed within four substitution boxes (S-boxes) [13, 14]. The process of Blowfish entails the following two phases [15, 16].

- Key generation phase: Blowfish uses many subkeys for generating the initial contents of an array called the P-array. These keys must be pre-computed before data encryption or decryption. The P-array comprises 18 32-bit (P_1, P_2, \dots, P_{18}) subarrays and four 32-bit S-boxes with 256 entries as follows:

$$S_{1,0}, S_{1,1}, \dots, S_{1,255}; S_{2,0}, S_{2,1}, \dots, S_{2,255}; S_{3,0}, S_{3,1}, \dots, S_{3,255}; S_{4,0}, S_{4,1}, \dots, S_{4,255} \quad (1)$$

- Encryption phase: 64-bit plaintext is encrypted using the key generated in the previous phase to produce a 64-bit message.

Figure 1 clarifies how the F-function in each round of the Blowfish algorithm depends on the substitution concepts and consists of four S-boxes. The input of the 32-bit is divided into four 8-bit values, which are mixed using an addition modulo and combined using XOR [3, 17]. Figure 1 shows the five operations included in a single round of two XOR, two addition modulo, 32, and addition. Accordingly, the evaluation computations for every round also includes only five operations. The work of Blowfish in steps is cited with details in [18].

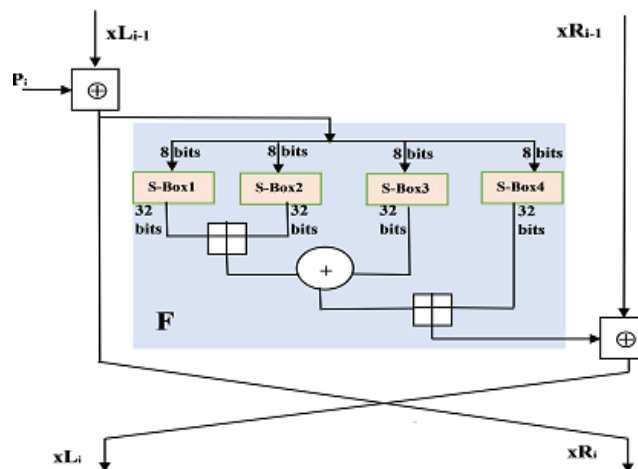


Figure 1. A structure of F-function in one round of blowfish algorithm

3. RELATED WORK

In this section, we overview the related literature on various modifications using four states tables in key generation and applied it in the Blowfish algorithm. In 2009 [11], researchers combined the curve security methods using the B-spline curve with quantum cryptography concepts to increase security and key space. Moreover, this modification on the base protocol focused on the manipulation of the bits in the message by replacing the original XOR operation with a new hash operation. However, the message was encoded with four different states (0, 1, 2, and 3) instead of two (0, 1) to make variations in the polarised angles used in the quantum description. These variations are encrypted in the four tables, and the output description incorporated the polarised state angles according to the table. Then manipulation cyphers convert the plaintext into ciphertext by replacing the current state pattern of each character using a new logical operator (#). In other words, the hash operation requires three inputs of the table number to be used for computing the result as well as the corresponding row and columns. The new encoding is applied using (#) operation which depends on four state tables shown in following Figure 2.

#0	0	1	2	3	#1	0	1	2	3
0	3	2	1	0	0	0	1	2	3
1	2	3	0	1	1	1	0	3	2
2	1	0	3	2	2	2	3	0	1
3	0	1	2	3	3	3	2	1	0

#2	0	1	2	3	#3	0	1	2	3
0	2	3	0	1	0	1	0	3	2
1	3	2	1	0	1	0	1	2	3
2	0	1	2	3	2	3	2	1	0
3	1	0	3	2	3	2	3	0	1

Figure 2. The truth table for # operation

In 2011, a new modification was introduced in [19] to increase the security and key space of Blowfish by replacing the predefined XOR operation applied in each round of the standard Feistel algorithm with a modified bit process of a new hash operation based on two keys. Here, each key consists of a combination of four states (0, 1, 2, 3) instead of the original two-state key (0, 1) using the truth tables proposed in [11]. The first key is used to determine the table number among the four available while the second key is used in the encryption algorithm. This replacement provides a new level of security to the algorithm against attackers by increasing its complexity. In our proposed, we extend block size of Blowfish algorithm from static block (2 blocks and four tables) into dynamic blocks (4 blocks and 16 tables) and (8 blocks and 256 tables).

4. PROPOSED MULTI-STATE BLOWFISH

Today, the Blowfish algorithm is considered insecure for many applications. For instance, only a single bit is used for manipulation (0 or 1), which does not contain sufficient randomness making the algorithm vulnerable to brute force attacks. To overcome this issue, we introduce a new modification to Blowfish to enhance its encryption performance and increase its complexity against attacks.

As described above, the XOR operation used in the Feistel network depends on two states (0 or 1) in the bit manipulation for mixing the input key and plaintext. This paper presents a modification to the Feistel algorithm by replacing the XOR with a new logical operation called #. This operation is based on a new bit manipulation for mixing the input key and the plaintext in variable bits sizes of 1, 2, 4, and 8-bits with block one's bit sizes operating on a two-state table (0 or 1), block two's bit sizes operates on a four-state table (0 or 1 and 2 or 3), block four's bit sizes operates on a 16-state table (0 or 1, or ... or 15), and block eight's bit sizes operating on a 256-state table (0 or 1 or ... 255). These tables are constructed from the addition operation in the Galois field $GF(2^n)$, where n is the value that depends on the block bit size specified by the key $number\ of\ bits$. In this work, four variable block bit sizes of 1, 2, 4, and 8 are used with a 2-state table (#0 and #1) for $GF(2^1)$, 4-state table (#0, #1, #2, and #3) for $GF(2^2)$, 16-state table (#0, #1, ..., #15) for $GF(2^4)$, and 256-state table (#0, #1, ..., #255) for $GF(2^8)$. Samples of these tables are presented in the next section as Tables 1 through 6.

To control the number of bits and states, the new # operation requires two additional keys, each generated independently. The first key, $key_{number\ of\ bits}$, is generated randomly in binary form for determining the block bit size taken from the input key and the plaintext. Then, the number of states is generated based on

the block bit size as selected by *keynumber of bits*. The second key, *keyno of table*, is also generated randomly in decimal form for selecting one of the state tables. After selecting the block bit size and state number, the # operation is applied with three inputs comprised of the state table number for calculating the result among the different state tables as well as the row and column numbers to cross-reference the result in the table.

Let, K_b = key number of bits, and K_c = key no. of table, then: $K_b = 000110110101010 \dots$ (generated randomly for encryption and decryption): At each round, take two bits from the K_b and check it:

- If $K_b = 00$, then the block bit size = 1 and recall 2- state tables: K_c select randomly one table either 0 or 1 for encrypting and decrypting.
- If $K_b = 01$, then the block bit size = 2 and recall 4-state tables: K_c select randomly one table among (0, 1, ..., 4) for encrypting and decrypting.
- If $K_b = 10$, then the block bit size = 4 and recall 16- state tables: K_c select randomly one table among (0, 1, ..., 15) for encrypting and decrypting.
- If $K_b = 11$, then the block bit size = 8 and recall 256-state tables: K_c select randomly one table among (0, 1, ..., 255) for encrypting and decrypting.

The process of the # operation comprises three steps for applying $(xL \# P)$ in each round in the original blowfish algorithm as shown in Figure 3. Figure 3 shows the overall operation during one modified round that consists of five operations (# with different states, XOR, two addition modulo 32, and addition) as described in Section 5.

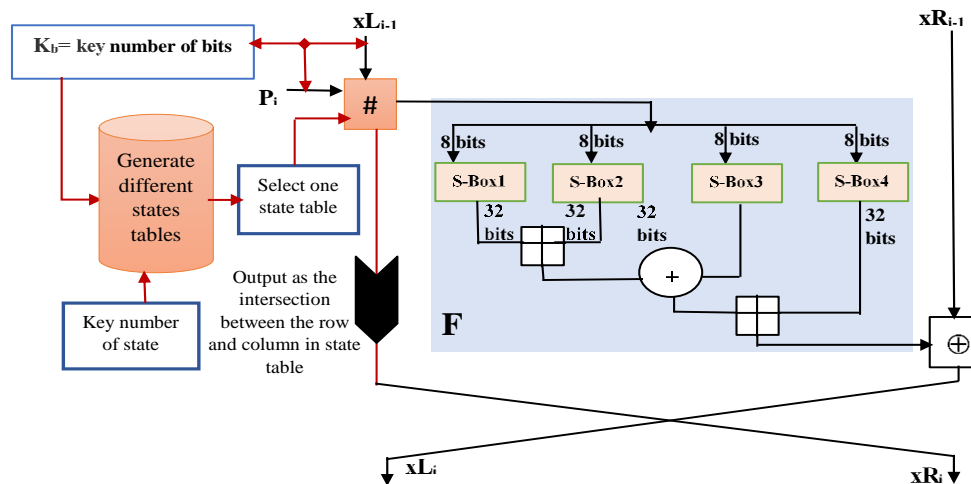


Figure 3. The modified process of the # operation in one round

4.1. Construction the state tables

This section shows sample tables that constructed based on the addition operation in the Galois field $GF(2^n)$. Tables 1-3 and Tables 4-6 represent the addition in $GF(2^4)$ and $GF(2^8)$ consecutively.

Table 1. State (#0) addition in $GF(2^4)$

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
#0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0001	1	1	0	3	2	5	4	7	6	9	8	11	10	13	12	15
0010	2	2	3	0	1	6	7	4	5	10	11	8	9	14	15	12
0011	3	3	2	1	0	7	6	5	4	11	10	9	8	15	14	13
0100	4	4	5	6	7	0	1	2	3	12	13	14	15	8	9	10
0101	5	5	4	7	6	1	0	3	2	13	12	15	14	8	9	10
0110	6	6	7	4	5	2	3	0	1	14	15	12	13	10	11	8
0111	7	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9
1000	8	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6
1001	9	9	8	11	10	13	12	15	14	1	0	3	2	5	4	7
1010	10	10	11	8	9	14	15	12	13	2	3	0	1	6	7	4
1011	11	11	10	9	8	15	14	13	12	3	2	1	0	7	6	5
1100	12	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2
1101	13	13	12	15	14	9	8	11	10	5	4	7	6	1	0	3
1110	14	14	15	12	13	10	11	8	9	6	7	4	5	2	3	0
1111	15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Table 2. State (#3) addition in GF (2⁴)

	#3	0	1	2	13	14	15
0000	0	3	2	1	12	13	14
0001	1	2	3	0	15	12	13
0010	2	1	0	3	12	15	14
0011	3	0	1	2	13	14	15
.
.
.
.
.
1100	12	13	12	0	3	2	
1101	13	12	13	14	1	2	3
1110	14	15	14	13	2	1	0
1111	15	14	15	12	3	0	1

Table 3. State (#15) addition in GF (2⁴)

	#15	0	1	2	13	14	15
0000	0	15	14	13	2	1	0
0001	1	2	3	0	15	12	13
0010	2	1	0	3	12	15	14
0011	3	4	5	6	9	10	11
.
.
.
.
.
.
1100	12	11	10	9	6	5	4
1101	13	14	15	12	3	0	1
1110	14	13	12	15	0	3	2
1111	15	0	1	2	13	14	15

Table 4. State (#0) addition in GF (2⁸)

	00000000	00000001	00000010	00000011	00010010	00010011	00110100	01111101	01111110	01111111	11111101	11111110	11111111
#0	0	1	2	3	50	51	52	125	126	127	253	254	255
00000000	0	1	2	3	50	51	52	125	126	127	253	254	255
00000001	1	0	3	2	51	50	53	124	127	126	252	255	254
00000010	2	3	0	1	48	49	54	127	124	125	255	252	253
00000011	3	2	1	0	49	48	55	126	125	124	254	253	252
00000100	4	5	6	7	54	55	48	121	122	123	249	250	251
00000101	5	4	7	6	55	54	49	120	123	122	148	251	250
.
.
.
.
.
11111010	250	251	248	249	200	201	206	135	132	133	7	4	5
11111011	251	250	249	248	201	200	207	134	133	132	6	5	4
11111100	252	253	254	255	206	207	200	129	130	131	1	2	3
11111101	253	252	255	253	207	206	201	128	131	130	0	3	2
11111110	254	255	252	253	204	205	202	131	128	129	3	0	1
11111111	255	254	251	252	205	204	203	130	129	128	2	1	0

Table 5. State (#170) addition in GF (2⁸)

	00000000	00000001	00000010	00000011	00000100	00000101	11111010	11111011	11111100	11111101	11111110	11111111	
#170	0	1	2	3	4	5	250	251	252	253	254	255	
00000000	0	170	171	168	169	174	175	80	81	86	87	84	85
00000001	1	2	3	0	1	6	7	248	249	254	255	252	253
00000010	2	1	0	3	2	5	4	251	250	253	252	255	254
00000011	3	4	5	6	7	0	1	254	255	248	249	250	251
00000100	4	3	2	1	0	7	6	249	248	255	254	253	252
00000101	5	6	7	4	5	2	3	252	253	250	251	248	249
.	
.	
.	
.	
.	
11111010	250	251	250	249	248	255	254	1	0	7	6	5	4
11111011	251	250	251	248	249	254	255	0	1	6	7	4	5
11111100	252	253	252	255	254	249	248	7	6	1	0	3	2
11111101	253	252	253	254	255	248	249	6	7	0	1	2	3
11111110	254	255	254	253	252	251	250	5	4	3	2	1	0
11111111	255	254	255	252	253	250	251	4	5	2	3	0	1

Table 6. State (#200) addition in GF (2⁸)

	00000000	00000001	00000010	00000011	00000100	00000101	11111010	11111011	11111100	11111101	11111110	11111111	
#200	0	1	2	3	4	5	250	251	252	253	254	255	
00000000	0	200	201	202	203	204	205	50	51	52	53	54	55
00000001	1	2	3	0	1	6	7	248	249	254	255	252	253
00000010	2	1	0	3	2	5	4	251	250	253	252	255	254
00000011	3	4	5	6	7	0	1	254	255	248	249	250	251
00000100	4	3	2	1	0	7	6	249	248	255	254	253	252
00000101	5	6	7	4	5	2	3	252	253	250	251	248	249
.	
.	
.	
.	
.	
11111010	250	251	250	249	248	255	254	1	0	7	6	5	4
11111011	251	250	251	248	249	254	255	0	1	6	7	4	5
11111100	252	253	252	255	254	249	248	7	6	1	0	3	2
11111101	253	252	253	254	255	248	249	6	7	0	1	2	3
11111110	254	255	254	253	252	251	250	5	4	3	2	1	0
11111111	255	254	255	252	253	250	251	4	5	2	3	0	1

4.2. Modified Blowfish algorithm in steps

This section introduces the proposed blowfish algorithm in steps as shown hereunder. Modified steps have been highlighted with the red colour:

Algorithm1-proposed blowfish using multi-level keys

Input: Plaintext 64-bit (X) and key

Output: Ciphertext 64-bit

Begin

Step1: sub key generation.

Step 2: Generate randomly control key 32 bits in binary form called K_{bi} (number of bits) for 16 round.

Step3: Encryption

begin

Step3.1: Divide x into two 32-bit halves: xL, xR

Step3.2: For $i=0$ to 16, compute xL and xR as follows:

begin

step3.2.1 Compute $xL_i = xL_i \# P_i$ as follows:

a. Split K_{bi} to two bits (N_b), then test as follows:

where n is the block bit size selected from xL_i and P_i

1. If $N_b = 00$, then $n = 1$; go to b;

2. If $N_b = 01$, then $n = 2$; go to b;

3. If $N_b = 10$, then $n = 4$; go to b;

4. If $N_b = 11$, then $n = 8$; go to b.

b. Depending on the previous step, recall states tables using addition on GF (2^n).

c. Generate random key (K_{ci}) for selecting one state table that recall in step (b).

d. Compute xL_i by applying the operation on ($xL_i \# P_i$) according to three inputs (index = number of state table, row = xL_i , and column = P_i).

e. The output of step (d) as the cross point between the row and column in the specified state table to give the result.

Step3.2.2: Compute $xR_i = F(xL_i) \oplus xR_i$; (F=function in Figure 1)

Step3.2.3: Swap the values (xR_i , xL_i)

endfor

Step3.3: Swap the values (xR , xL)

Step3.4: Compute $xR = xR \# P17$ and $xL = xL \# P18$, where # computed as previous

Step (3.2.1)

Step3.5: Recombine xL and xR to get the ciphertext

End

Step4: Decryption

Decryption is exactly the same as encryption, except that $P1, P2 \dots P18$ are used exactly in reverse order but should the keep the K_{bi} and K_{ci} for retrieving the plaintext.

End

5. EVALUATION

This section presents four evaluation metrics of complexity, encryption time and throughput, NIST tests and histogram of the proposed modified Blowfish algorithm.

5.1. Complexity computing

Algorithm complexity is computed against the attackers to estimate the key [20] followed by the encryption times. The complexity is calculated from the possible number of keys an attacker needs to decrypt the ciphertext with 64 bits. The complexity of the Blowfish algorithm in sixteen rounds is calculated as (the probability of plaintext) \times (the probability of $key^{no\ of\ round}$).

- First, we compute the complexity of the original Blowfish algorithm using a predefined XOR binary operation (0, 1), thereby giving the number of possible keys used in the encryption and decryption as:

$$2 \times (2)^8 \times 32 \times 2 = 2^{15} \quad (2)$$

- *Second*, when using the # operation in the modified Blowfish algorithm from [19] with four states (#0, #1, #2, and #3) and two bits instead of one, the number of the possible keys used in the encryption and decryption is:

$$(2^2)^{16} \times (2^2)^8 \times 2^2 \times 32 \times 2 = 2^{72} \tag{3}$$

- *Third*, we compute the complexity of the proposed algorithm using three keys as (the probability of plaintext)×(the probability of $key^{no. \text{ of round}}$)×(the probability of the state tables). The overall complexity of our proposed algorithm is as follows:

$$((2^1)^{32} \times (2^2)^{16} \times (2^4)^8 \times (2^8)^4) \times ((2^1)^{32} \times (2^2)^{16} \times (2^4)^8 \times (2^8)^4) \times (2 \times 2^2 \times 2^4 \times 2^8) \times 32 \times 2 = 2^{128} \times 2^{1024} \times 2^{15} \times 2^5 \times 2^1 = 2^{1173} \tag{4}$$

Table 7 summarises the results from computing the complexity of 16 rounds from the three algorithms (our proposed algorithm, the original Blowfish, and the previously modified algorithm from [19]).The algorithm complexity computation results are calculated in key size of 64-bits. Figure 4 explains how the proposed Blowfish algorithm features higher complexity compared to the original and modified Blowfish from [19] in sixteen rounds.

Table 7. The complexity comparison of original, previously modified, and proposed.

Blowfish algorithm for sixteen rounds	
Algorithm	The complexity
Well-known Blowfish	$2^{15} = 32,768$
Modified Blowfish [19]	$2^{72} = 4,722,366,482$
Our proposed Blowfish	$2^{1173} = 1.28287668946279217437411e+353$

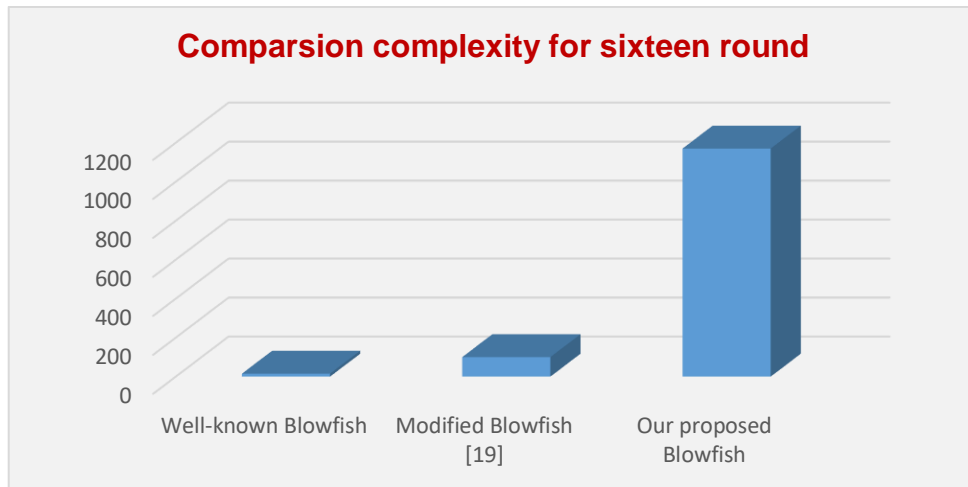


Figure 4. The overall comparison of the complexity

5.2. Encryption time and thruput

As another metric for measuring the performance of the algorithm, the encryption time is computed by the time required for converting the plaintext into an unrecognised form. While the thruput metric as applied in this context is calculated as [21, 22]:

$$\text{Throughput} = \frac{\text{plaintext size (in kilobyte)}}{\text{total encryption time (ms)}} \tag{5}$$

Table 8 and Figure 5 show the experimental results from calculating the encryption time and thruput for the proposed and the modified Blowfish algorithms. From these table and figure, the original, modified, and the proposed Blowfish algorithms are equivalent in terms of computation time. However, our proposed method offers more effective results related to the complexity evaluation against attacks, which enables our Blowfish algorithm to be more difficult for an attacker to retrieve the original message.

Table 8. The encryption time and throughput of original and proposed.
Blowfish algorithms

File sizes(kb)	Execution times (in milliseconds) for:	
	Original Blowfish algorithm	Our proposed Blowfish algorithm
15	8	8
25	13	12
50	31	31
80	46	46
150	50	49
500	72	72
Average time	36.66	36,33
Throughput	22.367	22.570

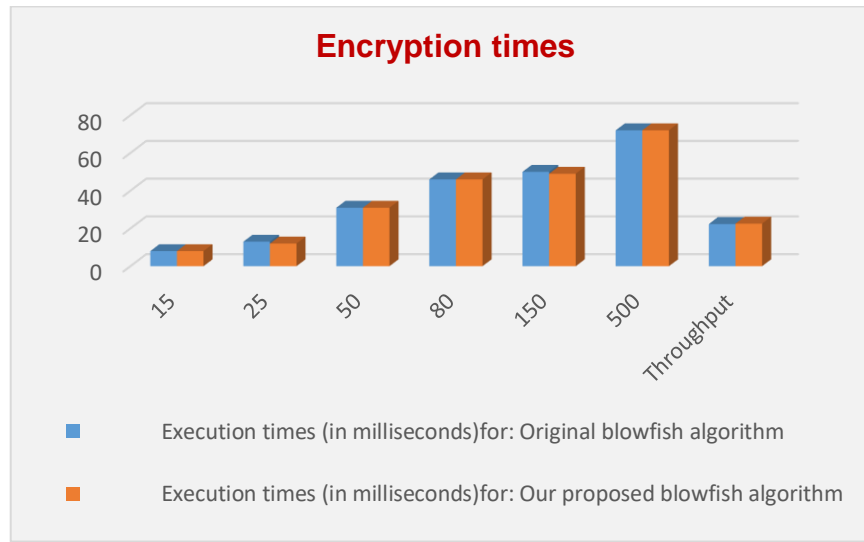


Figure 5. The overall analysis of encryption time

5.3. NIST analysis

The output of the encryption algorithm should be more random and unpredictable. Several methods exist for computing the randomness, such as NIST (National Institute of Standards and Technology), Diehard tests, and TestU01 [23, 24]. In this paper, we use 15 statistical tests from NIST statistical for testing the randomness of the binary sequences, as shown in Table 9. This and the modified tests are calculated over multiple cypher text produced from the original Blowfish. The probability value (p-value) is set to a value of 0.01 to confirm if the output is random. The average tests are computed and listed in Table 9.

Table 9. Result of Running NIST on the Generated Key by Blowfish and the proposed Blowfish

Test no.	Statistical Test Name	Well-known Blowfish		Proposed Blowfish	
		P-Value	Status	P-Value	Status
1	Approximate Entropy	-	Fail	0.848	pass
2	Block Frequency	0.073	pass	0.216	pass
3	Cumulative Sums	0.644	pass	0.133	pass
4	FFT	0.061	pass	0.867	pass
5	Frequency	0.043	pass	0.791	pass
6	Linear complexity	0.138	pass	0.998	pass
7	Longest Run	0.126	pass	0.367	pass
8	Non Overlapping Template	0.241	pass	0.490	pass
9	Overlapping Template	0.176	pass	0.506	pass
10	Random Excursions	0.520	pass	0.864	pass
11	Random Excursions Variant	0.329	pass	0.871	pass
12	Rank	0.187	pass	0.846	pass
13	Runs	0.180	pass	0.648	pass
14	Serial	-	Fail	0.490	pass
15	Universal	0.447	pass	0.940	pass

If the test results provide a p-value asymptotically approaching 1, then the output should appear to have complete randomness. A p-value equal to zero signifies that the output is non-random. The pass status represents that the p-value of these tests is greater than 0.001 and denotes the output is acceptable (e.g., offers good randomness). The p-values of most of the tests from the proposed Blowfish algorithm are greater than the p-values of the well-known Blowfish, as shown in Table 9.

5.4. Histogram analysis

A histogram is used to measure the security of the original, encrypted, and decrypted images [25] using the well-known and proposed Blowfish. The experimental results are shown in Figures 6 and 7 for two images (Baboon and Lena).

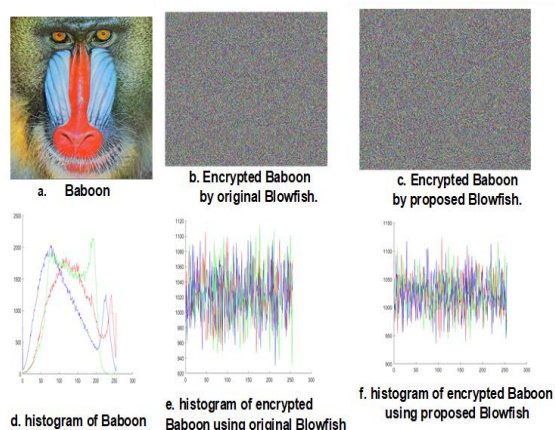


Figure 6. Results and histogram of well-known Blowfish and proposed Blowfish for Baboon image (original and encrypted)



Figure 7. Results and histogram of well-known Blowfish and proposed Blowfish for Lena image (original and encrypted)

6. CONCLUSION

A new method is proposed for enhancing the security and performance of the Blowfish algorithm achieved by an additional key and replacing the XOR with a new operation. These modifications provide more strength to the Blowfish algorithm and making it resistant against an attack. Using multiple keys instead of one increases the reliability of the key, which also increases the efficiency of the encryption and decreases the probability of breaking from brute force attacks. Moreover, a variable block bit size in each round increases the randomness and security of the algorithm. Based on our statistical results, we conclude that our proposed algorithm provides more complexity compared to the original Blowfish while maintaining approximately the same computation time.

REFERENCES

- [1] Swathi S, IILahari P., "Encryption algorithms: a survey," *International Journal of Advanced Research in Computer Science & Technology*, vol. 4, no. 2, pp. 81-88, 2016.
- [2] Septafiansyah D. P, Mario Y., Sarwono S., Yusuf K., Adang S. A., "Power analysis attack against encryption devices: a comprehensive analysis of AES, DES, and BC3," *TELKOMNIKA Telecommun Comput El Control*, vol. 17, no. 3, pp. 1282-1289, June 2019.
- [3] Sonia R., Harpreet K., "Technical survey on cryptography algorithms for network security," *International Journal of Advanced Research in Computer Science and Software Engineering* vol. 4, no. 9, pp. 204-209, 2016.
- [4] Christina L., Joe Irudayaraj V. S., "Optimized Blowfish encryption technique," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 2, no. 7, pp. 5009-5015, 2014.
- [5] M. A. Hameed, Ahmed I. Jaber, Jamhoor M. Alobaidy, Alaa A. Hajer, "Design and Simulation DES Algorithm of Encryption for Information Security," *American Journal of Engineering Research (AJER)*, vol. 7, no. 4, pp. 13-22, 2018.
- [6] Ayushi, "A Symmetric Key Cryptographic Algorithm," *International Journal of Computer Applications (0975-8887)*, vol. 1, no. 15, pp. 1-4, 2010.
- [7] William S., "Cryptography and network security: principles and practice," *Pearson Education/Prentice Hall*, 5th Edition, 2011.

- [8] Harshraj N, Aniruddha S, Shubham R. V, Rohit V, Vijay A., "A Review of Various Encryption Techniques," *International Journal of Engineering and Computer Science*, vol. 3, no. 9, pp. 8092-8096, 2014.
- [9] Gokhan D., Mehmet H., Ha_ze S_en C_ AKIR, "Increasing key space at little extra cost in RFID authentications," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 22, no. 1, pp. 155-165, 2014.
- [10] Rajdeep Bh. and Rahul H. "A Review and Comparative Analysis of Various Encryption Algorithms," *International Journal of Security and Its Applications*, vol. 9, no. 4, pp. 289-306, 2015.
- [11] Hala B., Abdul Monem S. Rahma, Hilal M., "Proposed new quantum cryptography system using quantum description techniques for generated curves," *In Proceedings of the 2009 International conference on security and management*, SAM, July 2009.
- [12] Priyadarshini P., Prashant N., Narayan D. G., Meena S. M., "A comprehensive evaluation of cryptographic algorithms: DES, 3DES, AES, RSA and Blowfish," *In Proceedings International Conference on Information Security & Privacy (ICISP2015)*, pp. 617-624, India, December 2015.
- [13] V. Kumara S., Prabhu B., "Predominance of Blowfish over triple data encryption standard symmetric key algorithm for secure integrated circuits using verilog HDL," *International Journal of Network Security & Its Applications*, vol. 9, no. 6, pp. 29-38, 2017.
- [14] S. Sweetlin, D. Mahendran, S. John Peter, "Interbit exchange and merge (IBEM) pattern of blowfish algo rithm," *International Journal of Recent Technology and Engineering*, vol. 7, no. 5, pp. 129-132, 2019.
- [15] Kirti A., Jaspal K., Harsh K., " Performance evaluation of RC6, Blow_sh, DES, IDEA, CAST-128 block ci-phers," *International Journal of Computer Applications*, vol. 68, no. 25, pp. 10-16, 2013.
- [16] Shafi'i A., Nafisat S., Mohammed A., Nadim R., Haruna Ch., and Dada G., "Development of Blowfish Encryption Scheme for Secure Data Storage in Public and Commercial Cloud Computing Environment," *2nd International Conference on Information and Communication Technology and Its Applications*, pp. 231-237, September 2018.
- [17] Smarajit Gh. and Vinod K., "Blowfish Hybridized Weighted Attribute-Based Encryption for Secure and Efficient Data Collaboration in Cloud Computing," *Applied Sciences MDPI*, pp. 1-15, 2018.
- [18] Monika A, Pradeep M., "A Modi_ed approach for symmetric key cryptography based on blowfish algorithm," *International Journal of Engineering and Advanced Technology*, vol. 1, no. 6, pp. 79-83, 2012.
- [19] Afaf M., Rehab F., "New Approach for Modifying Blowfish Algorithm by Using Multiple Keys," *International Journal of Computer Science and Network Security*, vol. 11, no. 3, pp. 21-26, 2011.
- [20] Ather M. Abbas, Abdul Monem S. Rahma, "A Modified Metrics Approach in Advanced Encryption Standard Algorithm," *Engineering and Technology Journal*, vol. 37, Part B, no. 3, pp. 86-91, 2019.
- [21] D. S. Abdul. Elminaam, H. M. Abdul Kader, M. M. Hadhoud, "Performance Evaluation of Symmetric Encryption Algorithms," *Communications of the IBIMA*. volume 8, pp. 58-64, ISSN: 1943-7765, 2009.
- [22] Narander K. and Priyanka Ch., "Performance Evaluation of Encryption/Decryption Mechanisms to Enhance Data Security," *Indian Journals of Science and Technology*, vol. 9, no. 20, pp. 2-10, 2016.
- [23] Alaa M. Riad and etc., "Evaluation of the RC4 Algorithm as a solution for Converged Networks," *Journal Of electrical engineering*, vol. 60, no.3, pp. 155-160, 2009.
- [24] V. B. Suresh, D. Antonioli and W. P. Burleson, "On-chip lightweight implementation of reduced NIST randomness test suite," *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pp. 93-98, 2013.
- [25] Mohammed F., "A Novel Encryption Method for Image Security," *Intarnational Journal of Security and Its Applications*, vol. 6, no. 1, pp. 1-8, 2012.