Cyber Security and Applications 2 (2024) 100030

Contents lists available at ScienceDirect

Cyber Security and Applications

journal homepage: http://www.keaipublishing.com/en/journals/cyber-security-and-applications/

Colour image encryption algorithm using Rubik's cube scrambling with bitmap shuffling and frame rotation

Aditi Nair[a], Diti Dalal[a], Ramchandra Mangrulkar[b,*]

[a] Department of Computer Engineering, D J Sanghvi College of Engineering, Mumbai, India
[b] Department of Information Technology, D J Sanghvi College of Engineering, Mumbai, India

ARTICLE INFO

Keywords:
Rubik'S cube scrambling
Bitplane decomposition
Frame rotation
Image encryption
Symmetric-key encryption
Cryptography

ABSTRACT

The 21st century has seen a significant increase in the usage of multimedia to transfer information. Algorithms used for the encryption of plain text are not suitable for multimedia encryption. This paper proposes a new and unique algorithm - Rubik's cube scrambling with Bitplane shuffling and Frame rotation (RBF) for the encryption of coloured images. RBF is a symmetric key algorithm that takes two 128-bit key inputs from the user. The algorithm uses Rubik's cube moves for generating scrambling sequences, followed by the shuffling of the bitplanes to generate the encrypted image. A distinctive frame rotation technique is applied to each of the bitplanes in the encryption process. The RBF algorithm is simple and straightforward to implement but produces satisfactory results. The algorithm is evaluated using a range of multimedia data. The paper also describes the various analysis tests and results of the algorithm in detail. Overall, the algorithm demonstrates its value for encrypting many types of multimedia data with practical applications.

1. Introduction

Every day, information is exchanged through multimedia in numerous areas around the world. Internet protocols transmit private data in packets along the same channels as public data, but sadly, hackers have developed methods for spying on the data being sent across the network. Security for web services becomes quite important, especially when it pertains to distributed system environments [1]. As a result, there is a continuing requirement for secure data transfer. There are several algorithms for this purpose, but with attackers attempting to intercept data, new methods that improve security are constantly encouraged. This is where encryption comes into play. The primary objective of an encryption algorithm should be to make encryption and decryption operations very fast while also withstanding any attempts to break them[2]. Encryption has proven useful in a variety of industries. Based on their industry, size, and budget, businesses invest a lot in data security because they need to safeguard their trade secrets, intellectual property, etc [3]. When the data is encrypted, the original data is scrambled to obscure the text's meaning while still leaving room for the data to be decrypted using a secret key. It is important to use encryption to make sure data is protected from tampering (data integrity), to ensure that people communicate with the people they think are communicating with (authentication), and to ensure that messages have been sent and received (non-repudiation) [4]. Stream cyphers and block cyphers are the two basic classifications of cryptographic algorithms that are used for encryption. Block cyphers work with groups of fixed-length bits known as blocks, whereas stream cyphers are designed to encrypt one bit at a time [5]. Besides that, cryptography can be categorized into two categories: symmetric encryption and asymmetric encryption.

In this paper, RBFRubik's cube scrambling with Bitplane shuffling and Frame rotation, a novel symmetric key algorithm for secure data transfer through multimedia data, which includes a three-stage encryption is introduced. The encryption and decryption processes in symmetric key cryptography employ the same key for both the sender and the receiver [6]. Due to its single key sharing scheme, symmetric key cryptosystems are substantially faster than asymmetric key cryptosystems [7]. Additionally, symmetric key encryption offers better security than asymmetric key encryption [8]. AES [9], DES [10], Blowfish, and RC5 [11] are some examples of symmetric cryptography algorithms. It is also important to know the algorithm's weaknesses and strengths along with its performance to appropriately use the algorithm for certain applications [12]. Thus various analyses of the algorithm related to performance are included in the paper.

The three stages of the proposed encryption method that will be discussed in this paper are Rubik's cube scrambling, frame rotation of bitplanes, and bitplane shuffling. The remainder of the article's structure is as follows. Existing work in the field of Rubik's cube encryption and bitplane decomposition is summarised in Section 2 of this paper. The proposed methodology is described in Section 3. A detailed explanation of the encryption and decryption methods is provided in Sections 4 and

* Corresponding author.
E-mail address: ramchandra.mangrulkar@djsce.ac.in (R. Mangrulkar).

https://doi.org/10.1016/j.csa.2023.100030
Received 3 May 2023; Received in revised form 15 July 2023; Accepted 9 August 2023
Available online 18 August 2023

5, respectively. The complete algorithm is presented in Section 6. The execution of RBF on various multimedia data, as well as the implementation and results of various analytical tests, are covered in Section 7. In Section 8, the study is concluded.

2. Related work

2.1. Rubik'S cube

This section provides an overview of a few studies in the field of cryptography that make use of the Rubik's cube principle. Here, the basic Rubik's cube notations are also explained. Official cubing competitions use computer-generated permutations to correctly scramble all of the puzzles and ensure that speedcubers have an equal opportunity [13]. Basic Notation is used to express each scrambled sequence. R signifies "turn the right face of the cube clockwise" in the Basic Notation system, and R2 stands for "flip the cube's right face 180 degrees." [14]. F stands for front, B for back, L for left, R for right, U for up, and D for down [14]. A letter by itself means a clockwise rotation of a face, while the letter followed by an apostrophe (') means a counterclockwise turn [13]. Fig. 1 Renusree Varma Mudduluri et al. developed a unique image encryption algorithm based on Rubik's cube principle. In their paper, using two randomly generated vectors, the pixels of the image are randomly shuffled in a manner similar to that of a Rubik's cube [15]. Li Zhang et al. combined the Logistic chaotic sequence and the common Rubik's Cube. Their algorithm divides an original image into a number of blocks and rotates these cubes in a manner resembling a Rubik's cube employing techniques produced by the logistic system. In the end, a new scrambled image is created using those cubes [16]. Xiao Feng et al. published a paper on how to optimize an image-scrambling method based on logistic sequence and Rubik's cube rotation. The improved algorithm resizes the original image, divides the resized image into six blocks, and then rotates the cubes in 25 steps applying 30 different rotating techniques that are managed by a chaotic system [17].

Adrian-Viorel Diaconu et al. suggested a newly proposed digital image cryptosystem that combines a digital chaotic cipher and the Rubik's cube approach. The original image is first scrambled using the Rubik's cube principle (rows and columns are alternatively shuffled and each row and column's circular-shifting direction and number of steps are derived from their intrinsic properties using different modulo operators), and then the XOR operator is applied to the rows and columns of the scrambled image using a chaos-based cipher (due to its proven diffusion properties) [18]. Khaled Loukhaoukha et al. suggested an effective image cryptosystem based on diffusion and permutation operations to strengthen the security of iris-based systems against replay threats. They first divide the original image into blocks, which are then permuted using a permutation key, and the Rubik's cube principle is then applied to each block to obtain a scrambled image [19].

2.2. BITPlane decomposition

The concept of bitplanes, as well as the various types of bitplanes, are described in this section. A summary of some of the work done in the field of image cryptography involving bitplane decomposition is stated further.

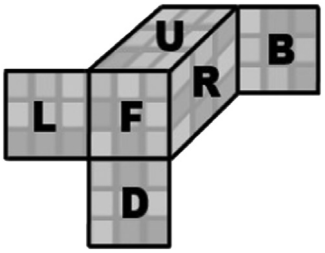


Fig. 1. Rubik's Cube Notation [13].

An image is made up of pixels of various values and is denoted in a matrix format. These values range from 0 to 255 for each colour of each pixel. Thus 1 byte i.e. 8 bits can be used to denote a pixel(in the case of grayscale images). There are various methods that can be used to generate the bitplanes, viz. Binary Bitplane Decomposition (BBD), Grey Code Bitplane Decomposition (GCBD), and Fibonacci p-code Bitplane Decomposition (FBD).

2.2.1. BBD

The Binary Bitplane Decomposition is the most conventional form of Bitplane Decomposition where a non-negative number N(1 byte) is decomposed into 8 bitplanes corresponding to its binary sequence.

$$N = \sum_{i=0}^{7} b_i 2^i = b_0 2^0 + b_1 2^1 + \ldots + b_7 2^7$$

2.2.2. GCBD

In Grey Code Bitplane Decomposition, the grey code is considered for bitplane decomposition that differs by 1 bit. This method also generates 8 bitplanes.

$$g_i = \begin{cases} b_i & \text{if } i = 7 \\ b_i \oplus b_i + 1 & \text{if } 0 \le i \le 6 \end{cases}$$

where g_i is the grey code of the corresponding binary of the i^{th} bit.

2.2.3. FBD

In Fibonacci p-code Bitplane Decomposition makes use of the p-Fibonacci Series. A natural number can be represented using p-code Fibonacci Series as follows:

$$N = \sum_{i=0}^{7} f_0 F_p(0) + f_1 F_p(1) + \ldots + f_7 F_p(7)$$

where f_i is a weight associated with $F_p(i)$ for uniqueness and

$$F_p(i) = \begin{cases} 0 & \text{if } i < 0 \\ 1 & \text{if } i = 1 \\ F_p(i-1) + f_p(i-1-p) & \text{if } i > 1 \end{cases}$$

Various values can be used for p. For example, when p = 1 the following Fibonacci series is obtained: 1,1,2,3,5,8,13,21...

The number of bitplanes generated is based on the value of p. For $p = 1$, 12 bitplanes can be generated, for $p = 2$, 14 bitplanes can be generated and so on.

Francesc Aul'-Llin'as et al. use bitplanes to create an image coding scheme as they use less computing power [20]. Few works have been done that use bitplanes for encryption, each employing a different approach. For example, Zhou et al. use bitplanes as security keys followed by bit-level scrambling for encryption of the image [21]. Similarly, Madhu and Kumari present a method for selecting a bitplane of the source image as a key for encrypting partial and all bitplanes of the original image [22]. Houas et al. decompose the bitplanes and encrypt and decrypt using a key-image obtained from the representation of the 8 bitplanes through a unique method [23]. Sun et al. propose an encryption technique that uses a random sequence to scramble the bitplane order [24]. Podesser et al. propose to only selectively encrypt certain bitplanes for data in mobile devices [25]. Liu et al. propose a novel technique for image sharing that includes compressing the bitplanes [26]. Some of the existing Encryption algorithms combine Bitplane decomposition and other techniques. The encryption method presented by Mozaffari, for example, combines bitplane decomposition with genetic algorithm that is used for permutations while Naveen and Satpute use the A5/1 Cipher with the bitplanes in their proposed image encryption algorithm [27,28]. Chen and Cheng propose a technique of reversible data hid-

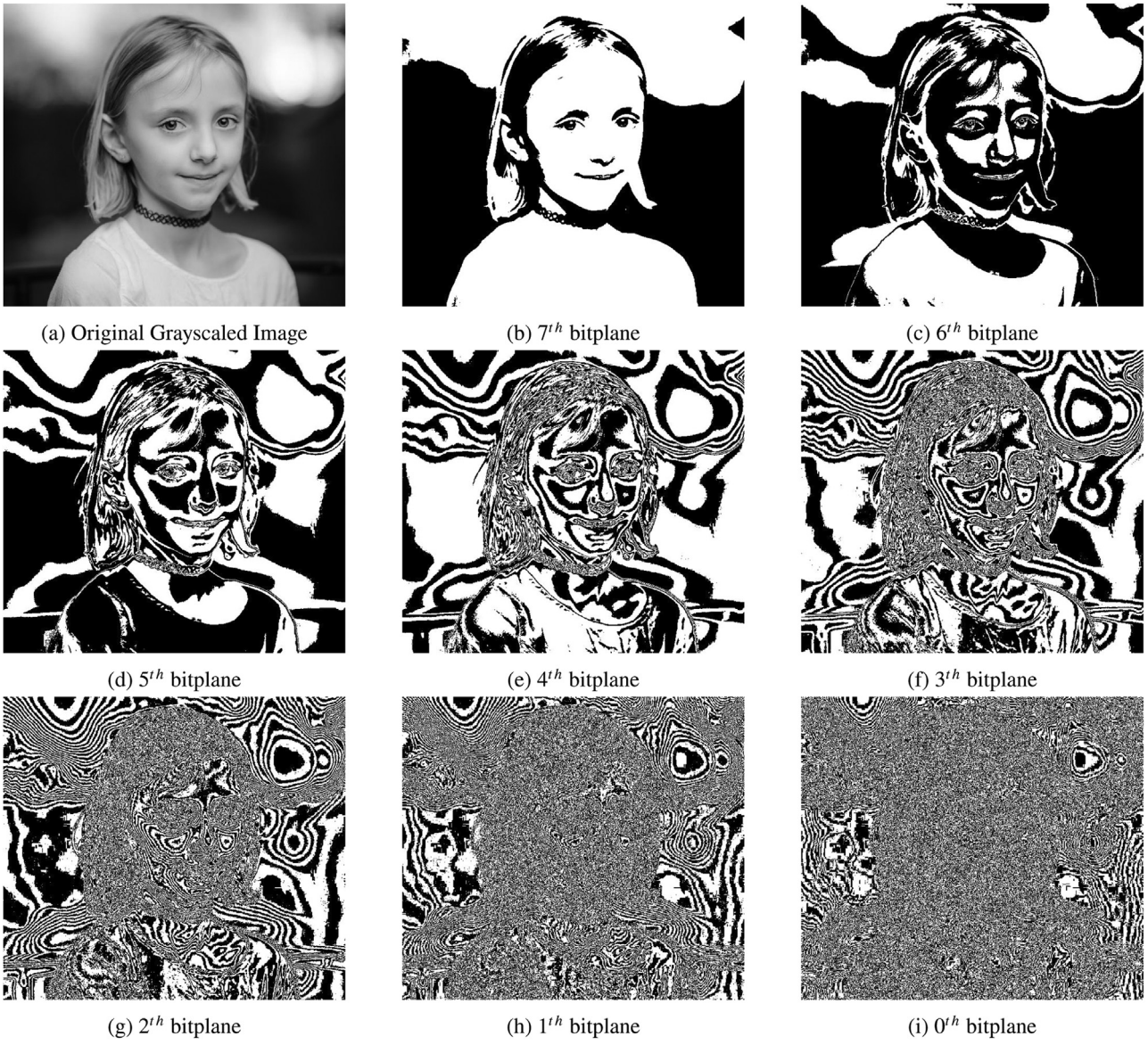


Fig. 2. Bitplanes of the Lena Image.

ing in encrypted images that preserves the correlation in the high-order bitplanes while compressing [29]. Song et al. propose an encryption algorithm that scrambles the bitplanes using chaotic logistic maps [30]. All of them employ the BBD (Binary Bitplane Decomposition) approach, while the suggested algorithms in [31,32], and [33] apply the Gray Code and Fibonacci p-code Bitplane Decomposition techniques[30].

In this paper, Binary Bitplane Decomposition is used in the encryption algorithm. Bitplane is the plane (2D matrix) generated by considering a bit's value of each pixel for an image. Thus using BBD, a grayscale image forms 8 bitplanes where bitplane 0 takes the least significant bit and bitplane 7 takes the most significant bit.

Considering the value of a pixel of a grayscaled image to be 200. Its binary representation will be 11001000. As it is observed, the value of the 3rd, 6th, and 7th bits are 1 while the value of the 0th,1st, 2nd, 4th, and 5th bits are 0. 8 bitplanes can be formed using these bit values, each replacing its pixel value for the corresponding bitplane. 0th bitplane will take the value of bit0, 1st of bit1, 2nd of bit2, and so on. Fig. 2 depicts the various 8-bitplanes that can be formed using the BBD (Binary Bitplane Decomposition) method for the grayscale Lena image.

As seen from Fig. 2, higher bitplanes that are formed using the MSB contain more information about the image than the other bitplanes.

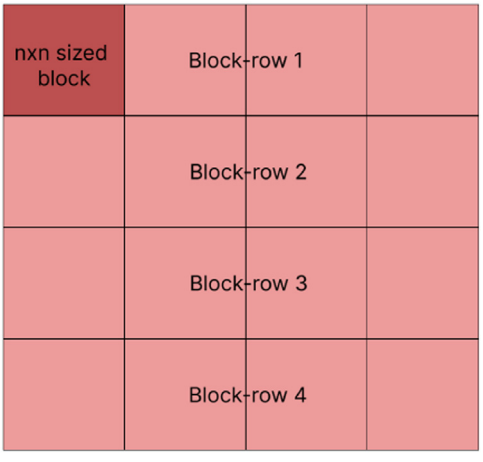


Fig. 3. Encryption-Decryption Flow.

About 94% percent of the image data is present in the first four bit-planes [31].

A coloured image consists of 3 values for every pixel, each for blue, green, and red colours, and each of these values ranges from 0 to 255. Thus a coloured image can be denoted in a 3D matrix, i.e. 3*2D matrices for every r,g, and b colour. The values of the pixel determine the colour and intensity of the image collectively. Hence a coloured image can be decomposed into 8*3 bitplanes i.e. 8 planes for each r,g, and b colour.

	00	01	10	11
00	L	R	U	D
01	F	L'	R'	U'
10	D'	F'	L2	R2
11	U2	D2	F2	L

Fig. 4. Encryption Matrix.

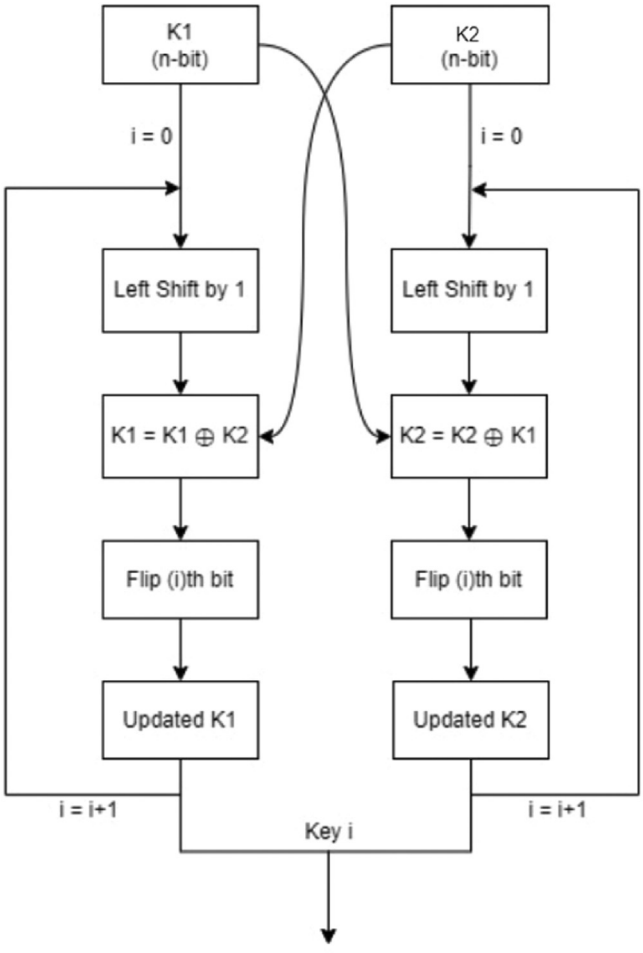


Fig. 5. Initial Matrix.

3. Proposed methodology

A novel and innovative technique for multimedia cryptography - RBF, is described in this section. This method takes two 128-bit unique keys and a coloured image as input to generate its corresponding encrypted image output. The encryption and decryption algorithms have 3 main stages as shown in Fig. 3.

The first step i.e. Rubik's Cube Scrambling uses the two keys to generate a scrambling sequence based on the techniques used in solving a Rubik's cube as mentioned in Section 4.1. In the second step, the image is decomposed into 24 bitplanes as described in Section 4.2, and frame rotation is applied for each of the bitplanes using the input keys. The third step involves scrambling the bitplanes in a specific sequence and integrating them to form the encrypted image. During the decryption procedure, the receiver uses the same two 128-bit keys to decrypt the received encrypted image.

4. Encryption

4.1. Rubik'S cube scrambling

This section provides a detailed explanation of the algorithm's first stage, i.e. the Rubik's cube scrambling. The fundamental idea behind this approach is to resemble the image as a Rubik's cube's face and then apply the scrambling operations generally used to scramble the cube. Here, however, just one face is taken into account rather than the complete cube. An image in the digital format is represented by a matrix. Therefore, when dealing with images for encryption, one would actually be dealing with a matrix of pixel values.

In this step, the image matrix is divided into n x n sized blocks. Each block represents the face of a n x n Rubik's cube. The scrambling operations will then be applied to these blocks based on the scrambling sequence generated. L, R, U, D, F, L', R', U', D', F', L2, R2, U2, D2, F2 are the basic operations that the encryption scrambling sequence is

(a) (b)

Fig. 6.

	0	1	2	3	4	5	6	7	8	9
0	20	41	35	98	125	10	5	61	44	32
1	126	42	73	64	11	66	75	49	12	55
2	23	56	48	101	56	45	88	41	65	45
3	70	63	12	64	75	69	85	36	45	35
4	42	65	20	73	48	52	66	30	16	71
5	145	113	87	65	34	22	56	62	32	54
6	200	145	180	64	44	41	123	150	206	120
7	212	86	34	111	146	137	37	21	69	47
8	128	60	34	96	56	60	26	34	80	46
9	125	100	145	122	24	61	40	94	193	45

Fig. 7. Key Update Logic.

composed of. A number attached with these operations will indicate the column or row number of the block that the operation is to be performed on (This number renders meaningless when the operation is F, F', or F2). Then, in the direction indicated by the letter in the scrambling sequence, the corresponding row or column is circularly shifted. A 2D matrix was formulated containing all the operations and there is a different matrix for encryption and decryption. Fig. 4.

The keys K1 and K2 are used to generate the scrambling sequence for each block-row of the image. The two keys are divided into sub-sequences of four digits. Each 4-digit sub-sequence from K1 gives the number indicating the row/column index. The corresponding 4-bit sub-sequence from K2 represents the index of the 2D operation matrix, resulting in the operation to be performed.

For example, when

$K1 = 01010100001000111 \ldots\ldots\ldots.00110011$

$K2 = 11010100011110100 \ldots\ldots\ldots.10111100$

First 4 bits of K1 $= 0101$
$= 5(decimal)$
First 4 bits of K2 $= 1101$
$= (row)11$ and $(col)01$

Therefore, the first operation $= 5D2$

Thus a sequence of 32 operations is obtained from the 2 keys. This sequence is used for scrambling all the blocks in a row. Next, the Key

	0	1	2	3	4	5	6	7	8	9
0	20	41	35	98	125	10	40	61	44	32
1	126	42	73	64	11	66	5	49	12	55
2	23	56	48	101	56	45	75	41	65	45
3	70	63	12	64	75	69	88	36	45	35
4	42	65	20	73	48	52	85	30	16	71
5	145	113	87	65	34	22	66	62	32	54
6	200	145	180	64	44	41	56	150	206	120
7	212	86	34	111	146	137	123	21	69	47
8	128	60	34	96	56	60	37	34	80	46
9	125	100	145	122	24	61	26	94	193	45

(a) After implementing "4R'"

	0	1	2	3	4	5	6	7	8	9
0	20	41	35	98	125	10	40	61	44	32
1	126	42	73	64	11	66	5	49	12	55
2	23	56	48	101	56	45	75	41	65	45
3	35	70	63	12	64	75	69	88	36	45
4	42	65	20	73	48	52	85	30	16	71
5	145	113	87	65	34	22	66	62	32	54
6	200	145	180	64	44	41	56	150	206	120
7	212	86	34	111	146	137	123	21	69	47
8	128	60	34	96	56	60	37	34	80	46
9	125	100	145	122	24	61	26	94	193	45

(b) After implementing "7D"

	0	1	2	3	4	5	6	7	8	9
0	20	100	35	98	125	10	40	61	44	32
1	126	41	73	64	11	66	5	49	12	55
2	23	42	48	101	56	45	75	41	65	45
3	35	56	63	12	64	75	69	88	36	45
4	42	70	20	73	48	52	85	30	16	71
5	145	65	87	65	34	22	66	62	32	54
6	200	113	180	64	44	41	56	150	206	120
7	212	145	34	111	146	137	123	21	69	47
8	128	86	34	96	56	60	37	34	80	46
9	125	60	145	122	24	61	26	94	193	45

(c) After implementing "2L"

	0	1	2	3	4	5	6	7	8	9
0	125	128	212	200	145	42	35	23	126	20
1	60	86	145	113	65	70	56	42	41	100
2	145	34	34	180	87	20	63	48	73	35
3	122	96	111	64	65	73	12	101	64	98
4	24	56	146	44	34	48	64	56	11	125
5	61	60	137	41	22	52	75	45	66	10
6	26	37	123	56	66	85	69	75	5	40
7	94	34	21	150	62	30	88	41	49	61
8	193	80	69	206	32	16	36	65	12	44
9	45	46	47	120	54	71	45	45	55	32

(d) After implementing "9F"

	0	1	2	3	4	5	6	7	8	9
0	125	128	212	200	145	42	35	23	126	20
1	60	86	145	113	65	70	56	42	41	100
2	145	34	34	180	87	20	63	48	73	35
3	122	96	111	64	65	73	12	101	64	98
4	24	56	146	44	34	48	64	56	11	125
5	61	60	137	41	22	52	75	45	66	10
6	26	37	123	56	66	85	69	75	5	40
7	49	61	94	34	21	150	62	30	88	41
8	193	80	69	206	32	16	36	65	12	44
9	45	46	47	120	54	71	45	45	55	32

(e) After implementing "3D2"

	0	1	2	3	4	5	6	7	8	9
0	60	128	212	200	145	42	35	23	126	20
1	145	86	145	113	65	70	56	42	41	100
2	122	34	34	180	87	20	63	48	73	35
3	24	96	111	64	65	73	12	101	64	98
4	61	56	146	44	34	48	64	56	11	125
5	26	60	137	41	22	52	75	45	66	10
6	49	37	123	56	66	85	69	75	5	40
7	193	61	94	34	21	150	62	30	88	41
8	45	80	69	206	32	16	36	65	12	44
9	125	46	47	120	54	71	45	45	55	32

(f) After implementing "1L'"

	0	1	2	3	4	5	6	7	8	9
0	60	128	212	200	145	42	35	23	126	20
1	145	86	145	113	65	70	56	42	41	100
2	122	34	34	180	87	20	63	48	73	35
3	24	96	111	64	65	73	12	101	64	98
4	61	56	146	44	34	48	64	56	11	125
5	26	60	137	41	22	52	75	45	66	10
6	37	123	56	66	85	69	75	5	40	49
7	193	61	94	34	21	150	62	30	88	41
8	45	80	69	206	32	16	36	65	12	44
9	125	46	47	120	54	71	45	45	55	32

(g) After implementing "7U"

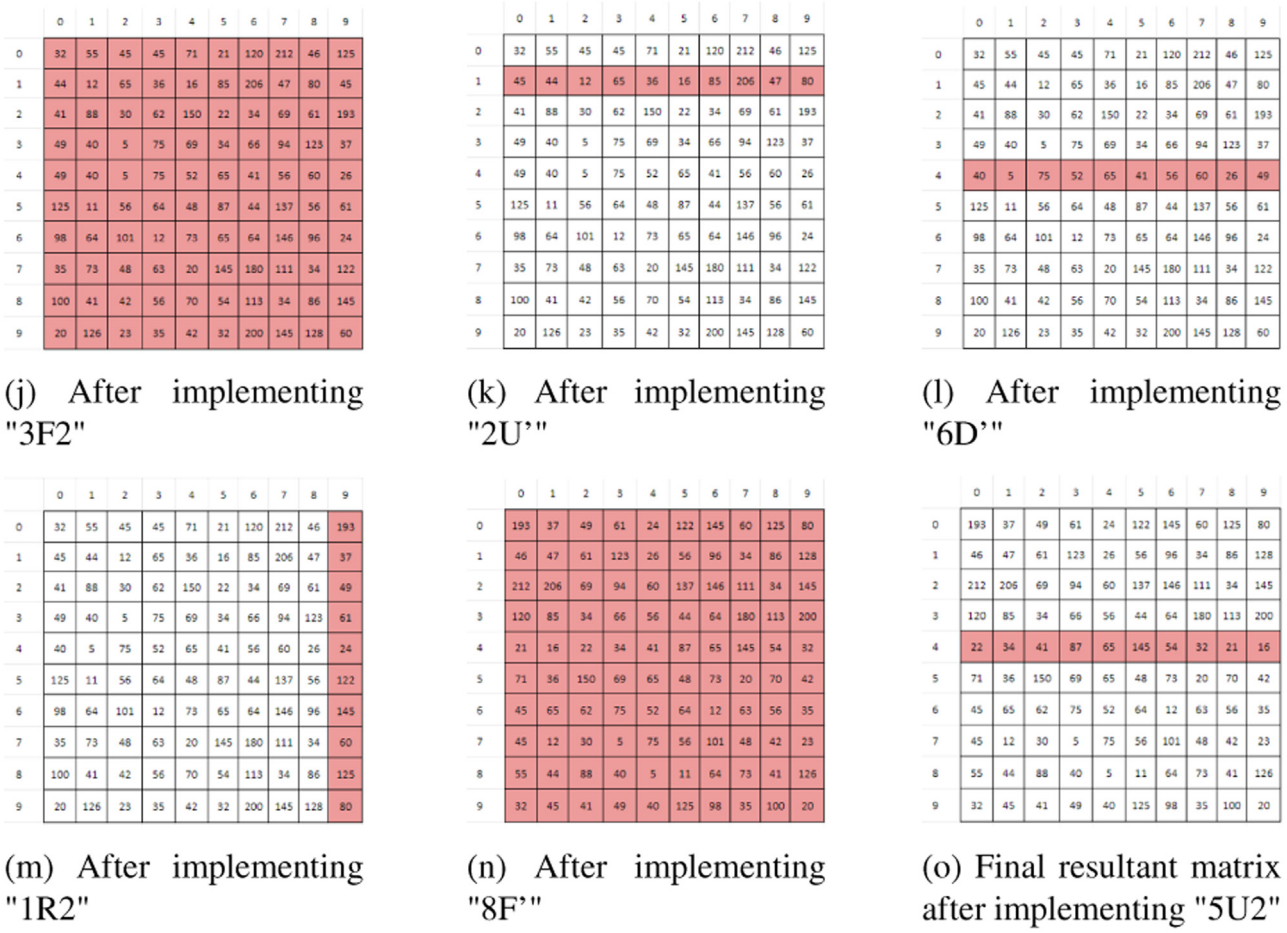
	0	1	2	3	4	5	6	7	8	9
0	60	128	212	200	32	42	35	23	126	20
1	145	86	145	113	54	70	56	42	41	100
2	122	34	34	180	145	20	63	48	73	35
3	24	96	111	64	65	73	12	101	64	98
4	61	56	146	44	87	48	64	56	11	125
5	26	60	137	41	65	52	75	45	66	10
6	37	123	56	66	34	69	75	5	40	49
7	193	61	94	34	22	150	62	30	88	41
8	45	80	69	206	85	16	36	65	12	44
9	125	46	47	120	21	71	45	45	55	32

(h) After implementing "5L2"

	0	1	2	3	4	5	6	7	8	9
0	60	128	145	200	32	42	35	23	126	20
1	145	86	34	113	54	70	56	42	41	100
2	122	34	111	180	145	20	63	48	73	35
3	24	96	146	64	65	73	12	101	64	98
4	61	56	137	44	87	48	64	56	11	125
5	26	60	56	41	65	52	75	45	66	10
6	37	123	94	66	34	69	75	5	40	49
7	193	61	69	34	22	150	62	30	88	41
8	45	80	47	206	85	16	36	65	12	44
9	125	46	212	120	21	71	45	45	55	32

(i) After implementing "8R"

Fig. 8. Rubik's cube scrambling (a) Original Image (b) Resultant Image.



(j) After implementing "3F2"

(k) After implementing "2U'"

(l) After implementing "6D'"

(m) After implementing "1R2"

(n) After implementing "8F'"

(o) Final resultant matrix after implementing "5U2"

Fig. 8. Continued

Fig. 9. Frame Rotation.

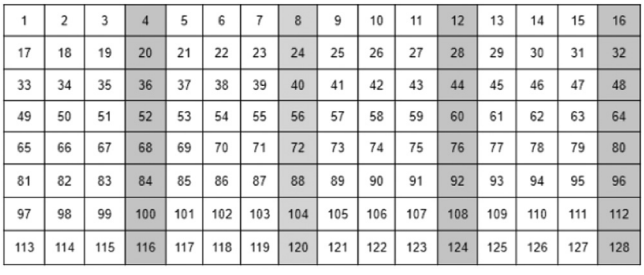


Fig. 10. Reduction Logic.

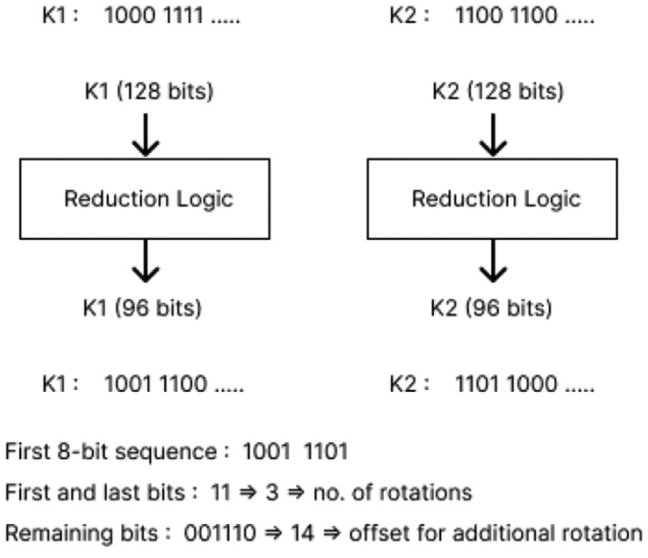


Fig. 11. Sequence Generation.

Update Logic is applied. The two updated keys of a row are similarly used to generate sequences for the next row.

4.1.1. Implementation of the Rubik's cube encryption sequence on a 10x10

Let the encryption sequence be EncSeq = [4R', 7D, 2L, 9F, 3D2, 1L', 7U, 5L2, 8R, 3F2, 2U', 6D', 1R2, 8F', 5U2]

The 10x10 block is displayed in Fig. 5 EncSeq[0] = 4R', so the fourth column from the right is located and rotated downwards by one cell. The resultant matrix can be seen in Fig. 6 a

EncSeq[1] = 7D, the seventh row from the bottom is located and right-shifted by one as shown in Fig. 6 b

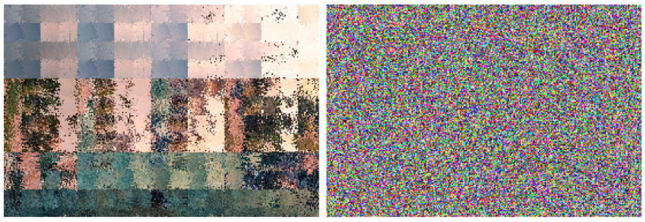


Fig. 12. (a) Resultant image after Rubik's cube scrambling (b) Final Encrypted Image.

	00	01	10	11
00	L'	R'	U'	D'
01	F'	L	R	U
10	D	F	L2'	R2'
11	U2'	D2'	F2'	L'

Fig. 13. Decryption Matrix.

EncSeq[2] = 2L, the second column from the left is rotated downwards by one cell. The resultant matrix is shown in Fig. 6 c

EncSeq[3] = 9F, when the operation is F, the integer renders meaningless. The matrix simply gets rotated clockwise as displayed in Fig. 6 d.

EncSeq[4] = 3D2, the third row from the bottom is right-shifted by two cells. The resultant matrix is displayed in Fig. 6 e

EncSeq[5] = 1L', the first column from the left is rotated upwards by one and the result is shown in Fig. 6 f

EncSeq[6] = 7U, the seventh row from the top will be left shifted by one. The resultant matrix is displayed in Fig. 6 g

EncSeq[7] = 5L2, the fifth column from the left will be rotated downwards by two cells as displayed in Fig. 6 h

EncSeq[8] = 8R, the eighth column from the right is rotated upwards by one. The resultant matrix is shown in Fig. 6 i

EncSeq[9] = 3F2, the integer before the operation F' renders meaningless. The matrix is simply rotated twice in clockwise direction and displayed in Fig. 6 j

EncSeq[10] = 2U', the second row from the top gets right-shifted by one as displayed in Fig. 6 k

EncSeq[11] = 6D', the sixth row from the bottom gets left-shifted by one and the resultant matrix is shown in Fig. 6 l

EncSeq[12] = 1R2, the first column from the right gets rotated upwards by two cells. The result is displayed in Fig. 6 m

EncSeq[13] = 8F', the integer in this case renders meaningless. The matrix is simply rotated counterclockwise as displayed in Fig. 6 n

EncSeq[14] = 5U2, the fifth row from the top will be left-shifted by two cells. The resultant matrix is shown in Fig. 6 o

Thus, the final resultant matrix is obtained after undergoing a set of operations from the EncSeq.

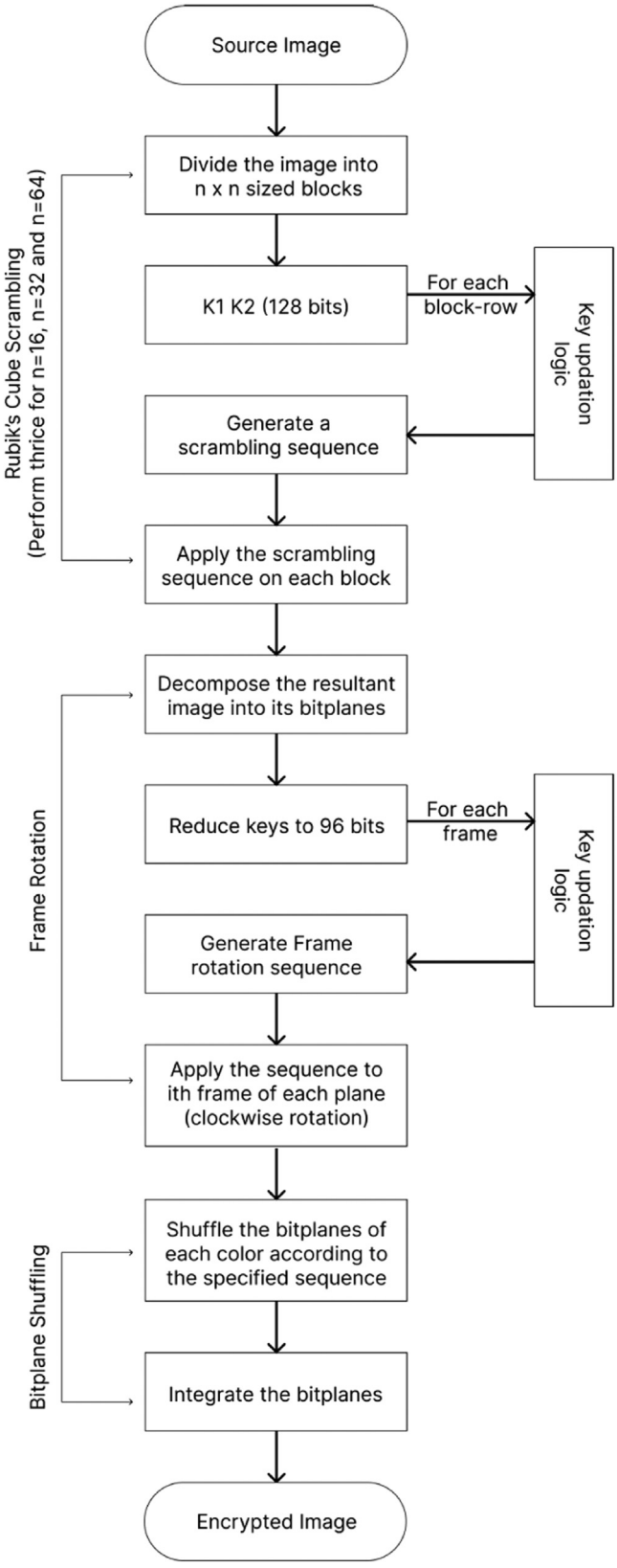


Fig. 14. Diagrammatic representation of the encryption algorithm.

4.1.2. Key updation

This section explains the key updating procedure, which helps generate new set of keys for each row. For the following row, the keys are left shifted by 1 bit and xored with the other original key (updated K1 is xored with K2 and vice versa). Then the bit at the location of the key number, i.e. the i^{th} bit is flipped. This operation ensures that no sequence is repeated. The logical flow is shown in Fig. 7. The remaining block at the end of the row or column having a rectangular size also goes through all the operations except for the F (face rotation) operation. To increase the complexity, the entire process is applied to the image thrice i.e. for three rounds. The block size in the first round is 16, 32 for the next round, and 64 for the last round. Fig. 8 a shows the original source image and Fig. 8 b shows the image after it has been scrambled using a Rubik's cube.

4.2. Bitplane shuffling and frame rotation

This section discusses the second and last stage of the RBF algorithm, which involves first separating the scrambled image into its bitplanes. Frame rotation is performed on these bitplanes, followed by bitplane shuffling.

4.2.1. Generating bitplanes

An image is represented as a matrix and is made up of pixels with different values. For each pixel of each colour, these values fall between 0 and 255. The planes (2D matrix) created by taking into account the bit value of each pixel in a picture are known as "bitplanes." As mentioned in section 2.2.1, bitplanes are generated using the BBD (Binary Bitplane Decomposition). Thus in this step, 24 (8*3) bitplanes are generated for coloured images. (0–7 for blue, 8–15 for green and 16–23 for red) for blue:

$$img[i,j,0] = 2^7 * b0[i,j] + 2^6 * b1[i,j] + 2^5 * b2[i,j] + 2^4 * b3[i,j] + 2^3 * b4[i,j] + 2^2 * b5[i,j] + 2^1 * b6[i,j] + 2^0 * b7[i,j]$$

Similarly, the bitplanes are generated for green and red components. The sequence of colour components may change based on the storing technique such as JPEG, PNG, etc.

4.2.2. Bitplane frame rotation

This stage proves to be quite helpful in order to boost the unpredictability and complexity of the encryption process. Using the original keys frame rotation is applied on each bit plane separately. A frame is considered as the outline 1-pixel border square/rectangle of a plane at a particular distance from the edge. The frames in each bitplane are then rotated by a certain count in a circular manner in the clockwise direction.

Two methods are applied to an image in this step - frame rotation and a certain shift. First, the rotation is applied followed by an offset of a few units. For i^{th} frame i.e. outline box of pixels at a distance of i from the end, the two steps are successively applied. As shown in Fig. 9 a 5x5 image and its 0^{th} frame are considered. The rotation of 1 followed by an offset of 2 units is applied in clockwise direction. The two 128-bit keys K1 and K2 are used to determine the count with which the frames in each bitplane need to be rotated. First, a reduction logic is applied to both keys, reducing them to 96 bits each. Then random sequences are generated using the reduced keys for each frame of a bitplane. This is followed by updating the reduced keys to generate new random sequences.

4.2.3. Reduction logic

From the 128-bit key, every 4th bit is discarded to produce a 96-bit key. That is bit positions 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92, 96, 100, 104, 108, 112, 116, 120, 124, 128 are discarded as shown in Fig. 10.

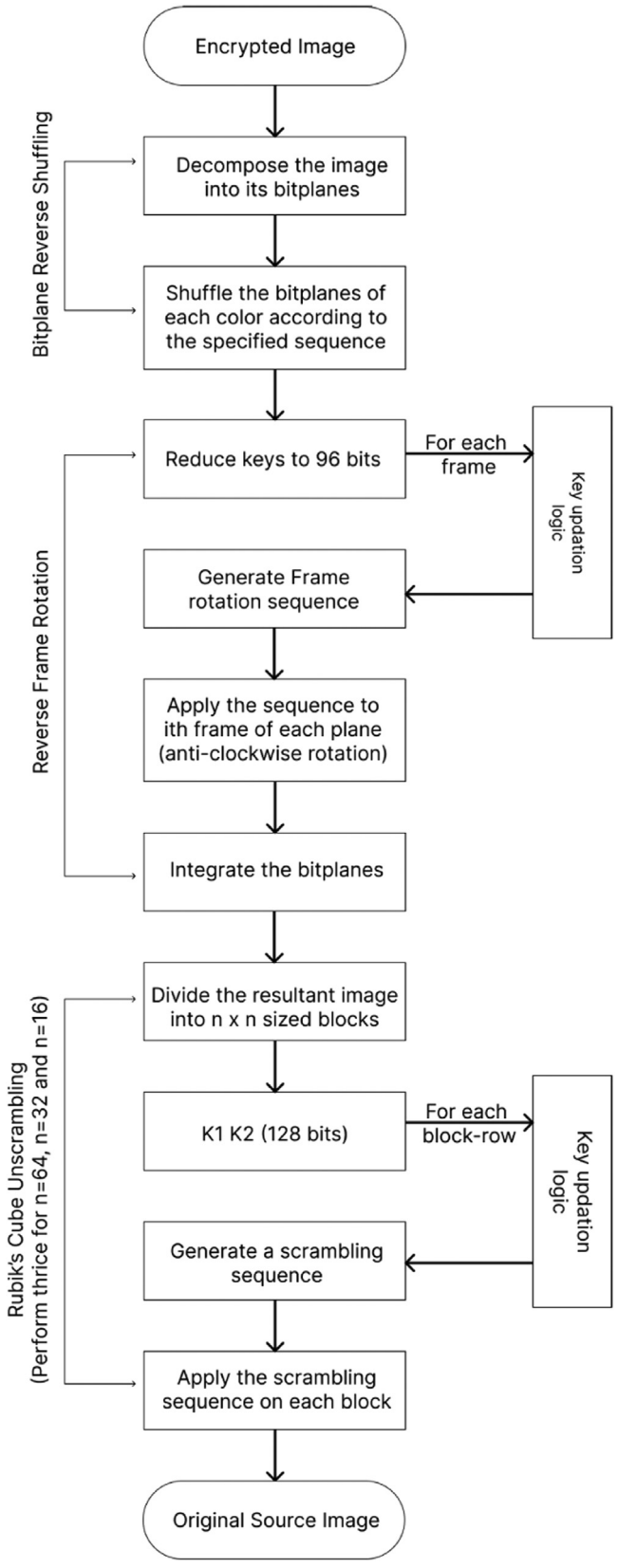


Fig. 15. Results of encryption and decryption of various images using the RBF algorithm.

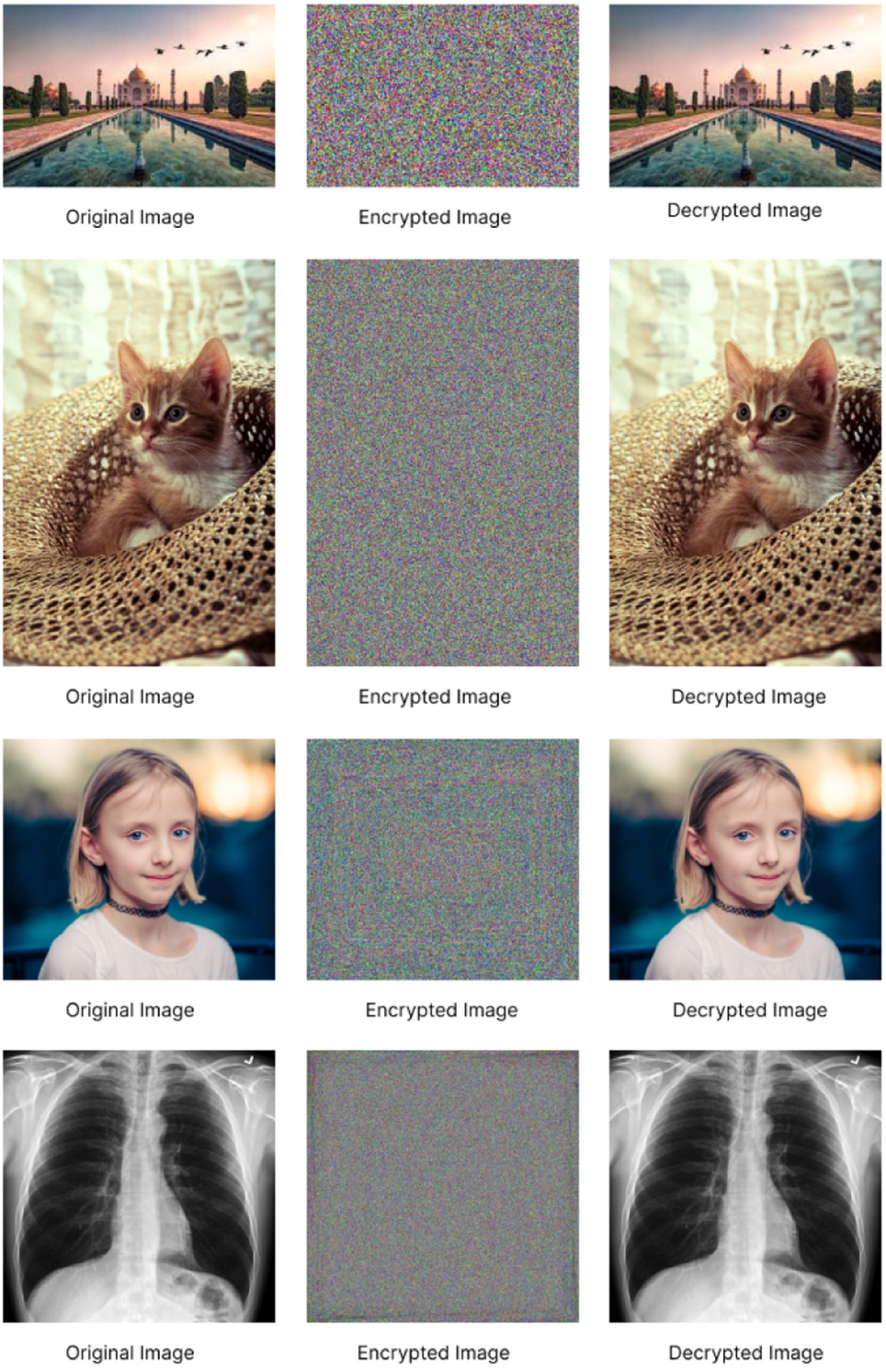


Fig. 16. Total time vs Image size.

4.2.4. Random sequence generation

The two 96-bit keys are used to generate 24 8-bit sequences corresponding to the 24-bitplanes for each frame. To create an 8-bit sequence, 4 bits from each of the keys are considered. For the first bitplane, the first 4 bits of K1 and K2 are considered, for the second bitplane, bits 4–8 of K1 and K2 are considered, and so on. The sequence is generated by appending the 4 bits of K2 after the 4 bits of K1. The first and last bits of this sequence is used to determine the number of rotations to be performed. And the 6 middle bits give the offset value as shown in Fig. 11.

The keys are updated using the same logic as mentioned in Section 4.1.2.

4.2.5. Bitplane shuffling

The frame rotation step is applied to all the 24 bitplanes. The order of the bitplanes is changed to generate a better-encrypted image when

integrated. The bitplanes for each colour are shuffled as follows:

blue: initial order : 0,1,2,3,4,5,6,7
shuffled order: 7,6,5,4,3,2,1,0
green: initial order : 8,9,10,11,12,13,14,15
shuffled order: 15,14,13,12,11,10,9,8
red: initial order : 16,17,18,19,20,21,22,23
shuffled order: 23,22,21,20,19,18,17,16

4.2.6. Generating the encrypted image

The bitplanes after being shuffled in the order mentioned before are integrated into an image. The bitplanes are integrated into an image by reversing a similar decomposition procedure for each colour component. for blue:

$$enc[i,j,0] = 2^7 * b7[i,j] + 2^6 * b6[i,j] + 2^5 * b5[i,j] + 2^4 * b4[i,j] + 2^3 * b3[i,j] + 2^2 * b2[i,j] + 2^1 * b1[i,j] + 2^0 * b0[i,j]$$

Similarly values for red and green components are calculated. These aggregated 2D matrices for each colour components are then stacked over each other to form a 3D matrix i.e. the Encrypted Image.

Fig. 12 a depicts the image formed after stage 1(Rubik's cube scrambling) and Fig. 12 b shows the final encrypted image after stage 2(Frame Rotation and Bitplane shuffling).

5. Decryption

The process of reconstructing the original image from the encrypted image using the same two 128-bit keys is briefly described in this section.

5.1. Reverse bitplane shuffling and frame rotation

5.1.1. Generating bitplanes

Same as Section 4.2.1

5.1.2. Bitplane reverse shuffling

The shuffled bitplanes are reshuffled to the original order. This step is similar to the step mentioned in Section 4.2.5. The bitplanes for each colour are reverse shuffled as follows:

blue: shuffled order : 0,1,2,3,4,5,6,7
final order : 7,6,5,4,3,2,1,0
green: shuffled order : 8,9,10,11,12,13,14,15
final order: 15,14,13,12,11,10,9,8
red: shuffled order : 16,17,18,19,20,21,22,23
final order: 23,22,21,20,19,18,17,16

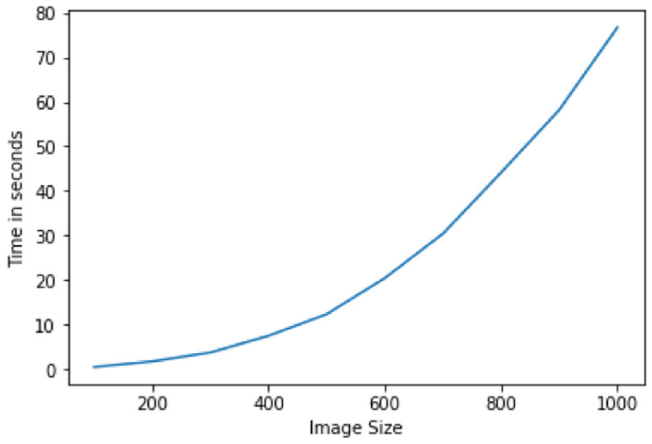


Fig. 17. Key Sensitivity Test (a) Original Image (b) Encrypted Image (with k1 and k2) (c) Encrypted Image (with k3 and k4) (d) Difference between Figs. 17 b and 17 c.

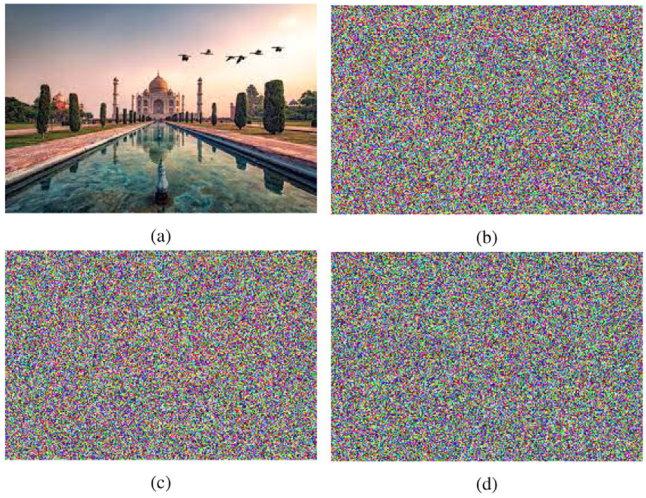


Fig. 18. Histogram Analysis (a) Original Image (b) Encrypted Image (c) Histogram of the source image (d) Histogram of the encrypted image.

5.1.3. Reverse frame rotation

For each bitplane, 8-bit sequences are generated, similar to encryption using the two 128-bit key inputs and the same logic for reduction and updating the keys. Here, first the frames are shifted in a circular manner by the offset in anti-clockwise direction and then perform the derived number of rotations, also in anti-clockwise direction.

The bitplanes are then integrated and sent to the next step.

5.2. Rubik'S cube unscrambling

The image received after the above processing is again divided into n x n sized blocks. The two 128-bit keys K1 and K2 are used to get started with the first block-row. They generate a sequence of 32 operations with the help of the 2D operations matrix for decryption. The same logic is applied for updating the keys Fig. 13.

This entire process is applied for three rounds. In the first round, the block size is 64x64, the block size in the next image is 32x32 and for the last round, the image is divided into 16x16 sized blocks. At the end of this step, the original image i.e. the decrypted image is retrieved.

6. Algorithms

For this version of the algorithm, perform the Rubik's cube encryption for 3 rounds with 3 different values of n:

1st round: n = 16
2nd round: n = 32
3rd round: n = 64

The diagrammatic representation of the encryption process can be seen in Fig. 14.

7. Results and discussion

In order to test the algorithm's performance on images from various categories, it is applied to a range of images. The RBF algorithm was implemented on pictures of monuments, portraits, animals, and even x-rays. This section discusses the results of these implementations. This section also includes demonstrations of several analysis tests. The results demonstrate how the proposed encryption technique is applicable across a wide range of domains. The source image, the resultant encrypted image and the final decrypted picture of various different images are displayed in Fig. 15.

It can be seen that the encrypted image successfully converts the original source image into a random assortment of pixels, rendering it

Algorithm 1: Rubik's Encryption.

```
Data: img, encMatrix, n, k1, k2
k1Reverse = reverse(k1)
K2Reverse = reverse(k2)
sequence = sequenceGeneration(encMatrix, k1, k2)
sequence1 = sequenceGeneration(encMatrix, k1Reverse,
k2Reverse)
foreach nxn sized block row do
    foreach nxn sized block do
        blockMatrix = values of the nxn sized block
        sequenceCalculation(sequence, blockMatrix)
        sequenceCalculation(seq1, blockMatrix)Update the
        corresponding block in the img with the values of the
        modified blockMatrix
    end
    newK1 = keyUpdation(k1)
    newK2 = keyUpdation(k2)
    newK1Reverse = reverse(newK1)
    newK2Reverse = reverse(newK2)newSequence =
    sequenceGeneration(encMatrix, newK1, newK2)
    newSequence1 = sequenceGeneration(encMatrix,
    newK1Reverse, newK2Reverse)
end
return img
```

Algorithm 2: sequenceGeneration.

```
Data: s, k1, k2
                                    ▷ s - list of possible operations
sequence = []
for i from 0 to 128 step 4 do
    num = substring of k1 from i to i + 3
    num = int(num) + 1operationIndex = substring of k2 from i
    to i + 3
    operationIndex = int(operationIndex)
    operation = s[operationIndex]append (str(num) +
    operationIndex) to sequence
end
return sequence
```

Algorithm 3: sequenceCalculation.

```
Data: sequence, blockMatrix
foreach term in sequence do
    row/column index = integer part of termoperation =
    remaining part of termapply the operation on the row/column
    of the block_matrix
end
return blockMatrix
```

impossible to identify the original image in any way while maintaining the image's integrity. To accurately evaluate the algorithm's performance, a number of additional analyses are carried out. Time complexity, key sensitivity analysis, histogram analysis, and recoverability analysis are all used to evaluate the algorithm.

7.1. Time complexity

The time complexity of an algorithm is primarily the total time required for the encryption algorithm to encrypt an image. The total time increases as the size of the image gets larger. Thus the time complexity is a function of the size of the image. The following results display the time recorded for the encryption process for various images.

Algorithm 4: Bitplane Shuffling.

```
Data: image, k1, k2
bp = generate bitplanes from image
reducedK1, reducedK2 = keyReduction(k1, k2)
newBp = frameRotation(bp, reducedK1, reducedK2)
//for each color in (R,G,B)
for i in [1,2,3] do
    bp[0*i] = new_bp[7*i]
    bp[1*i] = new_bp[6*i]
    bp[2*i] = new_bp[5*i]
    bp[3*i] = new_bp[4*i]
    bp[4*i] = new_bp[3*i]
    bp[5*i] = new_bp[2*i]
    bp[6*i] = new_bp[1*i]
    bp[7*i] = new_bp[0*i]
end
image = generate image from bitplanes bp
return image
```

Algorithm 5: frameRotation.

```
Data: bp, k1, k2
newK1, newK2 = k1, k2
foreach frame in image do
    i = frame number
    newK1, newK2 = keyUpdation(k1, k2, newK1, newK2, i)
    foreach bitplane in bp do
        j = bitplane number //0–24
        noOfRotations, offset = frameRotationSeqGeneration(k1,
        k2)x = noOfRotations * imgDimensions + offset
        newBp[j] = shift the contents of ith frame of bp[j] by x
        places
    end
end
return newBp
```

Algorithm 6: frameRotationSeqGeneration.

```
Data: k1, k2
newK1, newK2 = k1, k2
for i from 0 to len(k1) in steps of 4 do
    noOfRotations = int(k1[i] + k2[i+3])
    offset = int((substring of k1 from i+1 to i+3)+(substring of
    k2 from i to i+2))
end
return noOfRotations, offset
```

Algorithm 7: keyUpdation.

```
Data: k1Original, k2Original, k1, k2, i
left shift k1 by 1
left shift k2 by 1k1 = k1 exor k2Original
k2 = k2 exor k1Originalflip digit at ith position in k1 and
k2return k1, k2
```

Algorithm 8: keyReduction.

```
Data: k1, k2
foreach char in k1 and k2 do
    if the position of char is multiple of 4 then
        discard the char
    else
        append the char to newK1 and newK2
    end
end
return newK1, newK2
```

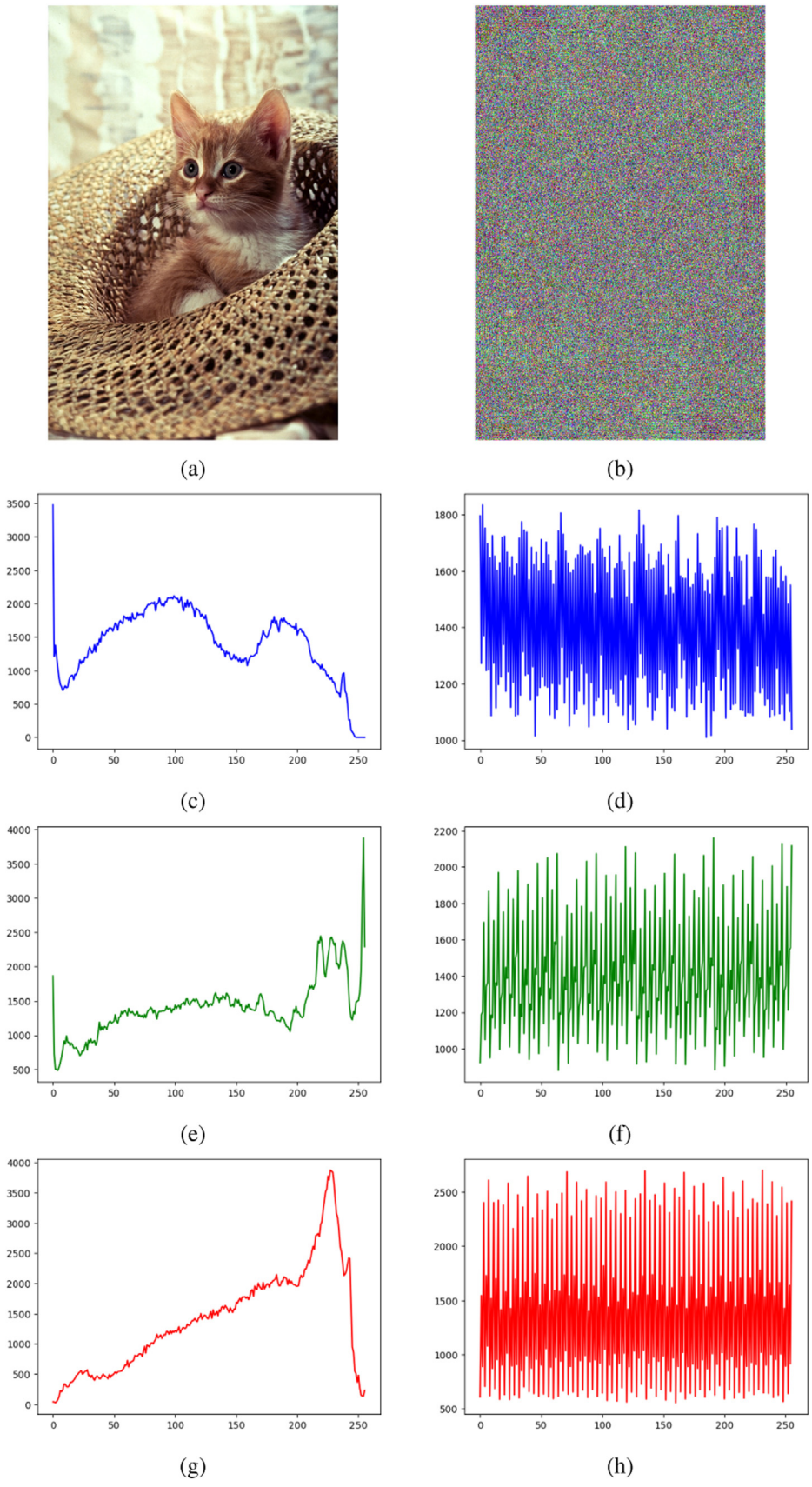


Fig. 19. Histogram Analysis (a) Original Image (b) Encrypted Image (c) Histogram of the source image (d) Histogram of the encrypted image.

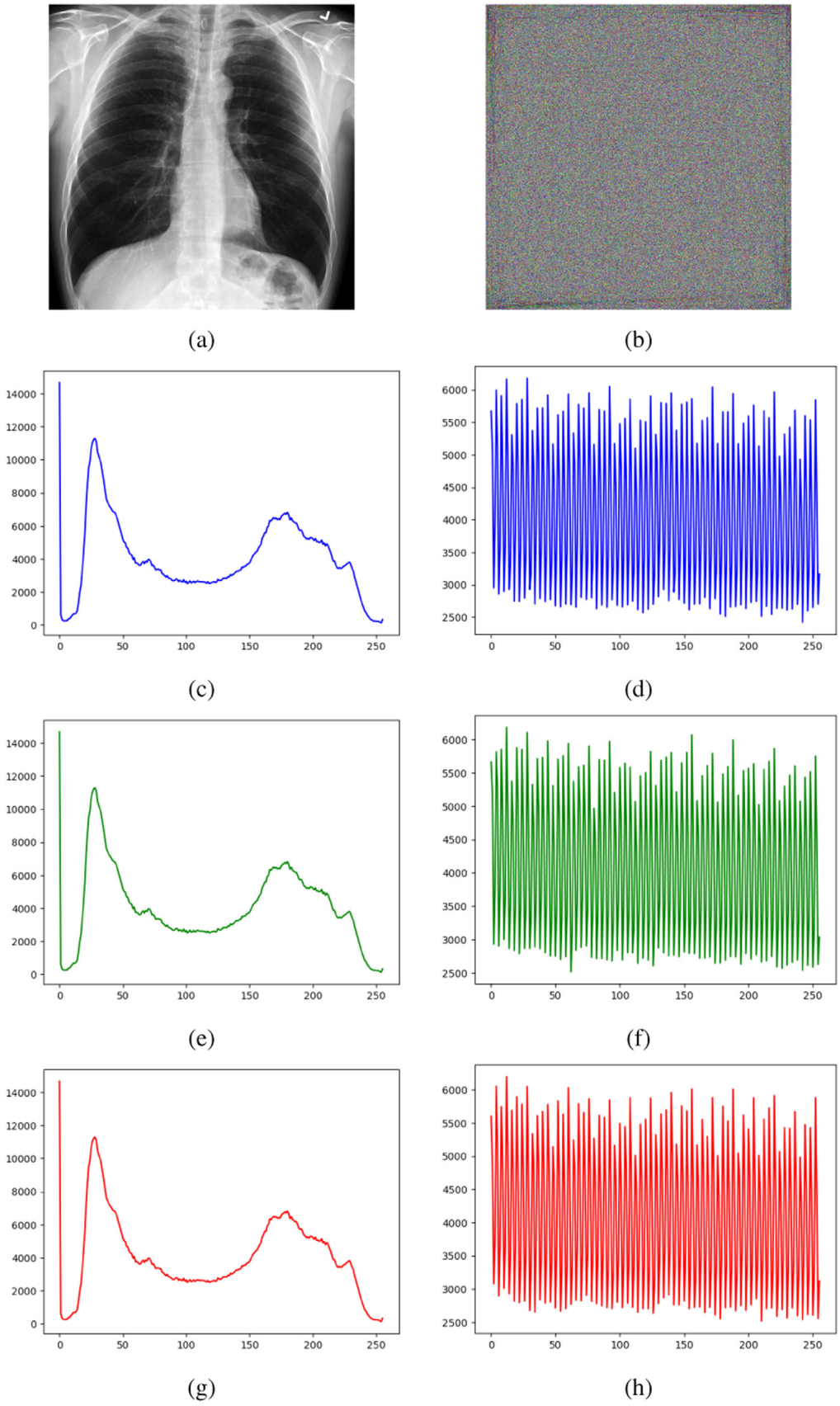


Fig. 20. Recoverability (a) Original Image (b) Decrypted Image (c) Difference between 20 a and 20 b.

Image	Image Dimensions (pixels)	Time Required (seconds)
monument	184 x 274	02.25
Lena	512 x 512	14.24
cat	733 x 490	20.99
girl portrait	576 x 650	25.80
x-ray	1024 x 1024	96.06

The graphical representation of the total time vs the size of the image is shown in Fig. 16.

7.2. Key sensitivity analysis

In this section, the aforementioned algorithm's sensitivity to changes in the user's initial two 128-bit keys is put to the test. The source image is encrypted with two different key combinations to get the resultant encrypted images. Then to test the sensitivity, the difference between those two encrypted images is taken.

Fig. 17 a shows the source image selected for encryption. The keys k1 and k2, taken for the first case are: k1="01010100011010000 11000010111010001110011001000000110110101111001001000000 10010110111010101101110011001110010000001000110011101" k2="11100010001100101111110011110001100100010001001010010001100010001011000101011001111001001110011011001111 10011010001010010011"

Fig. 17 b shows the resultant encrypted image using k1 and k2.

The keys taken for the second case for this analysis are: k3="10110100011010110110000101110100011100110010011001101101011110010010000100101011011010110111001100111001000000100011001110101" k4="11100010001100101111110011110001100010001001010100100010001100010001011000101011001101001001110011011010110011111001101000101001001 1".

The resultant encrypted image using the above keys k3 and k4 is shown in Fig. 17 c.

To show that the encrypted image differs greatly based on the keys used for the encryption process, the difference between the images is taken as shown in Fig. 17 b and 17 c. Hence, from Fig. 17 d, it can be concluded that change in the keys affects the results of the designed encryption process. Thus it can be said that the algorithm is key sensitive.

7.3. Histogram analysis

The distribution of intensity levels for each pixel value in an image is depicted by the histogram for that image. The intensity distribution differs from image to image when the histogram for them is plotted. The encrypted images' histograms are plotted in order to assess the intensity distribution of the images.

The source image is considered to be a portrait image as shown in Fig. 18 a. The original source image is then encrypted using the algorithm and the encrypted image can be seen in Fig. 18 b. Figs. 18 c and 18 d show the resulting histograms for both the images, which display the intensity distribution. Similarly, an x-ray image is analysed as shown in Fig. 19 a. The corresponding encrypted image can be seen in Fig. 19 b. The histograms depicting the intensity distribution for both the images are generated as shown in Figs. 19 c and 19 d. It is observed that after encryption, a similar but not same, bell shaped distribution for the images is obtained. Additionally, it is observed that the intensity distribution of the encrypted images is considerably more uniform than that of the original image.

7.4. Recoverability test

For any encryption algorithm, it is necessary that the algorithm is capable of effectively recovering the original source image after decryption. Thus, for the described encryption algorithm, it is to be proven that the original image is generated after decryption, using the same keys as in encryption. An approach to show 100% recoverability is to subtract the pixel values of the decrypted image from the corresponding pixel values of the original image.

(a) (b) (c)

Fig. 21. Recoverability (a) Original Image (b) Decrypted Image (c) Difference between 21 a and 21 b.

Fig. 20 a shows the original image and its decrypted image can be seen in Fig. 20 b. The difference between the two images is displayed in Fig. 20 c. It is a completely black image that indicates complete and lossless recoverability of the source image. Fig. 21 shows the same results with another image.

8. Conclusion

This paper presents a novel approach to image encryption for secure image transfer. The RBF algorithm consists mainly of three steps - 'Rubik's Cube Scrambling ', 'Bitplane Frame Rotation ', and 'Bitplane Scrambling '. The Rubik's cube scrambling principle is successfully incorporated into the proposed encryption algorithm, thus introducing a higher degree of randomness. As a future scope, the Rubik's cube scrambling can be done on nxn sized blocks with n as 32, 64 and 128 rather than 16, 32 and 64. This will help induce a greater degree of randomness in large images. The length of the scrambling sequence and hence, the size of the keys can also be changed accordingly. The complexity in the first stage is increased by applying the scrambling for three rounds on different-sized blocks. In the second stage, the bit planes of the image are separated and frame rotation is applied to the individual bitplanes. The encryption process is completed by the last stage where the bitplanes are combined according to a definite sequence. The entire process is reversed for the decryption of the image. Two 128-bit keys are taken as input from the user to uniquely encrypt and decrypt the image. This paper also describes a new technique that is employed for updating the keys between steps to increase the randomness.

The combination of these methods helps the algorithm sustain various attacks. The RBF algorithm is completely lossless as the information in the image can entirely be recovered after decryption. The proposed technique is flexible in nature as it can work on images of any size. The described algorithm is robust in nature and can be used in day-to-day life for a variety of images. The evaluation and analysis of the RBF algorithm are explained in depth throughout the paper.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] G. Kołaczek, J. Mizera-Pietraszko, Security framework for dynamic service-oriented it systems, J. Inform. Telecommun. 2 (4) (2018) 428–448, doi:10.1080/24751839.2018.1479926.

[2] K. Deshpande, J. Girkar, R. Mangrulkar, Security enhancement and analysis of images using a novel sudoku-based encryption algorithm, J. Inf. Telecommun. 0 (0) (2023) 1–34, doi:10.1080/24751839.2023.2183802.

[3] P. Vörös, D. Csubák, P. Hudoba, A. Kiss, Securing personal data in public cloud, J. Inform. Telecommun. 4 (1) (2020) 51–66, doi:10.1080/24751839.2019.1686684.
[4] Internet Society what is encryption, (https://www.internetsociety.org/issues/encryption/what-is/).
[5] D. Lambić, S-Box design method based on improved one-dimensional discrete chaotic map, J. Inform. Telecommun. 2 (2) (2018) 181–191, doi:10.1080/24751839.2018.1434723.
[6] H. Delfs, H. Knebl, Symmetric-Key Cryptography, 2015, pp. 11–48.
[7] M. Blaze, W. Diffie, R.L. Rivest, B. Schneier, T. Shimomura, Minimal key lengths for symmetric ciphers to provide adequate commercial security. A report by an ad hoc group of cryptographers and computer scientists, 1996.
[8] M. Agrawal, P. Mishra, A comparative survey on symmetric key encryption techniques, Int. J. Comput. Sci. Eng. 4 (2012).
[9] A. Abdullah, Advanced encryption standard (aes) algorithm to encrypt and decrypt data (2017).
[10] N. Kaur, S. Sodhi, Article: data encryption standard algorithm (des) for secure data transmission, IJCA Proc. Int. Conf. Adv. Emerg. Technol. ICAET 2016 (2) (2016) 31–37. Full text available
[11] R.L. Rivest, The rc5 encryption algorithm, in: B. Preneel (Ed.), Fast Software Encryption, Springer Berlin Heidelberg, Berlin, Heidelberg, 1995, pp. 86–96.
[12] B. Rahul, K. Kuppusamy, Efficiency analysis of cryptographic algorithms for image data security at cloud environment, IETE J. Res. 0 (0) (2021) 1–12, doi:10.1080/03772063.2021.1990141.
[13] Twisty puzzle scramble generators, (https://ruwix.com/puzzle-scramble-generators/).
[14] M. Deutsch, How to fairly scramble a rubik's cube, 2017, (https://medium.com/@maxdeutsch/m2m-day-88-how-to-fairly-scramble-a-rubiks-cube-8c715f38475a#:~:text=Each%20scramble%20is%20expressed%20using,of%20the%20cube%20180%20degrees%E2%80%9D).
[15] R.V. Mudduluri, A. Golla, S. Raghava, T.J. Sai, Advanced image encryption & decryption using Rubik's cube technology, Int. J. Eng. Adv. Technol. (IJEAT) 11 (3) (2022) 24–27, doi:10.35940/ijeat.C3331.0211322.
[16] l. Zhang, X. Tian, S. Xia, A scrambling algorithm of image encryption based on rubik's cube rotation and logistic sequence, volume 1, 2011, pp. 312–315, doi:10.1109/CMSP.2011.69.
[17] X. Feng, X. Tian, S. Xia, An improved image scrambling algorithm based on magic cube rotation and chaotic sequences (2011). doi:10.1109/CISP.2011.6100274.
[18] A.-V. Diaconu, K. Loukhaoukha, An improved secure image encryption algorithm based on rubik's cube principle and digital chaotic cipher, Math. Probl. Eng. 2013 (2013) 1–10, doi:10.1155/2013/848392.
[19] K. Loukhaoukha, N. Makram, K. Zebbiche, An efficient image encryption algorithm based on blocks permutation and rubik's cube principle for iris images, 2013, doi:10.1109/WoSSPA.2013.6602374.
[20] S. Mozaffari, Parallel image encryption with bitplane decomposition and genetic algorithm, Multimed. Tools Appl. 77 (19) (2018) 25799–25819.
[21] Y. Zhou, W. Cao, C. Chen, Image encryption using binary bitplane, Signal Process. 100 (2014) 197–207, doi:10.1016/j.sigpro.2014.01.020.
[22] S. Madhu, N. Kumari, Novel approaches for selection of bitplanes in image encryption, 2019.
[23] A. Houas, B. Rezki, Z. Mokhtari, An encryption algorithm for grey-scale image based on bit-plane decomposition and diffuse representation, J. Discr. Math. Sci. Cryptogr. 24 (1) (2021) 35–47, doi:10.1080/09720529.2019.1664382.
[24] Q. Sun, W. Yan, J. Huang, W. Ma, Image encryption based on bit-plane decomposition and random scrambling, in: 2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet), 2012, pp. 2630–2633, doi:10.1109/CECNet.2012.6201673.
[25] M. Podesser, H.-P. Schmidt, A. Uhl, Selective bitplane encryption for secure transmission of image data in mobile environments, Fifth IEEE Nordic Signal Process. Sympos. (2002).
[26] Y. Liu, Q. Zhong, J. Shen, C.-C. Chang, A novel image protection scheme using bitplane compression and secret sharing, J. Chinese Inst. Eng. 40 (2) (2017) 161–169, doi:10.1080/02533839.2017.1294994.
[27] S. Mozaffari, Parallel image encryption with bitplane decomposition and genetic algorithm, Multimed. Tools Appl. 77 (2018), doi:10.1007/s11042-018-5817-8.
[28] C. Naveen, V.R. Satpute, Image encryption technique using improved a5/1 cipher on image bitplanes for wireless data security, in: 2016 International Conference on Microelectronics, Computing and Communications (MicroCom), 2016, pp. 1–5, doi:10.1109/MicroCom.2016.7522451.
[29] K.-M. Chen, C.-C. Chang, High-capacity separable reversible data-hiding method in encrypted images based on block-level encryption and huffman compression coding, Conn. Sci. 33 (4) (2021) 975–994, doi:10.1080/09540091.2021.1926930.
[30] W. Song, C. Fu, Y. Zheng, M. Tie, J. Liu, J. Chen, A parallel image encryption algorithm using intra bitplane scrambling, Math. Comput. Simul. 204 (2023) 71–88, doi:10.1016/j.matcom.2022.07.029.
[31] Y. Zhou, K. Panetta, C. Chen, Image encryption using p-fibonacci transform and decomposition, Opt. Commun. 285 (2012) 594–608, doi:10.1016/j.optcom.2011.11.044.
[32] A. Kumar, P. Singh, Aggrandize bit plane coding using gray code method, Int. J. Comput. Appl. 20 (2011) 44–49, doi:10.5120/2434-3273.
[33] Y. Zhou, K. Panetta, S. Agaian, Image encryption algorithms based on generalized p-gray code bit plane decomposition, in: 2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers, 2009, pp. 400–404, doi:10.1109/ACSSC.2009.5469840.