sciendo

BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **23**, No 3
Sofia • 2023 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2023-0029

A Secure Peer-to-Peer Image Sharing Using Rubik's Cube Algorithm and Key Distribution Centre

Aswani Kumar Cherukuri[1], *Shria Sannuthi*[1], *Neha Elagandula*[1], *Rishita Gadamsetty*[1], *Neha Singh*[1], *Arnav Jain*[1], *I. Sumaiya Thaseen*[1], *V. Priya*[1], *Annapurna Jonnalagadda*[2], *Firuz Kamalov*[3]

[1]*School of Information Technology & Engineering, Vellore Institute of Technology, Vellore 632014, India*
[2]*School of Computer Science & Engineering, Vellore Institute of Technology, Vellore 632014, India*
[3]*Dept of Electrical Engineering, Faculty of Engineering, Canadian University Dubai, Dubai*
E-mail: cherukuri@acm.org

Abstract**:** *In this work, we build upon an implementation of a peer-to-peer image encryption algorithm: "Rubik's cube algorithm". The algorithm utilizes pixel-level scrambling and XOR-based diffusion, facilitated through the symmetric key. Empirical analysis has proven this algorithm to have the advantage of large key space, high-level security, high obscurity level, and high speed, aiding in secure image transmission over insecure channels. However, the base approach has drawbacks of key generation being handled client-side* (*at nodes*) *and the process is time-consuming due to dynamically generating keys. Our work solves these issues by introducing a Key Distribution Center* (*KDC*) *to distribute symmetric keys for transmission, increasing confidentiality, and reducing key-generation overhead on nodes. Three approaches utilizing the KDC are presented, communicating the dimensions with KDC to generate keys, standardizing any image to fixed dimensions to standardize key-generation, and lastly, using a single session key which is cyclically iterated over, emulating different dimensions.*

Keywords**:** *Image encryption, Key Distribution Center, Peer-to-peer encryption, Rubik's cube algorithm, Symmetric key.*

1. Introduction

In 2023, the global data generation rate exceeded 3.5 quintillion bytes per day, and it is growing rapidly due to the increasing use of smart devices [1]. M u n e and B h u r a [2] have highlighted that in this internet era, where almost every piece of data is stored somewhere permanently, data encryption has become of utmost importance to maintain privacy. M h a t r e et al. [3] mention that there are always malicious actors who try to gain unauthorized access to private data, with the intention of leaking it to cause significant losses to individuals and organizations, using it for blackmail or ransom, or just to cause trouble for the victim (M e h t a, D h i n g r a and

Mangrulkar [4], Sawant, Solkar and Mangrulkar [5]). Image encryption is the process of converting an image into a format that is unintelligible without the correct decryption key or password. It is important for protecting sensitive or private images. Image encryption typically involves transforming the image into an unreadable format using mathematical algorithms. The encrypted image can only be accessed by authorized parties who have the decryption key.

Digital data comprising a variety of information is being transmitted through the Internet in the post-pandemic world, offering consumers a great deal of ease. But this data exchange also puts it in danger of theft, leaking, and malicious attacks. Because they are two-dimensional and large, digital photographs provide a problem for traditional text-based encryption techniques. The algorithm includes adaptive image content-based initial random value generation to achieve high plain image sensitivity and overcome plain image-related attacks. The initial vectors of the Henon map are obtained from the random value, which is then iterated to obtain key sequences applied over the Rubik's cube row and column confusion processes. Vidhya and Brindha [6] have focused on how conventional encryption methods like AES, RSA, DES, etc., have been designed to encrypt text-based data and may not be suitable for multimedia data such as images and videos. Kumar and Rani [7] have inferred that when a picture is huge, traditional image encryption algorithms like the Data Encryption Standard (DES) have poor levels of efficiency. Images have unique characteristics, such as high correlation and redundancy, which can make them vulnerable to attacks.

Zhao et al. [8] have suggested that chaotic systems have become more and more important in a variety of industries. To overcome the shortcomings of existing chaotic maps, a novel 2D Logistic-Chebyshev Chaotic Map (2D-LCCM) is presented. Narrow parameter range, limited ergodicity, and uneven sequence distribution are some of the shortcomings. Deshpande, Girkar and Mangrulkar [9] suggest Sudoku puzzles, which are mathematical entities with inherent patterns, have been found to be effective in generating large and intricate key spaces that are difficult to crack. Helmy et al. [10] introduce a novel encryption algorithm based on the 3-D Rubik's cube for achieving 3D encryption of multiple images. The proposed method involves encrypting images separately using the RC6 algorithm, and further encrypting them with the Rubik's cube algorithm. The RC6 encrypted images serve as faces of the Rubik's cube, which adds a degree of permutation to the encryption process. The simulation results indicate that this method is efficient, robust, and secure. Gomathi and Kumar [11] suggest that with the Rubik's cube encryption technique, the LSB Steganography algorithm offers us exceptional security and safeguards the image during transmission. Image resolution essentially stays the same. Additionally, the speed of data embedding into images is very fast. Ionescu and Diaconu [12] presented an implementation of a communication system for multiple mobile devices with imaging sensors that encrypt images using the Rubik's cube encryption algorithm, along with a server. The study assesses the algorithm's performance and suitability for mobile devices through various tests.

A b i t h a and B h a r a t h a n have [13] discussed how more multimedia data is created and communicated across networks in the sphere of digital and multimedia applications. This work presents an efficient image encryption solution based on the Rubik's cube concept and a chaotic Baker map. It is made up of two layers. The proposed technique enhances the security level of the Rubik's cube encryption technique. A t t k a n, R a n g a and A h l a w a t [14] discuss the challenges of running high-end security protocols on resource-bound Edge devices in the context of the growing importance of the Internet of Things (IoT). To ensure peer-to-peer security among IoT network nodes, proper mutual authentication is necessary, and a secure session key must be established between source and destination nodes before sending sensitive data. The article proposes using a Rubik's cube puzzle-based cryptosystem to securely exchange parameters among peers and generate session keys. J i n and P a r k [15] have presented an AI-based approach that uses the state change of Rubik's cube to generate keys in a symmetric-key encryption system. By applying the Rubik's cube state change algorithm, the approach is able to induce and use the key without exchanging the pre-shared key, providing the advantage of secure key generation and management. Due to the key induction method, malicious attackers have limited information on the keys used for encryption and decryption, thus enhancing the confidentiality and integrity of the key generation and management process.

C h u m a n, S i r i c h o t e d u m r o n g and K i y a [16] have suggested that for EtC (Encryption and Transmission) systems, a novel picture encryption approach has been put forth that makes use of block scrambling techniques to increase security over existing schemes. The suggested approach allows for the usage of smaller block sizes as well as a greater number of blocks, enhancing invisibility and security against various assaults including jigsaw puzzles and brute-force attacks. This innovative method offers improved security and robustness for image transmission while offering considerable advantages over current encryption systems [17]. With the detailed literature analysis, we have learned that concerns about data privacy and security have been sparked by the increasing use of smart devices, which has accelerated the generation of data. Traditional text-based encryption methods are unsuitable for large and two-dimensional digital photographs in the context of image encryption. Researchers have proposed a variety of techniques based on chaotic systems, Rubik's cube, Sudoku puzzles, and block scrambling to address this issue.

In this paper, three methodologies are proposed that use the lightweight Rubik's Cube Algorithm along with KDC. The algorithm is made even more lightweight by eliminating the computational overhead on the nodes resulting from key generation. First, Alice sends Bob an image with the dimensions $M \times N$. The keys that Alice and Bob share, K_A and K_B, are produced by the KDC based on the dimensions that Alice has provided to it. Second, standardizing the image makes it simpler to use KDC's Rubik's Cube Algorithm by lowering variability and complexity. Lastly, by using a single session key, K_s, rather than several keys, which may be reused for photos of different dimensions. Each image's session key, K_s, is generated by the KDC depending on its dimensions and sent to the sender. The Rubik's Cube Algorithm for picture encryption offers great security and is impervious to statistical and brute-force attacks. The algorithm is also capable of fast encryption/decryption which is more

likely suitable for real-time image transmitting applications. Our work is different from existing algorithms for image sharing in the following ways:

- The key is generated and distributed by the KDC. This can reduce the communication overhead due to key generation.
- Enhanced security for image sharing.
- The proposed three methods can be used for different kinds of applications according to the need.

The remaining part of the paper is arranged as follows: The image encryption using Rubik's Cube Algorithm is given in Section 2. The proposed methodologies are given in Section 3. Section 4 provides an analysis of the suggested methods and is concluded in Section 5.

2. Image encryption using Rubik's Cube Algorithm

The outline of the Rubik's Cube Algorithm is given in Fig. 1. The algorithm consists of four phases. A random key generation process, confusion of the pixels in the plain image using the key sequence obtained for the row and columns, respectively, a simple XOR function applied on the confused image obtained in the previous step again by utilizing the key sequences generated for rows and columns, respectively. A detailed description of the phases mentioned is given below.

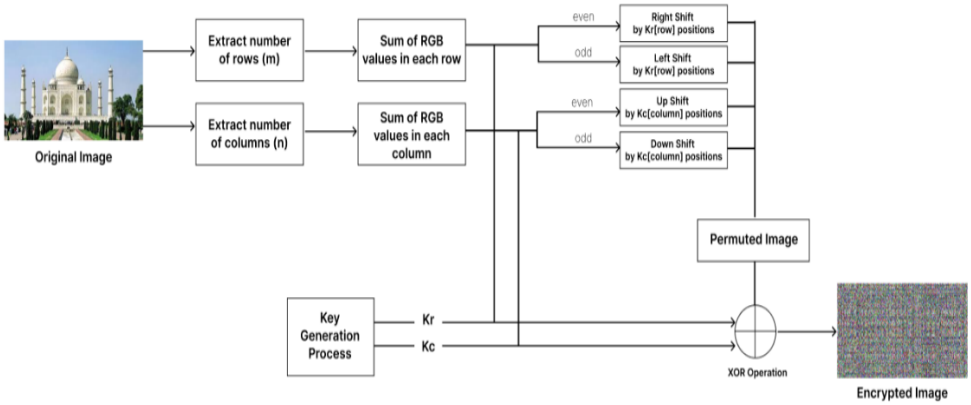


Fig. 1. Block-diagram of Rubik's Cube Algorithm

2.1. Random key generation process

This process involves using a random vector α to find prime factors and then storing the largest prime factor in a list called the prime store I. The user selects an increment factor β, which is added to the initial α to generate a new random number $\alpha = \alpha + \beta$. This process of finding prime factors and adding them to the prime store is then repeated Q times to generate a set of Q prime values. The size of Q is $Q=M\times N\times 3$ where Q is determined by the number of pixels in an image, with each pixel's R, G, and B values being XORed with a different prime value to prevent the pattern from being traced. It is crucial that the initial random seed for α is random in nature to be unpredictable to ensure the uniqueness of the prime store and prevent differential

attacks. The values in $I = \{I_1, I_2, I_3, \ldots, I_Q\}$ are converted to binary to create 8-bit sequences and transform into a new set of values $D = \{D_1, D_2, D_3, \ldots, D_Q\}$ that have no correlation with the previous bytes.

2.2. Encryption

In the Rubik's Cube Algorithm, the encryption happens as follows. Consider an image I_0 of the size $M \times N$. Image I_0 basically represents the pixel values in the matrix form of the image. Two keys K_R and K_C are randomly generated of length M and N, respectively, as discussed in Method 2.1 and stored in a vector format, $K_R[i]$ and $K_C[i]$. Both take random values from 0 to $2^{\alpha}-1$. For every row i of the image I_0, the sum of all the elements in the row i that is denoted by (i) is computed using the equation

$$\alpha(i) = \sum_{j=1} I_0(i,j), \text{where } i = 1, 2, 3, 4, \ldots, M. \quad (1)$$

Then, modulo 2 of (i) is computed which is denoted by $M_{\alpha}(i)$. Row i is then left, right, or circular shifted by $K_R(i)$ positions, i.e., the image pixels are moved $K_R(i)$ positions left or right based on whether $M_{\alpha}(i)$ =0. If $M(i)$ =0 then perform right circular shift, else – left circular shift. Similarly, for every column j of the image I_0, the sum of all the elements in the column j that is denoted as $\beta(j)$ is computed using

$$\beta(j) = \sum_{i=1} I_0(i,j), \text{where } j = 1, 2, 3, 4, \ldots, N. \quad (2)$$

Then, modulo 2 of $\beta(j)$ which is denoted by $M_{\beta}(j)$. Column j is then either down, up, or circular shifted by $K_C(i)$ positions based on whether $M_{\beta}(j) = 0$ or not if $M_{\beta}(j)$ =0 then perform an up circular shift, else – a down circular shift.

After executing the two steps mentioned above, a scrambled image is formed which is denoted as I_{SCR}. Then using vector K_C, the XOR operator is applied to each row of the scrambled image I_{SCR} like below:

$$I_1(2i-1, j) = I_{SCR}(2i-1, j) \oplus K_C(j),$$
$$I_1(2i, j) = I_{SCR}(2i, j) \oplus \text{rot180}(K_C(j)). \quad (3)$$

Similarly for Vector K_R the similar operations are performed as follows:

$$I_{ENC}(i, 2j-1) = I_{SCR}(i, 2j-1) \oplus K_R(j),$$
$$I_{ENC}(i, 2j) = I_{SCR}(i, 2j) \oplus \text{rot180}(K_R(j)). \quad (4)$$

If ITER=$ITER_{max}$, then the image that is encrypted (denoted as I_{ENC}) is created and the encryption process ends here, if ITER<$ITER_{max}$, the algorithm goes back to the first step. Vectors K_R, K_C and the max iteration number $ITER_{max}$ are secret keys in the proposed encryption algorithm.

2.3. Decryption

Decryption of an image using Rubik's Cube Algorithm is basically doing the reverse process of encrypting an image using the same algorithm which includes inversing all the operations.

The encrypted picture I_{ENC}, the secret keys, K_R, K_C, and $ITER_{max}$ are used to reconstruct the decrypted image, I_0. Initially the ITER variable is set to 0 and then incremented until it reaches $ITER_{max}$. The encrypted image I_{ENC}'s vector K_R and each column are subjected to the following bitwise XOR operation:

$$I_1(i, 2j-1) = I_{ENC}(i, 2j-1) \oplus K_R(j),$$
$$I_1(i, 2j) = I_{ENC}(i, 2j) \oplus \text{rot180}(K_R(j)). \quad (5)$$

Input: Image, K_R, K_C and $ITER_{max}$
Output: Encrypted Image
Algorithm-

1. Sender selects an image to send
2. Keys K_R and K_C are generated using Key generation process.
3. The key generation process involves:
   a) α to find prime factors and to store largest prime factor I
   b) Increment factor β
   c) This process is repeated Q times where Q = MxNx3
   d) The values in I are converted to binary bits D
4. Determine the number of iterations, ITERmax, and initialize the counter ITER to 0
5. **while** ITER < ITERmax:
6. For each row i of the image I_o

```
for i in range(image.shape[0]):
    compute the sum of all elements in the row i, denoted by α(i),
    compute α(i)%2, denoted by In(i),
    if In(i) == 0:
        circular shift row i right by KR [i] positions
    else
        circular shift row i left by KR [i] positions
    end
end
```

7. For each column j of the scrambled image I_{SCR}

```
for j in range(image.shape[1]) :
    compute the sum of all elements in the row i, denoted by β(j),
    compute β(j)%2, denoted by MQ(j),
    if MQ(j) == 0:
        circular shift column j up by KC [j] positions
    else
        circular shift column j down by KC [j] positions
    end
end
```

8. Apply the bitwise XOR operator to each row of I_{SCR}

```
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        if i % 2 == 0:
            KC key is rotated by 180 degrees using the rotate_bits function and stored in val2(Kc[j])
        else
            value of the KC key corresponding to the current column is stored in val1
        end
        image[i, j] = image[i, j] ^ val1 (bitwise operator is applied to the current pixel)
    end
end
```

9. Apply the bitwise XOR operator to each column of I

```
for j in range(image.shape[1]):
    for i in range(image.shape[0]):
        if j % 2 == 0:
            value of the KR key corresponding to the current column is stored in val1
        else
            KR key is rotated by 180 degrees using the rotate_bits function and stored in val2(KR[i])
        end
        image[i, j] = image[i, j] ^ val1^val2 (bitwise operator is applied to the current pixel)
    end
end
```

10. Increment the counter ITER by 1 and output the encrypted image I_{ENC}

Fig. 2. Rubik's Cube Algorithm

Next, every row in image I_1 is subjected to the bitwise XOR operator using the K_C vector.

(6)
$$I_{\mathrm{SCR}}(2i-1,j) = I_1(2i-1,j) \oplus \mathrm{K_C}(j),$$
$$I_{\mathrm{SCR}}(2i,j) = I_1(2i,j) \oplus \mathrm{rot180}(\mathrm{K_C}(j)).$$

For every column j of the scrambled image I_{SCR}, the total of all the elements in each column of the scrambled picture I_{SCR} is determined and indicated as

(7) $$\mathrm{SCR}(j) = M_i = I_{\mathrm{SCR}}(i,j)\,, \text{where } j = 1, 2, \ldots, N.$$

Modulo 2 of SCR(j), indicated by $M_{SCR}(j)$ and column j is circularly shifted downward or upward by $K_C(i)$ as follows – up-circular shift if $M_{SCR}(j) = 0$; else, down-circular shift I_{SCR} for each row i of the jumbled image. All the components in row I are added and indicated by

(8) $$\mathrm{SCR}(i) = N_j = I_{\mathrm{SCR}}(i,j)\,, \text{where } i = 1, 2, \ldots, M.$$

Modulo 2 of SCR(j) is again determined and denoted by $M_{SCR}(j)$. Row I is then circularly moved to the left or right by $K_R(i)$ as right circular shift if $M_{SCR}(j)=0$; else, left circular shift. Image I_{ENC} is decrypted and the decryption process is complete if ITER=$ITER_{max}$; otherwise, the algorithm loops back to the first step. Fig. 2 provides a detailed algorithm.

3. Proposed methodology

3.1. Communication of dimensions

In, this proposed method both the communicating nodes (i.e., Alice and Bob) communicate through each other via the Key Distribution Centre (KDC). At each stage of image sharing, i.e., for every image, the sender (i.e., Alice) will communicate the dimensions of the corresponding image with the KDC. According to the received dimensions, the KDC generates the key and sends it to Alice. This message is encrypted with the master key that is assumed to be previously shared between the KDC and Alice. For example, an image of dimensions $M \times N$ is being shared from Alice to Bob, K_R and K_C are the keys generated for the rows and columns of the image to be shared respectively using Rubik's Cube Algorithm. K_A and K_B are the keys that are previously shared by Alice and Bob with the KDC respectively.

The encryption and decryption process of the proposed methodology follows the same algorithm as the original Rubik's cube Method 2 for image encryption. However, in this approach, the row-wise and column-wise keys K_R and K_C are generated by the KDC using the dimensions $M \times N$ that are communicated for every image transmission. This eliminates the need to generate these keys at the sender and receiver ends, thus reducing computational overhead and ensuring the keys are securely managed. Fig. 4 presents the algorithm for the encryption process of this method.

The challenge of implementing this method is that for every image that is to be shared by the node, a contact to the KDC must be made, which leads to a communication overhead between KDC and the corresponding node, which can be resolved using Method 3.2 by compromising the quality as well as the dimensions of the image and by Method 3.3 through the usage of a single session key. Fig. 5 demonstrates the application of communicating dimensions to the KDC and the subsequent use of those keys to encrypt an image. The different stages of the encryption process such as permutation and diffusion is depicted here.

3.1.1. Block-diagram

In Fig. 3, Alice sends a request to the KDC for a key to protect the image that is being sent to Bob. The request includes ID_A, which is the identifier of Alice (e.g., its IP address or network address) and ID_B, which is the identifier of Bob so that the KDC knows that Alice wants to speak to Bob. This message also consists of the dimensions of the image that is to be sent.

(1) Alice → KDC: $ID_A \,||\, ID_B || \text{ Dimensions } || N_1$.

The KDC responds back to Alice with a message that is encrypted using a key that has been shared previously only by the KDC and Alice (i.e., K_A). This message includes K_R and K_C that KDC generated using the Rubik's cube algorithm according to the dimensions received in the previous transaction which Alice uses to encrypt the image using Rubik's Cube Algorithm. The other components in the message are

Another part of this message, which is intended for Bob, is encrypted using K_B, which is only shared by the KDC and Bob. The components in this message are ID_A, K_R, and K_C. Keys K_R and K_C are used by B to decrypt the message sent by Alice.

(2) KDC → Alice: $E[K_A, [K_R \,||\, K_C \,||\, ID_B \,||\, N_1 \,||\, E[K_B, [K_R \,||\, K_C \,||\, ID_A]]]$

Alice sends the message that is sent by KDC to Bob

(3) Alice → Bob: $E(K_B, [\, K_R \,||\, K_C \,||\, ID_A])$

Bob acknowledges the message received by Alice by sending a message encrypted with K_R and K_C and a nonce N_2

(4) Bob → Alice: $E([K_R \,||\, K_C], N_2)$

Alice then applies a function on N_2 and sends it to Bob for confirmation.

(5) Alice → Bob : $E([K_R \,||\, K_C\,], f(N_2))$

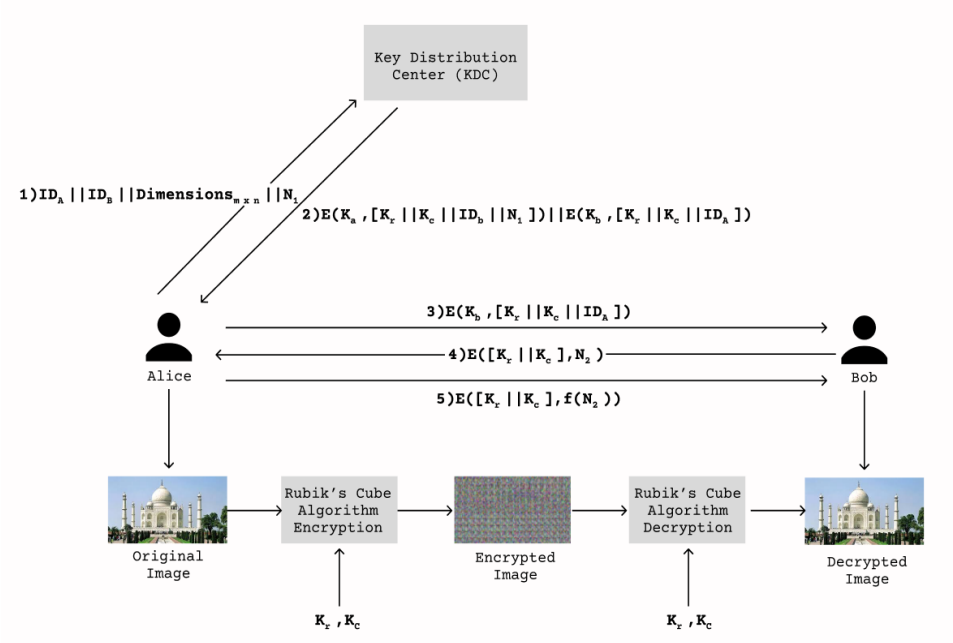


Fig. 3. Block diagram of Rubik's Cube Algorithm using KDC with communication of dimensions

3.2. Standardization

A new issue arises when using KDC, Rubik's cube generates keys based on the image and the number of pixels it contains, but the KDC does not have the image and does not know what size keys to generate. To address this, standardization can be used to transform the image into a standardized format with well-defined characteristics, such as resizing the image to a standard size, converting it to a specific colour space, normalizing the intensity values of the pixels, and removing noise or artefacts.

Input: Image, MxN, K_R, K_C and $ITER_{max}$
Output: Encrypted Image
Algorithm-

1. Sender selects an image to send and standardizes it to a fixed number of pixels
2. Key Distribution Centre generates two common keys K_R and K_C of length M and N for the row and column, respectively
3. Determine the number of iterations, ITERmax, and initialize the counter ITER to 0
4. **while** ITER < ITERmax:
5. For each row i of the image I_o

```
for i in range(image.shape[0]):
    compute the sum of all elements in the row i, denoted by α(i),
    compute α(i)%2, denoted by In(i),
    if In(i) == 0:
        circular shift row i right by KR [i] positions
    else
        circular shift row i left by KR [i] positions
    end
end
```

6. For each column j of the scrambled image I_{SCR}

```
for j in range(image.shape[1]) :
    compute the sum of all elements in the row i, denoted by β(j),
    compute β(j)%2, denoted by MQ(j),
    if MQ(j) == 0:
        circular shift column j up by KC [j] positions
    else
        circular shift column j down by KC [j] positions
    end
end
```

7. Apply the bitwise XOR operator to each row of I_{SCR}

```
for i in range(image.shape[0]):
    for j in range(image.shape[1]):
        if i % 2 == 0:
            KC key is rotated by 180 degrees using the rotate_bits function and stored in val2(Kc[j])
        else
            value of the KC key corresponding to the current column is stored in val1
        end
        image[i, j] = image[i, j] ^ val1 (bitwise operator is applied to the current pixel)
    end
end
```

8. Apply the bitwise XOR operator to each column of I

```
for j in range(image.shape[1]):
    for i in range(image.shape[0]):
        if j % 2 == 0:
            value of the KR key corresponding to the current column is stored in val1
        else
            KR key is rotated by 180 degrees using the rotate_bits function and stored in val2(KR[i])
        end
        image[i, j] = image[i, j] ^ val1^val2 (bitwise operator is applied to the current pixel)
    end
end
```

9. Increment the counter ITER by 1
10. Output the encrypted image I_{ENC}

Fig. 4. Rubik's Cube Algorithm using KDC by standardization of images

This process creates a scrambled image, which is then encrypted using a maximum number of iterations ($ITER_{max}$) and the secret keys K_R and K_C. The decryption process involves applying bitwise XOR operations to the K_R and K_C

vectors and to each row and column of the encrypted image I_{ENC}, using a similar set of formulas as the encryption process. The decrypted image Io is recovered from the encrypted image I_{ENC}, using the secret keys K_R and K_C and the maximum number of iterations $ITER_{max}$. The use of circular shifting and bitwise XOR operations creates a secure encryption algorithm that can be made more secure by increasing the maximum number of iterations, though this comes at the cost of reduced processing speed. Overall, this encryption algorithm can be used to secure images sent between two communicating nodes.

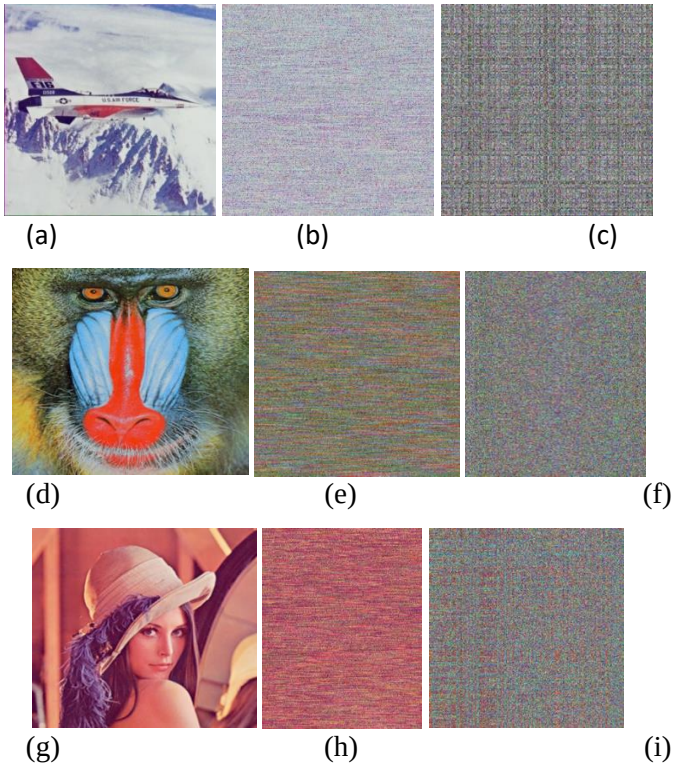


Fig. 5. Permutation and diffusion stages for various images: Jetplane-original (a); Jet plane-confused (b); Jet plane-encrypted (c); Baboon-original (d); Baboon-confused (e); Baboon-encrypted (f); Lena-original (g); Lena-confused (h); Lena-encrypted (i)

In terms of security, standardizing the image size and its pixels helps to ensure that the same level of security is applied to every image encryption. This is important because different image sizes and pixel values can result in different levels of security, which can make the encryption vulnerable to attacks. By standardizing the image size and pixel values, we can ensure that the same level of security is applied uniformly to all image encryptions. The significance of this method is that it simplifies the key distribution process and ensures that the encryption is uniformly secure for all images, which ultimately results in a more efficient and secure network communication. This method can be particularly useful in large-scale networks where a large number of nodes require cryptographic keys, as it can reduce the computational overhead and increase the overall security of the network.

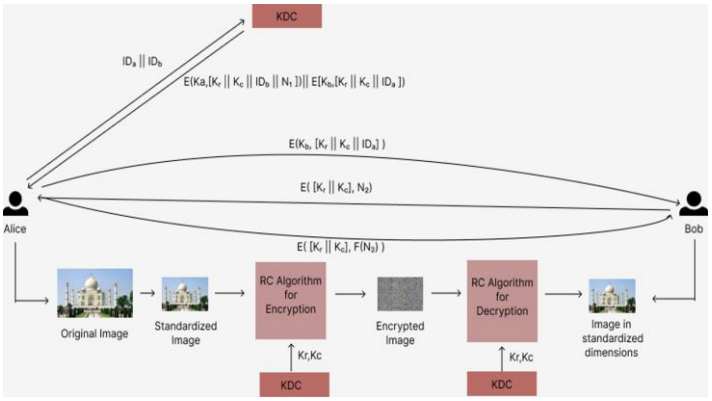


Fig. 6. Block-diagram of Rubik's Cube Algorithm using KDC by standardization of images

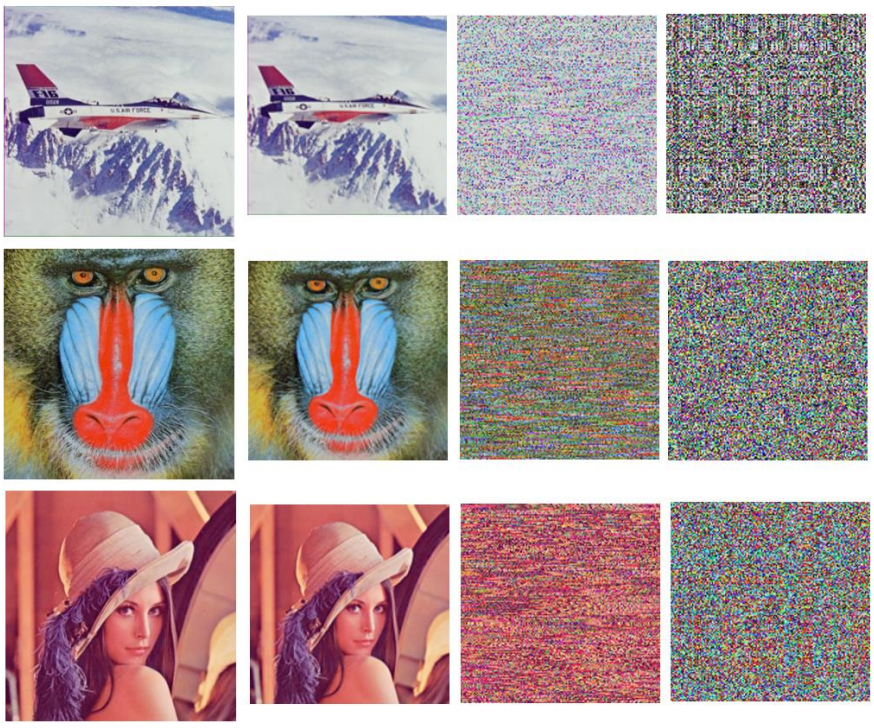
Input: Image, Standardized Dimensions (MxN), K_R, K_C and $ITER_{max}$
Output: Encrypted Image
Algorithm-

1. Sender selects an image to send and standardizes it to a fixed number of pixels
2. Key Distribution Centre generates two common keys K_R and K_C of fixed length M and N for the row and column, respectively
3. Determine the number of iterations, ITERmax, and initialize the counter ITER to 0
4. **while** ITER < ITERmax:
5. For each row i of the image I_o
   for i in range(image.shape[0]):
   compute the sum of all elements in the row i, denoted by α(i),
   compute α(i)%2, denoted by I_n (i),
   if I_n (i) == 0:
   circular shift row i right by K_R [i] positions
   else
   circular shift row i left by K_R [i] positions
   end
   end
6. For each column j of the scrambled image I_{SCR}
   for j in range(image.shape[1]) :
   compute the sum of all elements in the row i, denoted by β(j),
   compute β(j)%2, denoted by M_Q(j),
   if M_Q(j) == 0:
   circular shift column j up by K_C [j] positions
   else
   circular shift column j down by K_C [j] positions
   end
   end
7. Apply the bitwise XOR operator to each row of I_{SCR}
   for i in range(image.shape[0]):
   for j in range(image.shape[1]):
   if i % 2 == 0:
   K_C key is rotated by 180 degrees using the rotate_bits function and stored in val2(Kc[j])
   else
   value of the K_C key corresponding to the current column is stored in val1
   end
   image[i, j] = image[i, j] ^ val1 (bitwise operator is applied to the current pixel)
   end
   end
8. Apply the bitwise XOR operator to each column of I
   for j in range(image.shape[1]):
   for i in range(image.shape[0]):
   if j % 2 == 0:
   value of the K_R key corresponding to the current column is stored in val1
   else
   K_R key is rotated by 180 degrees using the rotate_bits function and stored in val2(K_R[i])
   end
   image[i, j] = image[i, j] ^ val1^val2 (bitwise operator is applied to the current pixel)
   end
   end
9. Increment the counter ITER by 1
10. Output the encrypted image I_{ENC}

Fig. 7. Rubik's Cube Algorithm using KDC by standardization of images

In Fig. 6, standard dimensions are set for all the images to be sent securely, node A sends the I_A and I_B to the KDC. These standard dimensions are used for every

image sent during that particular session. Moreover, the dimensions vary according to the application it is being used in. The KDC verifies the authentication credentials of nodes A and B and generates a new set of keys (K_R and K_C) for each row and column of the standardized image. The KDC encrypts (K_R and K_C) generated using the K_A. The KDC sends the encrypted keys to the node A and node A decrypts the keys using K_A and communicates the decrypted keys (K_R and K_C) to node B using K_B. Then, node B decrypts the keys using K_B. The image to be sent is standardized account to the set standard. Node A then encrypts the standardized image to be sent to node B using K_R and K_C. The encrypted image can now be used for secure communication between the nodes. Node B can now decrypt the image sent using K_R and K_C. The entire algorithm for this logic is given in Fig. 7. Fig. 8 depicts the initial standardization of the image and the subsequent utilization of the keys having standard dimensions from the KDC for the encryption process. The different stages of the process such as standardization, permutation, and diffusion are depicted here.



(a) (b) (c) (d)

Fig. 8. Permutation and diffusion stages for various images: Original image (a); Standardized image (b); Confused image (c); Encrypted image (d)

3.3. Single session key

The proposed methodology for single session keys has several advantages. One of the key benefits is that it reduces communication costs. This is achieved by replacing the use of multiple dimension-dependent keys, that is, the row-wise and column-wise keys K_R and K_C of *M* and *N* bits, respectively, with a single key K_s of length 128 bits. This single key is received from the KDC and used cyclically. The key iteration is

varied by applying a set of operations to the key while traversing it to accommodate varying image dimensions. Fig. 9 shows the block diagram of this proposal. Fig. 10 presents the algorithm for the encryption process of the single session key approach.

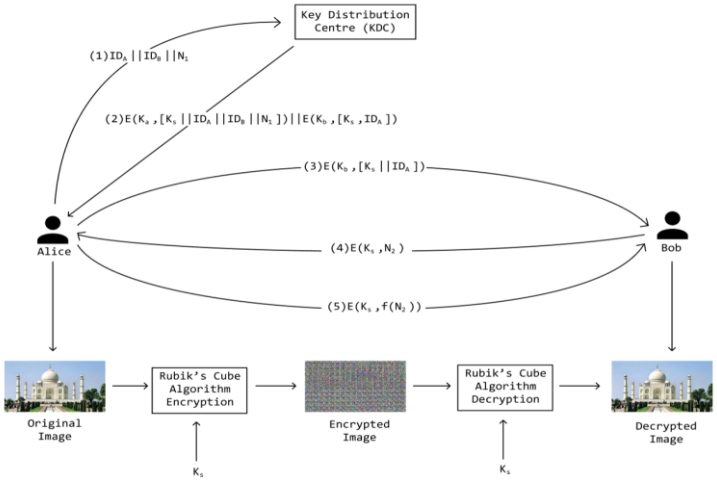


Fig. 9. Block-diagram of Rubik's Cube Algorithm with KDC using single session key

INPUT: Image, MxN, K_s and $ITER_{max}$
OUTPUT: Encrypted image
Algorithm:
1. Send selects image to send
2. Key Distribution Centre generates session key K_s of fixed 128 bits length
3. Determine number of iterations, $ITER_{max}$, number of 8-bit values in K_s , M as 16, and initialize counter ITER to 0
4. **for** k=0 to $ITER_{max}$ **do**
5. | **For** each row of image I_o,
   | **for** i in range(image.shape[0]):
   | | **if** sum(row) % 2 == 0:
   | | | circular shift row i right by K_s[i%M] positions
   | | **else**
   | | | circular shift row i left by K_s[i%M] positions
   | | **end**
   | **end**
   end
6. | **For** each column j of scrambled image I_{SCR}
   | **for** j in range(image.shape[1]):
   | | **if** sum(column) % 2 == 0:
   | | | circular shift column j up by K_s[i%N] positions
   | | **else**
   | | | circular shift column j down by K_s[i%N] positions
   | | **end**
   | **end**
   end
7. | Apply bitwise XOR operator to each row of I_{SCR}
   for i in range(image.shape[0]):
   | **for** j in range(image.shape[1]):
   | | **if** i % 2 == 0:
   | | | value of K_s corresponding to current row stored in val1
   | | **else**
   | | | K_s corresponding row component is rotated by 180 degrees using rotate_bits function and stored in val1
   | | **end**
   | | **if** j % 2 == 0:
   | | | value of K_s corresponding to current column stored in val2
   | | **else**
   | | | K_s corresponding column component is rotated by 180 degrees using rotate_bits function and stored in val2
   | | **end**
   | | image[i, j] = image[i, j] ^ val1 ^ val2 (bitwise operator is applied to the current pixel)
   | **end**
   end
end
8. Increment counter ITER by 1
9. Output encrypted image I_{ENC}

Fig. 10. Algorithm for single session key approach

The methodology of using a single key also solves the problem of having to generate a different key for every image transmission. Additionally, this approach provides an optimal balance between security, image quality, and communication cost. The use of a single key reduces communication overheads and streamlines the key management process. Fig. 11 demonstrates the implementation of the single session key approach to acquire keys from the KDC for the encryption of images. The different stages of the encryption process such as permutation and diffusion is depicted here. As shown in Fig. 9, Sender node initially contacts KDC to request for session key for transmission, upon which the KDC provides the key K_s as a message where one portion is encrypted using sender node's master key, and remaining portion with receiver node's master key so only these two parties gain access to the session key K_s.

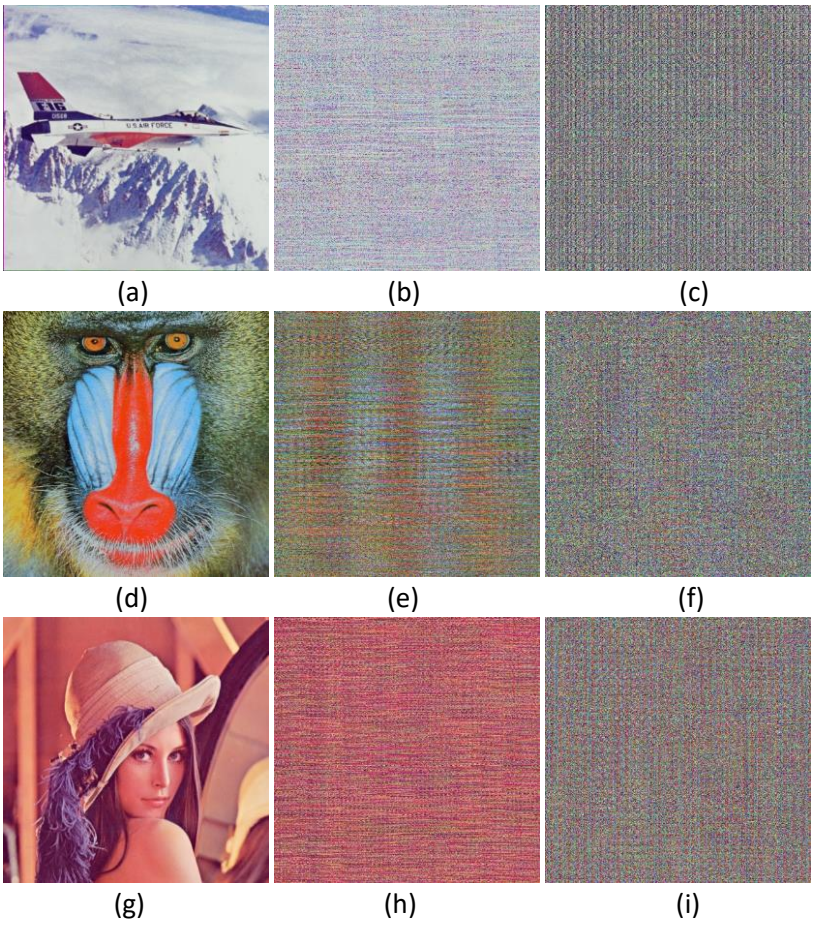


Fig. 11. Permutation and diffusion stages for various images: Jet plane-original (a); Jet plane-confused (b); Jet plane-encrypted (c); Baboon-original (d); Baboon-confused (e); Baboon-encrypted (f); Lena-original (g); Lena-confused (h); Lena-encrypted (i)

4. Security analysis

The proposed scheme has been analyzed in terms of security using several methods such as visual test, key sensitivity analysis, key space analysis, histogram analysis, and speed analysis.

4.1. Visual test

The effectiveness and security of the encryption scheme is initially analyzed by observing the differences and similarities between the plain image and the encrypted image. An encryption scheme is superior if the degree by which the encrypted image differs from the plain image is very high. For the three proposed methods, it is observed that the encrypted image and the plain image vary by a great extent.

4.2. Key space analysis

The key space refers to the total number of possible keys that can be used in an encryption algorithm. A larger key space generally implies additional security in the encryption algorithm. The proposed three methods use different key sizes catering to the needs of the algorithm.

4.2.1. Communication of dimensions of the image to KDC

Two keys are employed in this scheme namely, K_R and K_C. The length of K_R and K_C are dependent on the dimensions of the image. For an algorithm to be resistant to brute-force attacks, the key space must be greater than 2100. For example, if the image is of size 512×1024, an 8-bit key value is generated by the KDC for every pixel. For the given image, K_R generated would be of 4096 bits and K_C generated would be of 8192 bits. This results in the key-space of 212,288 keys. This is an extremely larger number and is hence impossible to perform a brute-force attack for an image encrypted using this method. Even for an image of size 256×256, the key space is 2512, which is greater than 2100, which satisfies the criteria required to be resistant to brute-force attacks.

4.2.2. Standardization of dimensions of the image

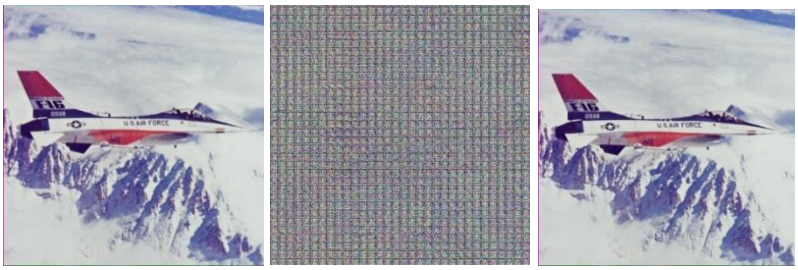
The key space for this method is dependent on the standard dimensions used by the algorithm. For an algorithm to be resistant to brute-force attacks, the key space must be greater than 2100. Consider the example of 500×1000 as the standard dimensions of the image. For the given dimensions, the KDC generates K_R and K_C, which would be valid for an entire session. Thus, the KDC would generate a 4000-bit K_R and 8000-bit K_C. This results in the key-space of 212,000 keys. Again, this is an extremely larger number and is hence impossible to perform a brute-force attack for an image encrypted using this method. Even with 150×150 as the standard dimensions, the key space is 2300, which is greater than 2100, which satisfies the criteria required to be resistant to brute-force attacks.

4.2.3. Encryption using a single session key

In this scheme, a single session key K_s is used for the encryption process. K_s is a set of sixteen 8-bit values. Hence, resulting in a single 128-bit key. The 128-bit key used in this scheme is irrespective of the dimensions of the image. The key space is hence 2128, which is approximately 340 undecillions. With current technology, it is estimated that it would take billions of years to exhaustively search the entire key space. Therefore, a 128-bit key is sufficient in preventing a brute-force attack.

4.3. Key sensitivity analysis

Key sensitivity analysis is a technique used to determine the impact of changes to a key on the output of a mathematical or statistical model. Here, we will observe the degree of difference in the encrypted image with respect to changes made in the key. In key sensitivity analysis, the model is run multiple times with different values for one or more input variables, while keeping all other variables constant. The results are then analyzed to determine how changes to each key affect the output of the model. Differential attacks are a type of cryptographic attack that exploit patterns in the differences between pairs of plaintexts and the corresponding cipher texts generated by a cipher. Firstly, the correct key is used for the encryption and decryption of the sample image. Fig. 12 shows the plain, encrypted and decrypted image using the correct key.



(a) (b) (c)

Fig. 12. Encryption and decryption: Plain image (a); Encrypted image (b); Decrypted image (c)

Secondly, a slight change in key values has been considered for the encryption and decryption process. Here, we consider a one-bit difference from the original key. Fig. 13 shows the images obtained after performing encryption using the correct and modified keys. Thirdly, the absolute difference in the encrypted image using the correct key and modified key is shown in Fig. 14.

From the analysis, we can conclude that the keys are highly sensitive and are hence resistant to differential attacks. Only the correct key can decrypt the encrypted image. The key sensitivity of the proposed scheme of communication of dimensions of the plain image and the proposed scheme of standardizing plain image dimensions is extremely high. The encryption key is a symmetric key; it is used only once and is truly random. There is no pattern whatsoever observed here. The key sensitivity analysis for the proposed scheme using a single session key is depicted in Figs 12-14. Further, the absolute intensity difference image in Fig. 14 shows that

there is significant difference in the encrypted image for a one-bit variation in the single session key. Hence, the proposed scheme is said to be resistant to differential attacks.

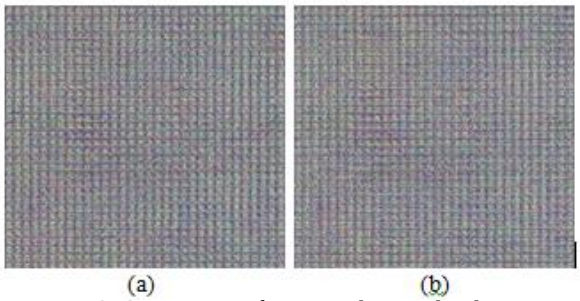


Fig. 13. Comparison of encrypted imaged : Plain image encrypted using correct key (a); Plain image encrypted using modified key (b)

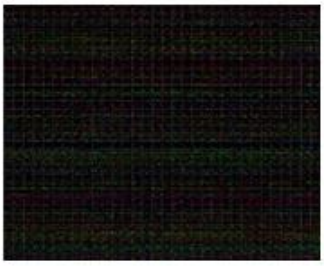
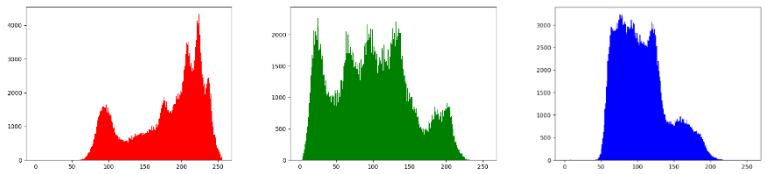


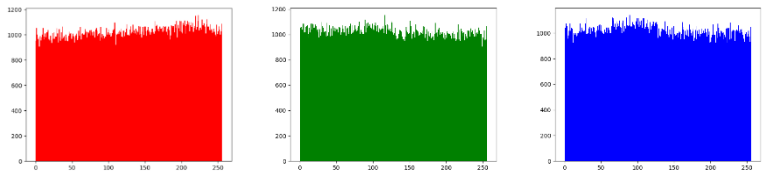
Fig. 14. Absolute intensity differences of encrypted images using correct and modified keys

4.4. Histogram analysis

For ensuring security, the input image and the encrypted image must not be statistically similar. Utilization of a histogram enables us to view the distribution of the image's pixel values. A more secure algorithm involves uniform distribution of pixel frequencies to provide adequate protection against statistical attacks. The red, green and blue color planes for the plain image and the encrypted image obtained using the different proposed schemes are shown in Figs 15-17. The variation in RGB values in the encrypted image is low as shown in Fig. 15, Fig. 16 and Fig. 17. The red, blue and green planes have been evaluated using histograms. From the experimental results, we can safely conclude that the given algorithm is resistant to statistical attacks.



(a)



(b)

Fig. 15. Histogram analysis for encryption using communication of dimensions – red, green, and blue color planes of image: Plain image (a); Encrypted image (b)

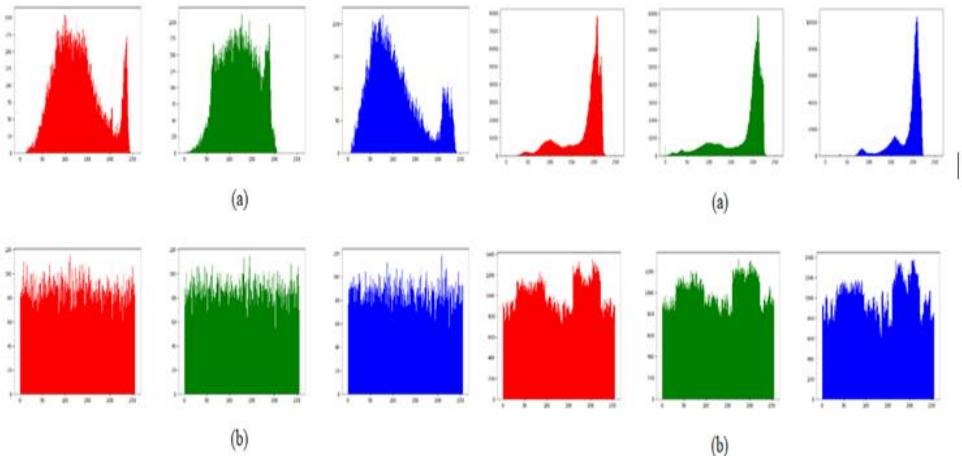


Fig. 16. Histogram analysis for encryption using standardization of dimensions – red, green and blue color planes of image: Plain image (a); Encrypted image (b)

Fig. 17. Histogram analysis for encryption using a single session key – red, green and blue color planes of image: Plain image (a); Encrypted image (b)

5. Conclusions

A secure peer to peer image sharing using Rubik's cube algorithm and KDC has been proposed in this paper. Use of KDC for key generation eliminates the overhead of key generation at the communicating nodes. Three different methods proposed use the Rubik's algorithm along with KDC, which can be used for different applications as required. Our main proposed method which uses single session key for image sharing can be used in cloud-based storages and also when using thin clients as well as social media platforms such as WhatsApp, Instagram because it can reduce both communication and processing overhead. The security analysis conducted on our proposed method proves that these methods have high security as well as low computation suitable for most of the applications. Overall, the proposed work helps in increasing the security of peer-to-peer image sharing using Rubik's cube algorithm along with KDC

References

1. Earthweb. How Much Data is Created Every Day? 2023 (Online). **https://earthweb.com/how-much-data-is-created-every-day/**
2. M u n e, R., S. A. B h u r a. An Analysis of Heterogeneous Data with Extreme Learning via Unsupervised Multiple Kernels. – In: Proc. of 2nd International Conference on Data, Engineering and Applications (IDEA'20), IEEE, 2020, pp. 1-7,
3. M h a t r e, M., H. K a s h i d, T. J a i n, P. C h a v a n. BCPIS: Blockchain-Based Counterfeit Product Identification System. – Journal of Applied Security Research, 2022, pp. 1-26 (Online).
4. M e h t a, K., G. D h i n g r a, R. M a n g r u l k a r. Enhancing Multimedia Security Using Shortest Weight First Algorithm and Symmetric Cryptography. – Journal of Applied Security Research, 2022, pp. 1-24. DOI: 10.1080/19361610.2022.2157193.

5. S a w a n t, V., A. S o l k a r, R. M a n g r u l k a r. Modified Symmetric Image Encryption Approach Based on Mixed Column and Substitution Box. – Journal of Applied Security Research, 2022, pp. 1-34.
6. V i d h y a, R., M. B r i n d h a. A Chaos-Based Image Encryption Algorithm Using Rubik's Cube and Prime Factorization Process (CIERPF). – Journal of King Saud University – Computer and Information Sciences, Vol. **34**, 2022, No 5, pp. 2000-2016. DOI: 10.1016/j.jksuci.2019.12.014.
7. K u m a r, A., M. R a n i. A Novel Chaotic Encryption Scheme for Images. – Chaos, Solitons & Fractals, Vol. **28**, October 2006, No 1, pp. 67-76. DOI: 10.1016/j.chaos.2005.05.022.
8. Z h a o, Y., R. M e n g, Y. Z h a n g, Q. Y a n g. Image Encryption Algorithm Based on a New Chaotic System with Rubik's Cube Transform and Brownian Motion Model. – Optik, Vol. **273**, 2023, 170342.
9. D e s h p a n d e, K., J. G i r k a r, R. M a n g r u l k a r. Security Enhancement and Analysis of Images Using a Novel Sudoku-Based Encryption Algorithm. – Journal of Information and Telecommunication, 2023, pp. 1-34
10. H e l m y, M., E. M. E l-R a b a i e, I. M. E l d o k a n y et al. 3D Image Encryption Based on Rubik's Cube and RC6 Algorithm. – 3D Res., Vol. **8**, 2017, No 38.
11. G o m a t h i, T., B. L. S. K u m a r. Multistage Image Encryption Using Rubik's Cube for Secured Image Transmission. – International Journal of Advanced Research in Computer Science, (S.l.), Vol. **6**, January 2017, No 6, p. 54-58. ISSN 0976-5697.
12. I o n e s c u, V. M., A. -V. D i a c o n u. Rubik's Cube Principle-Based Image Encryption Algorithm Implementation on Mobile Devices. – In: Proc. of 7th International Conference on Electronics, Computers and Artificial Intelligence (ECAI'2015), 2015, pp. P-31-P-34. DOI: 10.1109/ECAI.2015.7301247.
13. A b i t h a, K. A., P. K. B h a r a t h a n. Secure Communication Based on Rubik's Cube Algorithm and Chaotic Baker Map. – Procedia Technology, 2016.
14. A t t k a n, A., V. R a n g a, P. A h l a w a t. A Rubik's Cube Cryptosystem Based Authentication and Session Key Generation Model Driven in Blockchain Environment for IoT Security. – ACM Transactions on Internet of Things, 2023.
15. J i n, J., S. P a r k. Key Generation and Management Method Using AI Generated Rubik's Cube States. – In: Proc. of International Conference on Information Networking (ICOIN'2023), IEEE, 2023.
16. C h u m a n, T., W. S i r i c h o t e d u m r o n g, H. K i y a. Encryption-Then-Compression Systems Using Grayscale-Based Image Encryption for JPEG Images. – IEEE Transactions on Information Forensics and Security, 1-1, 2018. DOI:10.1109/tifs.2018.2881677.
17. A l s h e h r i, A. A., M. S. W o o n g. Cryptographic Algorithm for Image Encryption Based on Chaotic Map. – Mathematical Problems in Engineering, Vol. **2013**, February 2013, Article ID 848392. 10 p. DOI: 10.1155/2013/848392.

Received: 13.06.2023; Accepted: 29 .08.2023