International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

A Secure Medical Image Scrambling Using Argon2i, AES-CTR with Feedback-Based Diffusion

Ms. Lakshmamma T[1], Dr. Lakshmi J V N[2]
[1]Phd Scholar, School of Computer Science and Application, Reva University, Bangalore 560064, Karnataka, India.
[2]Associate Professor, School of Computer Science and Application, Reva University, Bangalore 560064, Karnataka, India.
Emails: *lakshmammat69@gmail.com[1], lakshmi.jvn@reva.edu.in[2]*

Abstract
The modern healthcare systems consider the secure transmission and storage of medical images as vital because of the sensitivity of patient data and the strict privacy regulations. This paper presents a secure and reversible RGB medical image scrambling and encryption scheme that aims to defend diagnostic images from unauthorized access, while, guaranteeing the integrity of the data. A 512- bit master key is generated through the memory-hard Argon2i algorithm which makes the system extremely resistant to brute-force and dictionary attacks. Domain-separated subkeys are generated using master key via AES in counter (CTR) mode. These subkeys are utilized at confusion and diffusion stages of the encryption and decryption process. The image confusion is performed with the use of a Fisher-Yates permutation which is controlled by cryptographically secure keystreams thereby disrupting spatial pixel relationships. The diffusion is done at both single colour channel and total colour channel levels utilizing a feedback-based mechanism that combines XOR operation, modular addition, and bit-level permutation. Both confusion and diffusion process ensures a strong avalanche effect across the entire image. Performance analysis shows that the proposed method provides an effective balance between security and computational efficiency, making it suitable for secure storage and transmission of medical images in telemedicine and healthcare information systems.
Keywords: *AES-CTR; Argon2i; Feedback-Based Diffusion; Medical Image encryption; Medical Image scrambling.*

1. Introduction
Healthcare systems are experiencing a rapid shift toward digital technologies. As part of this transformation, medical imaging has become central to clinical practice, with widespread use of endoscopic imaging, magnetic resonance imaging (MRI), computed tomography (CT), ultrasound, treatment planning, and long-term storage of patient records. These digital images play a vital role in supporting clinical decision-making. However, their increasing use has also introduced significant concerns related to data security and patient privacy. Unauthorized access, modification, or manipulation of these images constitutes a serious breach of privacy and may result in misdiagnosis, compromised treatment decisions, or legal consequences. For these reasons, protecting medical images has become an essential requirement in modern healthcare environments. Ensuring confidentiality, integrity, and authenticity during both storage and transmission is now a critical component of medical information systems. Conventional cryptographic techniques, which were primarily developed for text-based data, are not always suitable for securing medical images, because medical images are typically large in size and exhibit high redundancy and strong spatial correlation between adjacent pixels. Even a minimal loss or distortion of image details can adversely affect diagnostic accuracy. This has led to the development of image-specific encryption techniques. Confusion and diffusion is one of the image scrambling techniques used to effectively disrupt pixel

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

relationships in images to prevent the unauthorized interpretation of the image [12].

1.1. Literature Review

There has been a lot of research on scrambling-diffusion image encryption schemes, and the majority of these schemes rely on chaos for their pseudo-randomness. The initial surveys point out that chaos maps are very effective in image scrambling and encryption [1]. The chaotic image scrambling techniques such as those based on the Josephus cycle [13, 14], logistic map [23], Rossler system [21], and hyper-chaotic systems with multiple control parameters [15, 16] have been proposed for securing the images using chaos map features. Moreover, some researchers have researched on of employing DNA encoding and molecular mutation concepts as image encryption to increase the security level. Among the methods are double scrambling combined with DNA row–column operations [2], DNA-based key scrambling [3], multi-stream scrambling with DNA encoding [4], and closed-loop diffusion with DNA mutation [5]. They've paved the way for a significant resistance against the two most common attacks, namely statistical and differential ones. Also, medical-image encryption systems employing hyper-chaotic systems and DNA coding are reported to have very strong security performance for sensitive clinical data [6,25,26]. Additionally, biologically inspired approaches such as protein-chain-based cryptography have been explored to enhance cryptographic complexity [17]. At the same time, lightweight and block-based encryption algorithms were created to solve the problem of limited resources in IoT and telemedicine applications. These algorithms includes the multi-round confusion-diffusion cryptosystems [7], and spatio-color scrambling techniques compatible with JPEG [8], device-constrained color image encryption [9], chaos-based encryption combined with JPEG compression [10]. More advanced designs incorporate complex network scrambling and multi-directional diffusion [11], bit-level encryption with fully connected networks [12], and hyper-chaotic self-adaptive diffusion mechanisms [18]. New inventions in image security research have targeted deep learning, diffusion models, and hybrid encryption–compression frameworks as recent trends. These include diffusion-model-based image protection [28,29], deep-learning-assisted block-scrambling encryption for remote sensing images [30], CNN-assisted scrambling and steganography [31], and robust compression–encryption using scrambled block sampling [19].

1.2. Research Gap

Although there are a variety of image encryption methods available, there are still some limitations that can be highlighted. A lot of chaos-based algorithms are dependent on floating-point arithmetic, which makes these algorithms subject to the limitations of precise representation of numbers, numerical instability, and behavior that is dependent on the specific implementation. All these things can cause a reduction in the size of the possible key space and the ability to reproduce the results. Another problem is that a lot of schemes directly use user passwords or chaotic parameters as keys without proper key derivation methods, leaving them open to related-key, dictionary, and brute-force attacks. Besides that, even though the DNA-based and hyper-chaotic methods provide a remarkable amount of nonlinearity and diffusion strength, they are still quite computationally expensive compared to other methods. In addition, some of the lightweight encryption methods provide low-security levels in order to be efficient while others provide high security but not fast enough for real-time applications. All these drawbacks call for an encryption framework for medical images that is not only cryptographically secure but also efficient and lossless, and at the same time incorporates image-specific operations together with modern key management techniques.

1.3. Objectives

To address the above research gaps, this research aims to develop a secure, lossless, and efficient medical image encryption scheme that integrates modern cryptographic techniques with image-oriented confusion and diffusion mechanisms. The specific objectives of this research are as follows:

- To design a robust password-based master key generation using argon 2i, a password-

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

based key derivation function to resist brute-force and side channel attacks.

- To derive independent, domain-separated subkeys from the master key using HKDF (HMAC-based Key Derivation Function) with SHA-256 for using different keys for confusion and diffusion stages.
- To implement a pixel position scrambling method – confusion stage - for eliminating spatial correlations in medical images by using a secure permutation method.
- To develop a multi-stage diffusion framework**,** including channel-wise and combined RGB diffusion with nonlinear feedback and bit-level permutation, so that a strong avalanche effect can be assured.
- To guarantee perfect lossless decryption, to ensure diagnostic integrity and clinical reliability of medical images is preserved.

2. Method

This study is based on medical image scrambling, which involves scrambling medical images through confusion and diffusion processes.

2.1.Relevant Knowledge

2.1.1. Argon 2i and HKDF-Based Key Derivation

The proposed image encryption technique uses Argon2i and HKDF-Based Derivation mode for Master key and 5 Sub keys generation. Argon2i is a memory-hard password hashing function. It is designed for strong defense against side-channel attacks, making it ideal for key derivation [32, 33]. It generates master key based on the password provided by the user and a randomly selected salt.

General form

Argon2i(password, salt, time_cost, memory_cost, parallelism, hash_length)

Parameter Description

- **password**: User provided password
- **salt**: Random value combined with password to improve security
- **time_cost (t)**: Number of iterations over memory
- **memory_cost (m)**: Amount of memory used, typically in kilobytes
- **parallelism (p)**: Number of parallel lanes or threads
- **hash_length (l)**: Desired length of the output key in bytes

The addition of a random salt ensures that even two perfectly matching passwords will not yield the same master keys, thus stopping the precomputation and rainbow-table attacks. In the process of using Argon2i, a constant 256-bit master key is produced. HMAC-based Key Derivation Function (HKDF) is used to create a number of domain separated subkeys that are independent from each other and derived from master key.

General Form

hkdf = HKDF(hash, length, salt,info)
subkey = hkdf.derive(master_key)

HKDF Parameters

- **hash** : underlying hash function. We used SHA256 as its collision resistance and pseudorandom properties ensure secure key expansion.
- **length:**
  Specifies the length (in bytes) of the derived subkey. In this project, a 32-byte (256-bit) output is used for all cryptographic keys.
- **salt = None:**
  When no salt is provided, HKDF uses an implicit zero-valued salt. This is acceptable in this design because the input key material (the Argon2i-derived master key) already has high entropy and includes a salt from the Argon2i stage.
- **info:**
  The info parameter is a context-specific identifier used for domain separation. The info parameter is used to derive distinct subkeys for different stages of the encryption algorithm as given below,

Kc = HKDF(master_key, info = "confusion")

2.1.2. AES in Counter (CTR) Mode

AES in Counter (CTR) mode is a symmetric-key encryption mode that transforms the Advanced Encryption Standard (AES) block cipher into a stream cipher. AES can be initialized in Counter (CTR) mode using a user-specified key and nonce. This construction transforms AES into a stream

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

cipher capable of generating a cryptographically secure keystream. The keystreams generated using AES-CTR act as pseudorandom sequences which are using in confusion and diffusion phases [34]. In this research, a counter object is created with a total size of 64 bits and is initialized with an explicit nonce as its prefix. The nonce occupies the most significant 64 bits of the input block, while the remaining 64 bits are used as an incrementing counter starting from zero. This results in a 128-bit input block, which matches the AES block size. Formally, each counter block input to AES is constructed as:

- Input_block_i = nonce || counter_i
- where counter_i is a 64-bit integer that increments sequentially for each block, and || denotes concatenation.

The AES encryption function is then applied to each input block using the secret key to generate a keystream block. The resulting keystream is used for encryption or decryption by XORing it with the data stream. The explicit use of a nonce ensures that each AES-CTR instance produces a unique keystream, even when the same key is reused.

2.1.3. Fisher–Yates algorithm

In the proposed image encryption algorithm, the Fisher–Yates permutation is employed during the confusion phase to achieve strong spatial scrambling of image pixels. The Fisher–Yates algorithm generates a uniformly random permutation of a finite sequence. The following operations performed for the Fisher–Yates Permutation

- The input image is flattened into a one-dimensional array to allow global permutation across all pixel positions.
- Random indices for the permutation are derived from an AES-CTR–based keystream, providing cryptographically secure randomness.
- A unique nonce is used to initialize AES-CTR, ensuring permutation uniqueness across encryption sessions.
- Rejection sampling is applied during index selection to eliminate modulo bias and guarantee uniform distribution.
- During each iteration of the algorithm, a random index is selected from a shrinking range and swapped with the current element.
- The resulting permutation vector is stored as encryption metadata and inverted during decryption to recover the original pixel ordering.

2.2.Encryption Process

Encryption of image is achieved by conducting experiments on the endoscopic tested medical images. These images are center-cropped to a square region and resized to 256X256 pixels using bicubic interpolation. The following procedure is followed during the process.

2.2.1. Generation of master and sub key

Step 1. Salt Generation

- A 128-bit random salt is generated using a cryptographically secure random number generator:
- salt ∈ {0,1}^128
- The salt ensures that identical passwords result in distinct master keys and provides resistance against precomputation and rainbow-table attacks. The salt is stored as part of the encryption metadata.

Step 2. Master Key Derivation

- Using the input password and the generated salt, a master key is derived through a memory-hard key derivation function:
- master = Derive_Master_Key(password, salt)
- This function applies Argon2i with fixed time, memory, and parallelism parameters to produce a 256-bit master key.

Step 3. Domain-Separated Subkey Derivation

- From the master key, multiple independent subkeys are derived using a key derivation function with domain separation.
- Kc = Derive_Subkey(master, "confusion")
  Kr = Derive_Subkey(master, "diffusion-R")
  Kg = Derive_Subkey(master, "diffusion-G")
  Kb = Derive_Subkey(master, "diffusion-B")
  Krgb = Derive_Subkey(master, "diffusion-RGB")
- Image encryption is done in two phases: The Confusion Phase (Phase 1) and the Diffusion Phase (Phase 2).

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

2.2.2. Phase 1- Confusion Phase

During the confusion phase, a Fisher–Yates permutation is constructed based on the total pixels of the image. The following steps are followed.

Step 1. Nonce generation

- A nonce is generated randomly
- perm_nonce ∈ {0,1}^64
- This nonce ensures that a unique permutation is produced when the same key is reused and it will be stored as part of the encryption metadata

Step 2. AES-CTR Initialization

- Using the confusion subkey Kc and the generated nonce perm_nonce, an AES cipher is initialized in counter (CTR) mode:
- cipher = AES_CTR(Kc, perm_nonce)

Step 3. Permutation Array Initialization

for i = 0 to n − 1
perm[i] = i,

where n denotes the total number of image elements (n = image.size).

Step 4. Keystream Buffering

- To improve computational efficiency, a fixed-size keystream buffer is generated by encrypting a zero-filled byte array:
- BUFFER_SIZE = 1024 × 16 bytes buffer = AES_CTR(Kc, perm_nonce, BUFFER_SIZE)

Step 5. Fisher–Yates Shuffle with Rejection Sampling (To perform shuffling as shown in Figure 1)

For each index i from n − 1 down to 1
"A 32-bit random value is extracted from the keystream buffer:"
rnd = buffer[offset : offset + 4]
"To eliminate modulo bias, the extracted random value is accepted only if it satisfies the condition. If the condition is not met, the value is discarded and a new random value is drawn."

$$rnd < 2^{32} - (2^{32} \bmod (i + 1)) \quad (1)$$

"Once an acceptable random value is obtained, an unbiased index j is computed as"

$$j = rnd \bmod (i + 1) \quad (2)$$

"The elements at positions i and j in the permutation array are swapped"

$$perm[i] \leftrightarrow perm[j] \quad (3)$$

Step 6. Image Flattening

The input image is first converted into a one-dimensional array to enable global permutation across all pixel positions:

- image_flat = flatten(image)

Step 7. Permutation Application

The flattened image is permuted using the generated permutation vector perm as follows:

for i = 0 to n – 1
confused_flat[i] = image_flat[perm[i]] (4)

where n denotes the total number of elements in the image. Figure 1 shows Confusion stage. (a) Pixel Positions Before Confusion. (b) Pixel Positions After Confusion

Step 8. Image Reshaping

The permuted one-dimensional array is reshaped back into the original image dimensions to obtain the spatially confused image:

Ic = reshape(confused_flat, shape(image)) (5)

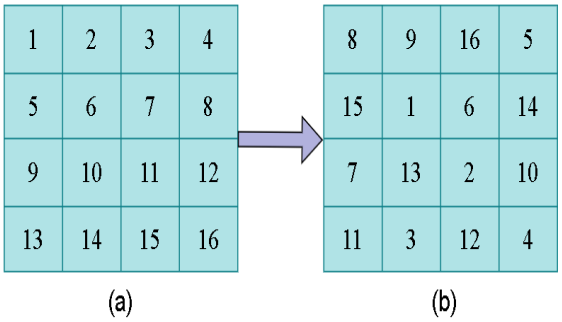

Figure 1 Confusion stage. (a) Pixel Positions Before Confusion. (b) Pixel Positions After Confusion

2.2.3. Phase 2- Diffusion Phase

After confusion, diffusion is done on confused image in two stages: one on a channel-wise diffusion and another on the diffusion of the whole RGB channels at once. For channel-wise diffusion, the red, green, and blue channels are processed independently. Each channel employs a unique AES-CTR keystreams generated using its corresponding subkey and nonce. The detailed steps of diffusion can be described as follows

Step 1.

The confused image – Ic - is decomposed into its three color channels as follows:

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

R = Ic(:,:,1)
G = Ic(:,:,2)
B = Ic(:,:,3)

Step 2.

Using the corresponding diffusion keys Kr, Kg, and Kb, independent AES-CTR keystreams are generated for each channel. The keystream generation is defined as:

ksR = AES_CTR(Kr, r_nonce, |R|)
ksG = AES_CTR(Kg, g_nonce, |G|)
ksB = AES_CTR(Kb, b_nonce, |B|)

where |R|, |G|, and |B| denote the number of pixels in the respective channels.

Step 3.

Build lookup table using BIT_LUT function

A fixed bit-permutation vector is defined as: perm = [2, 5, 1, 7, 0, 3, 6, 4].

A lookup table lut of size 256 is constructed

for x ← 0 to 255 do
  y ← 0
  for i ← 0 to 7 do
    "Extract the bit at position perm[i] from x"
    bit ← (x >> perm[i]) AND 1 (6)
    "Place this bit into position i of y"
    y ← y OR (bit << i) (7)
  end for
 "After processing all 8 bit positions, the resulting value y represents the bit-permuted version of x and is stored in the lookup table"
  lut[x] ← y
 end for

This lut lookup table is used in process of diffusion operation of single and merged RGB channels

Step 4.

Each color channel is flattened into a one-dimensional array.

For a flattened channel X in {R, G, B}, the diffusion operation is performed sequentially as follows:

$u_i = x_i \text{ XOR } k_i$ (8)

$v_i = (u_i + k_i + y_{(i-1)}) \bmod 256$ (9)

$y_i = BIT\_LUT[v_i]$ (10)

with the initial condition:

$y_{(-1)} = 0$.

Here, x_i represents the i-th pixel of the flattened channel, k_i is the corresponding keystream byte, $y_{(i-1)}$ is the previous encrypted output (feedback), and y_i is the encrypted output pixel.

Step 5.

The encrypted channels are merged to form the final encrypted image as:

merged = stack(R', G', B'),

where R', G', and B' denote the diffused red, green, and blue channels, respectively.

To further enhance inter-channel diffusion, a final combined diffusion stage is applied. First, a new independent 64-bit random nonce is generated:

$rgb\_nonce \in \{0,1\}^{64}$.

Using the RGB diffusion key Krgb and the generated nonce, an AES-CTR keystream is produced:

ksRGB = AES_CTR(Krgb, rgb_nonce, |merged|),

where |merged| denotes the total number of elements in the merged RGB image.

The merged image is then flattened into a one-dimensional array:

m_i = flatten(merged),

and encrypted using the same diffusion mechanism as:

$a_i = m_i \text{ XOR } k_i$ (11)

$b_i = (a_i + k_i + c_{(i-1)}) \bmod 256$ (12)

$c_i = BIT\_LUT[b_i]$ (13)

with the initial condition:

$c_{(-1)} = 0$,

where k_i denotes the i-th byte of ksRGB and BIT_LUT is the bit-permutation lookup table.

Finally, the diffused output is reshaped back to the original image dimensions to obtain the final encrypted image:

final = reshape(m_i, shape(merged)).

Final encrypted image, all nonces used in the encryption process, and the salt is retained as part of encryption metadata.

Algorithm for the Encryption of Image is as follows:

Algorithm 1. Encrypt_Image(image, password)

salt ← RandomBytes(16)
master ← Derive_Master_Key(password, salt)

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

```
    Kc ← Derive_Subkey(master, "confusion")
    Kr ← Derive_Subkey(master, "diffusion-R")
    Kg ← Derive_Subkey(master, "diffusion-G")
    Kb ← Derive_Subkey(master, "diffusion-B")
    Krgb ← Derive_Subkey(master, "diffusion-RGB")

    (perm, perm_nonce) ← Fisher_Yates_Permutation(size(image), Kc)

    confused ← Image is permuted according to generated permutation

    R ← confused image is decomposed into its red channel
    G ← confused image is decomposed into its green channel
    B ← confused image is decomposed into its blue channel

    r_nonce ← RandomBytes(8)
    g_nonce ← RandomBytes(8)
    b_nonce ← RandomBytes(8)

    ksR ← AES_CTR_Cipher(Kr, r_nonce, Length(R))
    ksG ← AES_CTR_Cipher(Kg, g_nonce, Length(G))
    ksB ← AES_CTR_Cipher(Kb, b_nonce, Length(B))

    BIT_PERM ← [2, 5, 1, 7, 0, 3, 6, 4]
    BIT_LUT ← Build_LUT(BIT_PERM)

    R ← Diffuse_Encrypt(Flatten(R), ksR, BIT_LUT)
    G ← Diffuse_Encrypt(Flatten(G), ksG, BIT_LUT)
    B ← Diffuse_Encrypt(Flatten(B), ksB, BIT_LUT)

    merged ← Stack(R, G, B)

    rgb_nonce ← RandomBytes(8)
    ksRGB ← AES_CTR_Cipher(Krgb, rgb_nonce, Length(merged))

    cipher ← Diffuse_Encrypt(Flatten(merged), ksRGB, BIT_LUT)
    return {cipher, perm, perm_nonce, r_nonce, g_nonce, b_nonce, rgb_nonce, salt}
End Algorithm
```

```
Function Derive_Master_Key(password, salt)
    master ← Argon2i(
                password,
                salt,
                time_cost = 3,
                memory_cost = 65536,
                parallelism = 2,
                hash_len = 32
            )
    return master
End Function
```

```
Function Derive_Subkey(master, info)
    key ← HKDF_SHA256(master, info, length = 32)
    return key
End Function
```

```
Function AES_CTR_Cipher(key, nonce, L)
    ctr ← InitializeCounter(64, nonce, 0)
    cipher ← AES(key, mode = CTR, counter = ctr)
    keystream ← Encrypt(cipher, ZeroBytes(L))
    return keystream
End Function
```

```
Function Fisher_Yates_Permutation(n, key)
    perm ← [0, 1, 2, ..., n−1]
    nonce ← RandomBytes(8)
    keystream ← AES_CTR_Cipher(key, nonce, buffer_size)
    for i ← n−1 downto 1 do
        j ← UnbiasedRandom(keystream, i + 1)
        swap(perm[i], perm[j])
    end for
    return (perm, nonce)
```

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

```
End Function
```

```
Function Build_LUT(perm[8])
   for x ← 0 to 255 do
      y ← 0
      for i ← 0 to 7 do
         bit ← (x >> perm[i]) AND 1
         y ← y OR (bit << i)
      end for
      lut[x] ← y
   end for
   return lut
End Function
```

```
Function Diffuse_Encrypt(arr, k, lut)
   prev ← 0
   for i ← 0 to Length(arr) − 1 do
      x ← arr[i] XOR k[i]
      x ← (x + k[i] + prev) mod 256
      out[i] ← lut[x]
      prev ← out[i]
   end for
   return out
End Function
```

2.3.Decryption Process

Image decryption is done in reverse order which consists of two phases: Diffusion Phase (Phase 1) and the Confusion Phase (Phase 2).

2.3.1. Diffusion Phase

The diffusion phase consists of successively undoing the combined RGB diffusion, the channel-wise diffusion. The following steps are followed during diffusion

Step1.Key generation

The encrypted metadata contains the cipher image, permutation, salt, and all nonces required for decryption. Using the input password and the stored salt, the master key is regenerated:

master = Derive_Master_Key(password, salt)

From the master key, the same set of subkeys used during encryption are re-derived:

Kc = Derive_Subkey(master, "confusion")
Kr = Derive_Subkey(master, "diffusion-R")
Kg = Derive_Subkey(master, "diffusion-G")
Kb = Derive_Subkey(master, "diffusion-B")
Krgb = Derive_Subkey(master, "diffusion-RGB")

These keys ensure cryptographic consistency between encryption and decryption.

Step 2. Reverse Combined RGB Diffusion

The cipher image is first processed to reverse the final combined RGB diffusion stage applied during encryption.

Using the stored RGB nonce (rgb_nonce) and the RGB diffusion key Krgb, an AES-CTR keystream is generated as:

ksRGB = AES_CTR(Krgb, rgb_nonce, |cipher|)

The cipher image is flattened into a one-dimensional array and decrypted using the inverse diffusion process with the inverse lookup table

BIT_PERM = [2, 5, 1, 7, 0, 3, 6, 4]
INV_BIT_PERM = [0,0,0,0,0,0,0,0]
for i ← 0 to 7 do
p ← BIT_PERM[i] (14)
INV_BIT_PERM[p] ← I (15)
end for

INV_BIT_LUT = Build_LUT((INV_BIT_PERM)

c_i = cipher_i
b_i = INV_BIT_LUT[c_i] (16)
a_i = (b_i − k_i − d_(i−1)) mod 256 (17)
m_i = a_i XOR k_i (18)

with the initial condition:

d_(−1) = 0

Here, k_i denotes the i-th byte of ksRGB, and d_(i−1) represents the previous decrypted output used as feedback.

The decrypted array is then reshaped back to the original image dimensions to obtain the merged RGB image:

merged = reshape(m_i, shape(cipher))

Step 3. Reverse Channel-wise Diffusion

The merged RGB image is decomposed into its three color channels:

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

R = merged(:,:,1)
G = merged(:,:,2)
B = merged(:,:,3)
using the stored nonces r_nonce, g_nonce, and b_nonce, AES-CTR keystreams are regenerated for each channel:
ksR = AES_CTR(Kr, r_nonce, |R|)
ksG = AES_CTR(Kg, g_nonce, |G|)
ksB = AES_CTR(Kb, b_nonce, |B|)
Each channel is flattened and decrypted independently using the inverse diffusion process:
For a flattened channel X ∈ {R, G, B}:
c_i = encrypted_pixel
b_i = INV_BIT_LUT[c_i] (19)
a_i = (b_i − k_i − d_(i−1)) mod 256 (20)
x_i = a_i XOR k_i (21)
with the initial condition:
d_(−1) = 0
Here, x_i represents the recovered pixel value of the channel, k_i is the corresponding keystream byte, and d_(i−1) is the feedback term.
After decryption, each channel is reshaped back to its original dimensions.

Step 4. Channel Merging

The decrypted red, green, and blue channels are merged to reconstruct the spatially confused image:
merged = stack(R, G, B)

2.3.2. Confusion Phase

Reverse confusion phase restores the original spatial arrangement of the image pixels by applying the inverse of the permutation on the merged image generated in the previous stage.

Step 1. Inverse Permutation Construction

The encrypted metadata contains the permutation array perm that was used during the confusion stage of encryption. To reverse this operation, the inverse permutation array inv_perm is constructed such that each index is mapped back to its original position.
The inverse permutation is computed as follows:
for i = 0 to n − 1
inv_perm[perm[i]] = I (22)
where n denotes the total number of elements in the image. This operation ensures that inv_perm represents the exact inverse mapping of the original permutation perm.

Step 2. Application of Inverse Permutation

The spatially diffused image merged is first flattened into a one-dimensional array:
merged_flat = flatten(merged)
The inverse permutation is then applied to reorder the pixel elements back to their original positions:
for i = 0 to n − 1
original_flat[i] = merged_flat[inv_perm[i]] (23)
where n denotes the total number of elements in the image.

Step 3. Image Reshaping

Finally, the permuted one-dimensional array is reshaped back into the original image dimensions to reconstruct the decrypted image:
original = reshape(original_flat) (24)

Algorithm for decrypting the image is as follows:

Algorithm 2. Decrypt_Image(bundle, password)

```
bundle = retrieve_from_storage()
cipher ← bundle["cipher"]
perm   ← bundle["perm"]
salt   ← bundle["salt"]

// Key regeneration

master ← Derive_Master_Key(password, salt)

Kc   ← Derive_Subkey(master, "confusion")
Kr   ← Derive_Subkey(master, "diffusion-R")
Kg   ← Derive_Subkey(master, "diffusion-G")
Kb   ← Derive_Subkey(master, "diffusion-B")
Krgb ← Derive_Subkey(master, "diffusion-RGB")

// Reverse combined RGB diffusion

ksRGB ← AES_CTR_Cipher(Krgb, bundle["rgb_nonce"], Length(cipher))

BIT_PERM ← [2, 5, 1, 7, 0, 3, 6, 4]
INV_BIT_PERM ← [0,0,0,0,0,0,0,0]
for i ← 0 to 7 do
   p ← BIT_PERM[i]
   INV_BIT_PERM[p] ← i
end for
```

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

```
    INV_BIT_LUT ← Build_LUT(INV_BIT_PERM)

    merged ← Diffuse_Decrypt(Flatten(cipher), ksRGB, INV_BIT_LUT)

    merged ← Reshape(merged, Shape(cipher))

    // Reverse channel-wise diffusion

    R ← merged[:,:,0]
    G ← merged[:,:,1]
    B ← merged[:,:,2]

    ksR ← AES_CTR_Cipher(Kr, bundle["r_nonce"], Length(R))
    ksG ← AES_CTR_Cipher(Kg, bundle["g_nonce"], Length(G))
    ksB ← AES_CTR_Cipher(Kb, bundle["b_nonce"], Length(B))

    R ← Diffuse_Decrypt(Flatten(R), ksR, INV_BIT_LUT)
    G ← Diffuse_Decrypt(Flatten(G), ksG, INV_BIT_LUT)
    B ← Diffuse_Decrypt(Flatten(B), ksB, INV_BIT_LUT)

    R ← Reshape(R, Shape(merged[:,:,0]))
    G ← Reshape(G, Shape(merged[:,:,1]))
    B ← Reshape(B, Shape(merged[:,:,2]))

    merged ← Stack(R, G, B)

    // Reverse confusion (inverse permutation)
    inv_perm ← Inverse_Permutation(perm)

    original ← Flatten(merged)[inv_perm]
    original ← Reshape(original, Shape(merged))
    return original
End Algorithm
```

```
Function Inverse_Permutation(perm)
    inv_perm ← Array of same length as perm
    for i ← 0 to Length(perm) − 1 do
        inv_perm[perm[i]] ← i
    end for
    return inv_perm
End Function
```

```
Function Diffuse_Decrypt(arr, k, inv_lut)
    prev ← 0
    for i ← 0 to Length(arr) − 1 do
        x ← inv_lut[arr[i]]
        x ← (x − k[i] − prev) mod 256
        out[i] ← x XOR k[i]
        prev ← arr[i]
    end for
    return out
End Function
```

Flowchart for Encryption and Decryption image is shown in Flowchart 1 and Flowchart 2.

3. Performance and Security Evaluation Results

In this section, we will check our encryption system in terms of key space analysis, key sensitivity analysis, histogram, entropy, correlation coefficient, NPCR, UACI, PSNR and other values. Table 1 shows the hardware, the software environment and the image source. Three images - img1, img2, and img3 – are used, which are related to endoscopy test images of a medical nature.

3.1.Key Space

Table 1 Specification Table

Specification	
Processor	12th Gen Intel(R) Core(TM) i5-12400 (2.50 GHz)
RAM	16.0 GB
Operating system	Windows 11 Home
Programming language	Python
Data source	The Kvasir Dataset [35]

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

The proposed scheme derives a 256-bit master key using Argon2i and HKDF, resulting in a cryptographic key space of 2256. With this, our scheme provides resistance against brute-force attacks due to its effective key space exceeding 2128. Table 1 shows Specification Table

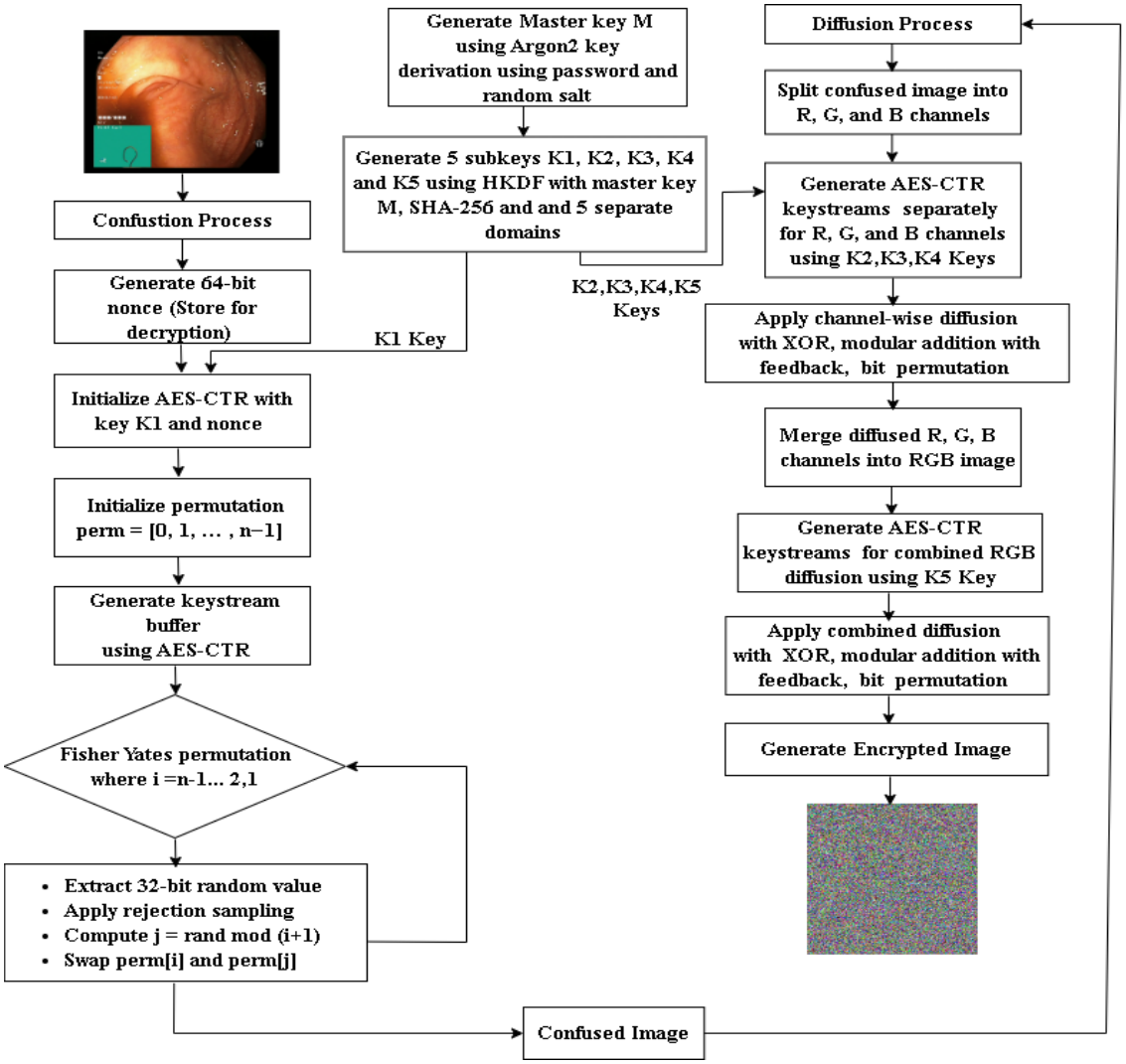

Figure 2 Image Encryption Process

3.2.Key sensitivity analysis

Key sensitivity analysis is a method used to assess the extent to which an encryption algorithm is influenced by minor changes made to the secret key. A secure algorithm shows high key sensitivity, to the change of one bit in the key, resulting in an entirely different cipher image. This is of utmost important as it stops the attackers from taking advantage of the similarities between the corresponding keys and guarantees the resistance to both brute-force and related-key attacks.

Two tests are performed by giving correct key in the first time, where

K1=
9**c**7480080e302ac765f37f97706fe6761d231b5783
b8e13513b88061fac2826

Second test is performed by giving wrong key (one bit change), where

K2=
9**d**7480080e302ac765f37f97706fe6761d231b5783b
b8e13513b88061fac2826

The Figure.2 illustrates the key sensitivity of the proposed encryption scheme: the encrypted image

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

appears random (Figure 2.b), decrypting the image with correct key –K1 restores the original image (Figure 2 c). However, decrypting with a slightly incorrect key –K2 produces a noise-like image with no recognizable structure (Figure 2 d).

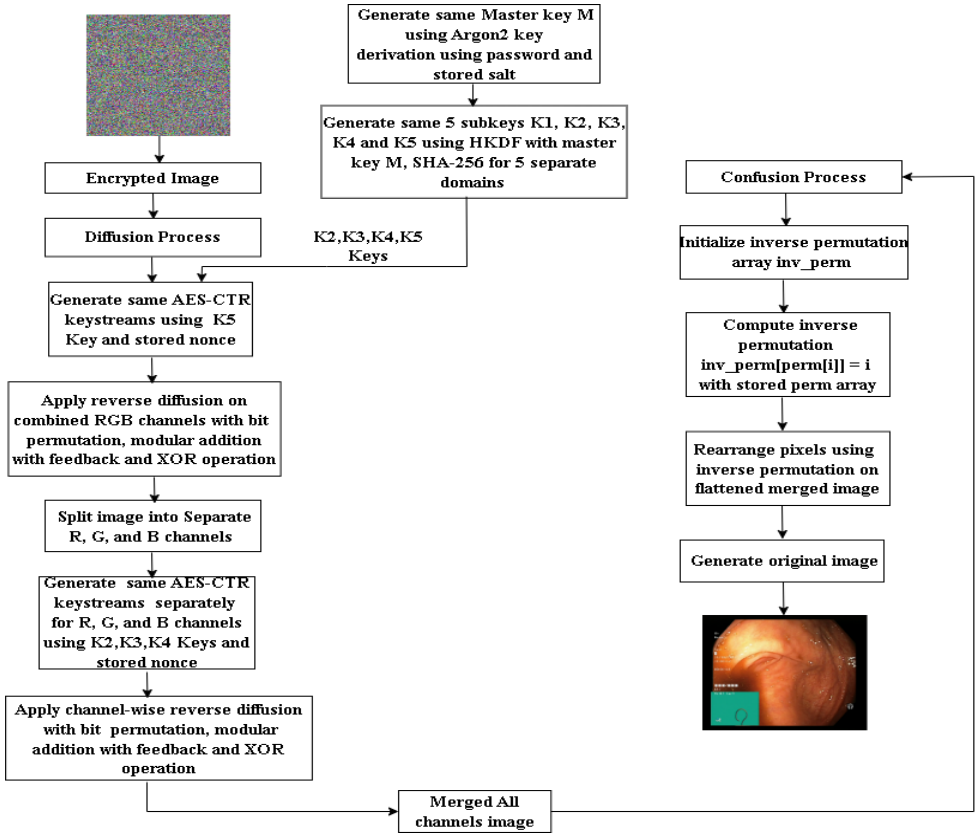

Figure 3 Image Decryption Process

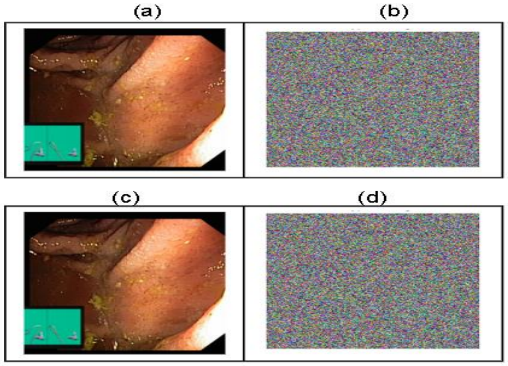

Figure 4 (a) Original Image (b) Encrypted Image (c) Decrypted Image with Correct Key (d) Decrypted Image with Wrong Key

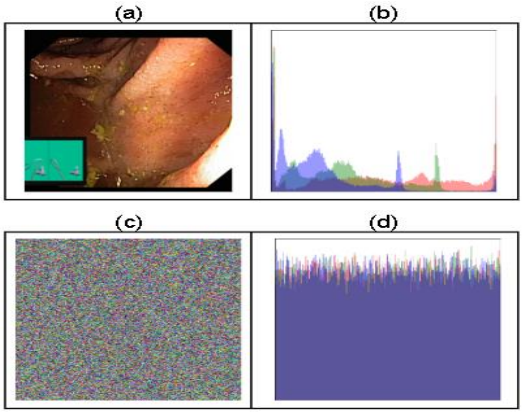

Figure 5 (a) Plain image (b) Histogram of plain image (c) Encrypted image (d) Histogram of encrypted image

3.3. Histogram Analysis

Histogram analysis studies the distribution of pixel intensity values before and after encryption [2].

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

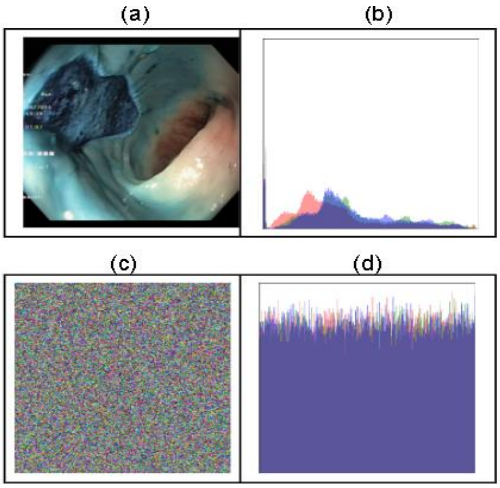

Figure 6 (a) Plain image img2 (b) Histogram of plain image (c) Encrypted image (d) Histogram of encrypted image

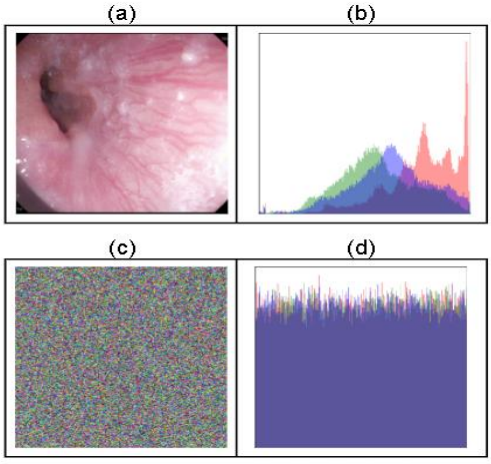

Figure 5 (a) Plain image img3 (b) Histogram of plain image (c) Encrypted image (d) Histogram of encrypted image

Referring to the Figure 3, 4 and 5, in the plain image, the histograms of all three channels are unevenly distributed with distinct peaks, reflecting the strong spatial and statistical correlations. After encryption, the histograms of the encrypted image are uniformly distributed. As a result, the encrypted image does not preserve any statistical characteristics of the original image. This behaviour effectively hides pixel intensity information and offers excellent resistance against histogram-based and statistical attacks. The Chi-square (χ^2) tests are done on histograms to confirm the effectiveness of image encryption against frequency attacks. This test checks if the histogram of an encrypted image follows a uniform distribution. The formula for the Chi-square (χ^2) test is shown below.

$$\text{Chi\_square} = \sum_{i=0}^{255} \frac{(O(i) - E(i))^2}{E(i)} \quad (25)$$

Where:

- O(i) = observed frequency of pixel value i
- E(i) = expected frequency of pixel value i
- i = 0 to 255 for an 8-bit image

The p-value is used as an another indicator to shows whether the observed Chi-square value significantly differs from a uniform distribution. For a secure encryption algorithm, the p-value should be greater than 0.05. In this case, the encrypted image acts like random noise. The Table 2 displays the Chi-square values and the corresponding p-values obtained from the histogram analysis of the encrypted images. For an 8-bit image, the expected Chi-square value for a uniform distribution is about equal to the degrees of freedom, which is 255. The observed Chi-square values (239.95, 265.42, and 268.01) are all close to this expected value, showing that the pixel intensity distributions of the encrypted images resemble a uniform distribution. The corresponding p-values (0.7423, 0.3139, and 0.2755) are all significantly higher than the commonly used significance threshold of 0.05. This indicates that the null hypothesis of a uniform distribution cannot be rejected for any of the test images. The consistently high p-values further confirm that the encrypted images show strong randomness and do not reveal statistical information through their histograms.

Table 2 Chi Square and p - Values

Image	Chi-square	p-value
img1	239.9453	0.7423
img2	265.4166	0.3139
Img3	268.0104	0.2755

3.4.Correlation analysis

In medical images, adjacent pixels (horizontal, vertical, and diagonal) are highly correlated due to spatial continuity. This correlation can be exploited

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

by attackers if it is preserved after encryption. Correlation analysis is used to evaluate how effectively the proposed encryption scheme eliminates the strong dependency between neighboring pixels in an image. For the measurement of this property, corresponding pixel pairs of adjacent pixels are selected in the horizontal, vertical, and diagonal directions, and the correlation coefficient is computed using the standard formula:

$$r = \frac{\mathrm{Cov}(x,y)}{\sqrt{\{\mathrm{Var}(x) * \mathrm{Var}(y)\}}} \qquad (26)$$

where x and y represent the color intensity values of two neighboring pixels [3]. Strong similarity between neighboring pixels is indicated by correlation coefficients for the plain image, which are usually close to 1. On the other hand, as shown in Figure 6, and Table 3, the encrypted image produced by the proposed scheme exhibits correlation coefficients close to 0 in all directions and across all RGB channels. This confirms the successful removal of spatial dependencies and transforms the image into statistically independent pixel values.

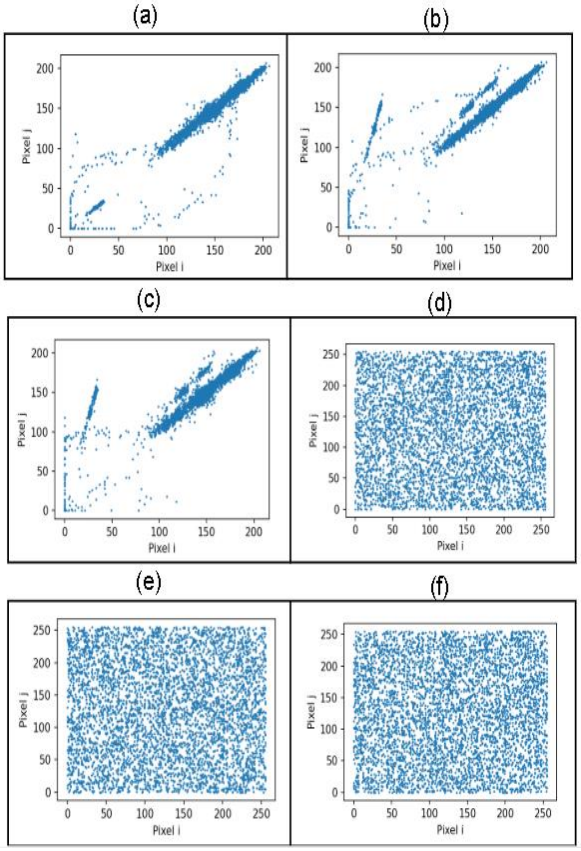

Figure 6 Adjacent pixel correlation of original image img3 (a) Horizontal (b) Vertical (c) Diagonal; Adjacent pixel correlation of encrypted image img3 (d) Horizontal (e) Vertical (f) Diagonal

Table 3 Correlation coefficients between adjacent pixels of Original image and Encrypted image

	Original Image			Encrypted Image		
	Horizontal	**Vertical**	**Diagonal**	**Horizontal**	**Vertical**	**Diagonal**
img1	0.9820	0.9842	0.9681	0.0002	0.0035	-0.0019
img2	0.9868	0.9829	0.9702	0.0031	-0.0005	0.0006
img3	0.9915	0.9786	0.9728	0.0016	-0.0020	0.0024

3.5.Differential Attack Analysis

Differential attack analysis is used to assess how well an image encryption scheme responds to minor changes in the plaintext image and whether those changes result in large, unpredictable variations in the encrypted output. A secure encryption algorithm should make sure that even a tiny change in the input image leads to significant changes in the encrypted image [22, 26]. In our research, resistance to differential attacks is obtained through key-dependent pixel permutation, AES-CTR based keystream mixing, nonlinear bit-level substitution,

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

and feedback-based diffusion. These methods ensure that a small alteration in the plain image spreads quickly across all pixels and color channels, creating a strong avalanche effect. We measure how well this works using NPCR (Number of Pixel Change Rate), which shows the percentage of pixels that change in the encrypted image, and UACI (Unified Average Changing Intensity), which assesses the average intensity difference between two encrypted images.

NPCR for R Channel

$$\mathbf{NPCR_r} = \left(\frac{1}{(H*W)}\right) * \sum_{i=1}^{H}\sum_{j=1}^{W} \mathbf{D_r(i,j)} * \mathbf{100\%} \quad \mathbf{(27)}$$

NPCR for G Channel

$$\mathbf{NPCR_g} = \left(\frac{1}{(H*W)}\right) * \sum_{i=1}^{H}\sum_{j=1}^{W} \mathbf{D_g(i,j)} * \mathbf{100\%} \quad \mathbf{(28)}$$

NPCR for B Channel

$$\mathbf{NPCR_b} = \left(\frac{1}{(H*W)}\right) * \sum_{i=1}^{H}\sum_{j=1}^{W} \mathbf{D_b(i,j)} * \mathbf{100\%} \quad \mathbf{(29)}$$

UACI for R Channel

$$\mathbf{UACI_r} = \left(\frac{1}{(H*W)}\right) * \sum_{i=1}^{H}\sum_{j=1}^{W} \frac{|C1_r(i,j)-C2_r(i,j)|}{255} * \mathbf{100\%} \quad \mathbf{(30)}$$

UACI for G Channel

$$\mathbf{UACI_g} = \left(\frac{1}{(H*W)}\right) * \sum_{i=1}^{H}\sum_{j=1}^{W} \frac{|C1_g(i,j)-C2_g(i,j)|}{255} * \mathbf{100\%} \quad \mathbf{(31)}$$

UACI for B Channel

$$\mathbf{UACI_b} = \left(\frac{1}{(H*W)}\right) * \sum_{i=1}^{H}\sum_{j=1}^{W} \frac{|C1_b(i,j)-C2_b(i,j)|}{255} * \mathbf{100\%} \quad \mathbf{(32)}$$

where H × W represents the height and width of image which represents the size of the image, C1 and C2 are two cipher images with only one different pixel. $C1_r$ and $C2_r$ represents the red channel of encrypted image. If $C1_r(i, j) \neq C2_r(i, j)$, $D_r(i, j) = 1$; otherwise, $D_r(i, j) = 0$. This is similar to G and B channels. The theoretical values of NPCR and UACI for 8 bit images are 99.6094% and 33.4635%, respectively. Referring Table 4. for the images img1, img2 and img3, the NPCR values are above 99.60%.and the UACI values are near the ideal 33%. This shows strong diffusion characteristics and great resistance to differential attacks.

Table 4 NPCR and UACI values for R, G and B Channels

	NPCR %				UACI%			
	R	G	B	Mean NPCR	R	G	B	Mean UACI
img1	99.5803	99.6459	99.6261	99.6174	33.5507	33.3853	33.5198	33.4853
img2	99.6337	99.5803	99.5956	99.6032	33.4857	33.483	33.543	33.5039
img3	99.6429	99.5956	99.6063	99.6149	33.4425	33.5551	33.4025	33.4667
House[22]	99.6105	99.6077	99.6076	99.6086	33.4547	33.4681	33.4735	33.4654
Img_f[26]	99.6094	99.6055	99.6177	99.6109	33.4635	33.4598	33.4702	33.4645

3.6.Information Entropy Analysis

Information entropy measures the randomness and uncertainty of pixel values. A higher entropy, which is closer to the theoretical maximum of 8 for grayscale, indicates better security [22, 26]. Mathematically it is represented as

$$Hc = \sum_{i=0}^{255} pc(i) \log 2\ pc(i) \quad (33)$$

where Hc is information entropy of a single color channel which can be Red, Green, or Blue. ∑ (summation) indicates that the expression is summed over all possible pixel intensity values. For gray-scale image variable **i** has the value from 0 to 255. The pc(i) represents the probability of occurrence of intensity

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

value i in channel c. In this research, the information entropy for each color channel is computed separately. The overall entropy of the RGB image is then defined as the average of the entropies of the red, green, and blue channels. Table 5. shows the entropy values of original image and encrypted image. After encryption, the entropy values of the cipher images for all three images img1, img2 and img3 are about 7.997. This is very close to the theoretical maximum entropy of 8 bits for an 8-bit image. This near-ideal entropy shows that the encrypted images have very uniform pixel intensity distributions and act similarly to random noise.

Table 5. Entropy of Original Image and Cipher Image

Image	Original Image	Cipher Image
img1	7.2916	7.9971
img2	7.4086	7.9972
img3	7.4295	7.9972
House[22]	-	7.9974
Img_f[26]	-	7.9994

3.7.Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR) Analysis

Mean Squared Error (MSE)

Mean Squared Error (MSE) measures the average squared difference between two images. In image encryption research, MSE is mainly used to check correct decryption (original vs. decrypted image) and to measure how different two images are at the pixel level. A lower MSE means that two images are very similar, while a higher MSE shows large differences.

$$\mathbf{MSE} = \frac{1}{(H*W)} \sum [\mathbf{I(i,j)} - \mathbf{K(i,j)}]^2 \quad \mathbf{(34)}$$

where I(i,j) = pixel value of the original image at position (i,j),
K(i,j) = pixel value of the decrypted image,
H = image height,
W = image width,
$\sum$ = summation over all pixels

Peak Signal-to-Noise Ratio (PSNR)

PSNR measures the quality of a reconstructed (decrypted) image compared to the original image. It is expressed in decibels (dB) and comes from MSE. In encryption, a high PSNR between the original and decrypted images indicates correct decryption. A low PSNR between the original and encrypted images indicates strong encryption. Mathematically it is represented as;

$$\mathbf{PSNR} = \mathbf{10} * \mathbf{log10}\left(\frac{\mathbf{MAX}^2}{\mathbf{MSE}}\right) \quad \mathbf{(35)}$$

Where:

- MAX = maximum possible pixel value (for 8-bit images, MAX = 255)
- MSE = mean squared error

Table 6 MSE and PSNR Values

Image	MSE Original vs Encrypted	MSE Original vs Decrypted	PSNR Original vs Encrypted	PSNR Original vs Decrypted
img1	12372.235	0	7.206322	∞
img2	10466.141	0	7.9329	∞
img3	9701.19	0	8.2625	∞

The Table 6 shows the Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR) values by comparing the original image with its encrypted and decrypted versions for three test images. For the Original vs Encrypted images, the MSE values are quite high, ranging from about 9701 to 12372. This indicates a large pixel-wise difference between the original and encrypted images. As a result, the PSNR values are very low, about 7 to 8 dB. This confirms that the encrypted images are heavily distorted and do not resemble the original images. This outcome shows strong confusion and diffusion of encryption

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

process. It makes the cipher images resilient to visual and statistical attacks. For the Original vs Decrypted images, the MSE values are zero for all test images, and the PSNR values are infinite (∞). An MSE of zero means there is no difference between the original and decrypted images at any pixel position. Consequently, PSNR becomes infinite, indicating perfect reconstruction. This shows that the decryption process is lossless and accurately recovers the original image without any loss.

3.8.Encryption speed analysis

The proposed method achieves an encryption time of 0.17108 seconds. This is much lower than the referred algorithms. This shows that the proposed algorithm is efficient and fits well for time-sensitive applications. In terms of encryption speed, the proposed scheme reaches 9.19 Mbps. This is significantly higher than the speeds of other referred algorithms listed in Table 7. This improvement comes from using efficient cryptographic tools like AES-CTR for generating keystreams and optimizing permutation and diffusion operations. Overall, the results show that the proposed encryption scheme not only offers strong security but also delivers better performance than current methods. This makes it suitable for real-time and large-scale image encryption scenarios.

Table 7 Encryption Time and Speed

	Encryption Time(s)	Encryption Speed (Mbps)
Ours	0.17108	9.19
Ref[2]	5.1	-
Ref[11]	0.324	1.618
Ref[12]	0.266	1.973

Conclusion

This study proposed a secure medical image encryption scheme to address privacy and security challenges associated with digital healthcare systems. This method, is rooted in the context of a confusion-diffusion architecture and the use of password-derived key generation. It is capable of effectively handling the peculiarities of the medical images having high redundancy and strong spatial correlation. Master key derivation based on Argon2, subkeys of different domains, secure pixel shuffling, and multi-layer diffusion are all parts of the encryption process. The design choices made here give rise to a very low pixel correlation and a considerable increase in the randomness of the cipher image. The results of the experiments show that the achieved entropy is close to the ideal level. The resistance to differential and key sensitivity attacks is very strong. The pixels are distributed uniformly. Additionally, the correct key ensures an exact reconstruction of the image with better performance. The proposed method has a great deal of security and low computational requirements; thus, it can be regarded as an excellent means for secure storage and transmission of medical images. The performance optimization and other types of medical data extension will be the subject of future research.

References

[1]. Agarwal, S. (2018). A review of image scrambling technique using chaotic maps. International Journal of Engineering and Technology Innovation, 8(2), 77–98.
[2]. Ran, W., Wang, E., & Tong, Z. (2022). A double scrambling–DNA row and column closed loop image encryption algorithm based on chaotic system. PLoS ONE, 17(7), e0267094.doi:10.1371/journal.pone.0267094
[3]. Uddin, M., Jahan, F., Islam, M. K., & Hassan, M. R. (2021). A novel DNA-based key scrambling technique for image encryption. Complex & Intelligent Systems, 7, 3241–3258.
[4]. Alsandi, N. S. A., Zebari, D. A., Haron, H., Zebari, R. R., & Zeebaree, S. R. M. (2021). A multi-stream scrambling and DNA encoding method based image encryption. Journal of Information Security and Applications, 58, 102735.
[5]. Gong, L.-H., Du, J., Wan, J., & Zhou, N.-R. (2021). Image encryption scheme based on

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

block scrambling, closed-loop diffusion, and DNA molecular mutation. Security and Communication Networks, 2021:6627005, 1–16. doi:10.1155/2021/6627005

[6]. Li, M., Pan, S., Mou, X., & Zhou, Y. (2019). Medical image encryption algorithm based on hyper-chaotic system and DNA coding. Journal of Medical Imaging and Health Informatics, 9(5), 1071–1082.

[7]. Anujaa, Amirtharajan, R., Thenmozhi, K., & Rayappan, J. B. B. (2022). Lightweight multi-round confusion–diffusion cryptosystem for securing images using a modified 5D chaotic system. Multimedia Tools and Applications, 81(8), 10819–10847.

[8]. Nakachi, T., Kato, Y., Fukuhara, T., & Watanabe, K. (2022). Privacy protection in JPEG XS: A lightweight spatio-color scrambling approach. IEEE Transactions on Circuits and Systems for Video Technology, 32(5), 2951–2965.

[9]. İnce, C., İnce, K., & Hanbay, D. (2022). Novel image pixel scrambling technique for efficient color image encryption in resource-constrained IoT devices. IEEE Internet of Things Journal, 9(5), 3554–3566.

[10]. Zang, W., Zhang, Y., Wang, X., & Zhu, Z. (2021). Chaos-based color image encryption with JPEG compression: Balancing security and compression efficiency. Signal Processing: Image Communication, 96, 116301.

[11]. Sheng, Y., Li, J., Zhang, Y., & Liu, Z. (2020). An image encryption algorithm based on complex network scrambling and multi-directional diffusion. Signal Processing, 171, 107484.
doi: 10.1016/j.sigpro.2020.107484

[12]. Sheng, Y., & Li, J. (2021). Bit-level image encryption algorithm based on fully-connected-like network and random modification of edge pixels. Information Sciences, 565, 343–360. doi: 10.1016/j.ins.2021.02.046

[13]. Zhang, H., Sun, W., & Lu, L. (2020). Chaotic encryption algorithm with scrambling diffusion based on the Josephus cycle. Nonlinear Dynamics, 99(3), 2291–2310. doi: 10.1007/s11071-019-05407-9

[14]. Guan, Z. (2021). A novel and fast encryption system based on improved Josephus scrambling and chaotic mapping. Optik, 231, 166406.
doi: 10.1016/j.ijleo.2021.166406

[15]. Huang, Y., & Bi, X. (2021). Development of a novel hyperchaos-based image encryption algorithm consisting of two scrambling–diffusion operations. Multimedia Tools and Applications, 80(4), 5417–5440.doi: 10.1007/s11042-020-09932-8

[16]. Tang, Y. (2019). Image encryption scheme based on hyper-chaotic map and self-adaptive diffusion. Signal Processing: Image Communication, 74, 189–201. doi: 10.1016/j.image.2019.02.005

[17]. Mawla, N. A., & Khafaji, H. K. (2023). Enhancing data security: A cutting-edge approach utilizing protein chains in cryptography and steganography. Journal of Information Security and Applications, 75, 103514.doi: 10.1016/j.jisa.2023.103514

[18]. Wang, T., Ge, B., Xia, C., & Dai, G. (2021). Multi-image encryption algorithm based on cascaded modulation chaotic system and block-scrambling-diffusion. Chaos, Solitons & Fractals, 148, 111024. doi: 10.1016/j.chaos.2021.111024

[19]. Chen, Z., Ma, C., Wang, T., Feng, Y., Hou, X., & Qian, X. (2021). Robust image compression–encryption via scrambled block Bernoulli sampling with diffusion noise. Signal Processing, 181, 107909. doi: 10.1016/j.sigpro.2020.107909

[20]. Satpute, N. R., & Hajare, H. (2018). Scrambling with image processing. International Journal of Computer Applications, 180(44), 15–19.

[21]. Zulfiqar, N., Ahmad, T., Ghazal, T. M., Ikram, A., & Khan, M. A. (2022). Securing digital images: A chaos-driven scrambling

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2026.0013
e ISSN: 2584-2854
Volume: 04 Issue: 01
January 2026
Page No: 075 - 093

algorithm using the Rössler system. Mathematics, 10(3), 1–20. doi: 10.3390/math10030443

[22]. Ge, B., Shen, Z., & Wang, X. (2020). Symmetric color image encryption using a novel cross-plane joint scrambling–diffusion method. Signal Processing: Image Communication, 83, 115773. doi: 10.1016/j.image.2019.115773

[23]. Thomas, M. Y. S. (2020). Image encryption algorithm with block scrambling based on logistic map. Procedia Computer Science, 171, 906–915. doi: 10.1016/j.procs.2020.04.098

[24]. Wang, J., & Liu, L. (2020). A novel chaos-based image encryption using magic square scrambling and octree diffusing. Multimedia Tools and Applications, 79(7), 4897–4923. doi: 10.1007/s11042-019-08306-4

[25]. Sanaboina, C. S. (2020). A novel chaos-based cryptographic scrambling technique to secure medical images. International Journal of Medical Engineering and Informatics, 12(3), 249–264. doi: 10.1504/IJMEI.2020.107331

[26]. Setiadi, D. R. I. M., Rachmawanto, E. H., & Sari, C. A. (2022). Medical image cryptosystem using dynamic Josephus sequence and chaotic-hash scrambling. Journal of King Saud University – Computer and Information Sciences, 34(5), 2030–2042. doi: 10.1016/j.jksuci.2020.10.006

[27]. Abusham, E. (2020). An integration of new digital image scrambling technique on PCA-based face recognition system. Journal of King Saud University – Computer and Information Sciences, 32(4), 507–516. doi: 10.1016/j.jksuci.2018.11.004

[28]. Liu, Y., Chen, Z., Zhang, X., & Zhao, J. (2023). A state-of-the-art review of diffusion model applications for microscopic image and micro-alike image analysis. Artificial Intelligence Review, 56(2), 1109–1150. doi: 10.1007/s10462-022-10250-9

[29]. Chen, H., Yang, Y., Zhong, N., & Ma, K. (2023). Hiding images in diffusion models by editing learned score functions. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 18900–18909). doi: 10.1109/CVPR52729.2023.01814

[30]. Alsubaei, F. S., Alhassan, A., & Alshahrani, A. M. (2022). Block-scrambling-based encryption with deep-learning-driven remote sensing image classification. Remote Sensing, 14(19), 4829. doi: 10.3390/rs14194829

[31]. Sharma, V. K., Sharma, A. K., & Kumar, R. (2021). Secret image scrambling and DWT-based image steganography using smoothing operation and convolution neural networks. Multimedia Tools and Applications, 80(19), 29767–29792. doi: 10.1007/s11042-021-10954-4

[32]. George, A. T., Kumar, S. S., & Kumar, R. S. (2020). Argon2: The secure password hashing function. International Journal of Engineering Research & Technology, 9(6), 721–725.

[33]. Eum, S., Kim, J., & Park, J. H. (2021). Optimized implementation of Argon2 utilizing the graphics processing unit. IEEE Access, 9, 148321–148334. doi: 10.1109/ACCESS.2021.3124206

[34]. Malliga, L., Priyanka, S., & Suresh, R. (2019). A new secure data hiding AES-CTR key modulation. International Journal of Advanced Research in Computer and Communication Engineering, 8(4), 38–43. doi: 10.17148/IJARCCE.2019.8406

[35]. Pogorelov, K., Randel, K. R., Griwodz, C., Eskeland, S. L., de Lange, T., Johansen, D., & Halvorsen, P. (2017). Kvasir: A multi-class image dataset for computer aided gastrointestinal disease detection. In Proceedings of ACM Multimedia Systems Conference (MMSys'17). ACM. doi: 10.1145/3083187.3083212. Kvasir Dataset (Kaggle). https://www.kaggle.com/datasets/yasserhessein/the-kvasir-dataset