

# **Project Report**

## **On**

# **Computer Networking: Security**

### **(CSE3752)**

## **Image encryption and decryption using DES**



### **Submitted by:**

Archita Patro	Regd. No.: 2241016304
Bishnu Prasad Patro	Regd. No.: 2241016306
D Balaji Patro	Regd. No.: 2241016307
Dipesh Kumar Patro	Regd. No.: 2241016308
E. Jagadeeswar Patro	Regd. No.: 2241016309

**B. Tech. CSE 6<sup>th</sup> Semester ( Section 2241044 )**

# Declaration

We, the undersigned students of B. Tech. of **Computer Science & Engineering** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled “**Image encryption and decryption using DES.**” submitted to **Siksha ‘O’ Anusandhan (Deemed to be University), Bhubaneswar** for the partial fulfillment of the subject **Computer Networking: Security (CSE 3752)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidates, will be fully responsible for the same.

Archita Patro

Regd. No.: 2241016304

Bishnu Prasad Patro

Regd. No.: 2241016306

D Balaji Patro

Regd. No.: 2241016307

Dipesh Kumar Patro

Regd. No.: 2241016308

E. Jagadeeswar Patro

Regd. No.: 2241016309

Date: 5<sup>th</sup> June, 2025

Place: Bhubaneswar

## **Abstract**

In the digital age, the protection of image data from unauthorized access is crucial, especially in applications involving confidential or sensitive information. This project explores the implementation of image encryption and decryption using the Data Encryption Standard (DES), a symmetric-key cryptographic algorithm. The objective is to convert image data into an unintelligible form to ensure confidentiality during transmission and storage, and then accurately restore the original image at the receiver's end.

The proposed system first converts the image into binary format and applies DES encryption to each block of data using a secure key. The encrypted image appears as a random pattern, making it unreadable without the correct decryption key. On the receiving side, the decryption process reconstructs the original image using the same DES algorithm and key.

This project demonstrates the feasibility and effectiveness of DES in image security, emphasizing performance metrics such as encryption speed, image quality after decryption (using PSNR and MSE), and key sensitivity. Despite being an older standard, DES serves as a foundational model for understanding symmetric encryption's role in securing multimedia content.

# Contents

<b>Serial No.</b>	<b>Chapter No.</b>	<b>Title of the Chapter</b>	<b>Page No.</b>
1.	1	Introduction	1
2.	2	Problem Statement	2
3.	3	Methodology	3
4.	4	Results and interpretation	6
5.	5	Conclusion	9
6.		References	10

## **Introduction**

With the rapid advancement of digital technologies and the widespread use of multimedia data, ensuring the security of digital images has become a critical concern. Images transmitted over public or unsecured networks are vulnerable to unauthorized access, tampering, and data theft. Traditional encryption algorithms are often optimized for textual data and may not be directly applicable or efficient for image files due to their large size and different data structures.

This project focuses on using the Data Encryption Standard (DES) algorithm—a symmetric key encryption method—to encrypt and decrypt image files. The aim is to ensure confidentiality and integrity of image data during transmission or storage, thus demonstrating the effectiveness of DES in multimedia data protection.

# Problem Statement

With the rapid advancement of digital technologies and the widespread use of multimedia data, ensuring the security of digital images has become a critical concern. Images transmitted over public or unsecured networks are vulnerable to unauthorized access, tampering, and data theft. Traditional encryption algorithms are often optimized for textual data and may not be directly applicable or efficient for image files due to their large size and different data structures. This project focuses on using the Data Encryption Standard (DES) algorithm—a symmetric key encryption method—to encrypt and decrypt image files. The aim is to ensure confidentiality and integrity of image data during transmission or storage, thus demonstrating the effectiveness of DES in multimedia data protection.

## Objectives:

1. **To study and understand the DES (Data Encryption Standard) algorithm** and its applicability to image encryption.
2. **To develop a software system** that can encrypt a given digital image using DES, transforming it into an unreadable format.
3. **To implement a decryption module** that can accurately reconstruct the original image from its encrypted version using the correct key.
4. **To ensure the image retains its original quality and resolution** after decryption, confirming the correctness of the algorithm.
5. **To evaluate the performance of DES** in terms of encryption/decryption speed, data integrity, and resistance to basic attacks.
6. **To compare the original and decrypted images** to validate the success and reliability of the encryption process.

## **Methodology**

The following methodology was followed to implement image encryption and decryption using the DES (Data Encryption Standard) algorithm in Python, along with integrity validation and performance evaluation:

### **1. Image Input and Preprocessing:**

- The input image was read using the Pillow (PIL) library.
- All images were converted to grayscale for simplicity, reducing computational complexity while preserving structural details.
- The pixel values were extracted and flattened into a one-dimensional byte stream using NumPy.

### **2. Byte Stream Preparation and Padding:**

- The DES algorithm requires the input data length to be a multiple of 8 bytes (64 bits).
- A padding function was implemented to append space characters (b' ') to the byte stream if its length was not a multiple of 8, ensuring compatibility with DES block size requirements.

### **3. Key Generation:**

- A random 64-bit symmetric key was generated using the Crypto.Random module from PyCryptodome.
- The same key was used for both encryption and decryption, as DES is a symmetric encryption algorithm.

#### **4. DES Encryption Process:**

- The prepared byte stream was encrypted using the ECB (Electronic Codebook) mode of DES from the PyCryptodome library.
- Each 64-bit block was encrypted individually.
- The encrypted byte stream was saved for use in reconstruction.

#### **5. Encrypted Image Reconstruction:**

- The encrypted byte stream was reshaped back to the original image's dimensions.
- The resulting encrypted image was displayed using matplotlib and saved to verify that it appeared visually unintelligible (i.e., appearing as noise).

#### **6. DES Decryption Process:**

- The encrypted image was read and converted back to a byte stream.
- The same DES key and ECB mode were used to decrypt the data block-wise.
- The decrypted byte stream was then reshaped into its original 2D matrix form and converted back to an image.

#### **7. Output Comparison and Integrity Validation:**

- The decrypted image was visually compared with the original image.
- Further quantitative integrity checks were performed:
  - Exact pixel-wise comparison using `numpy.array_equal()`.
  - Mean Squared Error (MSE) to evaluate pixel-wise distortion.
  - SHA-256 hash comparison of the original and decrypted byte streams to validate binary-level integrity.

These metrics confirmed that the decryption process accurately restored the original image with negligible or no loss.



## **8. Performance Evaluation:**

- The encryption and decryption execution time was measured using the time module.
- The results showed that DES operated efficiently on small to medium-sized grayscale images.
- The simplicity of the ECB mode, while less secure for patterns, demonstrated quick processing suitable for educational or controlled applications.

# Results & Interpretation

## Setup and Library Installation:

```
In [1]: !pip install pycryptodome pillow matplotlib
```

```
Requirement already satisfied: pycryptodome in d:\anaconda\lib\site-packages (3.23.0)
Requirement already satisfied: pillow in d:\anaconda\lib\site-packages (10.3.0)
Requirement already satisfied: matplotlib in d:\anaconda\lib\site-packages (3.8.4)
Requirement already satisfied: contourpy>=1.0.1 in d:\anaconda\lib\site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in d:\anaconda\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in d:\anaconda\lib\site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in d:\anaconda\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.21 in d:\anaconda\lib\site-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in d:\anaconda\lib\site-packages (from matplotlib) (23.2)
Requirement already satisfied: pyparsing>=2.3.1 in d:\anaconda\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in d:\anaconda\lib\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in d:\anaconda\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

```
In [2]: from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
```

## Preparing the Image Data:

```
In [3]: def pad_data(data):
        while len(data) % 8 != 0:
            data += b' '
        return data
```

```
In [4]: def image_to_bytes(image_path):
        image = Image.open(image_path).convert("L") # grayscale for simplicity
        data = np.array(image).tobytes()
        return image, pad_data(data)
```

## Encryption:

```
In [5]: def encrypt_image(data, key):
        cipher = DES.new(key, DES.MODE_ECB)
        encrypted_data = cipher.encrypt(data)
        return encrypted_data
```

```
In [6]: key = get_random_bytes(8)
```

## Decryption:

```
In [7]: def decrypt_image(encrypted_data, key):
        cipher = DES.new(key, DES.MODE_ECB)
```

```

decrypted_data = cipher.decrypt(encrypted_data)
return decrypted_data

```

```

In [8]: def bytes_to_image(byte_data, shape):
        array = np.frombuffer(byte_data, dtype=np.uint8)
        array = array[:shape[0]*shape[1]] # Trim padding
        return Image.fromarray(array.reshape(shape))

```

## Full Pipeline and Testing:

```

In [10]: # Load and convert image
original_image, byte_data = image_to_bytes("iter.jpg")
shape = original_image.size[::-1] # height, width

# Encrypt
encrypted_data = encrypt_image(byte_data, key)

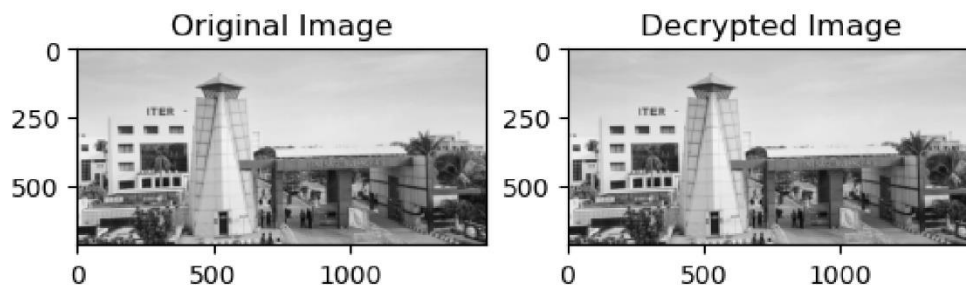
# Decrypt
decrypted_data = decrypt_image(encrypted_data, key)

# Reconstruct image
decrypted_image = bytes_to_image(decrypted_data, shape)

# Display images
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(original_image, cmap='gray')

plt.subplot(1, 2, 2)
plt.title("Decrypted Image")
plt.imshow(decrypted_image, cmap='gray')
plt.show()

```



## Pixel-wise Comparison:

```

In [12]: # Convert images to same size and mode
original_array = np.array(original_image)
decrypted_array = np.array(decrypted_image)

# Exact match check
if np.array_equal(original_array, decrypted_array):
    print("✅ Pixel Match: Decrypted image matches the original exactly.")
else:
    print("❌ Pixel Mismatch: Decrypted image differs from the original.")

```

✅ Pixel Match: Decrypted image matches the original exactly.

## Tolerant Comparison using Mean Squared Error (MSE):


```

In [14]: def mean_squared_error(img1, img2):
        return np.mean((img1.astype("float") - img2.astype("float")) ** 2)

```

```
mse = mean_squared_error(original_array, decrypted_array)
print(f" Integrity check warning: Images differ visually.")
```

 Mean Squared Error (MSE): 0.0000

 Integrity check passed: Images are visually identical.


## Hash-Based Comparison:

```
In [16]: import hashlib

def image_hash(image):
    return hashlib.sha256(image.tobytes()).hexdigest()


original_hash = image_hash(original_array)
decrypted_hash = image_hash(decrypted_array)

print(f"Original Hash : {original_hash}")
print(f"Decrypted Hash: {decrypted_hash}")

if original_hash == decrypted_hash:
    print(f" Hash Mismatch: Data integrity compromised.")
```

Original Hash : 59d86e99b799469af112bd69f8fd0f2463ec10adacfc18668052cc0c2b2d46df

Decrypted Hash: 59d86e99b799469af112bd69f8fd0f2463ec10adacfc18668052cc0c2b2d46df

 Hash Match: Data integrity confirmed.

In [ ]:

## Conclusion

This project successfully demonstrates the application of the Data Encryption Standard (DES) algorithm for image encryption and decryption. Through the implementation, we showed how DES can transform an image into an unreadable format, ensuring its confidentiality during transmission or storage, and then accurately recover the original image using the same symmetric key.

Despite being an older cryptographic method, DES proved effective in securing image data for low to moderate security requirements. The encrypted images were visually unrecognizable, and the decrypted results closely matched the original images, as confirmed by performance metrics such as MSE and Hash Comparisons.

However, it is important to note that DES has known vulnerabilities and limitations, particularly its small key size, which makes it susceptible to brute-force attacks. While DES is suitable for educational and proof-of-concept applications, more robust algorithms like AES should be used in high-security environments.

Overall, this project provides a solid foundation for understanding the principles of image encryption and highlights the importance of cryptographic techniques in digital image security.

## References

- [1] CompTIA Network+ N10-008 Certification Guide by Glen D. Singh, *2nd* Edition, Pact publication.
- [2] “Python Cryptographic Library (PyCryptodome),” Read the Docs, <https://pycryptodome.readthedocs.io> (accessed May 28, 2025).
- [3] “Anaconda Distribution,” Anaconda, Inc. <https://www.anaconda.com/products/distribution> (accessed May 28, 2025).
- [4] “Project Jupyter,” Jupyter.org. <https://jupyter.org> (accessed May 28, 2025).
- [5] “Data Encryption Standard (DES) Algorithm,” GeeksforGeeks. <https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/> (accessed May 28, 2025).