# Image Encryption and Decryption Project

**Overview:**

This project demonstrates how to encrypt and decrypt multiple images using the AES (Advanced Encryption Standard) algorithm. The encryption ensures that the images are securely stored and can only be decrypted with the correct key. This project also includes the management of image metadata to facilitate accurate decryption.

**Directory Structure:**

```
ImageEncryptionProject/
├── input_images/
|   ├── image1.jpg          # Example input image
|   ├── image2.png          # Example input image
|   └── ...                 # Other input images
|
├── encrypted_images/       # This will store encrypted images
|   └── ...                 # Encrypted images will be saved here
|
├── decrypted_images/       # This will store decrypted images
|   └── ...                 # Decrypted images will be saved here
|
├── main.py                 # Main Python script for encryption and decryption
```

**Prerequisites:**

PyCharm or any other preferred Python IDE

**Setup:**

**Step 1:** Create Directory Structure

Create the following directories in your project root:

input_images/

encrypted_images/

decrypted_images/

**Step 2:** Place Input Images

Place the images you want to encrypt in the input_images/ directory

**Step 3**: Create main.py

Create a file named main.py in the root directory of your project and copy the following code into it:

```
from PIL import Image

import numpy as np

import os

from Crypto.Cipher import AES

from Crypto.Random import get_random_bytes

from Crypto.Util.Padding import pad, unpad

import json

def image_to_byte_array(image_path):

    img = Image.open(image_path)

    img = img.convert('RGB')

    img_array = np.array(img)

    img_bytes = img_array.tobytes()
```

```
return img_bytes, img.size, img.mode

def encrypt_image(byte_array, key):

   cipher = AES.new(key, AES.MODE_CBC)

   ciphertext = cipher.encrypt(pad(byte_array, AES.block_size))

   return cipher.iv + ciphertext

def save_encrypted_image(encrypted_bytes, size, mode, output_path):

   metadata = {

      'size': size,

      'mode': mode

   }

   with open(output_path, 'wb') as file:

      file.write(encrypted_bytes)

   metadata_path = output_path + '.json'

   with open(metadata_path, 'w') as metadata_file:

      json.dump(metadata, metadata_file)

def decrypt_image(encrypted_bytes, key):

   iv = encrypted_bytes[:AES.block_size]

   ciphertext = encrypted_bytes[AES.block_size:]

   cipher = AES.new(key, AES.MODE_CBC, iv)

   decrypted_bytes = unpad(cipher.decrypt(ciphertext), AES.block_size)

   return decrypted_bytes

def byte_array_to_image(byte_array, size, mode):

   img_array = np.frombuffer(byte_array, dtype=np.uint8).reshape(size[1], size[0], 3)
```

```
    img = Image.fromarray(img_array, mode)
    return img
def main():
    # Generate a random AES key
    key = get_random_bytes(16)  # AES-128 key
    input_dir = 'input_images/'
    encrypted_dir = 'encrypted_images/'
    decrypted_dir = 'decrypted_images/'
    os.makedirs(encrypted_dir, exist_ok=True)
    os.makedirs(decrypted_dir, exist_ok=True)
    for image_name in os.listdir(input_dir):
        if image_name.endswith(('.jpg', '.jpeg', '.png', '.bmp')):
            input_image_path = os.path.join(input_dir, image_name)
            encrypted_image_path = os.path.join(encrypted_dir, f'{image_name}.enc')
            img_bytes, size, mode = image_to_byte_array(input_image_path)
            encrypted_bytes = encrypt_image(img_bytes, key)
            save_encrypted_image(encrypted_bytes, size, mode, encrypted_image_path)
            print(f"Image '{image_name}' encrypted and saved as '{encrypted_image_path}'")
    for encrypted_image_name in os.listdir(encrypted_dir):
        if encrypted_image_name.endswith('.enc'):
            encrypted_image_path = os.path.join(encrypted_dir, encrypted_image_name)
            decrypted_image_name = encrypted_image_name.replace('.enc', '')
            decrypted_image_path = os.path.join(decrypted_dir, decrypted_image_name)
            metadata_path = encrypted_image_path + '.json'
            with open(metadata_path, 'r') as metadata_file:
                metadata = json.load(metadata_file)
            size = tuple(metadata['size'])
            mode = metadata['mode']
            with open(encrypted_image_path, 'rb') as file:
                encrypted_bytes = file.read()
            decrypted_bytes = decrypt_image(encrypted_bytes, key)
            decrypted_img = byte_array_to_image(decrypted_bytes, size, mode)
            decrypted_img.save(decrypted_image_path)
            print(f"Image '{decrypted_image_name}' decrypted and saved as '{decrypted_image_path}'")
if __name__ == '__main__':
    main()
```

**Running the Project:**

To run the project, simply execute main.py. You can do this in PyCharm by right-clicking the main.py file and selecting "Run 'main'".

**Encryption Process:**

Load Image: Each image in the input_images/ directory is loaded and converted to a byte array.

Encrypt Image: The byte array is encrypted using AES (CBC mode) with a randomly generated key.

Save Encrypted Image: The encrypted byte array is saved to the encrypted_images/ directory, along with a JSON file containing the image's metadata (size and mode).

**Decryption Process:**

Load Encrypted Image: Each encrypted file in the encrypted_images/ directory is read.

Read Metadata: The metadata (size and mode) is read from the corresponding JSON file.

Decrypt Image: The encrypted byte array is decrypted using the same AES key.

Save Decrypted Image: The decrypted byte array is converted back to an image and saved to the decrypted_images/ directory.

**Notes:**

Security: Ensure that the AES key is kept secure. If the key is compromised, the encrypted images can be decrypted by unauthorized parties.

File Handling: The script assumes that the input images are in a format supported by the Pillow library (e.g., JPEG, PNG, BMP).

Error Handling: The script currently does not include extensive error handling. In a production environment, you should add error handling to manage issues such as file not found, read/write errors, etc.

**Conclusion:**

This project provides a practical implementation of image encryption and decryption using Python. It covers essential concepts of cryptography, such as AES encryption, and demonstrates how to handle and preserve metadata for accurate decryption.