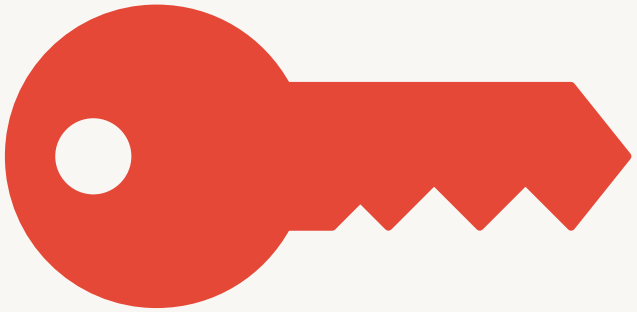
# Secure Photo Gallery Application

MATH319 Group project

- xxx
- xxx
- xxx
- xxx
- xxx



# Table of Contents

# Project idea

- Web App to upload and store photos safely by applying encryption at rest.

# App Functionalities

USER INTERFACE

USER AUTHENTICATION

PHOTO UPLOAD

AES ENCRYPTION

RETRIEVAL AND DECRYPTION

# Technologies Used

1. Python
2. Django
   - Frontend: Django Template Engine
   - Backend: Django & Python
3. SQLite

# Application Demo

# Encryption Used

- Advanced Encryption Standard (AES)
- Variable Key Size (128, 192, or 256 bits)
- Generates Unique Ciphertext
- Quick and Effective Encryption

# Applying AES Encryption

```python
def encrypt_image(data: bytes, key: str) -> bytes:
    iv = os.urandom(16)  # Generate a random IV
    key_bytes = bytes.fromhex(key)
    cipher = Cipher(algorithms.AES(key_bytes), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()

    # Pad the data to ensure it is a multiple of the block size
    padder = padding.PKCS7(algorithms.AES.block_size).padder()
    padded_data = padder.update(data) + padder.finalize()

    # Encrypt the data and prepend the IV
    encrypted_data = iv + encryptor.update(padded_data) + encryptor.finalize()
    return encrypted_data
```

# Applying AES Decryption

```python
def decrypt_image(encrypted_data: bytes, key: str) -> bytes:
    key_bytes = bytes.fromhex(key)

    # Extract the IV from the beginning of the encrypted data
    iv = encrypted_data[:16]  # First 16 bytes are the IV
    encrypted_content = encrypted_data[16:]  # The rest is the encrypted content

    cipher = Cipher(algorithms.AES(key_bytes), modes.CBC(iv), backend=default_backend())
    decryptor = cipher.decryptor()

    # Decrypt the data
    padded_data = decryptor.update(encrypted_content) + decryptor.finalize()

    # Unpad the data
    unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
    decrypted_data = unpadder.update(padded_data) + unpadder.finalize()
    return decrypted_data
```

# Challenges Faced

1. Key Management:

- Storing AES keys securely was challenging.

2. Applying AES in Python:

- Initial difficulties with encryption were solved using a library.

3. Image Processing:

- Linking encrypted images to users needed careful planning.

# Challenges Faced (Cont.)

4. Privacy and Usability:

- Balancing security with user-friendly design.

5. Feedback:

- Providing clear messages for user actions.

# Advantages AES Encryption

- High Security.
- Efficiency.
- Global Standard.
- Versatility.
- Hardware Support.

# Disadvantages AES Encryption

- High Power Consumption.
- Complex Implementation.
- Less Effective for Small Data.
- Vulnerable to Certain Attacks.

# Conclusion

Thanks for listening, any question ?