Encryption with Python Using the Cryptography Library- Yonase Project

Project Overview

This project demonstrates the use of the cryptography library in Python to encrypt a message. The goal is to generate a secure encryption key, encrypt a text message, and document the process and code used.

Objectives

- Generate a secure encryption key.
- Encrypt a text message using the generated key.
- Document the process and code used.

Tools and Technologies

- **Python:** The programming language used for the script.
- **cryptography library:** The Python library used for encryption tasks.

Steps Taken

1. **Setup**
    - Installed the cryptography library using the command:

```
bash
Copy code
pip install cryptography
```

2. **Key Generation**
    - **Function Used:** `write_key()`
    - **Description:** Generates a new encryption key and saves it to a file called `secret.key`.

```
python
Copy code
```

```
from cryptography.fernet import Fernet

def write_key():
    key = Fernet.generate_key()
    with open("secret.key", "wb") as key_file:
        key_file.write(key)
```

3. **Key Loading**
    - **Function Used:** `load_key()`
    - **Description:** Reads the encryption key from the `secret.key` file.

```
python
Copy code
def load_key():
    return open("secret.key", "rb").read()
```

4. **Message Encryption**
    - **Function Used:** `encrypt_message(message)`
    - **Description:** Encrypts the provided message using the loaded key.

```
python
Copy code
def encrypt_message(message):
    key = load_key()
    fernet = Fernet(key)
    encrypted_message = fernet.encrypt(message.encode())
    return encrypted_message
```

5. **Code Implementation**

```
python
Copy code
from cryptography.fernet import Fernet

def write_key():
    key = Fernet.generate_key()
    with open("secret.key", "wb") as key_file:
        key_file.write(key)
```

```
def load_key():
    return open("secret.key", "rb").read()

def encrypt_message(message):
    key = load_key()
    fernet = Fernet(key)
    encrypted_message = fernet.encrypt(message.encode())
    return encrypted_message

write_key()
message = "This is a secret message"
encrypted_message = encrypt_message(message)

print("Original Message: ", message)
print("Encrypted Message: ", encrypted_message)
```

Results

- **Original Message:** "This is a secret message"
- **Encrypted Message:** “gAAAAABm4ndyr0vk_Lm20LBdaleNUYGXvrSyIsTjhNswyzTC64S1XJF6ZtOB9yWFeWmWpG2-PUFINlAp-UZL2cIrAapPE3nhPWZ61uKns2P4d5mYJ6TCO74=”

Challenges and Solutions

- **Challenge:** Ensuring that the key is saved and loaded correctly.
    - **Solution:** Used binary mode for file operations to handle the key as binary data.
- **Challenge:** Properly handling the encoding of the message.
    - **Solution:** Converted the message to bytes before encryption.

Lessons Learned

- Understanding the importance of secure key management.
- Familiarity with encryption and decryption processes in Python.

Future Improvements

- Implement a decryption function to reverse the encryption.
- Expand the project to handle file encryption and decryption.