Perfect 👏 — this is a great project to present!  
Let’s make sure you can **explain it clearly and confidently** in your presentation.

We’ll go step-by-step so you understand what every part does, **why it’s needed**, and **how to describe it simply**.

**🧠 Overview — What this code does**

This Python script uses the **cryptography** library (specifically Fernet) to:

1. **Generate a secret encryption key**
2. **Encrypt a file** (convert readable data → unreadable format)
3. **Decrypt the file** (restore original data using the same key)

So it’s a simple, **secure file encryption system**.

**🧩 Part 1 — Imports**

from cryptography.fernet import Fernet

import os

* Fernet: A built-in encryption method in the cryptography library.
  + It provides **AES-128 encryption** with **authentication**.
  + Ensures data **confidentiality (encryption)** and **integrity (no tampering)**.
* os: Used to check if files exist (for key management).

🗣️ **In presentation:**

“We’re importing Fernet from Python’s cryptography library to handle encryption and decryption securely. The os module helps us manage files like checking if the key file already exists.”

**🔐 Part 2 — Key Management**

**1️⃣ Generate and save a key**

def generate\_key(key\_path='secret.key'):

key = Fernet.generate\_key()

with open(key\_path, 'wb') as key\_file:

key\_file.write(key)

* Fernet.generate\_key() creates a **random 32-byte secret key**.
* That key is saved in a file (secret.key) so you can reuse it later.
* 'wb' means **write in binary mode**.

🗣️ **In presentation:**

“The generate\_key() function creates a random encryption key and saves it in a file called secret.key. This key is required later for both encryption and decryption.”

**2️⃣ Load the key**

def load\_key(key\_path='secret.key'):

return open(key\_path, 'rb').read()

* Reads the saved key back from the file.
* 'rb' = **read binary**, since the key is not text.

🗣️ **In presentation:**

“The load\_key() function loads the previously saved key so we can use it to encrypt and decrypt files.”

**🔒 Part 3 — Encrypting a file**

def encrypt\_file(file\_path, key):

fernet = Fernet(key)

with open(file\_path, 'rb') as file:

original\_data = file.read()

encrypted\_data = fernet.encrypt(original\_data)

with open(file\_path + '.enc', 'wb') as enc\_file:

enc\_file.write(encrypted\_data)

print(f"File encrypted successfully: {file\_path}.enc")

**What happens:**

1. **Create a Fernet object** → fernet = Fernet(key)
2. **Read the file’s content** in binary form
3. **Encrypt the data** → fernet.encrypt(original\_data)
4. **Save** the encrypted bytes into a new file with .enc extension  
   (e.g., example.txt.enc)
5. **Print a success message**

🗣️ **In presentation:**

“This function reads the original file in binary, encrypts it using the Fernet key, and writes the encrypted data into a new file ending with .enc. This ensures the original data is now unreadable to others.”

**🔓 Part 4 — Decrypting a file**

def decrypt\_file(encrypted\_path, key, output\_path=None):

fernet = Fernet(key)

with open(encrypted\_path, 'rb') as enc\_file:

encrypted\_data = enc\_file.read()

decrypted\_data = fernet.decrypt(encrypted\_data)

if not output\_path:

output\_path = encrypted\_path.replace('.enc', '.dec')

with open(output\_path, 'wb') as dec\_file:

dec\_file.write(decrypted\_data)

print(f"File decrypted successfully: {output\_path}")

**What happens:**

1. Load the encrypted file
2. Use the same Fernet key to decrypt
3. Save the original data again (as .dec file — e.g., example.txt.dec)

🗣️ **In presentation:**

“This function performs the reverse process. It decrypts the .enc file using the same key and restores the original readable data into a new file ending with .dec.”

**⚙️ Part 5 — Running the script**

if \_\_name\_\_ == '\_\_main\_\_':

if not os.path.exists('secret.key'):

generate\_key()

key = load\_key()

encrypt\_file('example.txt', key)

decrypt\_file('example.txt.enc', key)

**Step-by-step:**

1. **Check if key exists**
   * If not → generate a new one.
2. **Load the key** for use.
3. **Encrypt a file** (example.txt)
4. **Decrypt** the resulting file (example.txt.enc)

🗣️ **In presentation:**

“The main block controls the flow. It first ensures the key is available, then loads it, encrypts example.txt, and finally decrypts the .enc version back to its original form.”

**📦 Files Created**

After running:

example.txt → original file

example.txt.enc → encrypted file

example.txt.dec → decrypted file (same as original)

secret.key → encryption key file

**💬 Simple Explanation Example (for your talk)**

“Our script uses the Fernet class from Python’s cryptography library to make file encryption easy and safe.  
First, it generates a secret key and stores it in a file called secret.key.  
Then it encrypts our data using this key and saves the encrypted version as .enc.  
Later, the same key is used to decrypt the file back to its original form.  
This ensures that only someone who has the correct key can read the contents.”

**🧠 Bonus — Concepts Used**

| **Concept** | **Example** | **Meaning** |
| --- | --- | --- |
| File Handling | open(..., 'rb') | Read/write files in binary mode |
| Functions | def encrypt\_file(...): | Reusable blocks of code |
| Conditionals | if not os.path.exists() | Check file existence |
| String Manipulation | .replace('.enc', '.dec') | Create new filenames |
| Exception Handling *(optional to add)* | try-except | Handle wrong keys or missing files |
| Encryption Library | cryptography.fernet | Provides secure AES-based encryption |

**Lock the Vault: Secure File Encryption Using Python’s Cryptography Library**

Build a digital vault to protect sensitive information using simple Python scripts.

[[](https://substackcdn.com/image/fetch/$s_!r7q3!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F77e4de00-0279-44ed-ab1a-0d1b6d6f4c49_1280x720.png)](https://substackcdn.com/image/fetch/$s_!r7q3!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2F77e4de00-0279-44ed-ab1a-0d1b6d6f4c49_1280x720.png" \t "_blank)

**1. Problem Statement – Secure File Encryption**

In today’s digital world, **sensitive data is frequently stored, transmitted, and processed** across various devices and networks. Without proper protection, these files are vulnerable to unauthorized access, tampering, or theft. Many individuals and organizations continue to store confidential files in plaintext, which can lead to **serious data breaches, financial loss, and compliance violations**.

Despite the availability of robust encryption tools, **non-technical users often lack simple, script-based solutions** that they can integrate into daily workflows or automation scripts. Furthermore, manually encrypting files is error-prone and not scalable.

This project aims to **solve the problem of securing sensitive files** by enabling users to:

* Generate and manage encryption keys securely.
* Encrypt files to prevent unauthorized access.
* Decrypt them reliably when access is required.

The solution will be implemented using **Python’s cryptography library**, providing a lightweight, easy-to-use, and secure method to **encrypt and decrypt files** on demand.

**2. Why We Need This Use Case**

In the digital era, **data security is critical**. Sensitive files like passwords, credentials, financial documents, health records, or business reports must be protected from unauthorized access. Plain-text storage or unsecured file sharing can lead to severe consequences such as data breaches, identity theft, or legal compliance violations.

This use case aims to help learners **build a secure, reusable encryption/decryption tool** using Python. It uses **symmetric encryption** via the cryptography library's Fernet module, which ensures confidentiality and integrity. Instead of relying on external tools, this script gives users complete control over the **key generation**, **file protection**, and **decryption process**, making it ideal for **local automation, secure archiving, or scripting pipelines.**

**3. When We Need This Use Case**

You need this use case in the following scenarios:

* ✅ When you're working with **confidential files** (e.g., .env, .pem, .csv with sensitive data).
* ✅ When transferring sensitive data over **email or unsecured channels**.
* ✅ When you're building **backup or archival scripts** for production environments.
* ✅ When working with **CI/CD pipelines** where secrets need to be encrypted in transit or storage.
* ✅ When storing files in a shared server or cloud bucket and want **extra encryption on top of standard security**.
* ✅ For internal tools or systems requiring **lightweight local encryption** without complex key management services.

**4. Challenge Questions**

**1. Scenario: Accidental Key Deletion**

Your script generated an encryption key and saved it in a file. But a teammate accidentally deleted the secret.key file.  
**Question**: How would you handle this situation and recover or regenerate the key to decrypt the existing files?

**2. Scenario: Secure File Transfer**

You need to send an encrypted file to a client over email.  
**Question**: How would you ensure both the encrypted file and the key are safely delivered without compromising security?

**3. Scenario: Multiple Files Encryption**

You are asked to encrypt all .txt files in a specific folder using the same key.  
**Question**: How would you modify the script to encrypt multiple files at once and handle potential errors?

**4. Scenario: Unauthorized Decryption Attempt**

You received an error while trying to decrypt a file: InvalidToken.  
**Question**: What are the possible reasons for this error, and how would you resolve it?

**5. Scenario: Encrypted File Naming Conflicts**

You encrypt a file named data.txt, and it creates data.txt.enc. Then you try to decrypt it again, but there’s a naming conflict.  
**Question**: How can you improve the script to avoid overwriting files during decryption?

**6. Scenario: Encryption at Rest for Compliance**

A data compliance officer asks you to ensure "encryption at rest" for your system logs.  
**Question**: How would you use this script in an automated pipeline to encrypt all logs daily?

**7. Scenario: Key Management Best Practices**

Your company rotates encryption keys every 90 days.  
**Question**: How would you design a solution that uses a new key every quarter while still being able to decrypt old files?

**8. Scenario: Accidental Encryption of System Files**

You accidentally encrypted system config files like /etc/hosts.  
**Question**: What preventive checks can you implement in the script to avoid encrypting system-critical files?

**9. Scenario: Cross-Platform Compatibility**

You developed the encryption script on Windows, but it must run on a Linux server.  
**Question**: What changes, if any, must be made for cross-platform compatibility?

**10. Scenario: Key Leak Detection**

Your encryption key was accidentally committed to a public Git repository.  
**Question**: What should be your immediate action steps to secure the encrypted data?

**11. Scenario: Folder-Level Encryption**

You want to encrypt a complete folder (and all subfolders) of files.  
**Question**: How would you enhance the script to recursively encrypt all files in a directory tree?

**12. Scenario: Backup Before Encryption**

Some files were corrupted during encryption due to interruptions.  
**Question**: How would you integrate a backup mechanism in your encryption flow to prevent data loss?

**13. Scenario: Multi-User Access Control**

Multiple team members need to decrypt files, but each with different keys.  
**Question**: How would you securely implement multi-user decryption using public-private key concepts?

**14. Scenario: Logging for Auditing**

Auditors request a log of all encrypted and decrypted files with timestamps.  
**Question**: How would you implement logging in the script for traceability?

**15. Scenario: Integration with Cloud Storage**

Your application stores files in AWS S3.  
**Question**: How would you use the script to encrypt files before uploading them to the cloud?

**16. Scenario: Password-Based Encryption**

You want to protect files using a user-defined password instead of a generated key file.  
**Question**: How would you modify the script to accept a password and derive a key securely?

**17. Scenario: Detect File Tampering**

You suspect an encrypted file was modified manually.  
**Question**: How would you verify the integrity of the encrypted file?

**18. Scenario: File Metadata Preservation**

After decryption, file metadata (e.g., timestamp, owner) is lost.  
**Question**: How can you modify the script to preserve original metadata?

**19. Scenario: Selective Decryption**

You want to decrypt only those files that contain specific keywords in the filename.  
**Question**: How would you design that feature into your decryption function?

**20. Scenario: Automating Key Generation**

You’re deploying the script on multiple servers. Each should have a unique key.  
**Question**: How would you automate key generation and storage securely during deployment?

**21. Scenario: File Integrity with Hashing**

You want to ensure that decrypted files match their original form exactly.  
**Question**: How would you implement file hash comparison in your encryption/decryption process?

**22. Scenario: Safe Key Storage**

You are not allowed to store the key as a plaintext .key file.  
**Question**: What are alternative secure storage mechanisms for encryption keys?

**23. Scenario: UI for File Encryption**

Non-technical users are confused by the CLI script.  
**Question**: How would you build a simple GUI or web interface for this file encryption tool?

**24. Scenario: Compression Before Encryption**

You want to reduce file size before encrypting.  
**Question**: How would you add file compression (e.g., zip) before encryption in the script?

**25. Scenario: Batch Decryption with Missing Keys**

A set of .enc files exists, but some keys are missing.  
**Question**: How would you handle batch decryption and alert the user for missing keys without breaking the entire process?

**5. Prerequisites for the Lab**

* 🐍 Basic knowledge of Python syntax, functions, and file handling
* 💻 Python 3 installed on your system
* 🔐 Familiarity with encryption/decryption concepts
* 🖥️ Access to command line or terminal
* 📦 cryptography module installed via pip install cryptography

**6. Advantages and Disadvantages of This Use Case**

**✅ Advantages:**

* Simple and reusable for various file types
* Uses strong encryption (Fernet = AES 128 in CBC mode with HMAC)
* Easy to integrate into automation scripts or DevOps pipelines
* Doesn’t require an external encryption tool
* Key can be stored securely in vaults or env variables

**❌ Disadvantages:**

* If secret.key is lost, the data is permanently inaccessible
* Not ideal for large files (e.g., >500MB) without optimization
* Not secure if the key is stored alongside encrypted files without protection
* Lacks key rotation or audit trail unless extended

**7. Step-by-Step Implementation Instructions**

A simple and secure Python script using the cryptography library to **encrypt and decrypt files**. This script will:

* Generate a key.
* Save/load the key.
* Encrypt a file.
* Decrypt a file.

**🔧 Required Library**

First, install the cryptography package (if not already installed):

pip install cryptography

🔐 Python Script: Encrypt & Decrypt Files

from cryptography.fernet import Fernet

import os

# ----------- KEY MANAGEMENT -----------

# Generate and save key to a file

def generate\_key(key\_path='secret.key'):

key = Fernet.generate\_key()

with open(key\_path, 'wb') as key\_file:

key\_file.write(key)

# Load the key from a file

def load\_key(key\_path='secret.key'):

return open(key\_path, 'rb').read()

# ----------- ENCRYPTION -----------

def encrypt\_file(file\_path, key):

fernet = Fernet(key)

with open(file\_path, 'rb') as file:

original\_data = file.read()

encrypted\_data = fernet.encrypt(original\_data)

with open(file\_path + '.enc', 'wb') as enc\_file:

enc\_file.write(encrypted\_data)

print(f"File encrypted successfully: {file\_path}.enc")

# ----------- DECRYPTION -----------

def decrypt\_file(encrypted\_path, key, output\_path=None):

fernet = Fernet(key)

with open(encrypted\_path, 'rb') as enc\_file:

encrypted\_data = enc\_file.read()

decrypted\_data = fernet.decrypt(encrypted\_data)

if not output\_path:

output\_path = encrypted\_path.replace('.enc', '.dec')

with open(output\_path, 'wb') as dec\_file:

dec\_file.write(decrypted\_data)

print(f"File decrypted successfully: {output\_path}")

# ----------- USAGE EXAMPLE -----------

if \_\_name\_\_ == '\_\_main\_\_':

# Step 1: Generate key (do it once and reuse later)

if not os.path.exists('secret.key'):

generate\_key()

# Step 2: Load key

key = load\_key()

# Step 3: Encrypt a file

encrypt\_file('example.txt', key)

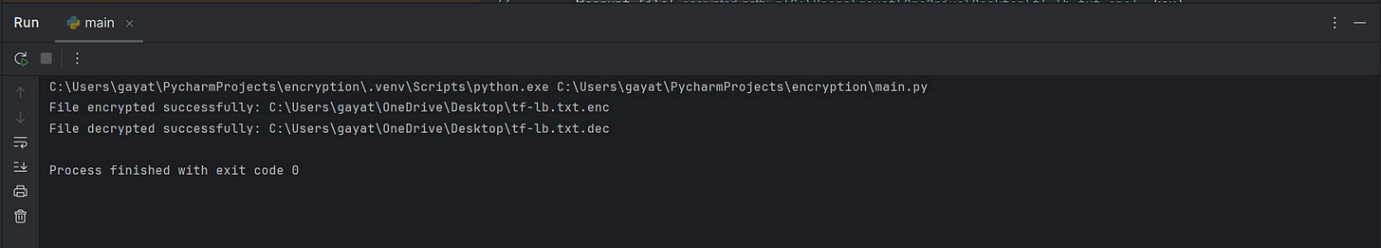
# Step 4: Decrypt a file

decrypt\_file('example.txt.enc', key)

Replace example.txt with your file path.

1. The script uses the cryptography library’s Fernet module for secure symmetric encryption.
2. It generates a key once and stores it in a secret.key file.
3. The encrypt\_file() function reads a file, encrypts its contents, and saves it with a .enc extension.
4. The decrypt\_file() function reads the encrypted file, decrypts it using the key, and writes the original content back.
5. This helps protect sensitive files by allowing easy encryption and decryption with the same key.

**EXPECTED OUTPUT:**

[[](https://substackcdn.com/image/fetch/$s_!pDSw!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2Ff367cd85-209d-4709-ae47-1a1f7e0fc81e_1823x327.png)](https://substackcdn.com/image/fetch/$s_!pDSw!,f_auto,q_auto:good,fl_progressive:steep/https%3A%2F%2Fsubstack-post-media.s3.amazonaws.com%2Fpublic%2Fimages%2Ff367cd85-209d-4709-ae47-1a1f7e0fc81e_1823x327.png" \t "_blank)

**8. Conclusion**

File encryption is **not just a security feature—it's a compliance requirement** in many industries. This use case empowers learners to write their own file encryption tool from scratch using **Python’s cryptography module**, promoting a deeper understanding of symmetric encryption, secure key handling, and real-world security practices.

The step-by-step lab teaches students how to:

* Work with encryption keys
* Securely encrypt/decrypt files
* Automate encryption in local workflows or pipelines

Once mastered, this use case becomes a building block for advanced topics like secure messaging, database encryption, or even integration with cloud key management services.