Course Name:	**Applied Cryptography (16U3263)**	**Semester:**	**V**
Date of Performance:	**28/07/25**	**DIV/ Batch No:**	**AC2**
Student Name:	**Aaryan Sharma**	**Roll No:**	**16010123012**

Experiment No: 1
Title: Encryption-Decryption programs using classical cryptography

Aim and Objective of the Experiment:

Using Cryptographic Fernet library for key management.

COs to be achieved:

CO2: Implement various Cryptographic arithmetic algorithms for securing systems

CO3: Analyze and Implement Symmetric and Symmetric Key Cryptography Algorithms

Books/ Journals/ Websites referred:

1. Stallings, W., Cryptography and Network Security: Principles and Practice, Second edition, Person Education
2. “Applied Cryptography” Bruce Schneier, Second edition
3. https://cryptography.io/en/latest/fernet/, last retrieved on July 28,2025
4. https://www.geeksforgeeks.org/python/fernet-symmetric-encryption-using-cryptography-module-in-python/, last retrieved on July 28,2025
5. https://www.tutorialspoint.com/fernet-symmetric-encryption-using-a-cryptography-module-in-python, last retrieved on July 28,2025

Theory:

Define Key management:

Key management is the process of creating, distributing, storing, using, rotating, and retiring cryptographic keys securely throughout their lifecycle. It ensures that keys remain protected from unauthorized access or misuse while being available for legitimate operations. Effective key management is essential for maintaining data confidentiality, integrity, and compliance with security standards.

List typical activities involved in key management:

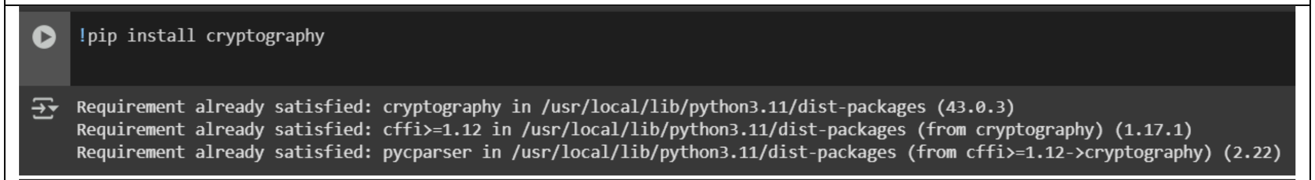
1. Key generation – Creating strong cryptographic keys.
2. Key distribution – Securely delivering keys to authorized parties.
3. Key storage – Protecting keys from unauthorized access while ensuring availability.
4. Key usage – Using keys only for their intended cryptographic operations.
5. Key rotation – Periodically replacing keys to limit exposure.
6. Key backup and recovery – Ensuring keys can be restored if lost or corrupted.

7. Key revocation – Invalidating compromised or expired keys.
8. Key destruction – Securely deleting keys that are no longer needed.
9. Auditing and logging – Tracking key usage for accountability and compliance.

Cryptography Fernet:
Fernet is a symmetric encryption algorithm that provides authenticated encryption to ensure both confidentiality and integrity of the data. It's part of the cryptography library and is designed to be easy to use for secure encryption and decryption.

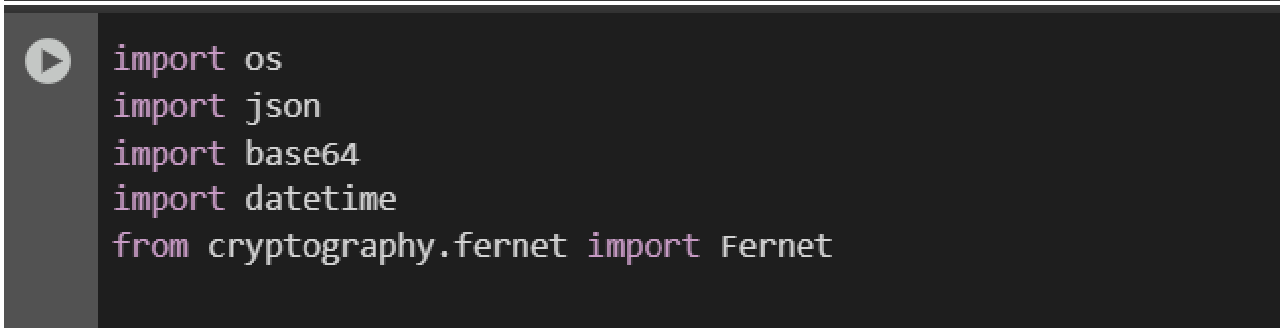
Features of Fernet:
1. Symmetric Encryption: Uses the same key for both encryption and decryption.
2. Authenticated Encryption: Ensures data integrity using HMAC, preventing tampering.
3. Uses AES Encryption: Employs AES in CBC mode for strong security.
4. Includes Timestamp: Allows verification of token age and optional expiration.
5. URL Safe: Encrypted tokens are URL-safe Base64 encoded strings.
6. Easy to Use: Handles key generation, encryption, and decryption with minimal setup.
7. Cross-Platform: Can be used across different systems and programming languages.

Code:

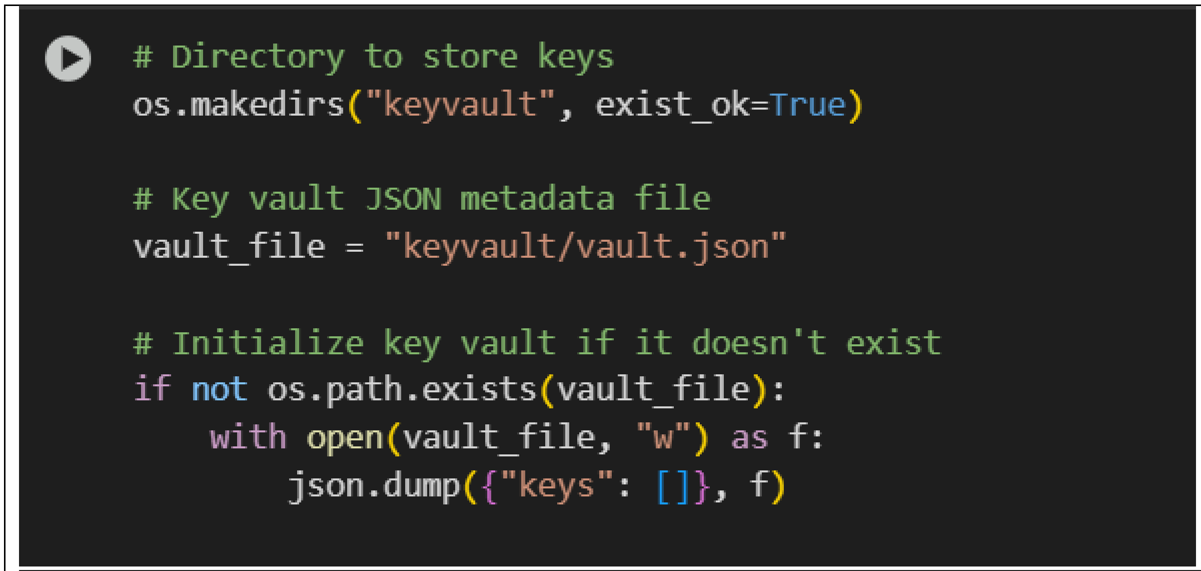


```
!pip install cryptography
```

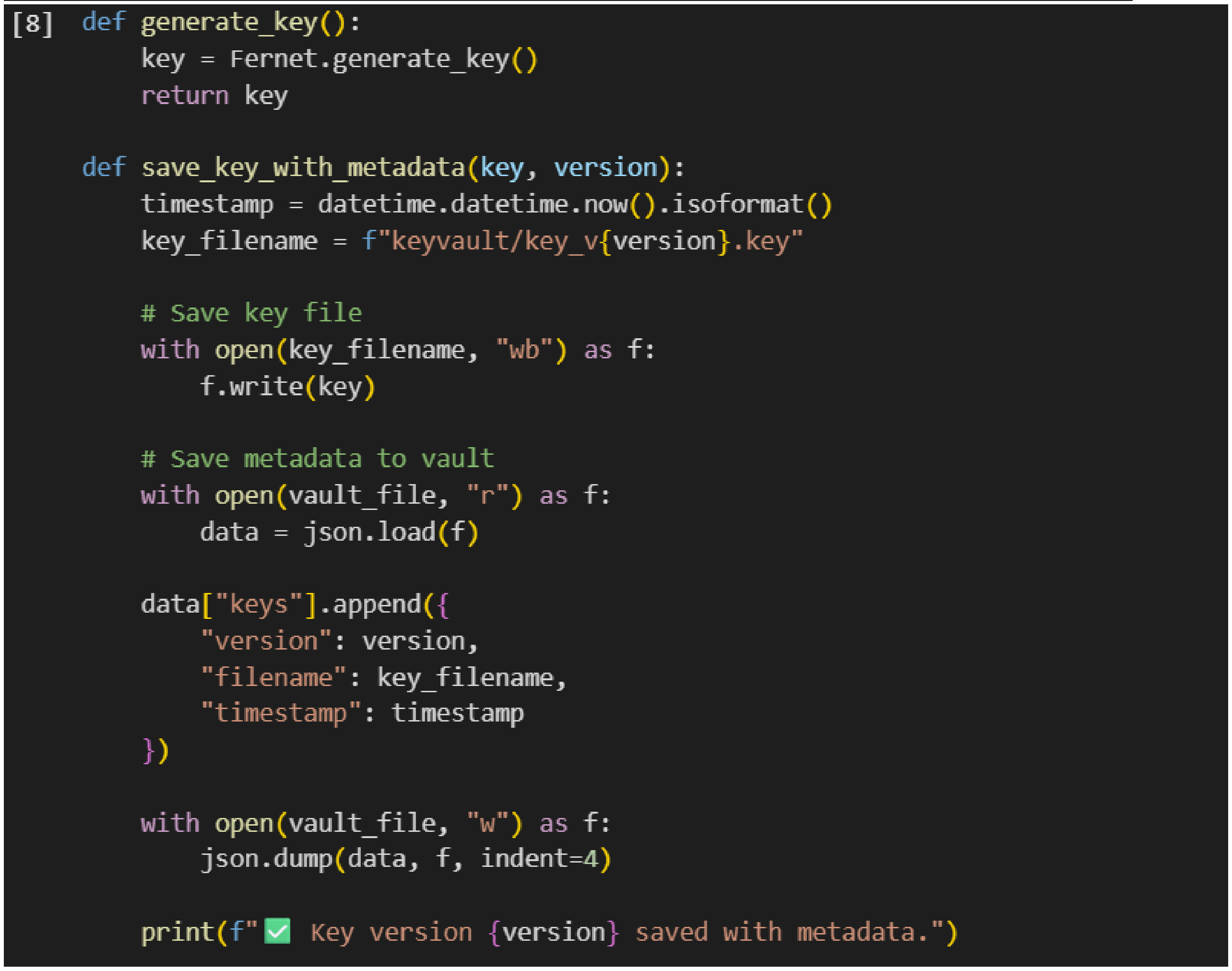
```
Requirement already satisfied: cryptography in /usr/local/lib/python3.11/dist-packages (43.0.3)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.11/dist-packages (from cryptography) (1.17.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.12->cryptography) (2.22)
```



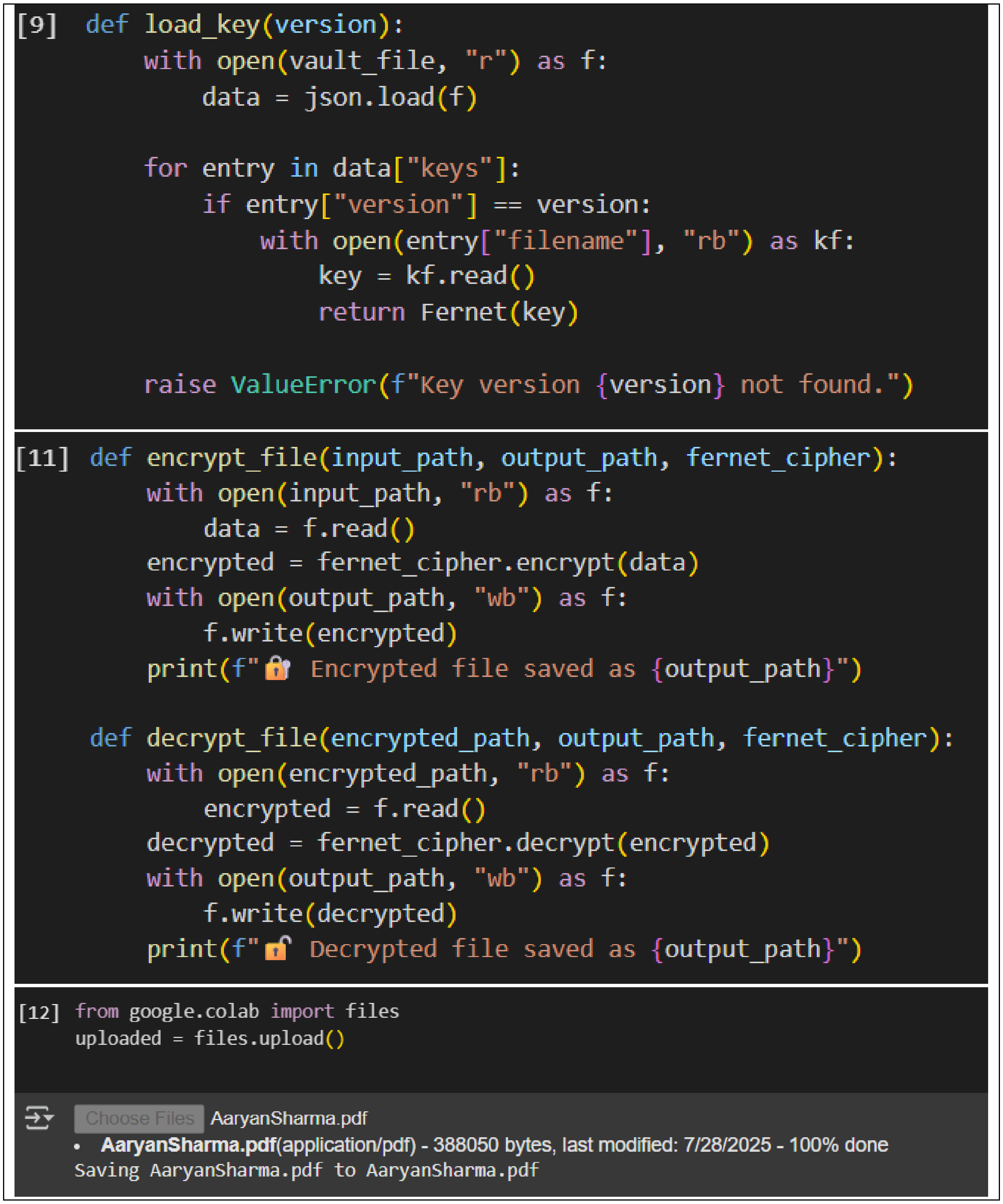
```python
import os
import json
import base64
import datetime
from cryptography.fernet import Fernet
```



```python
# Directory to store keys
os.makedirs("keyvault", exist_ok=True)

# Key vault JSON metadata file
vault_file = "keyvault/vault.json"

# Initialize key vault if it doesn't exist
if not os.path.exists(vault_file):
    with open(vault_file, "w") as f:
        json.dump({"keys": []}, f)
```



```python
[8] def generate_key():
        key = Fernet.generate_key()
        return key

    def save_key_with_metadata(key, version):
        timestamp = datetime.datetime.now().isoformat()
        key_filename = f"keyvault/key_v{version}.key"

        # Save key file
        with open(key_filename, "wb") as f:
            f.write(key)

        # Save metadata to vault
        with open(vault_file, "r") as f:
            data = json.load(f)

        data["keys"].append({
            "version": version,
            "filename": key_filename,
            "timestamp": timestamp
        })

        with open(vault_file, "w") as f:
            json.dump(data, f, indent=4)

        print(f"✅ Key version {version} saved with metadata.")
```



```
[9]  def load_key(version):
         with open(vault_file, "r") as f:
             data = json.load(f)

         for entry in data["keys"]:
             if entry["version"] == version:
                 with open(entry["filename"], "rb") as kf:
                     key = kf.read()
                     return Fernet(key)

         raise ValueError(f"Key version {version} not found.")
```

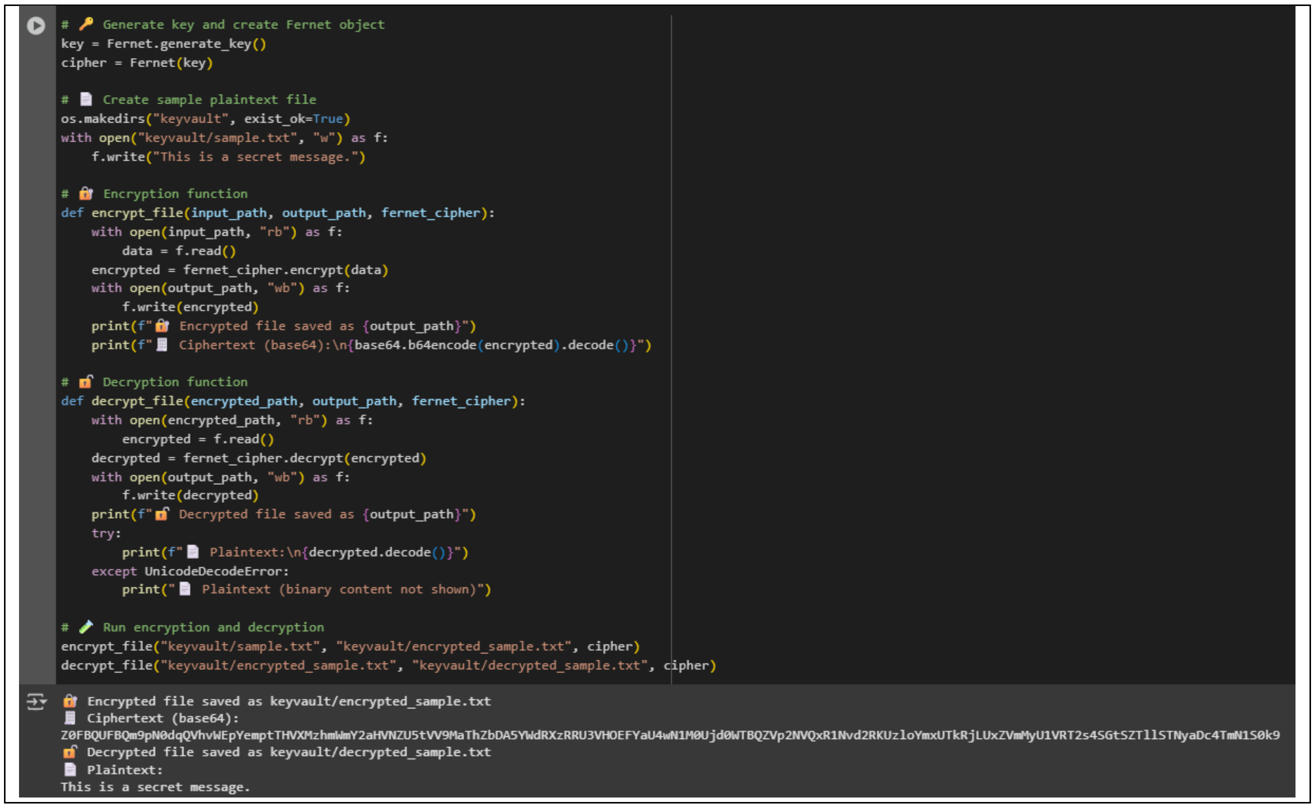
```
[11]  def encrypt_file(input_path, output_path, fernet_cipher):
          with open(input_path, "rb") as f:
              data = f.read()
          encrypted = fernet_cipher.encrypt(data)
          with open(output_path, "wb") as f:
              f.write(encrypted)
          print(f"🔐 Encrypted file saved as {output_path}")

      def decrypt_file(encrypted_path, output_path, fernet_cipher):
          with open(encrypted_path, "rb") as f:
              encrypted = f.read()
          decrypted = fernet_cipher.decrypt(encrypted)
          with open(output_path, "wb") as f:
              f.write(decrypted)
          print(f"🔓 Decrypted file saved as {output_path}")
```

```
[12]  from google.colab import files
      uploaded = files.upload()
```

Choose Files AaryanSharma.pdf

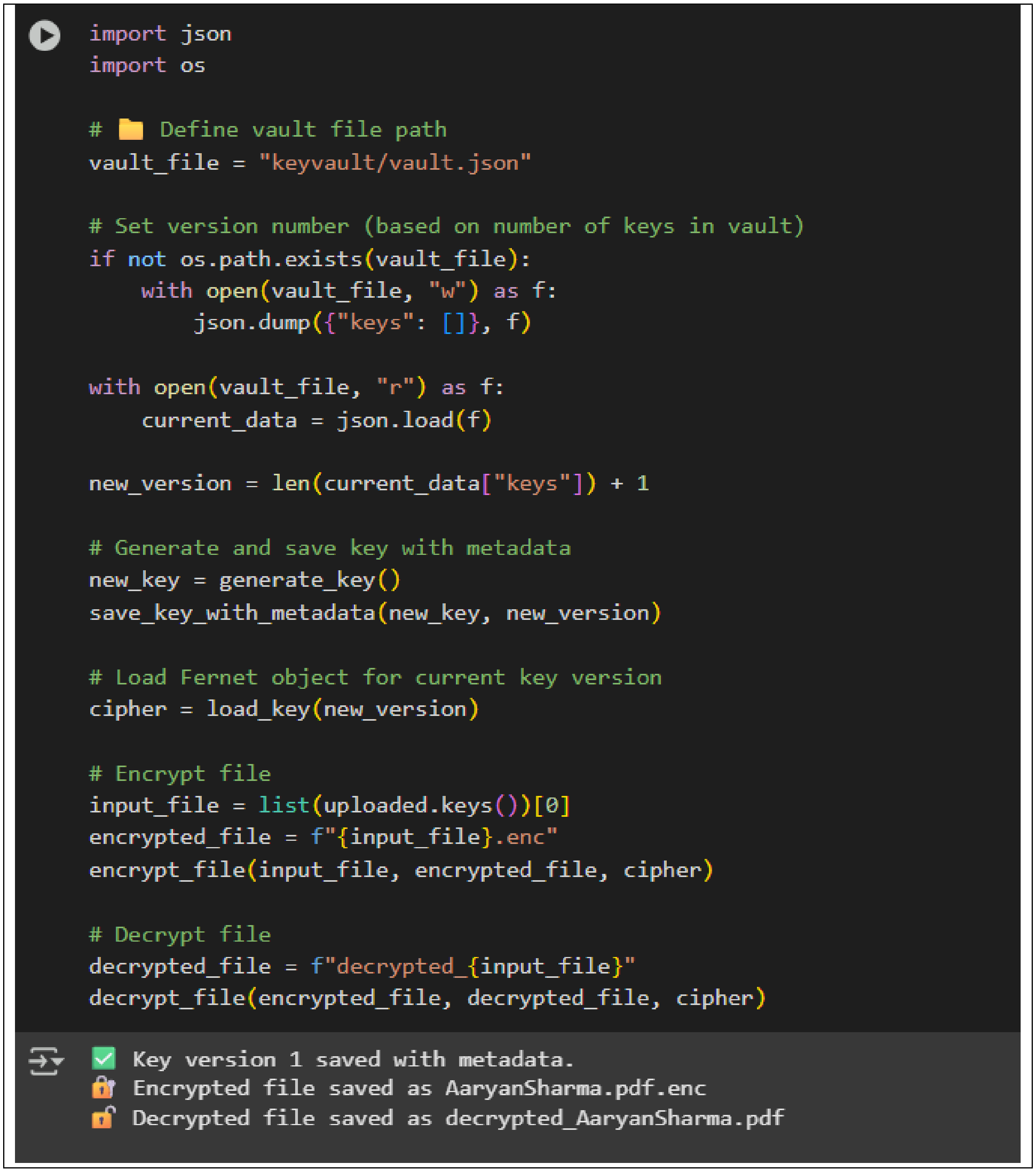
- **AaryanSharma.pdf**(application/pdf) - 388050 bytes, last modified: 7/28/2025 - 100% done

```
Saving AaryanSharma.pdf to AaryanSharma.pdf
```



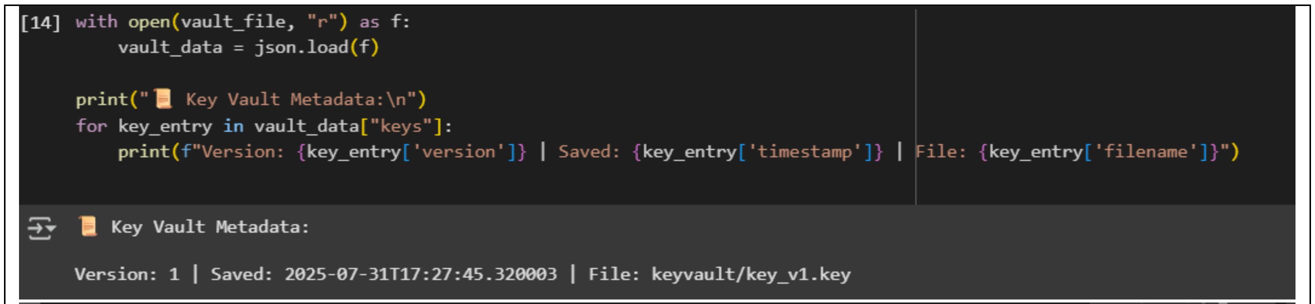
```python
# 🔑 Generate key and create Fernet object
key = Fernet.generate_key()
cipher = Fernet(key)

# 📄 Create sample plaintext file
os.makedirs("keyvault", exist_ok=True)
with open("keyvault/sample.txt", "w") as f:
    f.write("This is a secret message.")

# 🔐 Encryption function
def encrypt_file(input_path, output_path, fernet_cipher):
    with open(input_path, "rb") as f:
        data = f.read()
    encrypted = fernet_cipher.encrypt(data)
    with open(output_path, "wb") as f:
        f.write(encrypted)
    print(f"🔐 Encrypted file saved as {output_path}")
    print(f"📄 Ciphertext (base64):\n{base64.b64encode(encrypted).decode()}")

# 🔓 Decryption function
def decrypt_file(encrypted_path, output_path, fernet_cipher):
    with open(encrypted_path, "rb") as f:
        encrypted = f.read()
    decrypted = fernet_cipher.decrypt(encrypted)
    with open(output_path, "wb") as f:
        f.write(decrypted)
    print(f"🔓 Decrypted file saved as {output_path}")
    try:
        print(f"📄 Plaintext:\n{decrypted.decode()}")
    except UnicodeDecodeError:
        print("📄 Plaintext (binary content not shown)")

# 🧪 Run encryption and decryption
encrypt_file("keyvault/sample.txt", "keyvault/encrypted_sample.txt", cipher)
decrypt_file("keyvault/encrypted_sample.txt", "keyvault/decrypted_sample.txt", cipher)
```

```
🔐 Encrypted file saved as keyvault/encrypted_sample.txt
📄 Ciphertext (base64):
Z0FBQUFBQm9pN0dqQVhvWEpYemptTHVXMzhmWmY2aHVNZU5tVV9MaThZbDA5YWdRXzRRU3VHOEFYaU4wN1M0Ujd0WTBQZVp2NVQxR1Nvd2RKUzloYmxUTkRjLUxZVmMyU1VRT2s4SGtSZTllSTNyaDc4TmN1S0k9
🔓 Decrypted file saved as keyvault/decrypted_sample.txt
📄 Plaintext:
This is a secret message.
```



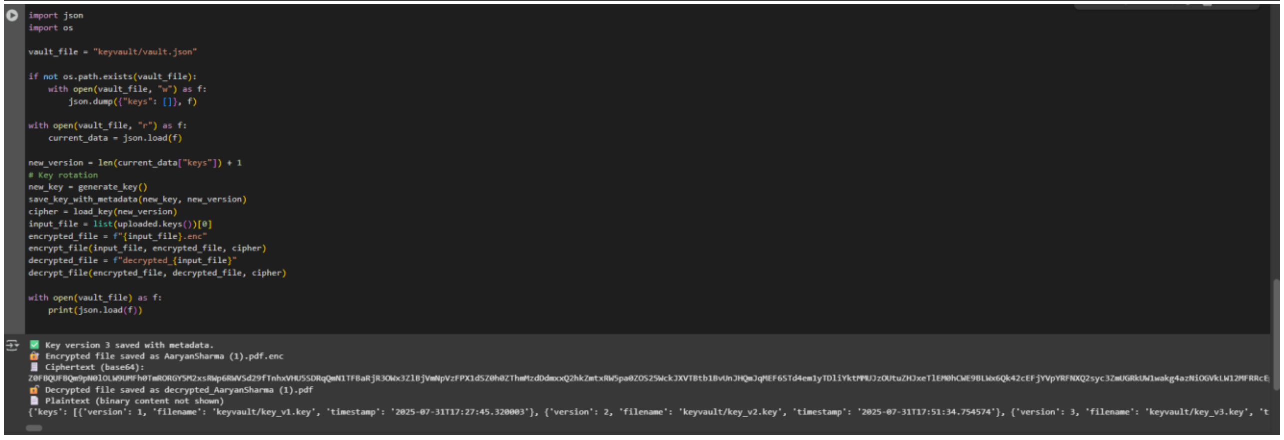
```python
import json
import os

# 📁 Define vault file path
vault_file = "keyvault/vault.json"

# Set version number (based on number of keys in vault)
if not os.path.exists(vault_file):
    with open(vault_file, "w") as f:
        json.dump({"keys": []}, f)

with open(vault_file, "r") as f:
    current_data = json.load(f)

new_version = len(current_data["keys"]) + 1

# Generate and save key with metadata
new_key = generate_key()
save_key_with_metadata(new_key, new_version)

# Load Fernet object for current key version
cipher = load_key(new_version)

# Encrypt file
input_file = list(uploaded.keys())[0]
encrypted_file = f"{input_file}.enc"
encrypt_file(input_file, encrypted_file, cipher)

# Decrypt file
decrypted_file = f"decrypted_{input_file}"
decrypt_file(encrypted_file, decrypted_file, cipher)
```

```
✅ Key version 1 saved with metadata.
🔐 Encrypted file saved as AaryanSharma.pdf.enc
🔓 Decrypted file saved as decrypted_AaryanSharma.pdf
```



```
[14] with open(vault_file, "r") as f:
         vault_data = json.load(f)

     print("📜 Key Vault Metadata:\n")
     for key_entry in vault_data["keys"]:
         print(f"Version: {key_entry['version']} | Saved: {key_entry['timestamp']} | File: {key_entry['filename']}")
```

```
📜 Key Vault Metadata:

Version: 1 | Saved: 2025-07-31T17:27:45.320003 | File: keyvault/key_v1.key
```



```
import json
import os

vault_file = "keyvault/vault.json"

if not os.path.exists(vault_file):
    with open(vault_file, "w") as f:
        json.dump({"keys": []}, f)

with open(vault_file, "r") as f:
    current_data = json.load(f)

new_version = len(current_data["keys"]) + 1
# Key rotation
new_key = generate_key()
save_key_with_metadata(new_key, new_version)
cipher = load_key(new_version)
input_file = list(uploaded.keys())[0]
encrypted_file = f"{input_file}.enc"
encrypt_file(input_file, encrypted_file, cipher)
decrypted_file = f"decrypted_{input_file}"
decrypt_file(encrypted_file, decrypted_file, cipher)

with open(vault_file) as f:
    print(json.load(f))
```

```
✅ Key version 3 saved with metadata.
🔐 Encrypted file saved as AaryanSharma (1).pdf.enc
📃 Ciphertext (base64):
Z0FBQUFBQm9pN0lOLW9UMFh0TmRORGY5M2xsRWp6RWVSd29fTnhxVHU5SDRqQmN1TFBaRjR3OWx3ZlBjVmNpVzFPX1dSZ0h0ZThmMzdDdmxxQ2hkZmtxRW5pa0ZOS25WckJXVTBtb1BvUnJHQmJqMEF6STd4em1yTDliYktMMUJzOUtuZHJxeTlEM0hCWE9BLWx6Qk42cEFjYVpYRFNXQ2syc3ZmUGRkUW1wakg4azNiOGVkLW12MFRRcE
🔓 Decrypted file saved as decrypted_AaryanSharma (1).pdf
📄 Plaintext (binary content not shown)
{'keys': [{'version': 1, 'filename': 'keyvault/key_v1.key', 'timestamp': '2025-07-31T17:27:45.320003'}, {'version': 2, 'filename': 'keyvault/key_v2.key', 'timestamp': '2025-07-31T17:51:34.754574'}, {'version': 3, 'filename': 'keyvault/key_v3.key', 't
```

Post Lab Subjective/Objective type Questions:

1. **Why is key versioning important in secure systems?**
   Key versioning is important in secure systems because it allows safe rotation and revocation of cryptographic keys without disrupting operations. If a key is compromised, versioning ensures only that version is invalidated, limiting damage. It also enables systems to decrypt or verify older data using the correct key version, ensuring backward compatibility. Additionally, versioning improves auditing by tracking key usage and helps meet regulatory requirements for key management and security.

2. **What are the challenges in securely distributing and storing encryption keys?**
   The main challenges in securely distributing and storing encryption keys include preventing unauthorized access, ensuring keys are not exposed during transmission, and protecting them from theft or misuse when stored. Keys must be distributed over secure channels to avoid interception, and proper authentication is needed to ensure they reach the intended recipient. Storing keys securely is also difficult, as they must be protected from insiders, malware, and system breaches while still being accessible for legitimate use. Additionally, managing key lifecycles (rotation, revocation, and expiry) adds complexity, especially in large, distributed systems.

3. **Describe how key destruction and key recovery procedures are handled in secure systems.**
   In secure systems, key destruction and key recovery are carefully managed to maintain data security. Key destruction involves permanently erasing cryptographic keys so they cannot be

reconstructed or retrieved. This is done when keys expire, are compromised, or are no longer needed. Secure deletion methods include overwriting memory, using hardware-based zeroization, or physically destroying storage media to ensure the key cannot be recovered. Key recovery ensures that legitimate users can regain access to encrypted data if keys are lost or corrupted. This is typically handled through key escrow (storing copies of keys in a secure location), recovery agents, or splitting keys using secret-sharing techniques. Recovery procedures require strict authentication and authorization to prevent misuse.

Together, these processes protect data confidentiality while allowing continuity of access when necessary.

Conclusion:

I have successfully completed the experiment on encryption and decryption using the Cryptography Fernet library. Through this experiment, I learned how Fernet provides authenticated encryption ensuring both confidentiality and integrity of data.