Course Name:	Applied Cryptography (16U3263)	Semester:	V
Date of Performance:	31 / 07 / 2025	DIV/ Batch No:	D2
Student Name:	Shreyans Tatiya	Roll No:	16010123325

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 refers to the process of handling cryptographic keys in a secure manner. This includes their generation, exchange, storage, use, rotation, and destruction. The goal is to protect the confidentiality, integrity, and availability of keys throughout their lifecycle.

List typical activities involved in key management:

Key Generation – Creating strong cryptographic keys using secure algorithms.
Key Distribution – Safely delivering keys to intended recipients.
Key Storage – Securely storing keys to prevent unauthorized access.
Key Usage – Using keys for encryption, decryption, signing, or authentication.
Key Rotation – Periodically replacing keys to reduce the risk of compromise.
Key Revocation – Disabling keys that are no longer trusted.
Key Destruction – Securely deleting keys when they are no longer needed.

Cryptography Fernet :

Fernet is a symmetric encryption method provided by the cryptography library in Python. It ensures that a message encrypted using Fernet cannot be manipulated or read without the key. It uses AES in CBC mode with a 128-bit key for encryption and HMAC using SHA256 for authentication.

Features of Fernet:

- **Symmetric encryption** – The same key is used for encryption and decryption.
- **Secure by default** – Uses AES-128 encryption and SHA256 for HMAC.
- **Timestamp included** – Helps track when the token was generated.
- **Easy to use** – Simple API for encrypting and decrypting data.
- **Prevents tampering** – Ensures message integrity and authenticity.

Code :

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

def generate_key():
    return Fernet.generate_key()

def encrypt_message(message, key):
    f = Fernet(key)
    encrypted = f.encrypt(message.encode())
    return encrypted

def decrypt_message(encrypted_message, key):
    f = Fernet(key)
    return f.decrypt(encrypted message).decode()

def create_key_metadata(key_version, key):
    metadata = {
        "version": key version,
        "created_at": datetime.datetime.utcnow().isoformat(),
        "key": key.decode()
    }
```

```
    return metadata

def rotate_key(old_key, new_key, encrypted_data):

    old_fernet = Fernet(old_key)
    plaintext = old_fernet.decrypt(encrypted_data)

    new_fernet = Fernet(new_key)
    new_encrypted_data = new_fernet.encrypt(plaintext)
    return new_encrypted_data

key_v1 = generate_key()
metadata_v1 = create_key_metadata("v1", key_v1)
print("Key V1 Metadata:\n", json.dumps(metadata_v1, indent=2))

message = "Confidential: Key management is critical!"
encrypted_msg_v1 = encrypt_message(message, key_v1)
print("\nEncrypted Message V1:\n", encrypted_msg_v1)

decrypted_msg_v1 = decrypt_message(encrypted_msg_v1, key_v1)
print("\nDecrypted Message V1:\n", decrypted_msg_v1)

key_v2 = generate_key()
metadata_v2 = create_key_metadata("v2", key_v2)
print("\nKey V2 Metadata:\n", json.dumps(metadata_v2, indent=2))

encrypted_msg_v2 = rotate_key(key_v1, key_v2, encrypted_msg_v1)
print("\nEncrypted Message V2 (after key rotation):\n", encrypted_msg_v2)

decrypted_msg_v2 = decrypt_message(encrypted_msg_v2, key_v2)
print("\nDecrypted Message V2:\n", decrypted_msg_v2)
```

Output:

```
Key V1 Metadata:
 {
  "version": "v1",
  "created_at": "2025-07-31T17:35:14.952696",
  "key": "-ZgfZhC25rXMNK12AiCFKNfphBOETE6PqdaMfsZJs6Q="
}

Encrypted Message V1:
 b'gAAAAABoi6lSh2nMkxgeT2huPM5ZYz8Pr8ie4EqB3VltnOh6BX4XVlmJmcQ-jeDLiWEuvijG8s0qXelqGOR__kGq7sYx83CvhLDvPqq2G97kzVSTjNK7cOQ-Bzbgw6iHGlKFWeGXIlrG'

Decrypted Message V1:
 Confidential: Key management is critical!

Key V2 Metadata:
 {
  "version": "v2",
  "created_at": "2025-07-31T17:35:14.953673",
  "key": "wR-EBzw2Vf6TB5XEdv_JHhU_Uv1aQBaL36JrIV7X-F0="
}

Encrypted Message V2 (after key rotation):
 b'gAAAAABoi6lSKf9PVBMPG8TwC9oYYbsr8OcRLqNo3kZYioq8A1zMqfP8mrGFYlhfSligf8U4Cu4BitnfOXbVRkM21FJ3sAB6VdcWBp4JagnZ4MF0E-icPsEmLi_YEW2KVLzvrvVGDPdm'

Decrypted Message V2:
 Confidential: Key management is critical!
```

Post Lab Subjective/Objective type Questions:

1. Why is key versioning important in secure systems?

Key versioning involves assigning a unique version identifier to each cryptographic key. It is important because:

- **Supports Key Rotation:** Systems can rotate keys (replace old keys with new ones) without affecting encrypted data, as the version number helps determine which key to use for decryption.
- **Backwards Compatibility:** Older data encrypted with previous keys can still be decrypted if the key version is known and the old key is retained securely.
- **Auditability:** Versioning allows tracking of which key was used when, improving traceability and accountability.
- **Security Response:** If a key is compromised, it can be revoked or rotated without impacting the entire system. Data associated with a specific version can be isolated.

Example: Token "abc123" is encrypted using key version `v3`, so the system fetches `v3` key from the key store for decryption.

2. What are the challenges in securely distributing and storing encryption keys?

Key distribution and storage are **critical** for maintaining system security, but face several challenges:

🔄 **Distribution Challenges:**

- **Man-in-the-Middle Attacks:** If keys are sent over insecure channels, attackers can intercept them.
- **Lack of Trust:** Ensuring both sender and receiver trust the source of the key (authentication).
- **Scalability:** Securely distributing keys to a large number of users or services can be complex.

Solutions: Use secure protocols like TLS, pre-shared keys, hardware security modules (HSM), or public-key infrastructure (PKI).

🗄️ **Storage Challenges:**

- **Unauthorized Access:** Storing keys in plaintext or insecure environments can lead to data breaches.
- **Key Leakage via Logs or Memory:** Poor practices might expose keys through logs, dumps, or debugging info.
- **Backup & Recovery:** Keys must be backed up securely; loss of keys means permanent loss of encrypted data.

Solutions: Store keys in HSMs, cloud KMS (e.g., AWS KMS, GCP KMS), or OS-level key vaults (e.g., TPM, Apple Secure Enclave).

3. Describe how key destruction and key recovery procedures are handled in secure systems.

Key Destruction:

- **Goal:** Permanently delete keys so that encrypted data cannot be decrypted.
- **Methods:**

- Overwriting keys in memory before releasing.
- Secure erase from storage using specialized software/hardware.
- Revoking keys from key management systems (KMS), marking them as "expired" or "disabled."

- **Policy Enforcement:** Organizations define retention policies and compliance guidelines for destruction (e.g., NIST SP 800-88).

Without proper destruction, attackers might retrieve "deleted" keys from memory dumps or storage.

Key Recovery:

- **Goal:** Retrieve a lost or compromised key or re-establish data access.

- **Methods:**

  - **Key Escrow:** Keys are securely stored by a trusted third party.

  - **Shamir's Secret Sharing:** A key is split into parts and distributed across multiple custodians. A threshold number of parts can reconstruct it.

  - **Backup Systems:** Keys are encrypted and stored in secure backup systems, often with access controls and audit trails.

- **Security Checks:** Recovery often requires multi-factor authentication and logging to prevent misuse.

Recovery must balance **security and availability**: if too restrictive, data may be lost; if too lenient, attackers may abuse it.

Conclusion:

This lab demonstrated practical key management with Fernet, covering generation, versioning, encryption/decryption, and seamless key rotation. By applying secure storage and lifecycle practices, we reinforced the importance of safeguarding keys at every stage. The exercise solidified our grasp of symmetric-key cryptography and prepared us to implement these techniques in real-world systems.