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:

```
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": "2025-08-11T11:19:01.296387",
  "key": "_0mjUlvp7xdCHh2d08FYkS11AH1pq0F1E6GgZAHQsAU="
}

Encrypted Message V1:
 b'gAAAAABomdGlPOOy6Y84_LotPGDTmIjE1hK2sn4arVwAWYL-xP1O7xYw5OCn1997WzoOy5abZEnI1_TypdNLNFimAeamIVW-6iO-PxfUsESntJnRmwpwicqtUs-1M6dqDTGIz8W1OAts'

Decrypted Message V1:
 Confidential: Key management is critical!

Key V2 Metadata:
 {
  "version": "v2",
  "created_at": "2025-08-11T11:19:01.302887",
  "key": "jwdY0Hw5HMpWZfCa7ez9eNR74onksJLu97xGr4qWAlo="
}

Encrypted Message V2 (after key rotation):
 b'gAAAAABomdGlRu2X-Kkqu4wlFaBbsjVUfsQVUIGo0VrQeCDTdsdykbhAjOFchzMDXArnL69ootDNJUwqkfLPW4ja6MTuPjWb8KNaf0Q4esJLMPktW9pEuNqMg7d17uj72d4FkA2o_CSt'

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 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.

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.

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. 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.