Walchand College of Engineering, Sangli *(Government Aided Autonomous Institute)*	
AY 2025-26	
Course Information	
Programme	B.Tech. (Computer Science and Engineering)
Class, Semester	Final Year B. Tech., Sem VII
Course Code	6CS451
Course Name	Cryptography and Network Security Lab
PRN	22510016

Experiment No. 06

Title - Apply AES algorithm for practical applications

Objectives:

Study the structure of AES, including key sizes (128, 192, 256 bits), encryption rounds, and the internal processes (Sub Bytes, Shift Rows, Mix Columns, Add Round Key).

Problem Statement: In an era where digital communication and data storage have become integral to everyday life,
ensuring the security and privacy of sensitive information is a critical challenge. With the increasing frequency of cyberattacks, data breaches, and unauthorized access, traditional methods of data protection are no longer sufficient. Individuals, businesses, and governments alike require robust encryption techniques to safeguard confidential data.

The ***Advanced Encryption Standard (AES)*** is a symmetric key encryption algorithm that has been widely accepted as a secure and efficient standard for encrypting digital data. Despite its proven security, many real-world systems either lack proper encryption or use outdated or weak encryption mechanisms. Moreover, improper implementation of AES can lead to vulnerabilities that compromise the overall security of the system.

This project addresses the need to ***apply the AES algorithm effectively in practical applications***, such as secure file storage, encrypted messaging, and protected communication in IoT systems. By integrating AES into these applications, the goal is to enhance data confidentiality and security without significantly impacting system performance or usability.

Equipment/Tools:

Python 3.8+ (recommended 3.10 or later)
`pycryptodome` library (`pip install pycryptodome`)
Text editor / IDE (VS Code, PyCharm) and terminal

Theory:

AES (Advanced Encryption Standard) is a symmetric block cipher standardized by NIST. Key points:

Block size: 128 bits (16 bytes)

Key sizes: 128, 192, 256 bits

Number of rounds: depends on key size

- AES-128: 10 rounds
- AES-192: 12 rounds
- AES-256: 14 rounds

Each round (except the final) applies four transformations:

1. ***SubBytes*** — Non-linear byte substitution using an S-box (a lookup table) applied to each byte.
2. ***ShiftRows*** — Cyclically shift rows of the 4x4 state matrix.
3. ***MixColumns*** — Linear mixing of each column using GF(2^8) arithmetic.
4. ***AddRoundKey*** — XOR the state with a round key derived from the cipher key using the key schedule.

Final round omits MixColumns. AES encryption transforms a 128-bit plaintext block into a 128-bit ciphertext block using the round transformations above.

Important implementation notes:

AES itself is ***not*** authenticated — use an AEAD mode (e.g., GCM) or combine AES with an HMAC for authenticity.

Never reuse IV/nonce with the same key for modes like GCM or CTR.

Use a secure key source (e.g., os.urandom(32) for 256-bit key) and protect keys in storage.

Procedure:

Install Python and PyCryptodome.

Write Python code to generate a secret key.

Use the AES algorithm to encrypt data.

Save the ciphertext.

Use the same key to decrypt and get back the original data.

Test with files or text messages.

Steps:

Install library:

pip install pycryptodome

Write simple Python program for AES encryption and decryption.

Run the program with sample input text or files.

Observe that encrypted text is unreadable.

Decrypt with the same key and verify it matches the original.

Program:

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import padding
import os

# Generate key and IV
key = os.urandom(32)  # 256-bit key
iv = os.urandom(16)   # 128-bit IV

# AES cipher
cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())

# Pad plaintext to block size (16 bytes)
padder = padding.PKCS7(128).padder() # 128 bits = 16 bytes block
plaintext = b"Lets take AES Demo"
padded_data = padder.update(plaintext) + padder.finalize()

# Encrypt
encryptor = cipher.encryptor()
ciphertext = encryptor.update(padded_data) + encryptor.finalize()
print("Ciphertext:", ciphertext)

# Decrypt
decryptor = cipher.decryptor()
decrypted_padded = decryptor.update(ciphertext) + decryptor.finalize()

# Remove padding
unpadder = padding.PKCS7(128).unpadder()
decrypted = unpadder.update(decrypted_padded) + unpadder.finalize()
print("Decrypted:", decrypted.decode())
```

Output:

```
PS C:\Users\aishw\OneDrive\Documents\Final Year\CNS Practicals\Assignment_6_23520001> python aesdemo.py
Ciphertext: b'x\xd1\r\x857f\x8d\xd94\x014\x01_\xe9j\x15\xd3\x9b\xbf\x00\x99\xccqcD\x93Q`\x024}\x84\xe9\xd9\x8b\xefa\xeb\xd8bMr\x19\xb4\xd9\x947\xe7'
Decrypted: Hello welcome to the world for cryptography
PS C:\Users\aishw\OneDrive\Documents\Final Year\CNS Practicals\Assignment_6_23520001> python aesdemo.py
Ciphertext: b'\xc2\x82\x80\x87q\xad\x9d\x90y1I\x87+\xcbjr$\xb5\n\x05\x83C|G\x91\x03\xef\xf5\xf2h\x0c\xca'
Decrypted: Lets take AES Demo
PS C:\Users\aishw\OneDrive\Documents\Final Year\CNS Practicals\Assignment_6_23520001>
```

Observations and Conclusion:

Encrypted text looks like random bytes and cannot be understood.
Decrypted text matches the original message exactly.
AES requires the same key for both encryption and decryption.

AES is a strong and widely used encryption algorithm. It is suitable for securing files, messages, and communication. With correct implementation (unique IVs and secure key storage), it protects sensitive data effectively.