Walchand College of Engineering, Sangli (Government Aided Autonomous Institute)

AY 2025-26

Course Information Programme B.Tech. (Computer Science and Engineering)

Final Year B. Tech., Sem VII Class,

Semester Course Code 6CS451

Course Name Cryptography and Network Security Lab

22510064 PARSHWA HERWADE BATCH 1

EXPERIMENT NO. 6

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 Despite its proven security, many real-world systems either lack proper encryption or use outdated or weak encryption mechanisms.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.

Theory

AES Overview

The **Advanced Encryption Standard (AES)** is a symmetric block cipher standardized by NIST in 2001. It operates on **128-bit blocks** with key sizes of **128, 192, or 256 bits**.

- **AES-128** → 10 rounds
- **AES-192** → 12 rounds
- **AES-256** → 14 rounds

AES Structure (Each Round)

1. **SubBytes** – Byte substitution using an S-box (non-linear substitution).
2. **ShiftRows** – Cyclically shifts rows of the state array.
3. **MixColumns** – Mixes each column of the state matrix using polynomial multiplication (not applied in final round).
4. **AddRoundKey** – XORs the round key with the state matrix.

Modes of AES

AES is often used with block cipher modes like ECB, CBC, CFB, and GCM. In this experiment, we use **AES with ECB mode + PKCS5 Padding** for simplicity.

Equipment/Tools

- Hardware: Computer/Laptop
- Software:
    - JDK (Java Development Kit 8+)
    - IDE (Eclipse, IntelliJ, or any text editor)

- Command line for execution

Procedure

1. Study AES algorithm structure (rounds, key sizes, operations).
2. Write Java code for AES encryption and decryption.
3. Accept **plaintext** and **secret key** as user input.
4. Generate AES secret key and initialize cipher.
5. Encrypt the plaintext and display **cipher text** (in Base64 encoding).
6. Decrypt the cipher text and display the **original plaintext**.
7. Verify that the decrypted text matches the original.

Steps

1. Compile the program: javac AESDemo.java
2. Run the program: java AESDemo
3. Enter plaintext message and secret key at runtime.
4. Observe encrypted (cipher text) and decrypted (original) outputs.

CODE:

```
import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
import java.util.Scanner;

public class AES {

    // Method to generate AES key of given size
    public static SecretKey generateKey(int n) throws Exception {
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
```

```
        keyGen.init(n); // key size: 128, 192, or 256
        return keyGen.generateKey();
    }

    // Convert string key to SecretKeySpec
    public static SecretKeySpec getKeyFromString(String key) throws
Exception {
        byte[] keyBytes = key.getBytes("UTF-8");
        // Use only first 16 bytes for 128-bit key
        byte[] keyBytes16 = new byte[16];
        System.arraycopy(keyBytes, 0, keyBytes16, 0,
Math.min(keyBytes.length, 16));
        return new SecretKeySpec(keyBytes16, "AES");
    }

    // Encrypt text
    public static String encrypt(String plainText, SecretKeySpec
secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] cipherText = cipher.doFinal(plainText.getBytes("UTF-
8"));
        return Base64.getEncoder().encodeToString(cipherText);
    }

    // Decrypt text
    public static String decrypt(String cipherText, SecretKeySpec
secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] plainText =
cipher.doFinal(Base64.getDecoder().decode(cipherText));
        return new String(plainText, "UTF-8");
    }

    public static void main(String[] args) {
        try (Scanner sc = new Scanner(System.in)) {
            System.out.println("Enter plaintext message:");
            String plainText = sc.nextLine();

            System.out.println("Enter secret key (any string, min 16
characters recommended):");
```

```
            String keyInput = sc.nextLine();
            SecretKeySpec secretKey = getKeyFromString(keyInput);

            // Encryption
            String cipherText = encrypt(plainText, secretKey);
            System.out.println("\nEncrypted Text (Base64): " +
cipherText);

            // Decryption
            String decryptedText = decrypt(cipherText, secretKey);
            System.out.println("Decrypted Text: " + decryptedText);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:

```
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A6> javac AES.java
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A6> java AES
Enter plaintext message:
HelloWorld
Enter secret key (any string, min 16 characters recommended):
mysecretpassword

Encrypted Text (Base64): mi/6kTPamna+MKu0WSyRhQ==
Decrypted Text: HelloWorld
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A6> java AES
Enter plaintext message:
CryptographyAndNetworkSecurity
Enter secret key (any string, min 16 characters recommended):
securekey1234567

Encrypted Text (Base64): Fyw6NL1HMwu1avq3gUGM1UTv0e3ggyhwZmk64dkXoR4=
Decrypted Text: CryptographyAndNetworkSecurity
```

```
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A6> java AES
Enter plaintext message:
AES is very secure!
Enter secret key (any string, min 16 characters recommended):
thisisaverystrongkey

Encrypted Text (Base64): H8vChV9mMSr8sZYDS0L/czRFhtOrItCNhvL3fm9BZfA=
Decrypted Text: AES is very secure!
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A6>
```

Observations

- AES successfully encrypts plaintext into unreadable cipher text.
- The same key must be used for both encryption and decryption.
- Changing even one character in the key produces a totally different cipher text.
- AES provides strong **confidentiality** and is resistant to brute-force attacks with 128+ bit keys.
- Performance is efficient and suitable for practical applications (messaging, storage, IoT).

Conclusion

AES is a **robust, efficient, and secure** symmetric encryption algorithm.

- It ensures confidentiality of sensitive information.
- Proper implementation in real-world applications can **prevent unauthorized access and data breaches**.
- This experiment demonstrates practical usage of AES in **Java**, confirming that it can be integrated into applications like **secure messaging, file protection, and IoT communication** with minimal performance overhead.