Name: Shreeyash S. Dongarkar

PRN: 22510025

BTech Final Year CSE

Cryptography and Network Security Lab (B – 1)

Assignment 5 (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 Despite its proven security, many real-world systems either lack proper encryption or use outdated or weak encryption mechanisms.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.

1. import javax.crypto.Cipher;

2. import javax.crypto.KeyGenerator;

3. import javax.crypto.SecretKey;

4. import javax.crypto.spec.SecretKeySpec;

5. import java.util.Base64;

6. import java.util.Scanner;

7. import java.util.logging.Level;

8. import java.util.logging.Logger;

9.

10. public class AESExample {

11.     private static final Logger logger = Logger.getLogger(AESExample.class.getName());

12.

13.     public static SecretKey generateKey(int keySize) throws Exception {

14.         KeyGenerator keyGen = KeyGenerator.getInstance("AES");

15.         keyGen.init(keySize);

16.         return keyGen.generateKey();

17.     }

18.

19.     public static SecretKey buildKeyFromString(String keyStr, int keySize) {

20.         byte[] keyBytes = keyStr.getBytes();

21.         byte[] keyBytesPadded = new byte[keySize / 8];

22.         System.arraycopy(keyBytes, 0, keyBytesPadded, 0, Math.min(keyBytes.length, keyBytesPadded.length));

23.         return new SecretKeySpec(keyBytesPadded, "AES");

24.     }

25.

26.     public static String encrypt(String plaintext, SecretKey key) throws Exception {

27.         Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

28.         cipher.init(Cipher.ENCRYPT\_MODE, key);

29.         byte[] encryptedBytes = cipher.doFinal(plaintext.getBytes());

30.         return Base64.getEncoder().encodeToString(encryptedBytes);

31.     }

32.

33.     public static String decrypt(String ciphertext, SecretKey key) throws Exception {

34.         Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

35.         cipher.init(Cipher.DECRYPT\_MODE, key);

36.         byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(ciphertext));

37.         return new String(decryptedBytes);

38.     }

39.

40.     public static void main(String[] args) {

41.         try (Scanner scanner = new Scanner(System.in)) {

42.             System.out.println("==== AES Encryption/Decryption ====");

43.             System.out.println("1. Encrypt");

44.             System.out.println("2. Decrypt");

45.             System.out.print("Enter your choice: ");

46.             int choice = scanner.nextInt();

47.             scanner.nextLine();

48.

49.             System.out.print("Enter key (any string): ");

50.             String keyStr = scanner.nextLine();

51.

52.             System.out.print("Enter key size (128, 192, 256): ");

53.             int keySize = scanner.nextInt();

54.             scanner.nextLine();

55.

56.             SecretKey key = buildKeyFromString(keyStr, keySize);

57.

58.             if (choice == 1) {

59.                 System.out.print("Enter plaintext message: ");

60.                 String plaintext = scanner.nextLine();

61.                 String encryptedText = encrypt(plaintext, key);

62.                 System.out.println("Encrypted (Ciphertext): " + encryptedText);

63.             } else if (choice == 2) {

64.                 System.out.print("Enter ciphertext (Base64): ");

65.                 String ciphertext = scanner.nextLine();

66.                 String decryptedText = decrypt(ciphertext, key);

67.                 System.out.println("Decrypted (Plaintext): " + decryptedText);

68.             } else {

69.                 System.out.println("Invalid choice. Please select 1 or 2.");

70.             }

71.         } catch (Exception e) {

72.             logger.log(Level.SEVERE, "Error in AES operation", e);

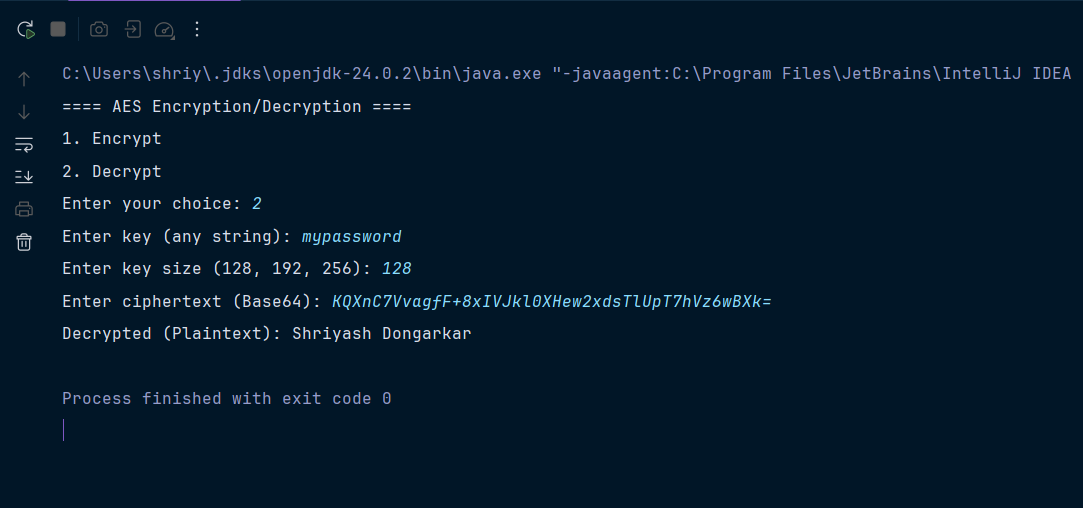
73.         }

74.     }

75. }

76.





**Tools:**

** Software: Java Development Kit (JDK ≥ 8), any IDE (Eclipse, IntelliJ, NetBeans) or text editor**

** Hardware: Computer system with minimum 4 GB RAM**

** Libraries/Packages: javax.crypto, java.util.Base64, java.util.Scanner, java.util.logging**

**Theory:**

The **Advanced Encryption Standard (AES)** is a **symmetric block cipher** standardized by NIST in 2001 to replace DES. It is widely used in secure communication, banking, IoT, and file encryption.

**Key Points:**

* **Block Size**: 128 bits (fixed)
* **Key Sizes**: 128, 192, or 256 bits
* **Encryption Rounds**:
  + 10 rounds for 128-bit keys
  + 12 rounds for 192-bit keys
  + 14 rounds for 256-bit keys
* **Internal Operations**:
  + **SubBytes** – Non-linear byte substitution using an S-box.
  + **ShiftRows** – Cyclic shift of rows in the state matrix.
  + **MixColumns** – Mixing operation that provides diffusion (not in final round).
  + **AddRoundKey** – XOR with round subkey derived from the main key.

AES is secure against brute-force attacks due to large key sizes and is the **current industry standard** for symmetric key encryption.

**Procedure:**

1. Write a **Java program** that implements AES encryption and decryption using the javax.crypto package.
2. Accept **plaintext input** from the user.
3. Accept a **key string** and **key size** (128, 192, 256 bits) from the user.
4. Convert the key string into an AES SecretKeySpec of the required size.
5. Encrypt the plaintext using **AES/ECB/PKCS5Padding** mode.
6. Display the **Base64-encoded ciphertext**.
7. Decrypt the ciphertext using the same key and display the **restored plaintext**.

**Steps:**

1. Import the required Java libraries (javax.crypto, java.util.Base64, etc.).
2. Define methods to:
   * Generate/build AES keys.
   * Encrypt plaintext into ciphertext.
   * Decrypt ciphertext back into plaintext.
3. In the main() method:
   * Display a menu: **1 → Encrypt, 2 → Decrypt**.
   * Read user input (plaintext or ciphertext, key, and key size).
   * Perform the selected operation.
   * Print the result.
4. Run the program for different key sizes (128, 192, 256) and note the results.

**Observations:**

* AES successfully encrypts plaintext into unreadable ciphertext.
* The ciphertext differs each time because of padding and block-level transformation, but decryption with the **same key and key size** restores the original plaintext.
* If a **wrong key** or **different key size** is used during decryption, the process fails.
* Larger key sizes (192, 256) result in stronger security but slightly higher computational cost.

**Conclusion:**

* The AES algorithm was successfully implemented in Java for both encryption and decryption.
* It demonstrates how **symmetric key cryptography** ensures **data confidentiality** in real-world communication.
* AES is far stronger than DES due to larger key sizes and more complex round functions, making it the **preferred industry standard** for securing sensitive data.
* The experiment highlights how AES can be integrated into applications such as **secure file storage, encrypted messaging, and IoT security systems**.