CRYPTO LAB-3

NAME: ABISHEK K G **DATE: 26/01/2026**

REGNO: 23BCE1739 **SLOT: L29+L30**

Source code:

```
import java.io.*;

import java.net.*;

import java.util.Scanner;


public class AESSecureChat {


   // S-box for SubBytes

   private static final int[] SBOX = {

      0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab,
0x76,

      0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72,
0xc0,

      0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31,
0x15,

      0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2,
0x75,

      0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f,
0x84,

      0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58,
0xcf,

      0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f,
0xa8,
```

```
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3,
0xd2,

    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19,
0x73,

    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b,
0xdb,

    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4,
0x79,

    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae,
0x08,

    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b,
0x8a,

    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d,
0x9e,

    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28,
0xdf,

    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb,
0x16

  };


  // Inverse S-box for InvSubBytes

  private static final int[] INV_SBOX = {

    0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7,
0xfb,

    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9,
0xcb,

    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3,
0x4e,

    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1,
0x25,
```

```
    0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6,
0x92,

    0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d,
0x84,

    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45,
0x06,

    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a,
0x6b,

    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6,
0x73,

    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf,
0x6e,

    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe,
0x1b,

    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a,
0xf4,

    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec,
0x5f,

    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c,
0xef,

    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99,
0x61,

    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c,
0x7d
  };


  // Round constants for key expansion
  private static final int[] RCON = {
    0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36
  };
```

```
    // ==================== AES ENCRYPTION/DECRYPTION METHODS
====================

    // Generate 11 round keys from the original key
    public static int[][] keyExpansion(int[] key) {
        int[][] roundKeys = new int[11][16];
        System.arraycopy(key, 0, roundKeys[0], 0, 16);

        for (int round = 1; round <= 10; round++) {
            int[] prevKey = roundKeys[round - 1];
            int[] newKey = new int[16];

            int[] temp = {prevKey[12], prevKey[13], prevKey[14], prevKey[15]};

            // RotWord
            int t = temp[0];
            temp[0] = temp[1];
            temp[1] = temp[2];
            temp[2] = temp[3];
            temp[3] = t;

            // SubWord
            for (int i = 0; i < 4; i++) {
                temp[i] = SBOX[temp[i]];
            }
```

```
        // XOR with Rcon
        temp[0] ^= RCON[round - 1];

        // Generate new key
        for (int i = 0; i < 4; i++) {
            newKey[i] = prevKey[i] ^ temp[i];
        }
        for (int i = 4; i < 16; i++) {
            newKey[i] = prevKey[i] ^ newKey[i - 4];
        }

        roundKeys[round] = newKey;
    }

    return roundKeys;
}

private static void subBytes(int[][] state) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            state[i][j] = SBOX[state[i][j]];
        }
    }
}
```

```
private static void invSubBytes(int[][] state) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            state[i][j] = INV_SBOX[state[i][j]];
        }
    }
}

private static void shiftRows(int[][] state) {
    int temp;

    temp = state[1][0];
    state[1][0] = state[1][1];
    state[1][1] = state[1][2];
    state[1][2] = state[1][3];
    state[1][3] = temp;

    temp = state[2][0];
    state[2][0] = state[2][2];
    state[2][2] = temp;
    temp = state[2][1];
    state[2][1] = state[2][3];
    state[2][3] = temp;

    temp = state[3][3];
    state[3][3] = state[3][2];
```

```
    state[3][2] = state[3][1];
    state[3][1] = state[3][0];
    state[3][0] = temp;
}

private static void invShiftRows(int[][] state) {
    int temp;

    temp = state[1][3];
    state[1][3] = state[1][2];
    state[1][2] = state[1][1];
    state[1][1] = state[1][0];
    state[1][0] = temp;

    temp = state[2][0];
    state[2][0] = state[2][2];
    state[2][2] = temp;
    temp = state[2][1];
    state[2][1] = state[2][3];
    state[2][3] = temp;

    temp = state[3][0];
    state[3][0] = state[3][1];
    state[3][1] = state[3][2];
    state[3][2] = state[3][3];
    state[3][3] = temp;
```

```
}

private static int gmul(int a, int b) {
   int p = 0;
   for (int i = 0; i < 8; i++) {
      if ((b & 1) != 0) {
         p ^= a;
      }
      boolean hiBitSet = (a & 0x80) != 0;
      a <<= 1;
      if (hiBitSet) {
         a ^= 0x1b;
      }
      b >>= 1;
   }
   return p & 0xFF;
}

private static void mixColumns(int[][] state) {
   for (int c = 0; c < 4; c++) {
      int s0 = state[0][c];
      int s1 = state[1][c];
      int s2 = state[2][c];
      int s3 = state[3][c];

      state[0][c] = gmul(s0, 2) ^ gmul(s1, 3) ^ s2 ^ s3;
```

```
        state[1][c] = s0 ^ gmul(s1, 2) ^ gmul(s2, 3) ^ s3;
        state[2][c] = s0 ^ s1 ^ gmul(s2, 2) ^ gmul(s3, 3);
        state[3][c] = gmul(s0, 3) ^ s1 ^ s2 ^ gmul(s3, 2);
    }
}

private static void invMixColumns(int[][] state) {
    for (int c = 0; c < 4; c++) {
        int s0 = state[0][c];
        int s1 = state[1][c];
        int s2 = state[2][c];
        int s3 = state[3][c];

        state[0][c] = gmul(s0, 14) ^ gmul(s1, 11) ^ gmul(s2, 13) ^ gmul(s3, 9);
        state[1][c] = gmul(s0, 9) ^ gmul(s1, 14) ^ gmul(s2, 11) ^ gmul(s3, 13);
        state[2][c] = gmul(s0, 13) ^ gmul(s1, 9) ^ gmul(s2, 14) ^ gmul(s3, 11);
        state[3][c] = gmul(s0, 11) ^ gmul(s1, 13) ^ gmul(s2, 9) ^ gmul(s3, 14);
    }
}

private static void addRoundKey(int[][] state, int[] roundKey) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            state[i][j] ^= roundKey[j * 4 + i];
        }
    }
```

```
}

private static int[][] toStateMatrix(int[] block) {
    int[][] state = new int[4][4];
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            state[i][j] = block[j * 4 + i];
        }
    }
    return state;
}

private static int[] fromStateMatrix(int[][] state) {
    int[] block = new int[16];
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            block[j * 4 + i] = state[i][j];
        }
    }
    return block;
}

public static int[] encryptBlock(int[] plainBlock, int[][] roundKeys) {
    int[][] state = toStateMatrix(plainBlock);

    addRoundKey(state, roundKeys[0]);
```

```
    for (int round = 1; round <= 9; round++) {
        subBytes(state);
        shiftRows(state);
        mixColumns(state);
        addRoundKey(state, roundKeys[round]);
    }

    subBytes(state);
    shiftRows(state);
    addRoundKey(state, roundKeys[10]);

    return fromStateMatrix(state);
}

public static int[] decryptBlock(int[] cipherBlock, int[][] roundKeys) {
    int[][] state = toStateMatrix(cipherBlock);

    addRoundKey(state, roundKeys[10]);

    for (int round = 9; round >= 1; round--) {
        invShiftRows(state);
        invSubBytes(state);
        addRoundKey(state, roundKeys[round]);
        invMixColumns(state);
    }
```

```
    invShiftRows(state);
    invSubBytes(state);
    addRoundKey(state, roundKeys[0]);

    return fromStateMatrix(state);
}

public static int[] padMessage(String message) {
    byte[] bytes = message.getBytes();
    int paddedLength = ((bytes.length + 15) / 16) * 16;
    int[] padded = new int[paddedLength];

    for (int i = 0; i < bytes.length; i++) {
        padded[i] = bytes[i] & 0xFF;
    }

    int paddingValue = paddedLength - bytes.length;
    for (int i = bytes.length; i < paddedLength; i++) {
        padded[i] = paddingValue;
    }

    return padded;
}

public static String unpadMessage(int[] padded) {
```

```
    int paddingValue = padded[padded.length - 1];
    int messageLength = padded.length - paddingValue;

    byte[] bytes = new byte[messageLength];
    for (int i = 0; i < messageLength; i++) {
        bytes[i] = (byte) padded[i];
    }

    return new String(bytes);
}

public static int[] encrypt(String plaintext, int[] key) {
    int[] padded = padMessage(plaintext);
    int[][] roundKeys = keyExpansion(key);
    int[] ciphertext = new int[padded.length];

    for (int i = 0; i < padded.length; i += 16) {
        int[] block = new int[16];
        System.arraycopy(padded, i, block, 0, 16);
        int[] encrypted = encryptBlock(block, roundKeys);
        System.arraycopy(encrypted, 0, ciphertext, i, 16);
    }

    return ciphertext;
}
```

```
public static String decrypt(int[] ciphertext, int[] key) {
    int[][] roundKeys = keyExpansion(key);
    int[] plaintext = new int[ciphertext.length];

    for (int i = 0; i < ciphertext.length; i += 16) {
        int[] block = new int[16];
        System.arraycopy(ciphertext, i, block, 0, 16);
        int[] decrypted = decryptBlock(block, roundKeys);
        System.arraycopy(decrypted, 0, plaintext, i, 16);
    }

    return unpadMessage(plaintext);
}

public static String toHex(int[] bytes) {
    StringBuilder sb = new StringBuilder();
    for (int b : bytes) {
        sb.append(String.format("%02x", b));
    }
    return sb.toString();
}

public static int[] fromHex(String hex) {
    int[] bytes = new int[hex.length() / 2];
    for (int i = 0; i < bytes.length; i++) {
        bytes[i] = Integer.parseInt(hex.substring(i * 2, i * 2 + 2), 16);
```

```
    }
    return bytes;
  }

  // ==================== SERVER MODE ====================

  public static void runServer() {
    try {
      ServerSocket serverSocket = new ServerSocket(8888);
      System.out.println("Server started. Waiting for client connection...\n");

      Socket socket = serverSocket.accept();
      System.out.println("Client connected!\n");

      BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
      PrintWriter output = new PrintWriter(socket.getOutputStream(), true);
      Scanner scanner = new Scanner(System.in);

      // Generate 128-bit key
      int[] key = new int[16];
      for (int i = 0; i < 16; i++) {
        key[i] = (int)(Math.random() * 256);
      }

      // Send key to client
```

```
output.println(toHex(key));
System.out.println("Encryption Key: " + toHex(key) + "\n");

// Receive encrypted message
String cipherHex = input.readLine();
System.out.println("Ciphertext received: " + cipherHex);

// Decrypt message
int[] ciphertext = fromHex(cipherHex);
String plaintext = decrypt(ciphertext, key);
System.out.println("Decryption Key: " + toHex(key));
System.out.println("Decrypted Plaintext: " + plaintext + "\n");

// Send response
System.out.print("Enter response message: ");
String response = scanner.nextLine();
int[] encryptedResponse = encrypt(response, key);
String responseHex = toHex(encryptedResponse);

System.out.println("Sending encrypted acknowledgment to the client...");
System.out.println("Ciphertext: " + responseHex + "\n");
output.println(responseHex);

socket.close();
serverSocket.close();
scanner.close();
```

```
            System.out.println("Connection closed.");

        } catch (Exception e) {
            System.out.println("Server error: " + e.getMessage());
            e.printStackTrace();
        }
    }

    // ==================== CLIENT MODE ====================

    public static void runClient() {
        try {
            Socket socket = new Socket("localhost", 8888);
            System.out.println("Connected to server!\n");

            BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter output = new PrintWriter(socket.getOutputStream(), true);
            Scanner scanner = new Scanner(System.in);

            // Receive key
            String keyHex = input.readLine();
            int[] key = fromHex(keyHex);
            System.out.println("Encryption Key: " + keyHex + "\n");

            // Get plaintext from user
```

```
System.out.print("Enter plaintext message: ");
String plaintext = scanner.nextLine();

System.out.println("\nPlaintext message: " + plaintext);

int[] padded = padMessage(plaintext);
System.out.println("Plaintext (padded): " + toHex(padded));

// Encrypt message
int[] ciphertext = encrypt(plaintext, key);
String cipherHex = toHex(ciphertext);

System.out.println("Ciphertext: " + cipherHex);
System.out.println("\nSending ciphertext to the server...");

// Send encrypted message
output.println(cipherHex);

// Wait for response
System.out.println("Waiting for the server's response...\n");
String responseCipherHex = input.readLine();

// Decrypt response
System.out.println("Ciphertext received from server: " + responseCipherHex);
int[] responseCipher = fromHex(responseCipherHex);
String decryptedResponse = decrypt(responseCipher, key);
```

```
            System.out.println("Server response (decrypted): " + decryptedResponse);

            socket.close();
            scanner.close();
            System.out.println("\nConnection closed.");

        } catch (Exception e) {
            System.out.println("Client error: " + e.getMessage());
            e.printStackTrace();
        }
    }

    // ==================== MAIN METHOD ====================

    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Usage:");
            System.out.println("  java AESSecureChat server   - Run as server");
            System.out.println("  java AESSecureChat client   - Run as client");
            return;
        }

        String mode = args[0].toLowerCase();

        if (mode.equals("server")) {
            runServer();
```

```
        } else if (mode.equals("client")) {
            runClient();
        } else {
            System.out.println("Invalid mode. Use 'server' or 'client'.");
        }
    }
}
```

Detailed explanation:

Detailed Explanation of AES Encryption and Decryption Steps

Overview

Advanced Encryption Standard (AES) is a symmetric block cipher that operates on 128-bit blocks of data using a 128-bit key. The algorithm consists of multiple rounds of transformations applied to the data, with each round using a different round key derived from the original key.

---

1. Key Expansion (Rijndael Key Schedule)

The key expansion process generates 11 round keys (each 128 bits) from the original 128-bit encryption key. This is accomplished through the following operations:

Steps:

1. **Initial Round Key: The first round key is the original encryption key itself.**
2. **Generate 10 Additional Round Keys: For rounds 1 through 10, each new round key is generated using:**

a) RotWord: The last 4 bytes of the previous round key are rotated left by one byte.

- **Example: [A, B, C, D] becomes [B, C, D, A]**

b) SubWord: Each byte from RotWord is substituted using the S-box lookup table.

- **The S-box is a predefined 16×16 lookup table that provides non-linear byte substitution**

c) Rcon (Round Constant): The first byte is XORed with a round-specific constant.

- **Round constants: [0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36]**

d) XOR Operation: The result is XORed with the corresponding bytes from the previous round key to generate the new round key.

Purpose:

Key expansion ensures that each round uses a different key, providing diffusion and making the cipher resistant to various cryptographic attacks.

---

2. AES Encryption Process

The encryption process transforms plaintext into ciphertext through an initial round, 9 main rounds, and a final round.

Input Preparation:

1. **Padding: The plaintext is padded to a multiple of 16 bytes using PKCS7 padding.**
2. **State Matrix: The 16-byte block is arranged into a 4×4 matrix (State) in column-major order.**

Initial Round:

AddRoundKey: The State matrix is XORed with the first round key (Round 0).

Main Rounds (Rounds 1-9):

Each of the 9 main rounds applies four transformations in sequence:

a) SubBytes Transformation

- **Purpose: Provides non-linearity and confusion**
- **Operation: Each byte in the State matrix is replaced with a corresponding byte from the S-box lookup table**
- **Example: If State[i][j] = 0x19, it is replaced with SBOX[0x19] = 0xd4**
- **Security: This step protects against linear cryptanalysis**

b) ShiftRows Transformation

- **Purpose: Provides diffusion by mixing bytes across columns**
- **Operation: Cyclically shifts the rows of the State matrix:**
    - **Row 0: No shift**
    - **Row 1: Shift left by 1 byte**
    - **Row 2: Shift left by 2 bytes**
    - **Row 3: Shift left by 3 bytes**
- **Effect: Ensures that bytes from one column affect multiple columns in subsequent rounds**

c) MixColumns Transformation

- **Purpose: Provides diffusion within each column**
- **Operation: Each column of the State matrix is treated as a polynomial and multiplied with a fixed polynomial in the Galois Field GF(2^8)**
- **Matrix Multiplication:**

[2 3 1 1] [s0]

[1 2 3 1] × [s1]

[1 1 2 3] [s2]

[3 1 1 2] [s3]

- **Galois Field Multiplication: Special multiplication where:**
    - **Multiplication by 2: Left shift with XOR 0x1b if overflow occurs**
    - **Multiplication by 3: (multiply by 2) XOR original value**
- **Effect: Each output byte depends on all four input bytes of the column**

d) AddRoundKey Transformation

- **Purpose: Incorporates the round key into the state**
- **Operation: The State matrix is XORed with the current round key**
- **Security: This is the only step that uses the key, making it essential for security**

Final Round (Round 10):

The final round omits the MixColumns transformation:

1. **SubBytes: Apply S-box substitution**
2. **ShiftRows: Shift rows cyclically**
3. **AddRoundKey: XOR with the final round key (Round 10)**

Output:

The final State matrix is converted back from column-major order into a 16-byte ciphertext block.

---

3. AES Decryption Process

Decryption reverses the encryption process using inverse transformations applied in reverse order.

Input:

The 16-byte ciphertext block is arranged into a 4×4 State matrix.

Initial Round:

AddRoundKey: XOR the State with the final round key (Round 10).

- **Note: AddRoundKey is its own inverse since XOR is self-inverting**

Main Rounds (Rounds 9-1, in reverse order):

Each round applies four inverse transformations:

a) InvShiftRows Transformation

- **Purpose: Reverse the row shifting**
- **Operation: Cyclically shifts rows in the opposite direction:**
    - **Row 0: No shift**
    - **Row 1: Shift right by 1 byte**
    - **Row 2: Shift right by 2 bytes**
    - **Row 3: Shift right by 3 bytes**

b) InvSubBytes Transformation

- **Purpose: Reverse the S-box substitution**

- **Operation: Each byte is replaced using the inverse S-box lookup table**
- **Example: If State[i][j] = 0xd4, it is replaced with INV_SBOX[0xd4] = 0x19**

c) AddRoundKey Transformation

- **Purpose: XOR with the current round key**
- **Operation: Same as in encryption (XOR is its own inverse)**

d) InvMixColumns Transformation

- **Purpose: Reverse the column mixing**
- **Operation: Multiply each column with the inverse fixed polynomial in GF(2^8)**
- **Matrix Multiplication:**
  - **[14 11 13 9] [s0]**
  - **[9 14 11 13] × [s1]**
  - **[13 9 14 11] [s2]**
  - **[11 13 9 14] [s3]**
- **Galois Field Multiplication: Uses multipliers 9, 11, 13, and 14**

Final Round (Round 0):

Completes the decryption without InvMixColumns:

1. **InvShiftRows: Shift rows right**
2. **InvSubBytes: Apply inverse S-box**
3. **AddRoundKey: XOR with the original key (Round 0)**

Output:

1. **The State matrix is converted back to a 16-byte plaintext block**
2. **Unpadding: PKCS7 padding is removed to retrieve the original plaintext**

---

4. Mathematical Foundation

Galois Field GF(2^8)

- **AES operations are performed in the finite field GF(2^8)**

- **Irreducible polynomial: x^8 + x^4 + x^3 + x + 1 (0x11b in hexadecimal)**
- **Multiplication requires modular reduction using XOR with 0x1b when overflow occurs**

S-box Construction

- **The S-box provides non-linear transformation**
- **Constructed using multiplicative inverse in GF(2^8) followed by an affine transformation**
- **Provides resistance against differential and linear cryptanalysis**

---

5. Security Properties

Confusion

Achieved through:

- **SubBytes: Non-linear substitution makes the relationship between key and ciphertext complex**
- **Key Expansion: Each round uses a different derived key**

Diffusion

Achieved through:

- **ShiftRows: Spreads bytes across columns**
- **MixColumns: Ensures each byte affects multiple bytes in the next round**
- **Multiple Rounds: 10 rounds ensure thorough mixing**

Avalanche Effect

- **A single bit change in plaintext or key results in approximately 50% of ciphertext bits changing**
- **Achieved through the combination of all transformations over multiple rounds**

---

6. Implementation Details

Padding Scheme (PKCS7)

- **If message length is not a multiple of 16 bytes, padding is added**

- **Padding value equals the number of bytes added**
- **Example: If 5 bytes of padding needed, add [0x05, 0x05, 0x05, 0x05, 0x05]**

State Matrix Layout

- **16 bytes arranged in 4×4 matrix in column-major order:**

  [b0 b4 b8 b12]

  [b1 b5 b9 b13]

  [b2 b6 b10 b14]

  [b3 b7 b11 b15]

Socket Communication

- **Client: Encrypts plaintext and sends ciphertext to server**
- **Server: Receives ciphertext, decrypts it, and can send encrypted response**
- **Key Exchange: Server generates key and sends it to client (in production, use secure key exchange protocols)**

7. Why AES is Secure

1. **Multiple Rounds: 10 rounds provide sufficient complexity**
2. **Non-linearity: S-box prevents linear relationships**
3. **Diffusion: MixColumns and ShiftRows spread influence**
4. **Key Mixing: AddRoundKey integrates the secret key**
5. **No Known Practical Attacks: AES-128 remains secure against all known practical attacks**
6. **Symmetric Nature: Both parties use the same key, ensuring fast encryption/decryption**

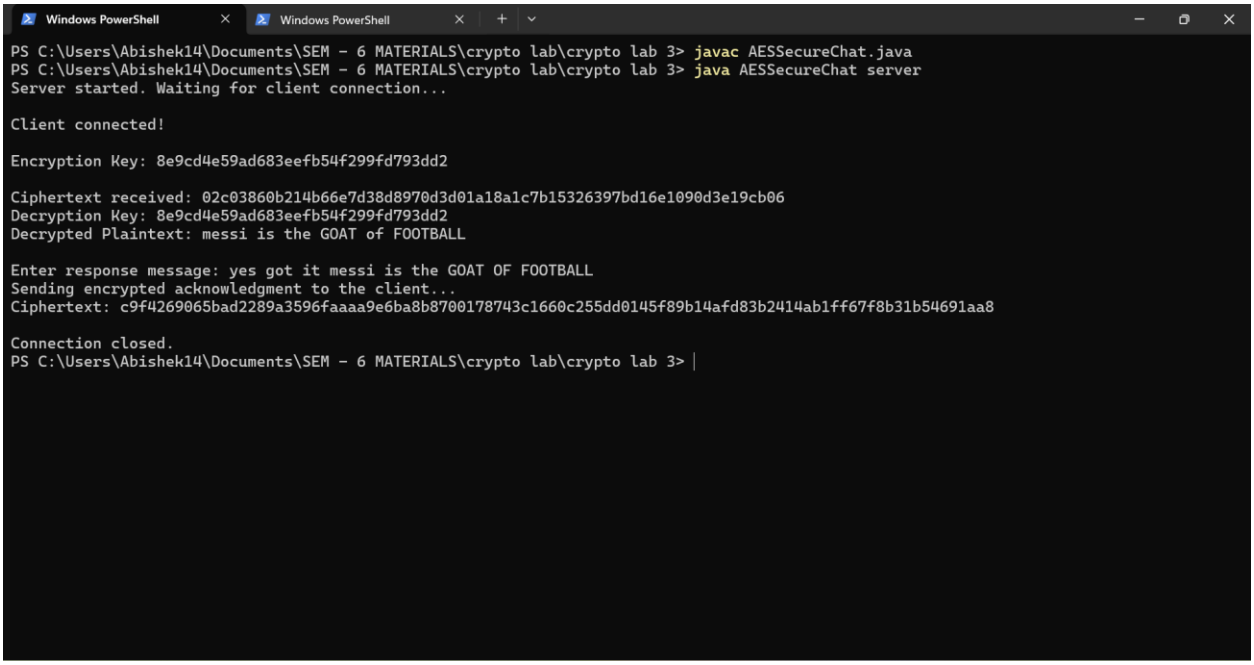
Conclusion

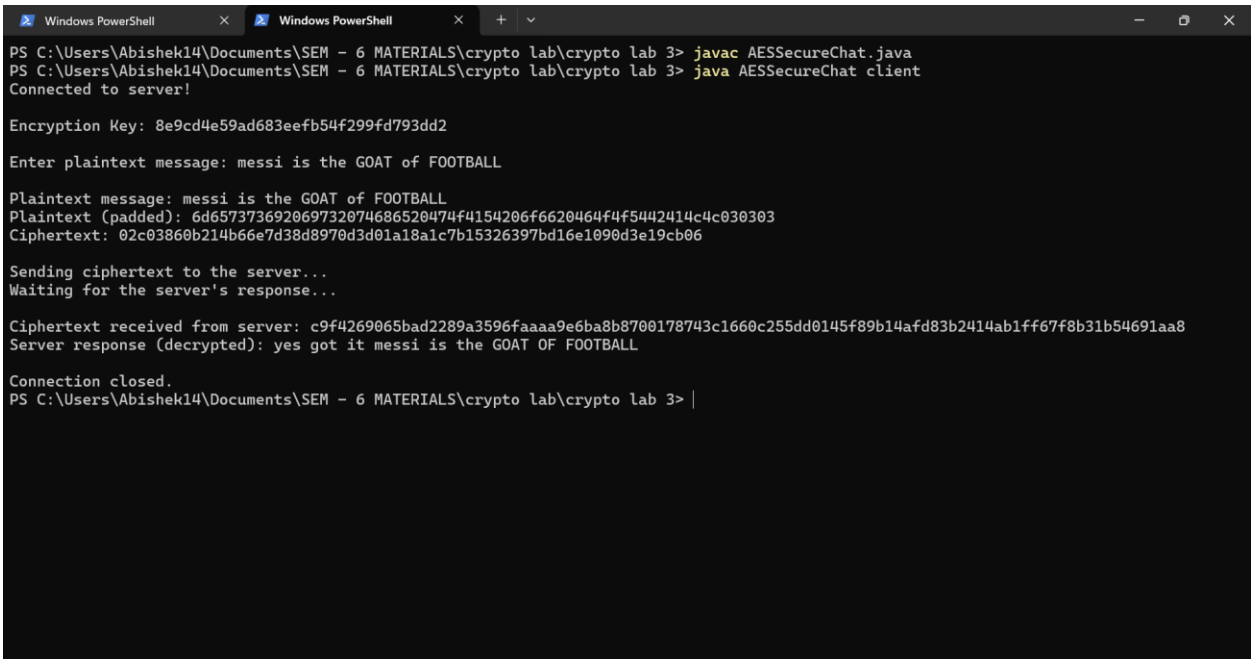
The AES algorithm combines substitution, permutation, mixing, and key addition operations over 10 rounds to provide robust encryption. The encryption process transforms plaintext through multiple layers of confusion and diffusion, while decryption reverses these operations using inverse transformations. The mathematical foundation in Galois Field

arithmetic and the carefully designed S-box ensure that AES remains one of the most secure and widely used encryption standards in modern cryptography.

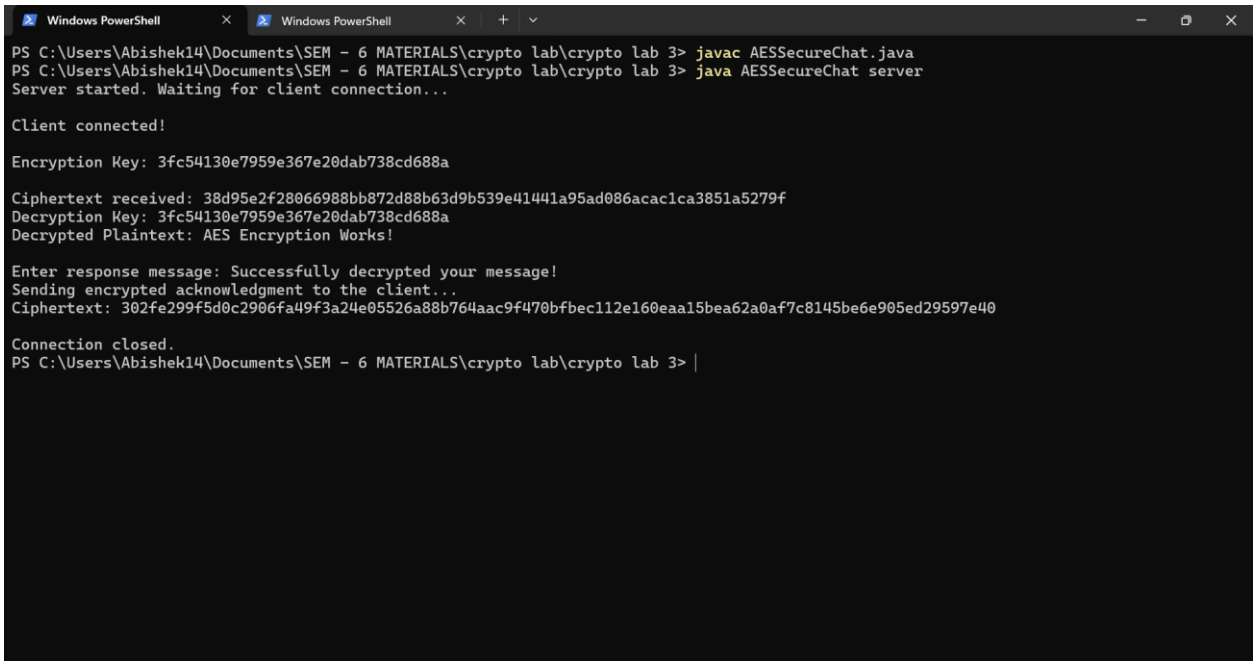
Output Screenshots:

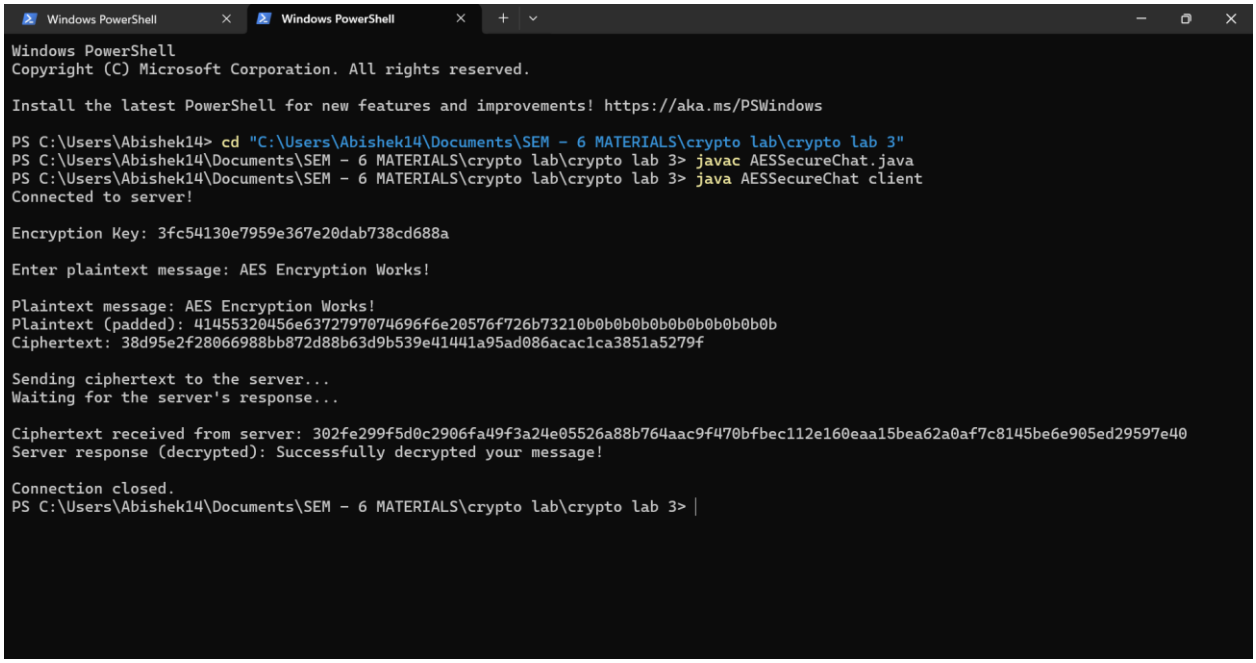
TESTCASE 1:





TESTCASE2:


Windows PowerShell
Windows PowerShell
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 3> javac AESSecureChat.java
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 3> java AESSecureChat server
Server started. Waiting for client connection...
Client connected!
Encryption Key: 3fc54130e7959e367e20dab738cd688a
Ciphertext received: 38d95e2f28066988bb872d88b63d9b539e41441a95ad086acac1ca3851a5279f
Decryption Key: 3fc54130e7959e367e20dab738cd688a
Decrypted Plaintext: AES Encryption Works!
Enter response message: Successfully decrypted your message!
Sending encrypted acknowledgment to the client...
Ciphertext: 302fe299f5d0c2906fa49f3a24e05526a88b764aac9f470bfbec112e160eaa15bea62a0af7c8145be6e905ed29597e40
Connection closed.
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 3>


Windows PowerShell
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS C:\Users\Abishek14> cd "C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 3"
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 3> javac AESSecureChat.java
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 3> java AESSecureChat client
Connected to server!
Encryption Key: 3fc54130e7959e367e20dab738cd688a
Enter plaintext message: AES Encryption Works!
Plaintext message: AES Encryption Works!
Plaintext (padded): 41455320456e6372797074696f6e20576f726b73210b0b0b0b0b0b0b0b0b0b
Ciphertext: 38d95e2f28066988bb872d88b63d9b539e41441a95ad086acac1ca3851a5279f
Sending ciphertext to the server...
Waiting for the server's response...
Ciphertext received from server: 302fe299f5d0c2906fa49f3a24e05526a88b764aac9f470bfbec112e160eaa15bea62a0af7c8145be6e905ed29597e40
Server response (decrypted): Successfully decrypted your message!
Connection closed.
PS C:\Users\Abishek14\Documents\SEM - 6 MATERIALS\crypto lab\crypto lab 3>