NAME: SAKSHI VEER

ROLL NO: 4101066

BATCH: BA2

**Problem Statement:** **To implement a Simplified Advanced Encryption Standard (S-AES) algorithm.**

**CODE:**

**package** ics;

**import** java.util.Arrays;

**import** java.util.Scanner;

**import** java.util.\*;

**import** java.util.regex.\*;

**public** **class** SimplifiedAdvancedEncryptionStandard {

**private** **static** **final** String[][] ***SBOX*** = { {"1001","0100","1010","1011"},{"1101","0001","1000","0101"},{"0110","0010","0000","0011"},{"1100","1110","1111","0111"} };

**private** **static** **final** String[][] ***SBOX\_INV*** = { {"1010","0101","1001","1011"},{"0001","0111","1000","1111"},{"0110","0000","0010","0011"},{"1100","0100","1101","1110"} };

**private** **static** String *key0* = **null**, *key1* = **null**, *key2* = **null**;

**private** **static** **int** *encryptionConstantMatrix*[][] = { {1, 4}, {4, 1} };

**private** **static** **int** *decryptionConstantMatrix*[][] = { {9, 2}, {2, 9} };

**public** SimplifiedAdvancedEncryptionStandard(String key) {

generateKeys(key);

}

**private** **int** binaryToDecimal(String binary) {

**return** Integer.*parseInt*(binary, 2);

}

**private** String decimalToBinary(**int** decimal, **int** binaryStringSize) {

**return** String.*format*("%" + binaryStringSize + "s", Integer.*toBinaryString*(decimal & 0xFF)).replace(' ', '0');

}

**public** String stringXOR(String a, String b) {

StringBuilder sb = **new** StringBuilder();

**for**(**int** i = 0; i < a.length(); i++) {

sb.append(a.charAt(i) ^ b.charAt(i));

}

**return** sb.toString();

}

// Galois field Multiplication

**private** **int** gfMul(**int** a, **int** b) {

**int** product = 0; //the product of the multiplication

**while** (b > 0) {

**if** ((b & 1) != 0) //if b is odd then add the first num i.e a into product result

product = product ^ a;

a = a << 1; //double first num

//if a overflows beyond 4th bit

**if** ((a & (1 << 4)) != 0)

a = a ^ 0b10011; // XOR with irreducible polynomial with high term eliminated

b = b >> 1; //reduce second num

}

**return** product;

}

**private** String nibbleSubstitution(String input, String[][] SBOX) {

StringBuilder sb = **new** StringBuilder();

**for**(**int** i = 0 ; i < input.length() / 4 ; i++) {

String str = input.substring(i\*4, (i\*4)+4);

sb.append(SBOX[binaryToDecimal(str.substring(0,2))][binaryToDecimal(str.substring(2,4))]);

}

**return** sb.toString();

}

**private** String shiftRow(String str) {

// Swap 2nd and 4th nibble

StringBuilder sb = **new** StringBuilder();

sb.append(str.substring(0,4));

sb.append(str.substring(12, 16));

sb.append(str.substring(8,12));

sb.append(str.substring(4,8));

**return** sb.toString();

}

**private** String rotateNibble(String word) {

**return** word.substring(4,8) + word.substring(0,4);

}

**private** **void** generateKeys(String key) {

String w0 = key.substring(0,8);

String w1 = key.substring(8,16);

String w2 = stringXOR(stringXOR(w0, "10000000"), nibbleSubstitution(rotateNibble(w1), ***SBOX***));

String w3 = stringXOR(w2, w1);

String w4 = stringXOR(stringXOR(w2, "00110000"), nibbleSubstitution(rotateNibble(w3), ***SBOX***));

String w5 = stringXOR(w4, w3);

*key0* = w0 + w1;

*key1* = w2 + w3;

*key2* = w4 + w5;

}

**private** String getKeys() {

StringBuilder sb = **new** StringBuilder();

sb.append("Key0: "+*key0* + "\n");

sb.append("Key1: "+*key1* + "\n");

sb.append("Key2: "+*key2* + "\n");

**return** sb.toString();

}

**public** String encrypt(String plainText) {

// Round 0 - Add Key

String roundZeroResult = stringXOR(plainText, *key0*);

// Round 1 - Nibble Substitution -> Shift Row -> Mix Columns -> Add Key

String shiftRowResult = shiftRow(nibbleSubstitution(roundZeroResult, ***SBOX***));

String matrix[][] = **new** String[2][2];

matrix[0][0] = shiftRowResult.substring(0,4);

matrix[0][1] = shiftRowResult.substring(8,12);

matrix[1][0] = shiftRowResult.substring(4,8);

matrix[1][1] = shiftRowResult.substring(12,16);

StringBuilder sb = **new** StringBuilder();

**for**(**int** i = 0 ; i < *encryptionConstantMatrix*.length ; i++) {

**for**(**int** j = 0 ; j < matrix.length ; j++) {

String tempResults[] = **new** String[2];

**for**(**int** k = 0 ; k < 2 ; k++) {

tempResults[k] = decimalToBinary(gfMul(*encryptionConstantMatrix*[i][k],binaryToDecimal(matrix[k][j])), 4);

}

sb.append(stringXOR(tempResults[0], tempResults[1]));

}

}

String res = sb.toString();

String mixColumnsResult = res.substring(0,4) + res.substring(8,12) + res.substring(4,8) + res.substring(12, 16);

String roundOneResult = stringXOR(mixColumnsResult, *key1*);

// Round 2 - Nibble Substitution -> Shift Row -> Add Key

String roundTwoResult = stringXOR(shiftRow(nibbleSubstitution(roundOneResult, ***SBOX***)), *key2*);

**return** roundTwoResult;

}

**public** String decrypt(String cipherText) {

// Round 0 - Add Key

String roundZeroResult = stringXOR(cipherText, *key2*);

// Round 1 - Shift Row -> Nibble Substitution -> Add Key -> Mix Columns

String addKeyResult = stringXOR(nibbleSubstitution(shiftRow(roundZeroResult), ***SBOX\_INV***), *key1*);

String matrix[][] = **new** String[2][2];

matrix[0][0] = addKeyResult.substring(0,4);

matrix[0][1] = addKeyResult.substring(8,12);

matrix[1][0] = addKeyResult.substring(4,8);

matrix[1][1] = addKeyResult.substring(12,16);

StringBuilder sb = **new** StringBuilder();

**for**(**int** i = 0 ; i < *decryptionConstantMatrix*.length ; i++) {

**for**(**int** j = 0 ; j < matrix.length ; j++) {

String tempResults[] = **new** String[2];

**for**(**int** k = 0 ; k < 2 ; k++) {

tempResults[k] = decimalToBinary(gfMul(*decryptionConstantMatrix*[i][k],binaryToDecimal(matrix[k][j])), 4);

}

sb.append(stringXOR(tempResults[0], tempResults[1]));

}

}

String res = sb.toString();

String mixColumnsResult = res.substring(0,4) + res.substring(8,12) + res.substring(4,8) + res.substring(12, 16);

// Round 2 - Shift Row -> Nibble Substitution -> Add Key

String roundTwoResult = stringXOR(nibbleSubstitution(shiftRow(mixColumnsResult), ***SBOX\_INV***), *key0*);

**return** roundTwoResult;

}

**public** **static** **void** main(String[] args) {

String key = **null**, msg = **null**;

Scanner sc = **new** Scanner(System.***in***);

System.***out***.print("Enter 16-bit key: ");

key = sc.next();

System.***out***.print("Enter 16-bit binary form message for encryption: ");

msg = sc.next();

SimplifiedAdvancedEncryptionStandard simplifiedAdvancedEncryptionStandard = **new** SimplifiedAdvancedEncryptionStandard(key);

System.***out***.println(simplifiedAdvancedEncryptionStandard.getKeys());

System.***out***.println("\n\*\*\*\*\* ENCRYPTION \*\*\*\*\*");

String encryptedMsg = simplifiedAdvancedEncryptionStandard.encrypt(msg);

System.***out***.println("Encrypted Message: "+encryptedMsg);

System.***out***.println("\n\*\*\*\*\* DECRYPTION \*\*\*\*\*");

String decryptedMsg = simplifiedAdvancedEncryptionStandard.decrypt(encryptedMsg);

System.***out***.println("Decrypted Message: "+decryptedMsg);

}

}

**OUTPUT:**

Enter 16-bit key: 0100101011110101

Enter 16-bit binary form message for encryption: 1101011100101000

Key0: 0100101011110101

Key1: 1101110100101000

Key2: 1000011110101111

\*\*\*\*\* ENCRYPTION \*\*\*\*\*

Encrypted Message: 0010010011101100

\*\*\*\*\* DECRYPTION \*\*\*\*\*

Decrypted Message: 1101011100101000