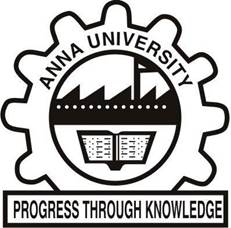
**MADRAS INSTITUTE OF TECHNOLOGY**

**ANNA UNIVERSITY, CHENNAI**

**CHENNAI – 600 044.**

****

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**IT5711 MOBILE AND SECURITY LABORATORY**

**RECORD NOTEBOOK**

**7/8 INFORMATION TECHNOLOGY**

**Name       : Rahul Prasanth D**

**Reg.No.   : 2020506070**

**Semester : 7/8 IT**

**Year        : 4**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exp. No** | **Date** | **Experiment Name** | **Page No** | **Signature** |
| 1 | 31-07-2023 | Classic Cipher Algorithms  a.     Shift Cipher  b.    Affine Cipher  c.     Hill Cipher  d.    Transposition Cipher | 3  5  8  13 |  |
| 2 | 22-08-2023 | Cryptanalysis  a.     Shift Cipher  b.    Affine Cipher  c.     Hill Cipher | 16  18  21 |  |
| 3 | 01-09-2023 | Simplified Data Encryption Standard | 26 |  |
| 4 | 06-10-2023 | Simplified Advanced Encryption Standard | 32 |  |
| 5 | 06-10-2023 | RSA(Rivest-Shamir-Adleman) encryption | 41 |  |
| 6 | 13-10-2023 | Digital Signature Standard | 46 |  |
| 7 | 13-10-2023 | Secure Hash Algorithm | 51 |  |

**Exp No**:1       **CLASSIC CIPHER ALGORITHMS**

**Date**:31-07-2023

**Aim**

To write a program to implement classic cipher algorithms and encrypt the given plaintext and decrypt the same.

**1a. Caesar Cipher**

**Algorithm**

**Encryption:**

1. Create a new empty string to store the encrypted text. This is where we will store the encrypted characters as we loop through the plaintext.
2. Loop through each character of the plaintext. We will use a for loop to iterate over each character of the plaintext.
3. Shift the character by the given shift amount and add it to the encrypted text. If the character is uppercase, we will shift it by the given shift amount and add it to the encrypted text. If the character is lowercase, we will also shift it by the given shift amount and add it to the encrypted text.
4. Return the encrypted text. Once we have looped through all of the characters in the plaintext, we will return the encrypted text.

**Decryption:**

1. Create a new empty string to store the decrypted text. This is where we will store the decrypted characters as we loop through the encrypted text.
2. Loop through each character of the encrypted text. We will use a for loop to iterate over each character of the encrypted text.
3. Unshift the character by the given shift amount and add it to the decrypted text. If the character is uppercase, we will unshift it by the given shift amount and add it to the decrypted text. If the character is lowercase, we will also unshift it by the given shift amount and add it to the decrypted text.
4. Return the decrypted text. Once we have looped through all of the characters in the encrypted text, we will return the decrypted text

**Source Code:**

import java.io.\*;

import java.util.\*;

public class shiftCipher {

public static String encrypt(String text, int s) {

StringBuilder encryptedText = new StringBuilder();

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

char ch = text.charAt(i);

if (Character.isLetter(ch))

{

if (Character.isUpperCase(ch)) {

ch = (char) (((int) ch + s - 65) % 26 + 65);

} else {

ch = (char) (((int) ch + s - 97) % 26 + 97);

}

encryptedText.append(ch);

}

else{

encryptedText.append(ch);

}

}

return encryptedText.toString();

}

public static String decrypt(String text, int s) {

StringBuilder plaintext = new StringBuilder();

for (char character : text.toCharArray()) {

if (Character.isLetter(character)) {

boolean isUpperCase = Character.isUpperCase(character);

char lowerCaseChar = Character.toLowerCase(character);

int charCode = (lowerCaseChar - 'a' - s + 26) % 26 + 'a';

if (isUpperCase) {

plaintext.append((char) (charCode - 32));

} else {

plaintext.append((char) charCode);

}

} else {

plaintext.append(character);

}

}

return plaintext.toString();

}

public static void main(String[] args)

{

String text = "HI I AM RAHUL";

int s = 7;

System.out.println("\n\nText : " + text);

System.out.println("Shift : " + s);

String cip=encrypt(text, s);

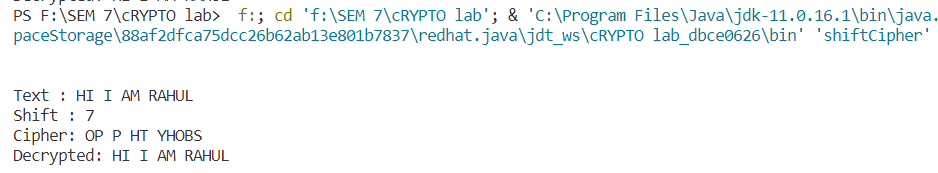
System.out.println("Cipher: " + cip);

System.out.println("Decrypted: " + decrypt(cip, s)+"\n\n");

}

}

**Output**



**1b. Affine Cipher**

**Algorithm**

**Encryption:**

1. Choose two integers, 'a' and 'b', where 'a' must be coprime with the length of the alphabet (usually 26 for the English alphabet). This means that 'a' and 26 have no common factors other than 1. 'b' is the shift value.
2. Convert the plaintext message to uppercase and remove any spaces or punctuation.
3. For each letter 'P' in the plaintext:

a. Compute the numerical value of 'P' using a standard mapping, where 'A' is 0, 'B' is 1, and so on.

b. Apply the encryption formula to get the ciphertext letter 'C': C = (a \* P + b) % 26

c. Convert 'C' back to a letter using the standard mapping.

1. Repeat step 3 for each letter in the plaintext, and you'll get the ciphertext message.

**Decryption:**

1. Given the ciphertext and the values of 'a' and 'b,' which were used for encryption, compute the modular multiplicative inverse of 'a.' Let's call this value 'a\_inv.' 'a\_inv' is the integer such that (a \* a\_inv) % 26 = 1. Note: 'a\_inv' exists if and only if 'a' is coprime with 26.
2. Convert the ciphertext message to uppercase and remove any spaces or punctuation.
3. For each letter 'C' in the ciphertext:

a. Compute the numerical value of 'C' using the standard mapping.

b. Apply the decryption formula to get the plaintext letter 'P': P = (a\_inv \* (C - b)) % 26

c. Convert 'P' back to a letter using the standard mapping.

4.Repeat step 3 for each letter in the ciphertext, and you'll get the plaintext message.

**Source Code**

import java.io.\*;

import java.util.\*;

class AffineCipher

{

    public static String Encrypt(String plain,int a,int b)

    {

        String cipher="";

        //ax+b

        for(int i=0;i<plain.length();++i)

        {

            char c=plain.charAt(i);

            cipher+=(char)(((a\*(c) +b)%26) + 'A');

        }

        return cipher;

    }

    public static int Inverse(int a)

    {

        //USING EXTENDED EUCLEDIAN

        int q;

        int r1=26,r2=a;

        int r,t;

        int t1=0,t2=1;

        while(r2>0)

        {

            q=r1/r2;

            r=r1%r2;

            t=t1-q\*t2;

            r1=r2;

            r2=r;

            t1=t2;

            t2=t;

        }

        return (t1<0)?26+t1 : t1;

    }

    public static String Decrypt(String cipher,int a,int b)

    {

        String ans="";

        //a^-1 (x-b)

        int inv=Inverse(a);

        System.out.println("Inverse = "+inv);

        for(int i=0;i<cipher.length();++i)

        {

            ans = ans + (char) (((inv \* ((cipher.charAt(i) - b)) % 26)) + 65);

        }

        return ans;

    }

    public static void main(String[] args) {

        Scanner scn=new Scanner(System.in);

        System.out.print("\n\nEnter the plain text = ");

        String plain=scn.nextLine();

        int a,b;

        System.out.println("Enter the values of a and b = ");

        a=scn.nextInt();

        b=scn.nextInt();

        //where a and b are the keys

        String cipher=Encrypt(plain,a,b);

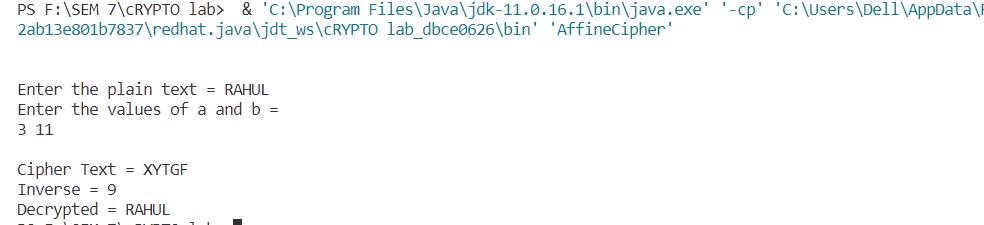
        System.out.println("\nCipher Text = "+cipher);

        String dec=Decrypt(cipher,a,b);

        System.out.println("Decrypted = "+dec);

    }}

**Output**



**1c. Hill Cipher**

**Algorithm**

**Encryption:**

1. Choose a key matrix 'K' of size n x n, where n is the size of the group of letters you want to encrypt (e.g., 2x2 or 3x3 matrices are common).
2. Convert the plaintext message to uppercase and remove any spaces or punctuation.
3. Break the plaintext message into blocks of n letters each, where n is the size of the key matrix. If the last block is shorter than n, pad it with additional letters.
4. Convert each block of plaintext letters into a column vector, where each letter corresponds to its numerical value (A=0, B=1, ..., Z=25).
5. Multiply each column vector of plaintext letters by the key matrix 'K':
   1. C = K \* P (mod 26)
6. 'C' is the resulting column vector of ciphertext letters, and 'P' is the column vector of plaintext letters. The multiplication is done modulo 26 to ensure that the result stays within the range of the alphabet.
7. Convert the resulting column vector 'C' back to a group of letters, where each numerical value is converted to its corresponding letter (0=A, 1=B, ..., 25=Z).
8. Repeat steps 3-6 for each block of the plaintext, and you'll get the ciphertext message

**Decryption:**

1. To decrypt a Hill Cipher, you need the key matrix 'K' and its inverse 'K\_inv.' Calculate the modular multiplicative inverse of the determinant of 'K' modulo 26. If it does not exist (e.g., if the determinant is 0 or not coprime with 26), you cannot decrypt the message.
2. Compute the adjugate (adjoint) matrix 'K\_adj' of the key matrix 'K.' The adjugate matrix is obtained by taking the cofactors of each element in 'K' and then transposing the resulting matrix.
3. Calculate the inverse matrix 'K\_inv' by multiplying 'K\_adj' by the modular multiplicative inverse of the determinant of 'K':
   1. K\_inv = K\_adj \* (determinant of K)^(-1) (mod 26)
4. Convert the ciphertext message to uppercase and remove any spaces or punctuation.
5. Break the ciphertext message into blocks of n letters each, where n is the size of the key matrix 'K.'
6. Convert each block of ciphertext letters into a column vector, where each letter corresponds to its numerical value (A=0, B=1, ..., Z=25).
7. Multiply each column vector of ciphertext letters by the inverse key matrix 'K\_inv':
   1. P = K\_inv \* C (mod 26)
   2. 'P' is the resulting column vector of plaintext letters, and 'C' is the column vector of ciphertext letters.
8. Convert the resulting column vector 'P' back to a group of letters, where each numerical value is converted to its corresponding letter (0=A, 1=B, ..., 25=Z).
9. Repeat steps 5-8 for each block of the ciphertext, and you'll get the plaintext message

**Source Code:**

import java.util.ArrayList;

import java.util.Scanner;

public class HillCipher {

    private static int[][] getKeyMatrix() {

        Scanner scn = new Scanner(System.in);

        System.out.println("Enter the key matrix:");

        String key = scn.nextLine().toUpperCase();

        int matrixSize = (int) Math.sqrt(key.length());

        int[][] keyMatrix = new int[matrixSize][matrixSize];

        int k = 0;

        for (int i = 0; i < matrixSize; i++) {

            for (int j = 0; j < matrixSize; j++) {

                keyMatrix[i][j] = key.charAt(k) - 'A';

                k++;

            }

        }

        return keyMatrix;

    }

    private static void validateKeyMatrix(int[][] keyMatrix) {

        int determinant = keyMatrix[0][0] \* keyMatrix[1][1] - keyMatrix[0][1] \* keyMatrix[1][0];

        if (determinant == 0) {

            throw new RuntimeException("Invalid key matrix (Determinant is zero).");

        }

    }

    private static int[][] calculateInverseKeyMatrix(int[][] keyMatrix) {

        int determinantMod26 = (keyMatrix[0][0] \* keyMatrix[1][1] - keyMatrix[0][1] \* keyMatrix[1][0]) % 26;

        int factor;

        int[][] inverseKeyMatrix = new int[2][2];

        for (factor = 1; factor < 26; factor++) {

            if ((determinantMod26 \* factor) % 26 == 1) {

                break;

            }

        }

        inverseKeyMatrix[0][0] = keyMatrix[1][1] \* factor % 26;

        inverseKeyMatrix[0][1] = (26 - keyMatrix[0][1]) \* factor % 26;

        inverseKeyMatrix[1][0] = (26 - keyMatrix[1][0]) \* factor % 26;

        inverseKeyMatrix[1][1] = keyMatrix[0][0] \* factor % 26;

        return inverseKeyMatrix;

    }

    private static void displayResult(String label, int adder, ArrayList<Integer> phrase) {

        System.out.print(label);

        for (int i = 0; i < phrase.size(); i += 2) {

            System.out.print((char) (phrase.get(i) + (65 + adder)));

            System.out.print((char) (phrase.get(i + 1) + (65 + adder)));

            if (i + 2 < phrase.size()) {

                System.out.print("-");

            }

        }

        System.out.println();

    }

    public static void encrypt(String text, boolean useZeroBasedAlphabet) {

        int adder = useZeroBasedAlphabet ? 0 : 1;

        int[][] keyMatrix;

        ArrayList<Integer> textToNumbers = new ArrayList<>();

        ArrayList<Integer> encryptedText = new ArrayList();

        text = text.replaceAll("[^a-zA-Z]", "").toUpperCase();

        if (text.length() % 2 == 1) {

            text += "Q";

        }

        keyMatrix = getKeyMatrix();

        validateKeyMatrix(keyMatrix);

        for (char c : text.toCharArray()) {

            textToNumbers.add(c - (65 + adder));

        }

        for (int i = 0; i < textToNumbers.size(); i += 2) {

            int x = (keyMatrix[0][0] \* textToNumbers.get(i) + keyMatrix[0][1] \* textToNumbers.get(i + 1)) % 26;

            int y = (keyMatrix[1][0] \* textToNumbers.get(i) + keyMatrix[1][1] \* textToNumbers.get(i + 1)) % 26;

            encryptedText.add(useZeroBasedAlphabet ? x : (x == 0 ? 26 : x));

            encryptedText.add(useZeroBasedAlphabet ? y : (y == 0 ? 26 : y));

        }

        displayResult("Encrypted text: ", adder, encryptedText);

    }

    public static void decrypt(String cipher, boolean useZeroBasedAlphabet) {

        int adder = useZeroBasedAlphabet ? 0 : 1;

        int[][] keyMatrix, inverseKeyMatrix;

        ArrayList<Integer> cipherToNumbers = new ArrayList<>();

        ArrayList<Integer> decryptedText = new ArrayList<>();

        cipher = cipher.replaceAll("[^a-zA-Z]", "").toUpperCase();

        keyMatrix = getKeyMatrix();

        validateKeyMatrix(keyMatrix);

        for (char c : cipher.toCharArray()) {

            cipherToNumbers.add(c - (65 + adder));

        }

        inverseKeyMatrix = calculateInverseKeyMatrix(keyMatrix);

        for (int i = 0; i < cipherToNumbers.size(); i += 2) {

            int decryptedX = (inverseKeyMatrix[0][0] \* cipherToNumbers.get(i) + inverseKeyMatrix[0][1] \* cipherToNumbers.get(i + 1)) % 26;

            int decryptedY = (inverseKeyMatrix[1][0] \* cipherToNumbers.get(i) + inverseKeyMatrix[1][1] \* cipherToNumbers.get(i + 1)) % 26;

            decryptedText.add(decryptedX);

            decryptedText.add(decryptedY);

        }

        displayResult("Decrypted text: ", adder, decryptedText);

    }

    public static void main(String[] args) {

        Scanner scn = new Scanner(System.in);

        System.out.println("1. Encryption");

        System.out.println("2. Decryption");

        System.out.print("ch : ");

        String choice = scn.nextLine();

        switch (choice) {

            case "1":

                System.out.print("Text: ");

                String textToEncrypt = scn.nextLine();

                encrypt(textToEncrypt, true);

                break;

            case "2":

                System.out.print("Cipher: ");

                String cipherToDecrypt = scn.nextLine();

                decrypt(cipherToDecrypt, true);

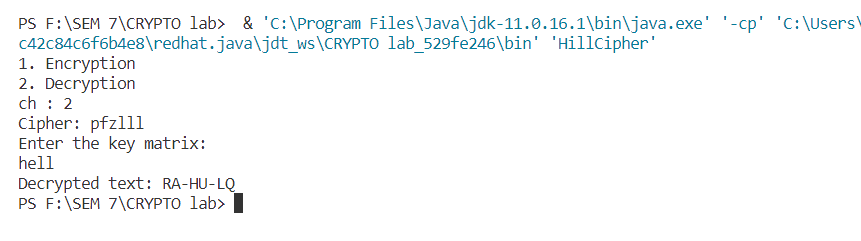
                break;

        }

    }

}

**Output**



**1d. Transposition Cipher**

**Algorithm**

**Encryption:**

1. Choose a keyword or phrase. This keyword will determine the order in which the columns of the plaintext will be written.
2. Write down the keyword in alphabetical order, removing any duplicate letters. This reordered keyword is used for column ordering.
3. Write the plaintext message horizontally in a grid, row by row. The number of columns in the grid is equal to the length of the keyword.
4. Read the grid vertically, column by column, according to the alphabetical order of the letters in the keyword. This is your ciphertext.

**Decryption:**

1. Start by choosing the same keyword that was used for encryption.
2. Write down the keyword in alphabetical order, removing any duplicate letters. This reordered keyword will be used for column ordering during decryption.
3. Write the ciphertext message horizontally in a grid, row by row. The number of columns in the grid is equal to the length of the keyword.
4. Read the grid vertically, column by column, according to the alphabetical order of the letters in the keyword. This is your plaintext.

**Source Code**

import java.util.\*;

class Transposition {

    public static String Encrypt(String str)

    {

        char[] arr=str.toCharArray();

        String odd = "",eve="";

        for(int i=0;i<str.length();++i)

        {

            if(i%2!=0) odd+=arr[i];

            else eve+=arr[i];

        }

        return eve+odd;

    }

    public static String Decrypt(String cipher)

    {

        String ans="";

        //Hello -> Hloel

        //abcd->acbd

        int first=0,second=cipher.length()/2;

        if(cipher.length()%2!=0)

        {

            second=cipher.length()/2 +1;

        }

        int l=first,r=second;

        while(l<second && r<cipher.length())

        {

            ans=ans+cipher.charAt(l)+cipher.charAt(r);

            l++;r++;

        }

        if(l<second) ans=ans+cipher.charAt(l);

        return ans;

    }

    public static void main(String[] args)

    {

        String inp;

        Scanner scn=new Scanner(System.in);

        System.out.print("\n\nEnter the string = ");

        inp=scn.next();

        String enc = Encrypt(inp);

        String dec=Decrypt(enc);

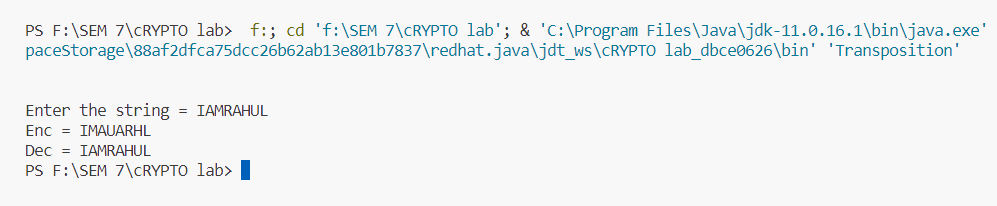
        System.out.println("Enc = "+enc);

        System.out.println("Dec = "+dec);

    }

}

**Output**

****

**Result**

Thus, the program for the implementation of Classic Cipher Algorithms is completed and output is displayed and verified.

**Exp No**:2               **Cryptanalysis**

**Date**: 22-08-2023

**Aim**

To implement the crytanalysis attack for shift cipher, affine cipher and hill cipher.

**2a) Shift cipher:**

**Algorithm:**

1. Initialize the ciphertext and plaintext with the known values.
2. Create a function findCaesarCipherKey that takes the ciphertext and plaintext as inputs and returns the Caesar cipher key.
3. Initialize a variable shift to 0.
4. Loop while shift is less than 26 (since there are 26 possible shifts in the Caesar cipher):
   1. Inside the loop, call the decryptCaesarCipher function with the current ciphertext and shift as arguments to attempt decryption.
   2. Compare the decrypted text with the known plaintext.
   3. If the decrypted text matches the known plaintext, return the current shift as the key.
   4. If there is no match, increment the shift by 1 and repeat the loop.
5. If the loop completes without finding a match (all 26 shifts have been tried), return -1 to indicate that the key was not found.
6. The main function calls findCaesarCipherKey, and if the returned key is not -1, it prints the key. Otherwise, it prints "Key not found."
7. End of the algorithm.

**Code:**

import java.io.\*;

import java.util.\*;

public class shiftCipher {

    public static String encrypt(String text, int s) {

        StringBuilder encryptedText = new StringBuilder();

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

            char ch = text.charAt(i);

            if (Character.isLetter(ch))

            {

                if (Character.isUpperCase(ch)) {

                ch = (char) (((int) ch + s - 65) % 26 + 65);

                } else {

                    ch = (char) (((int) ch + s - 97) % 26 + 97);

                }

                encryptedText.append(ch);

            }

            else{

                encryptedText.append(ch);

            }

        }

        return encryptedText.toString();

    }

public static String Decryption(String s,int shift){

        String ans="";

        for(char c:s.toCharArray()){

            if(Character.isUpperCase(c)){

                int t1=(c-'A'-shift)%26;

                char t2=(char)(t1+'A');

                ans+=t2;

            }

            else if(Character.isLowerCase(c)){

                int t1=(c-'a'-shift)%26;

                char t2=(char)(t1+'a');

                ans+=t2;

            }

            else return("");

        }

        return ans;

    }

    public static void main(String[] args)

    {

        Scanner inp=new Scanner(System.in);

        System.out.print("Enter the plain text = ");

        String plain=inp.nextLine();

        System.out.print("Enter the cipher text = ");

        String cip=inp.nextLine();

        for(int i=0;i<26;i++){

            String ans=Decryption(cip,i);

            if(ans.equals(plain))

            {

                System.out.println("Plain text = "+plain);

                System.out.println("Cipher text = "+cip);

                System.out.println("Key = "+i);

                break;

            }

        }

    }

}

**Output:**



**2b) Affine cipher:**

**Algorithm:**

1. Initialize a scanner to read user input.
2. Prompt the user to enter the ciphertext.
3. Start a loop that iterates through all possible values of a from 1 to 25 (inclusive).
4. Inside the a loop:
   1. Check if a is coprime with 26 (i.e., gcd(a, 26) equals 1). If not, skip this iteration of the loop.
   2. Start another loop that iterates through all possible values of b from 0 to 25
   3. Inside the b loop:
      1. Decrypt the ciphertext using the current a and b values.
      2. If the decryption result appears to be a meaningful plaintext (e.g., resembles English text):
      3. Print the values of a and b along with the decrypted plaintext.
5. Continue looping through all combinations of a and b, systematically checking for valid decryptions.
6. After checking all combinations, close the scanner.
7. End of the algorithm.

**Code:**

import java.util.Scanner;

public class AffineCrypt {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the ciphertext: ");

        String ciphertext = scanner.nextLine();

        System.out.print("Enter the plaintext: ");

        String text = scanner.nextLine();

        for (int a = 1; a < 26; a++) {

            if (gcd(a, 26) == 1) {

                for (int b = 0; b < 26; b++) {

                    String plaintext = decryptAffineCipher(ciphertext, a, b,text);

                    // System.out.println("a = " + a + ", b = " + b + ": " + plaintext);

                    if(plaintext.equals(text)){

                        System.out.println("Key = "+a+" "+b);

                        return;

                    }

                }

            }

        }

        scanner.close();

    }

    public static String decryptAffineCipher(String ciphertext, int a, int b,String text) {

        StringBuilder plaintext = new StringBuilder();

        for (char character : ciphertext.toCharArray()) {

            if (Character.isLetter(character)) {

                boolean isUpperCase = Character.isUpperCase(character);

                char lowerCaseChar = Character.toLowerCase(character);

                int charCode = ((modInverse(a) \* (lowerCaseChar - 'a' - b)) % 26 + 26) % 26;

                // charCode += 'a';

                if (isUpperCase) {

                    plaintext.append((char) (charCode+'A'));

                } else {

                    plaintext.append((char) charCode+'a');

                }

            } else {

                plaintext.append(character);

            }

        }

        return plaintext.toString();

    }

    public static int gcd(int a, int b) {

        if (b == 0) {

            return a;

        }

        return gcd(b, a % b);

    }

    public static int modInverse(int r2) {

        if(gcd(26,r2)==1){

            int r1=26;

            int t1=0;

            int t2=1

                    ;

            int q,r,t;

            while(r2!=0){

                q=r1/r2;

                r=r1%r2;

                t=t1-(q\*t2);

                r1=r2;

                r2=r;

                t1=t2;

                t2=t;

            }

            if(t1<0){

                return t1+26;

            }

            else{

                return t1;

            }

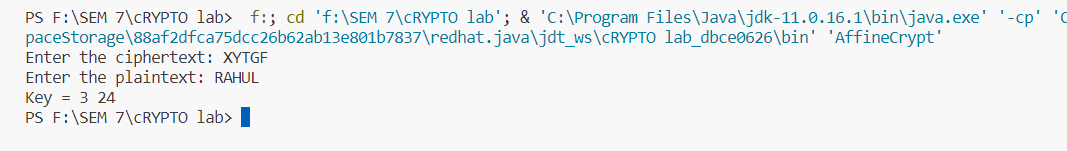
        }

        return -1;

    }

}

**Output:**



**2c) Hill cipher:**

**Algorithm:**

**Inverse function:**

1. Accept a matrix `matrix` as input.
2. Create an augmented matrix `augmentedMatrix` that is twice as wide as the input matrix. The left half contains the input matrix, and the right half is an identity matrix.
3. Use Gaussian elimination to transform the left side of the augmented matrix into the identity matrix while applying the same row operations to the right side.
4. Once the left side becomes the identity matrix, the right side contains the inverse matrix.
5. Return the inverse matrix.

**Main method:**

1. Define a plaintext matrix (`text`) and a ciphertext matrix (`cipher`).
2. Calculate the modular inverse of the `text` matrix using the `inverse` function.
3. Perform matrix multiplication of the `cipher` matrix with its modular inverse using the `multiplyMatricesMod26` function. This step is typically done during decryption in a Hill cipher.
4. Print the modular inverse and the result of the matrix multiplication, which should yield the original plaintext matrix.

**Code:**

import java.io.\*;

import java.util.\*;

public class HillCrypt

{

    public static int[][] multiply(int[][] matrixA, int[][] matrixB) {

        int rowsA = matrixA.length;

        int colsA = matrixA[0].length;

        int colsB = matrixB[0].length;

        int[][] result = new int[rowsA][colsB];

        for (int i = 0; i < rowsA; i++) {

            for (int j = 0; j < colsB; j++) {

                int sum = 0;

                for (int k = 0; k < colsA; k++) {

                    sum += matrixA[i][k] \* matrixB[k][j];

                }

                result[i][j] = sum % 26; // Take the result modulo 26

            }

        }

        return result;

    }

    private static int modInverse(int a, int m) {

        for (int x = 1; x < m; x++) {

            if ((a \* x) % m == 1) {

                return x;

            }

        }

        throw new ArithmeticException("Modular inverse does not exist.");

    }

    public static int[][] inverse(int[][] matrix){

        int n = matrix.length;

        int[][] augmentedMatrix = new int[n][2 \* n];

        // Create an augmented matrix [matrix | I] where I is the identity matrix

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                augmentedMatrix[i][j] = matrix[i][j];

                augmentedMatrix[i][j + n] = (i == j) ? 1 : 0;

            }

        }

        // Apply Gaussian elimination to transform the left side into the identity matrix

        for (int i = 0; i < n; i++) {

            int pivotIdx = -1;

            for (int j = i; j < n; j++) {

                if (augmentedMatrix[j][i] != 0) {

                    pivotIdx = j;

                    break;

                }

            }

            if (pivotIdx == -1) {

                throw new ArithmeticException("Matrix is not invertible modulo 26.");

            }

            // Swap rows to make the pivot element non-zero

            int[] temp = augmentedMatrix[i];

            augmentedMatrix[i] = augmentedMatrix[pivotIdx];

            augmentedMatrix[pivotIdx] = temp;

            int pivot = augmentedMatrix[i][i];

            int pivotInverse = modInverse(pivot, 26);

            // Scale the row to make the pivot element 1

            for (int j = 0; j < 2 \* n; j++) {

                augmentedMatrix[i][j] = (augmentedMatrix[i][j] \* pivotInverse) % 26;

            }

            // Eliminate other rows

            for (int j = 0; j < n; j++) {

                if (j != i) {

                    int factor = augmentedMatrix[j][i];

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

                        augmentedMatrix[j][k] = (augmentedMatrix[j][k] - (factor \* augmentedMatrix[i][k])) % 26;

                        if (augmentedMatrix[j][k] < 0) {

                            augmentedMatrix[j][k] += 26;

                        }

                    }

                }

            }

        }

        // Extract the right side of the augmented matrix as the inverse

        int[][] inverseMatrix = new int[n][n];

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                inverseMatrix[i][j] = augmentedMatrix[i][j + n];

            }

        }

        return inverseMatrix;

    }

    public static void main(String[] args) {

        System.out.println("\nInput is : HELL");

        int [][] text = new int[][] {{7,11},{4,11}};

        int [][] cipher = new int[][] {{3,9},{17,8}};

        System.out.println("INVERSE:");

        int[][] inverse = inverse(text);

        for (int[] row : inverse) {

            for (int value : row) {

                System.out.print(value + " ");

            }

            System.out.println();

        }

         System.out.println("\nKEY MATRIX:");

        int [][]mulMat = multiply (cipher, inverse);

        for (int[] row : mulMat) {

            for (int value : row) {

                System.out.print(value + " ");

            }

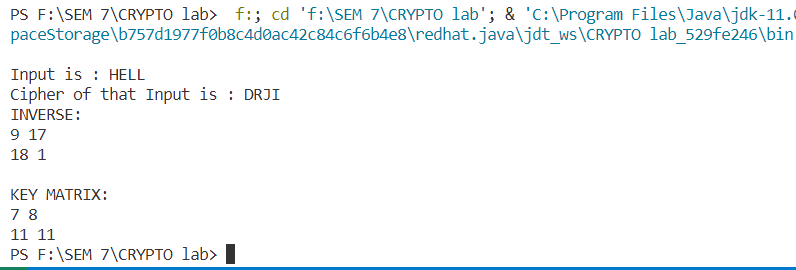
            System.out.println();

        }

    }

}

**Output:**

****

**Result**

Thus, the program for the implementation of cryptanalysis is completed and output is displayed and verified.

**Exp No**:3               **SIMPLIFIED DATA ENCRYPTION STANDARD**

**Date**:01-09-2023

**Aim**

To implement the Simplified DES using java and perform the encryption and decryption.

**Algorithm:**

**Key Generation:**

1. Start with a 10-bit key.
2. Permute the key.
3. Split the key into two 5-bit subkeys (K1 and K2).
4. Perform a left shift on both subkeys.
5. Combine the shifted subkeys to create K1+K2.

**Encryption:**

1. Start with an 8-bit plaintext.
2. Permute the plaintext.
3. Split the plaintext into two 4-bit halves (L0 and R0).
4. Perform Feistel Network rounds using subkeys (K1 and K2).
5. Swap values and perform a final permutation.

**Decryption:**

1. Start with an 8-bit ciphertext.
2. Permute the ciphertext.
3. Split the ciphertext into two 4-bit halves (L0 and R0).
4. Perform Feistel Network rounds using subkeys (K2 and K1).
5. Perform a final permutation to get the plaintext.

**Code:**

import java.util.Scanner;

public class SDES {

    public static String circularLeftShift(String input, int shiftValue) {

        String result = "";

        for (int shift = 0; shift < shiftValue; shift++) {

            result = "";

            for (int i = 0; i < input.length() - 1; i++) {

                result = result + input.charAt(i + 1);

            }

            result = result + input.charAt(0);

            input = result;

        }

        return result;

    }

    public static String[] generateSubkeys(String plaintext, String key) {

        int[] compressionTable = {6, 3, 7, 4, 8, 5, 10, 9};

        int[] SP = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};

        String[] subkeys = new String[2];

        String keyPermuted = "";

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

            keyPermuted = keyPermuted + key.charAt(SP[i] - 1);

        }

        String left = keyPermuted.substring(0, 5);

        String right = keyPermuted.substring(5, 10);

        for (int i = 0; i < 2; i++) {

            String leftShifted = circularLeftShift(left, i + 1);

            String rightShifted = circularLeftShift(right, i + 1);

            String combined = leftShifted + rightShifted;

            String subkey = "";

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

                subkey = subkey + combined.charAt(compressionTable[j] - 1);

            }

            subkeys[i] = subkey;

            left = leftShifted;

            right = rightShifted;

        }

        return subkeys;

    }

    public static String XOR(String a, String b) {

        String result = "";

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

            if (a.charAt(i) == b.charAt(i)) {

                result = result + "0";

            } else {

                result = result + "1";

            }

        }

        return result;

    }

    public static String SBOX(String input, int sBoxNumber) {

        int[][] sBox1 = {{1, 0, 3, 2}, {3, 2, 1, 0}, {0, 2, 1, 3}, {3, 1, 3, 2}};

        int[][] sBox2 = {{0, 1, 2, 3}, {2, 0, 1, 3}, {3, 0, 1, 0}, {2, 1, 0, 3}};

        String rowBits = "" + input.charAt(0) + input.charAt(3);

        String colBits = "" + input.charAt(1) + input.charAt(2);

        int rowIndex = Integer.parseInt(rowBits, 2);

        int colIndex = Integer.parseInt(colBits, 2);

        String result;

        if (sBoxNumber == 1) {

            result = Integer.toBinaryString(sBox1[rowIndex][colIndex]);

        } else {

            result = Integer.toBinaryString(sBox2[rowIndex][colIndex]);

        }

        if (result.length() == 1) {

            result = "0" + result;

        }

        return result;

    }

    public static String Feistal(String input, String[] subkeys, int sBoxNumber) {

        int[] EP = {4, 1, 2, 3, 2, 3, 4, 1};

        int[] SP = {2, 4, 3, 1};

        String expandedInput = "";

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

            expandedInput += input.charAt(EP[j] - 1);

        }

        String xorResult = XOR(expandedInput, subkeys[sBoxNumber]);

        String leftBits = xorResult.substring(0, 4);

        String rightBits = xorResult.substring(4, 8);

        String sBoxResult = SBOX(leftBits, 1) + SBOX(rightBits, 2);

        String output = "";

        for (int k = 0; k < SP.length; k++) {

            output = output + sBoxResult.charAt(SP[k] - 1);

        }

        return output;

    }

    public static String process(String data, String[] subkeys, boolean isEncryption) {

        int[] IP = {2, 6, 3, 1, 4, 8, 5, 7};

        int[] FP = {4, 1, 3, 5, 7, 2, 8, 6};

        String processedData = "";

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

            processedData += data.charAt(IP[j] - 1);

        }

        String leftBits = processedData.substring(0, 4);

        String rightBits = processedData.substring(4, 8);

        for (int i = 0; i < 2; i++) {

            String functionResult;

            if (isEncryption) {

                functionResult = Feistal(rightBits, subkeys, i);

            } else {

                functionResult = Feistal(rightBits, subkeys, subkeys.length - i - 1);

            }

            String newRightBits = XOR(leftBits, functionResult);

            if (i == 0) {

                leftBits = rightBits;

                rightBits = newRightBits;

            } else {

                leftBits = newRightBits;

            }

        }

        String finalData = leftBits + rightBits;

        String outputData = "";

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

            outputData += finalData.charAt(FP[j] - 1);

        }

        return outputData;

    }

    public static void main(String[] args) {

        String data;

        String key;

        Scanner scanner = new Scanner(System.in);

        System.out.println("Data : ");

        data = scanner.next();

        System.out.println("--------------------");

        System.out.println("Key : ");

        key = scanner.next();

        String[] subkeys = generateSubkeys(data, key);

        System.out.println("Generated Subkeys in 2 rounds:");

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

            System.out.println("Subkey " + i + " " + subkeys[i]);

        }

        System.out.println("--------------------");

        String encryptedData = process(data, subkeys, true);

        System.out.println("Encrypted Data: " + encryptedData);

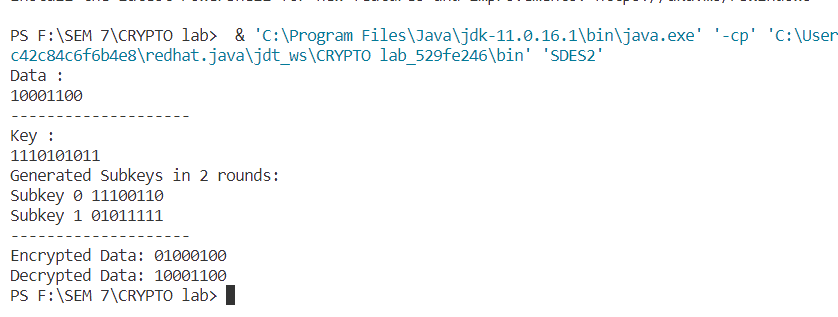
        String decryptedData = process(encryptedData, subkeys, false);

        System.out.println("Decrypted Data: " + decryptedData);

    }

}

**Output:**

****

**Result:**

Thus, the program for the implementation of simplified DES is completed and output is displayed and verified

**Exp No**:4           **SIMPLIFIED ADVANCED ENCRYPTION STANDARD**

**Date**:06-10-2023

**Aim**

To implement the Simplified AES using java and perform the encryption and decryption.

**Algorithm:**

**Encryption:**

1. Generate Round Keys: Create round keys from the master key using a key schedule, similar to AES.
2. Initial Round Key Addition: XOR the plaintext with the first round key (Round 1 key).
3. SubBytes: Replace each byte in the state with the corresponding value from the S-Box.
4. Shift Rows: Shift the rows in the state matrix.
5. Mix Columns (Not used in S-AES).
6. Round Key Addition: XOR the state with the round key for the current round (Round 2 key).
7. Repeat Steps 3 to 6 for a total of three rounds (Round 2 and Round 3).
8. Ciphertext: The final state matrix is the ciphertext.

**Decryption:**

1. Generate Round Keys: Recreate the round keys used in encryption.
2. Initial Round Key Addition: XOR the ciphertext with the last round key (Round 3 key).
3. Inverse Shift Rows: Reverse the shift rows operation.
4. Inverse SubBytes: Apply the inverse S-Box operation to each byte in the state.
5. Round Key Addition: XOR the state with the round key for the current round (Round 2 key).
6. Inverse Shift Rows: Reverse the shift rows operation.
7. Inverse SubBytes: Apply the inverse S-Box operation again.
8. Round Key Addition: XOR the state with the first round key (Round 1 key).
9. Plaintext Recovery: The final state matrix is the decrypted plaintext.

**Code:**

import java.util.\*;

public class SAES {

    static HashMap<String, String> map = new HashMap<>();

    static HashMap<Integer, String> mixColMat = new HashMap<>();

    static HashMap<String, String> invSbox = new HashMap<>();

    public static String Rotate(String a){

        String ans=a.substring(4,8)+a.substring(0,4);

        return ans;

    }

    public static String SBOX(String a){

        String a1=a.substring(0,4);

        String a2=a.substring(4,8);

        map.put("0000", "1001");

        map.put("0001", "0100");

        map.put("0010", "1010");

        map.put("0011", "1011");

        map.put("0100", "1101");

        map.put("0101", "0001");

        map.put("0110", "1000");

        map.put("0111", "0101");

        map.put("1000", "0110");

        map.put("1001", "0010");

        map.put("1010", "0000");

        map.put("1011", "0011");

        map.put("1100", "1100");

        map.put("1101", "1110");

        map.put("1110", "1111");

        map.put("1111", "0111");

        String ans1=map.get(a1);

        String ans2=map.get(a2);

        return ans1+ans2;

    }

    public static String invSBOX(String a){

        String a1=a.substring(0,4);

        String a2=a.substring(4,8);

        invSbox.put( "1001","0000");

        invSbox.put("0100","0001");

        invSbox.put("1010","0010");

        invSbox.put("1011","0011");

        invSbox.put("1101","0100");

        invSbox.put("0001","0101");

        invSbox.put("1000","0110");

        invSbox.put("0101","0111");

        invSbox.put("0110","1000");

        invSbox.put( "0010","1001");

        invSbox.put("0000","1010");

        invSbox.put("0011","1011");

        invSbox.put("1100","1100");

        invSbox.put("1110","1101");

        invSbox.put("1111","1110");

        invSbox.put("0111","1111");

        String ans1=map.get(a1);

        String ans2=map.get(a2);

        return ans1+ans2;

    }

    public static String XOR(String a,String b){

        String ans="";

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

            if(a.charAt(i)==b.charAt(i)){

                ans=ans+'0';

            }

            else{

                ans=ans+'1';

            }

        }

        return ans;

    }

    public static String[] keyGeneration(String key){

        String[] keys=new String[3];

        String w0=key.substring(0,8);

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

        keys[0]=w0+w1;

        String r1="10000000";

        String r2="00110000";

        String key1\_steps=SBOX(Rotate(w1));

        String w2 = XOR(XOR(w0, r1), key1\_steps);

        String w3= XOR(w2, w1);

        String key2\_steps=SBOX(Rotate(w3));

        String w4=XOR(XOR(w2, r2), key2\_steps);

        String w5=XOR(w4, w3);

        keys[1]=w2+w3;

        keys[2]=w4+w5;

        return keys;

    }

    public static String substitution(String a){

        String w1=a.substring(0,4);

        String w2=a.substring(4,8);

        String w3=a.substring(8,12);

        String w4=a.substring(12,16);

        String ans=map.get(w1)+map.get(w2)+map.get(w3)+map.get(w4);

        System.out.println("Substitution : "+ans);

        return ans;

    }

    public static String invsubstitution(String a){

        String w1=a.substring(0,4);

        String w2=a.substring(4,8);

        String w3=a.substring(8,12);

        String w4=a.substring(12,16);

        String ans=invSbox.get(w1)+invSbox.get(w2)+invSbox.get(w3)+invSbox.get(w4);

        System.out.println("Inv Substitution : "+ans);

        return ans;

    }

    public static String shiftRows(String a){

        String w1=a.substring(0,4);

        String w2=a.substring(4,8);

        String w3=a.substring(8,12);

        String w4=a.substring(12,16);

        String ans=w1+w4+w3+w2;

        System.out.println("Shift Rows : "+ans);

        return ans;

    }

    public static String Multiply(String a){

        String ans="0000";

        for(int i=3;i>=0;i--){

            if(a.charAt(3-i)=='1'){

                ans  =XOR(ans,mixColMat.get(i+2));

            }

        }

        return ans;

    }

     public static String mixer\_2(String a){

        String ans="0000";

        for(int i=3;i>=0;i--){

            if(a.charAt(3-i)=='1'){

                ans  =XOR(ans,mixColMat.get(i+1));

            }

        }

        return ans;

    }

     public static String mixer\_9(String a){

        String ans="0000";

         for(int i=3;i>=0;i--){

            if(a.charAt(3-i)=='1'){

                ans  =XOR(ans,mixColMat.get(i));

                ans  =XOR(ans,mixColMat.get(i+3));

            }

        }

        return ans;

    }

    public static String mixColumns(String a){

        mixColMat.put(0, "0000");

        mixColMat.put(1, "0001");

        mixColMat.put(2, "0100");

        mixColMat.put(3, "1000");

        mixColMat.put(4, "0011");

        mixColMat.put(5, "0110");

        mixColMat.put(6, "1100");

        String w1=a.substring(0,4);

        String w2=a.substring(4,8);

        String w3=a.substring(8,12);

        String w4=a.substring(12,16);

        String ans1=XOR(w1,Multiply(w2));

        String ans2=XOR(Multiply(w1), w2);

        String ans3=XOR(w3,Multiply(w4));

        String ans4=XOR(Multiply(w3), w4);

        System.out.println("Mix Column : "+ans1+" "+ans2+" "+ans3+" "+ans4);

        return ans1+ans2+ans3+ans4;

    }

    public static String inv\_mixColumns(String a){

        mixColMat.put(0, "0001");

        mixColMat.put(1, "0010");

        mixColMat.put(2, "0100");

        mixColMat.put(3, "1000");

        mixColMat.put(4, "0011");

        mixColMat.put(5, "0110");

        mixColMat.put(6, "1100");

        String w1=a.substring(0,4);

        String w2=a.substring(4,8);

        String w3=a.substring(8,12);

        String w4=a.substring(12,16);

        String ans1=XOR(mixer\_9(w1),mixer\_2(w2));

        String ans2=XOR(mixer\_2(w1), mixer\_9(w2));

        String ans3=XOR(mixer\_9(w3),mixer\_2(w4));

        String ans4=XOR(mixer\_2(w3), mixer\_9(w4));

        System.out.println("Inv Mix Column : "+ans1+" "+ans2+" "+ans3+" "+ans4);

        return ans1+ans2+ans3+ans4;

    }

    public static String round(String plaintext,String key){

        String val=substitution(plaintext);

        val=shiftRows(val);

        val=mixColumns(val);

        val=XOR(val, key);

        System.out.println("Round : "+val);

        return val;

    }

    public static String finalround(String plaintext,String key){

        String val=substitution(plaintext);

        val=shiftRows(val);

        val=XOR(val, key);

        System.out.println("Final Round : "+val);

        return val;

    }

    private static String roundDec(String text,String key){

        text=shiftRows(text);

        text=invsubstitution(text);

        text=XOR(text,key);

        text=inv\_mixColumns(text);

        System.out.println("Round Dec : "+text);

        return text;

    }

    private static String finalroundDec(String text,String key){

        text=shiftRows(text);

        text=invsubstitution(text);

        text=XOR(text,key);

        System.out.println("Final Round Dec : "+text);

        return text;

    }

    public static void main(String[] args){

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the Plaintext(16 bits): ");

        String plaintext=sc.next();

        System.out.println("Enter the Key(16 bits): ");

        String key=sc.next();

        String temp=invSBOX("11111111");

        String[] keys=keyGeneration(key);

        System.out.println("\nKEYS: ");

        int ct=0;

        for(String i:keys){

            System.out.println("key"+ct+": "+i);

            ct++;

        }

        plaintext = XOR(plaintext, keys[0]);

        //round 1

        plaintext = round(plaintext, keys[1]);

        //FinalRound

        String ciphertext = finalround(plaintext,keys[2]);

        System.out.println("\nEncryption");

        System.out.println("Ciphertext: "+ciphertext);

        System.out.println("\nDecryption");

        ciphertext=XOR(ciphertext,keys[2]);

        ciphertext=roundDec(ciphertext,keys[1]);

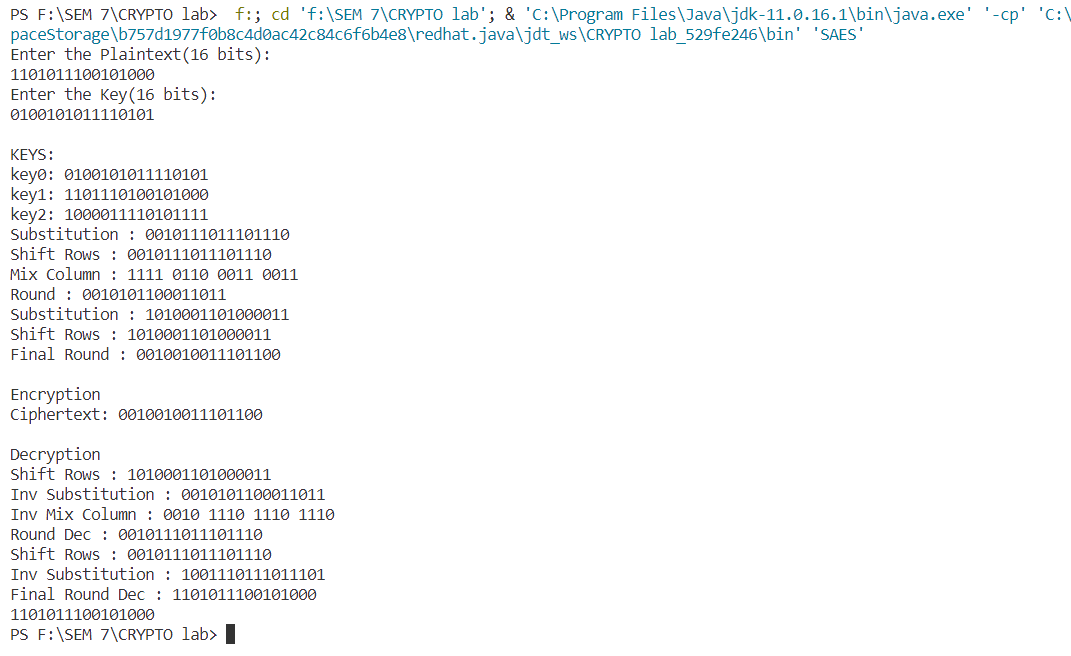
        ciphertext=finalroundDec(ciphertext,keys[0]);

        System.out.println(ciphertext);

    }

}

**Output:**

****

**Result:**

Thus, the program for the implementation of simplified AES is completed and output is displayed and verified

**Exp No**:5               **RSA(Rivest–Shamir–Adleman) Encryption**

**Date**:06-10-2023

**Aim**

To implement the RSA algorithm using java and perform the encryption and decryption.

**Algorithm:**

**Key Generation**:

1. Select two large prime numbers, **p** and **q**.
2. Calculate **n = p \* q** as the modulus.
3. Calculate Euler's totient function: **φ(n) = (p - 1) \* (q - 1)**.
4. Choose a public exponent **e** such that **1 < e < φ(n)** and **e** is relatively prime to **φ(n)**. Common values for **e** are 3 or 65537.
5. Calculate the private exponent **d** as the modular multiplicative inverse of **e** modulo **φ(n)**, i.e., **(d \* e) % φ(n) = 1**.
6. Public Key: **(n, e)**
7. Private Key: **(n, d)**

**Encryption**:

1. Convert the plaintext message **M** into a numerical value **m** where **0 <= m < n**.
2. Calculate the ciphertext **C** as: **C = (m^e) % n**.

**Decryption**:

1. Given the ciphertext **C**, compute the plaintext message **m** as: **m = (C^d) % n**.
2. Convert **m** back to text to retrieve the original message **M**.

**RSA Encryption (Sender)**:

1. Obtain the recipient's public key **(n, e)**.
2. Convert the message **M** to a numerical value **m**.
3. Calculate the ciphertext **C** using the recipient's public key: **C = (m^e) % n**.
4. Send the ciphertext **C** to the recipient.

**RSA Decryption (Recipient)**:

1. Use the private key **(n, d)** to decrypt the ciphertext **C**.
2. Compute the plaintext message **m** as: **m = (C^d) % n**.
3. Convert **m** back to text to retrieve the original message **M**.

**Code:**

import java.util.Scanner;

public class RSA {

    public static long SqMul(long base, long exponent, long modulus) {

        long result = 1;

        base = base % modulus;

        while (exponent > 0) {

            if (exponent % 2 == 1) {

                result = (result \* base) % modulus;

            }

            base = (base \* base) % modulus;

            exponent = exponent / 2;

        }

        return result;

    }

    public static int Inverse(int a, int n) {

        int m0 = n;

        int x0 = 0;

        int x1 = 1;

        if (n == 1) {

            return 0;

        }

        while (a > 1) {

            int q = a / n;

            int t = n;

            n = a % n;

            a = t;

            t = x0;

            x0 = x1 - q \* x0;

            x1 = t;

        }

        if (x1 < 0) {

            x1 += m0;

        }

        return x1;

    }

    public static int GCD(int a, int b) {

        int r1 = a;

        int r2 = b;

        if (a > b) {

            r1 = a;

            r2 = b;

        } else {

            r2 = a;

            r2 = b;

        }

        int q, r = 0;

        while (r2 != 0) {

            q = r1 / r2;

            r = r1 % r2;

            r1 = r2;

            r2 = r;

        }

        return r1;

    }

      public static long RSA(long p, int e, int n) {

        long ans = SqMul(p, e, n);

        return ans;

    }

    public static long [] ToNum(String str,int e,int n){

        long[] arr=new long[str.length()];

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

            long c=(long)str.charAt(i);

            long enc=RSA(c,e,n);

            arr[i]=enc;

        }

        return arr;

    }

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.print("\nString :");

        String text=sc.next();

        int p = 3271;

        int q = 3461;

        int n = p \* q;

        int phi\_n = (p - 1) \* (q - 1);

        int e = 10;

        while (e < phi\_n) {

            if (GCD(e, phi\_n) == 1) {

                break;

            }

            else {

                e++;

            }

        }

        int d = Inverse(e, phi\_n);

        System.out.println("Public key: " + e);

        System.out.println("Private key: " + d);

        long[] enc=ToNum(text,e,n);

        System.out.print("Encrypted text: ");

        for(long i:enc){

            System.out.print(i+" ");

        }

        String ans="";

        for(long i:enc){

            long decrypt=RSA(i, d, n);

            ans=ans+(char)decrypt;

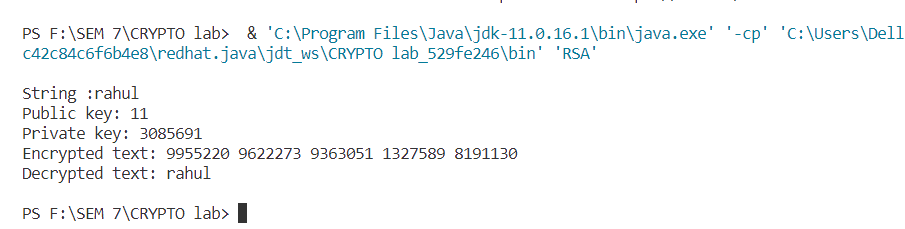
        }

        System.out.println("\nDecrypted text: "+ans+"\n");

    }

}

**Output:**

****

**Result:**

Thus, the program for the implementation of RSA is completed and output is displayed and verified.

**Exp No**:6               **Digital Signature Standard**

**Date**:13-10-2023

**Aim**

To implement the DSS algorithm using java and perform the validation of the signature.

**Algorithm:**

**Key Generation:**

1. Select two large prime numbers, **p** and **q**, such that **q** is a prime divisor of **(p - 1)**.
2. Choose a generator **g** for a subgroup of the multiplicative group modulo **p**. This subgroup has order **q**, and **g** raised to the power of **q** modulo **p** is congruent to 1.
3. Select a private key **x** such that **0 < x < q**.
4. Calculate the corresponding public key **y** as **y = g^x mod p**.

**Signature Generation:**

1. Calculate a random value **k** such that **0 < k < q**.
2. Compute **r** as **r = (g^k mod p) mod q**.
3. Calculate the hash value of the message **H(m)**.
4. Calculate **s** as **s = (k^(-1) \* (H(m) + x \* r)) mod q**, where **k^(-1)** is the modular multiplicative inverse of **k** modulo **q**.
5. The signature is **(r, s)**.

**Signature Verification:**

1. Receive the message **m** and the signature **(r, s)**.
2. Calculate the hash value of the message, **H(m)**.
3. Verify that **r** and **s** are in the range **0 < r < q** and **0 < s < q**.
4. Calculate **w** as **w = s^(-1) mod q**, where **s^(-1)** is the modular multiplicative inverse of **s** modulo **q**.
5. Calculate **u1** as **u1 = (H(m) \* w) mod q**.
6. Calculate **u2** as **u2 = (r \* w) mod q**.
7. Calculate **v** as **v = ((g^u1 \* y^u2) mod p) mod q**.
8. If **v** is equal to **r**, the signature is valid; otherwise, it's invalid.

**Code:**

import java.util.Scanner;

public class DSS {

    public static int modInv(int b, int m) {

        int t = 0, t2 = 1;

        int r = b, r2 = m;

        while (r2 != 0) {

            int quotient = r / r2;

            int tmpT = t2;

            t2 = t - quotient \* t2;

            t = tmpT;

            int tmpR = r2;

            r2 = r - quotient \* r2;

            r = tmpR;

        }

        if (r > 1) {

            throw new IllegalArgumentException("a is not invertible");

        }

        if (t < 0) {

            t = t + b;

        }

        return t;

    }

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        System.out.print("SHA-1(m) = ");

        int hf = sc.nextInt();

        System.out.print("P (prime modulus)= ");

        int p = sc.nextInt();

        System.out.print("Q (prime divisor of p - 1)= ");

        int q = sc.nextInt();

        System.out.print("h (1 < h < p-1) = ");

        int h = sc.nextInt();

        System.out.print("k = ");

        int k = sc.nextInt();

        System.out.print("x (0 < x < q) = ");

        int x = sc.nextInt();

        int pow = (p - 1) / q;

        int g = 1;

        for (int i = 0; i < pow; i++) {

            g = (g \* h) % p;

        }

        System.out.println("-----------\ng: " + g);

        System.out.println("Global Public keys: " + p + ", " + q + ", " + g);

        int y = 1;

        for (int i = 0; i < x; i++) {

            y = (y \* g) % p;

        }

        System.out.println("Public key:(y) " + y);

        int r1 = 1;

        int r = 0;

        for (int i = 0; i < k; i++) {

            r1 = (r1 \* g) % p;

            r = r1 % q;

        }

        System.out.println("r = " + r);

        int invert = modInv(k, q);

        int t = x \* r;

        int s1 = invert \* (hf + t);

        int st = -s1;

        while (st > q) {

            st = st % q;

        }

        int s = q - st;

        System.out.println("s = " + s);

        int w = modInv(s, q);

        System.out.println("w =  " + w);

        int n1 = (hf \* w) % q;

        int n2 = (r \* w);

        int st1 = -n2;

        if (st1 > q) {

            st1 = st1 % q;

        }

        int n21 = q - st1;

        int v1 = 1;

        for (int i = 0; i < n1; i++) {

            v1 = v1 \* g;

        }

        int v2 = 1;

        for (int i = 0; i < n21; i++) {

            v2 = v2 \* y;

        }

        int v3 = (v1 \* v2) % p;

        int v = v3 % q;

        System.out.println("V = " + v);

        if (r == v) {

            System.out.println("r==v Signature is verified");

        } else {

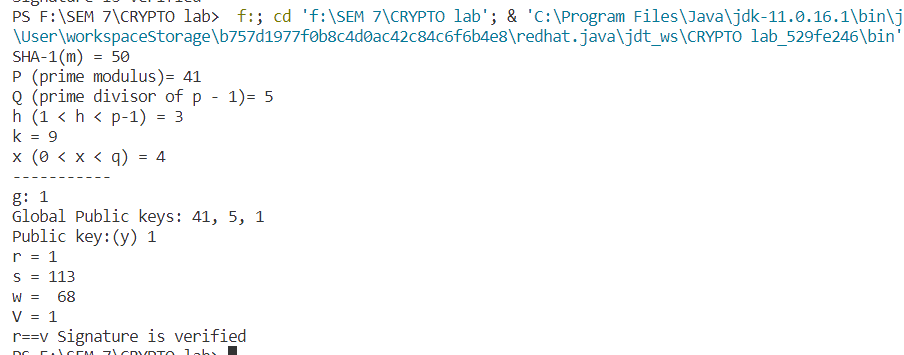
            System.out.println("r!=v Signature is not verified");

        }

    }

}

**Output:**

****

**Result:**

Thus, the program for the implementation of DSS is completed and output is displayed and verified.

**Exp No**:7               **Secure Hash Algorithm**

**Date**:13-10-2023

**Aim**

To implement the SHA512 algorithm using java

**Algorithm:**

Input:

- Message M (as a byte array)

Output:

- 512-bit hash value (as a byte array)

Constants:

- Initialize a set of constants based on the SHA-512 specification.

1. Preprocessing:

1.1. Append padding to the message:

- Add a '1' bit to the end of the message.

- Append '0' bits to the message until its length is congruent to 896 (mod 1024).

- Append the original message length as a 128-bit binary representation.

1.2. Break the padded message into 1024-bit blocks.

2. Initialize Hash Values:

2.1. Initialize eight 64-bit hash values (h0, h1, h2, h3, h4, h5, h6, h7) with predefined initial values.

3. Main Loop:

3.1. For each message block:

- Prepare a message schedule (W[0..79]) from the 1024-bit block.

- Extend the message schedule to 80 words (W[80..159]) using bitwise operations.

- Initialize working variables (a, b, c, d, e, f, g, h) with the current hash values (h0..h7).

- Perform 80 rounds of processing:

- Update the working variables a, b, c, d, e, f, g, h using a series of bitwise and arithmetic operations.

- Update the hash values (h0..h7) with the final working variables.

4. Compute the Final Hash:

4.1. Concatenate the final hash values h0, h1, h2, h3, h4, h5, h6, and h7 into a single 512-bit hash.

4.2. Convert the hash to a byte array format for output.

5. Output:

- The output is the 512-bit hash value as a byte array.

**Code:**

import java.io.IOException;

import java.math.BigInteger;

import java.util.Scanner;

class Constants {

protected static long[] IV = new long[] {

0x6A09E667F3BCC908L,

0xBB67AE8584CAA73BL,

0x3C6EF372FE94F82BL,

0xA54FF53A5F1D36F1L,

0x510E527FADE682D1L,

0x9B05688C2B3E6C1FL,

0x1F83D9ABFB41BD6BL,

0x5BE0CD19137E2179L

};

protected static long[] K = new long[] {

0x428A2F98D728AE22L, 0x7137449123EF65CDL, 0xB5C0FBCFEC4D3B2FL, 0xE9B5DBA58189DBBCL, 0x3956C25BF348B538L,0x59F111F1B605D019L, 0x923F82A4AF194F9BL, 0xAB1C5ED5DA6D8118L, 0xD807AA98A3030242L, 0x12835B0145706FBEL,0x243185BE4EE4B28CL, 0x550C7DC3D5FFB4E2L, 0x72BE5D74F27B896FL, 0x80DEB1FE3B1696B1L, 0x9BDC06A725C71235L, 0xC19BF174CF692694L, 0xE49B69C19EF14AD2L, 0xEFBE4786384F25E3L, 0x0FC19DC68B8CD5B5L, 0x240CA1CC77AC9C65L,0x2DE92C6F592B0275L, 0x4A7484AA6EA6E483L, 0x5CB0A9DCBD41FBD4L, 0x76F988DA831153B5L, 0x983E5152EE66DFABL,0xA831C66D2DB43210L, 0xB00327C898FB213FL, 0xBF597FC7BEEF0EE4L, 0xC6E00BF33DA88FC2L, 0xD5A79147930AA725L,

0x06CA6351E003826FL, 0x142929670A0E6E70L, 0x27B70A8546D22FFCL, 0x2E1B21385C26C926L, 0x4D2C6DFC5AC42AEDL,0x53380D139D95B3DFL, 0x650A73548BAF63DEL, 0x766A0ABB3C77B2A8L, 0x81C2C92E47EDAEE6L, 0x92722C851482353BL,0xA2BFE8A14CF10364L, 0xA81A664BBC423001L, 0xC24B8B70D0F89791L, 0xC76C51A30654BE30L, 0xD192E819D6EF5218L,

0xD69906245565A910L, 0xF40E35855771202AL, 0x106AA07032BBD1B8L, 0x19A4C116B8D2D0C8L, 0x1E376C085141AB53L,0x2748774CDF8EEB99L, 0x34B0BCB5E19B48A8L, 0x391C0CB3C5C95A63L, 0x4ED8AA4AE3418ACBL, 0x5B9CCA4F7763E373L,0x682E6FF3D6B2B8A3L, 0x748F82EE5DEFB2FCL, 0x78A5636F43172F60L, 0x84C87814A1F0AB72L, 0x8CC702081A6439ECL,

0x90BEFFFA23631E28L, 0xA4506CEBDE82BDE9L, 0xBEF9A3F7B2C67915L, 0xC67178F2E372532BL, 0xCA273ECEEA26619CL,0xD186B8C721C0C207L, 0xEADA7DD6CDE0EB1EL, 0xF57D4F7FEE6ED178L, 0x06F067AA72176FBAL, 0x0A637DC5A2C898A6L,0x113F9804BEF90DAEL, 0x1B710B35131C471BL, 0x28DB77F523047D84L, 0x32CAAB7B40C72493L, 0x3C9EBE0A15C9BEBCL,

0x431D67C49C100D4CL, 0x4CC5D4BECB3E42B6L, 0x597F299CFC657E2AL, 0x5FCB6FAB3AD6FAECL, 0x6C44198C4A475817L

};

};

class Logic {

public static long Ch(long x, long y, long z) {

return (x & y) ^ (~x & z);

}

public static long Maj(long x, long y, long z) {

return (x & y) ^ (x & z) ^ (y & z);

}

public static long rotate(long x, int l) {

return (x >>> l) | (x << (Long.SIZE - l));

}

public static long Sigma0(long x) {

return rotate(x, 28) ^ rotate(x, 34) ^ rotate(x, 39);

}

public static long Sigma1(long x) {

return rotate(x, 14) ^ rotate(x, 18) ^ rotate(x, 41);

}

public static long \_Sigma0(long x) {

return rotate(x, 1) ^ rotate(x, 8) ^ (x >>> 7);

}

public static long \_Sigma1(long x) {

return rotate(x, 19) ^ rotate(x, 61) ^ (x >>> 6);

}

public static byte[] pad(byte[] input) {

int size = input.length + 17;

while (size % 128 != 0) {

size += 1;

}

byte[] out = new byte[size];

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

out[i] = input[i];

}

out[input.length] = (byte) 0x80;

byte[] lenInBytes = BigInteger.valueOf(input.length \* 8).toByteArray();

for (int i = lenInBytes.length; i > 0; i--) {

out[size - i] = lenInBytes[lenInBytes.length - i];

}

return out;

}

public static long arrToLong(byte[] input, int j) {

long v = 0;

for (int i = 0; i < 8; i++) {

v = (v << 8) + (input[i + j] & 0xff);

}

return v;

}

public static long[][] toBlocks(byte[] input) {

long[][] blocks = new long[input.length / 128][16];

for (int i = 0; i < input.length / 128; i++) {

for (int j = 0; j < 16; j++) {

blocks[i][j] = arrToLong(input, i \* 128 + j \* 8);

}

}

return blocks;

}

public static long[][] Message(long[][] M) {

long[][] W = new long[M.length][80];

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

for (int j = 0; j < 16; j++) {

W[i][j] = M[i][j];

}

for (int j = 16; j < 80; j++) {

W[i][j] = \_Sigma1(W[i][j-2]) + W[i][j-7] + \_Sigma0(W[i][j-15]) + W[i][j-16];

}

}

return W;

}

}

public class SHA512\_80 {

public static int DEBUG = 0;

public static void main(String[] args) throws IOException {

Scanner sc = new Scanner(System.in);

DEBUG = 2;

System.out.print("Enter the string:");

String input = sc.next();

input = input.replaceAll("\r\n", "\n");

String hashed = SHA512\_80.hash(input.getBytes());

System.out.println(hashed);

}

public static String hash(byte[] input) {

input = Logic.pad(input);

long[][] blocks = Logic.toBlocks(input);

long[][] W = Logic.Message(blocks);

long[] buffer = Constants.IV.clone();

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

long a = buffer[0]; long b = buffer[1];long c = buffer[2];long d = buffer[3];

long e = buffer[4];long f = buffer[5];long g = buffer[6]; long h = buffer[7];

for (int j = 0; j < 80; j++) {

long t1 = h + Logic.Sigma1(e) + Logic.Ch(e, f, g) + Constants.K[j] + W[i][j];

long t2 = Logic.Sigma0(a) + Logic.Maj(a, b, c);

h = g;

g = f;

f = e;

e = d + t1;

d = c;

c = b;

b = a;

a = t1 + t2;

}

buffer[0] = a + buffer[0];

buffer[1] = b + buffer[1];

buffer[2] = c + buffer[2];

buffer[3] = d + buffer[3];

buffer[4] = e + buffer[4];

buffer[5] = f + buffer[5];

buffer[6] = g + buffer[6];

buffer[7] = h + buffer[7];

}

String result = "";

for (int i = 0; i < 8; i++) {

result += String.format("%016x", buffer[i]);

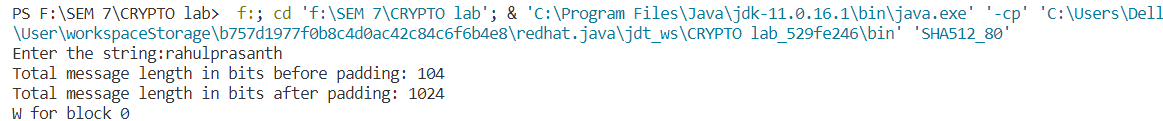
}

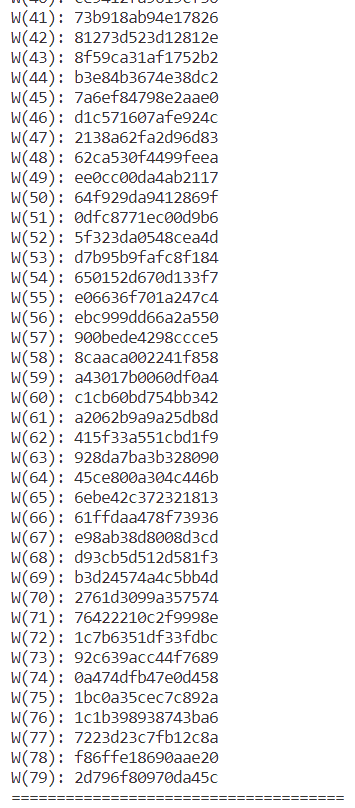
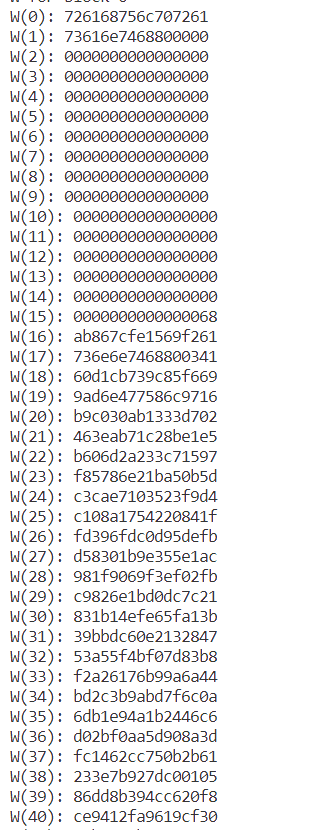
return result;

}

}

**Output:**

****

****

****

****

**Result:**

Thus SHA512 has been implemented successfully in java.