1. Caesar Cipher

class CaesarCipher{

public static String encrypt(String plainText,int key){

char [] plainChars = plainText.toCharArray();

for(int idx = 0; idx<plainText.length();idx++){

if(plainChars[idx]>=65 && plainChars[idx]<=91){

plainChars[idx] = (char)((plainChars[idx] + key - 65)%26 + 65) ;

}

else if(plainChars[idx]>=97 && plainChars[idx]<=122){

plainChars[idx] = (char)((plainChars[idx] + key - 97)%26 + 97) ;

}

}

return String.valueOf(plainChars);

}

public static String decrypt(String cipherText,int key){

char [] cipherChars = cipherText.toCharArray();

for(int idx = 0; idx<cipherText.length();idx++){

if(cipherChars[idx]>=65 && cipherChars[idx]<=91){

cipherChars[idx] = (char)((cipherChars[idx] + 26 - key - 65)%26 + 65) ;

}

else if(cipherChars[idx]>=97 && cipherChars[idx]<=122){

cipherChars[idx] = (char)((cipherChars[idx] + 26 - key - 97)%26 + 97) ;

}

}

return String.valueOf(cipherChars);

}

}

public class Main

{

public static void main(String[] args) {

String s = CaesarCipher.encrypt("ABCabcxyZAlu",10);

System.out.println(s);

String k = CaesarCipher.decrypt(s,10);

System.out.println(k);

}

}

1. PlayFair Cipher

class PlayFairCipher {

char [][]matrix = new char[5][5];

PlayFairCipher(String key){

int alphabets[] = new int[26];

int i = 0, j = 0;

key = key.toUpperCase();

for(int idx=0;idx<key.length();idx++){

char ch = key.charAt(idx);

if(ch =='j'){

ch = 'i';

}

alphabets[ch-'A']++;

if(alphabets[ch-'A'] == 1){

matrix[i][j] = ch;

j++;

if(j==5){

i++;

j=0;

}

if(i==5){

i=-1;

break;

}

}

}

if(i != -1){

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

if(alphabets[idx] == 0 && idx != 9){

matrix[i][j] = (char)(idx+'A');

j++;

if(j==5){

i++;

j=0;

}

if(i==5){

break;

}

}

}

}

}

int[] findIndices(char a){

if(Character.isAlphabetic(a) == true){

for(int i=0;i<5;i++){

for(int j=0;j<5;j++){

if(Character.toUpperCase(a) == matrix[i][j]){

int []arr = new int[2];

arr[0] = i;

arr[1] = j;

return arr;

}

}

}

return null;

}

else{

return null;

}

}

String digramEncoding(char a, char b){

int arr1[] = findIndices(a);

int arr2[] = findIndices(b);

if(arr1 == null || arr2 == null){

return a+""+b;

}

if(arr1[0] == arr2[0]){

int row = arr1[0];

return matrix[row][(arr1[1]+1)%5] + "" + matrix[row][(arr2[1]+1)%5] ;

}

else if(arr1[1] == arr2[1]){

int col = arr1[1];

return matrix[(arr1[0]+1)%5][col] + "" + matrix[(arr2[0]+1)%5][col];

}

else{

return matrix[arr1[0]][arr2[1]] + "" + matrix[arr2[0]][arr1[1]] ;

}

}

String encrypt(String plainText){

String encryptedText = "";

int i=0;

while(i<(plainText.length()-1)){

char ch1 = plainText.charAt(i);

char ch2 = plainText.charAt(i+1);

// System.out.println(ch1+":"+ch2+":"+i);

if(ch1 == ch2){

encryptedText = encryptedText + digramEncoding(ch1,'x');

i+=1;

}

else{

encryptedText = encryptedText + digramEncoding(ch1,ch2);

i+=2;

}

}

if(i == plainText.length()-1){

char ch1 = plainText.charAt(i);

encryptedText += digramEncoding(ch1,'z');

}

else if(i == plainText.length()-2){

char ch1 = plainText.charAt(i);

char ch2 = plainText.charAt(i+1);

encryptedText += digramEncoding(ch1,ch2);

}

return encryptedText;

}

String digramDecoding(char a, char b){

int arr1[] = findIndices(a);

int arr2[] = findIndices(b);

if(arr1 == null || arr2 == null){

return a+""+b;

}

if(arr1[0] == arr2[0]){

int row = arr1[0];

return matrix[row][( (arr1[1]-1)%5 +5)%5] + "" + matrix[row][ ((arr2[1]-1)%5 +5)%5] ;

}

else if(arr1[1] == arr2[1]){

int col = arr1[1];

return matrix[( (arr1[0]-1)%5 +5)%5][col] + "" + matrix[ ((arr2[0]-1)%5 +5)%5][col];

}

else{

return matrix[arr1[0]][arr2[1]] + "" + matrix[arr2[0]][arr1[1]] ;

}

}

String decrypt(String cipherText){

String decryptedText = "";

int i=0;

while(i<(cipherText.length()-1)){

char ch1 = cipherText.charAt(i);

char ch2 = cipherText.charAt(i+1);

// System.out.println(ch1+":"+ch2+":"+i);

decryptedText = decryptedText + digramDecoding(ch1,ch2);

i+=2;

}

return decryptedText;

}

void display(){

for(int i=0;i<5;i++){

for(int j=0;j<5;j++){

System.out.print(matrix[i][j]+" ");

}

System.out.println();

}

}

}

public class Main

{

public static void main(String[] args) {

PlayFairCipher pfc = new PlayFairCipher("Monarchy");

System.out.println("PlayFair Cipher Matrix : ");

pfc.display();

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

String enc = pfc.encrypt("Balloon");

System.out.println(enc);

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

System.out.println(pfc.decrypt(enc));

}

}

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

/\*\*

\*

\* @author itlab

\*/

class HillCipher {

int[][] matrix;

float[][] inverse;

int size;

static final int N = 3;

public void keyInit(String key, int n) {

this.matrix = new int[n][n];

this.size = n;

int index = 0;

key = key.toUpperCase();

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

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

if (key.charAt(index) >= 'A' && key.charAt(index) <= 'Z')

this.matrix[i][j] = (int)(key.charAt(index) - 'A');

index++;

}

}

this.inverseMat(this.matrix);

}

public void inverseMat(int A[][]) {

int i, j;

float det = 0;

this.inverse = new float[N][N];

System.out.println("Enter elements of matrix row wise:");

for(i = 0; i < 3; i++)

det = det + (A[0][i] \* (A[1][(i+1)%3] \* A[2][(i+2)%3] - A[1][(i+2)%3] \* A[2][(i+1)%3]));

System.out.println("\ndeterminant = " + det);

for(i = 0; i < 3; ++i) {

for(j = 0; j < 3; ++j){

System.out.println( (((A[(j+1)%3][(i+1)%3] \* A[(j+2)%3][(i+2)%3]) - (A[(j+1)%3][(i+2)%3] \* A[(j+2)%3][(i+1)%3])))%det + " ");

this.inverse[i][j] = (((A[(j+1)%3][(i+1)%3] \* A[(j+2)%3][(i+2)%3]) - (A[(j+1)%3][(i+2)%3] \* A[(j+2)%3][(i+1)%3]))%det);

}

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

}

}

public void display() {

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

for (int j = 0; j < this.size; j++) {

System.out.print(this.matrix[i][j] + " ");

}

System.out.println();

}

System.out.println("Inverse");

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

for (int j = 0; j < this.size; j++) {

System.out.print(this.inverse[i][j] + " ");

}

System.out.println();

}

}

public String encrypt(String plainText) {

int ptMatrix[] = new int[plainText.length()];

plainText = plainText.toUpperCase();

// PlainText Matrix Generation

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

if (plainText.charAt(i) >= 'A' && plainText.charAt(i) <= 'Z')

ptMatrix[i] = (int)(plainText.charAt(i) - 'A');

}

// kEY and PlainText Matrix Multiplication

int encrypted[] = new int[this.size];

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

for (int j = 0; j < 1; j++) {

for (int k = 0; k < this.size; k++) {

encrypted[i] += (this.matrix[i][k] \* ptMatrix[k]);

}

}

}

String encryptedText = "";

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

System.out.println(encrypted[i] % 26);

encryptedText += (char)(encrypted[i] % 26 + 'A');

}

return encryptedText;

}

public String decrypt(String cipherText) {

int ctMatrix[] = new int[cipherText.length()];

cipherText = cipherText.toUpperCase();

// PlainText Matrix Generation

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

if (cipherText.charAt(i) >= 'A' && cipherText.charAt(i) <= 'Z')

ctMatrix[i] = (int)(cipherText.charAt(i) - 'A');

}

int decrypted[] = new int[this.size];

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

for (int j = 0; j < 1; j++) {

for (int k = 0; k < this.size; k++) {

decrypted[i] += (this.inverse[i][k] \* ctMatrix[k]);

}

}

}

String decryptedText = "";

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

System.out.println(decrypted[i] % 26);

decryptedText += (char)(decrypted[i] % 26 + 'A');

}

return decryptedText;

}

}

public class Main

{

public static void main(String[] args) {

HillCipher hc = new HillCipher();

hc.keyInit("ACTIVATED", 3);

String s = hc.encrypt("PEN");

String d = hc.decrypt(s);

System.out.println(d);

}

}

DES

AES

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package cryptography;

/\*\*

\*

\* @author itlab

\*/

import java.security.SecureRandom;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.IvParameterSpec;

import javax.xml.bind.DatatypeConverter;

/\*\*

\*

\* @author anton

\*/

class AESImplement {

private static final String AES

= "AES";

// We are using a Block cipher(CBC mode)

private static final String AES\_CIPHER\_ALGORITHM

= "AES/CBC/PKCS5PADDING";

// Function to create a

// secret key

public static SecretKey createAESKey()

throws Exception {

SecureRandom securerandom = new SecureRandom();

KeyGenerator keygenerator = KeyGenerator.getInstance(AES);

keygenerator.init(256, securerandom);

SecretKey key

= keygenerator.generateKey();

return key;

}

// Function to initialize a vector

// with an arbitrary value

public static byte[] createInitializationVector() {

// Used with encryption

byte[] initializationVector

= new byte[16];

SecureRandom secureRandom

= new SecureRandom();

secureRandom.nextBytes(initializationVector);

return initializationVector;

}

// This function takes plaintext,

// the key with an initialization

// vector to convert plainText

// into CipherText.

public static byte[] do\_AESEncryption(

String plainText,

SecretKey secretKey,

byte[] initializationVector)

throws Exception {

Cipher cipher

= Cipher.getInstance(

AES\_CIPHER\_ALGORITHM);

IvParameterSpec ivParameterSpec

= new IvParameterSpec(

initializationVector);

cipher.init(Cipher.ENCRYPT\_MODE,

secretKey,

ivParameterSpec);

return cipher.doFinal(

plainText.getBytes());

}

// This function performs the

// reverse operation of the

// do\_AESEncryption function.

// It converts ciphertext to

// the plaintext using the key.

public static String do\_AESDecryption(

byte[] cipherText,

SecretKey secretKey,

byte[] initializationVector)

throws Exception {

Cipher cipher

= Cipher.getInstance(

AES\_CIPHER\_ALGORITHM);

IvParameterSpec ivParameterSpec= new IvParameterSpec(initializationVector);

cipher.init(

Cipher.DECRYPT\_MODE,

secretKey,

ivParameterSpec);

byte[] result = cipher.doFinal(cipherText);

return new String(result);

}

}

public class AES {

public static void main(String args[]){

try{

SecretKey Symmetrickey = AESImplement.createAESKey();

System.out.println("Symmetric Key : "+DatatypeConverter.printHexBinary(Symmetrickey.getEncoded()));

byte[] initializationVector= AESImplement.createInitializationVector();

System.out.println("Initialization Vector :" + DatatypeConverter.printHexBinary(initializationVector));

byte[] encrypted= AESImplement.do\_AESEncryption("HELLO", Symmetrickey, initializationVector);

System.out.println("Encrypted Data : "+DatatypeConverter.printHexBinary(encrypted));

String decryptedText = AESImplement.do\_AESDecryption(encrypted,Symmetrickey,initializationVector);

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

}catch(Exception e){

System.out.println(e.getStackTrace());

}

}

}

6. Diffie hellman

// Diffie Hellman Key Exchange algorithm

import java.util.\*;

public class Main

{

public static void main(String[] args) {

long q ,alpha , xa, xb,ya,yb,ka,kb;

Scanner s = new Scanner(System.in);

System.out.println("Enter prime number : ");

q = s.nextLong();

System.out.println("Enter primitive root of q : ");

alpha = s.nextLong();

// USER A Key Generation

System.out.println("Enter a number less than primitive root : ");

xa = s.nextLong();

ya = (long)Math.pow(alpha,xa);

ya = ya%q;

// USER B Key Generation

System.out.println("Enter a number less than primitive root : ");

xb = s.nextLong();

yb = (long)Math.pow(alpha,xb);

yb = yb%q;

//Secret key Generation

ka = ((long)Math.pow(yb,xa) )%q;

kb = ((long)Math.pow(ya,xb))%q;

System.out.println("Private key of Alice " + xa);

System.out.println("Public key of Alice " + ya);

System.out.println("Secret key of Alice " + ka);

System.out.println("Private key of Bob " + xb);

System.out.println("Public key of Bob " + yb);

System.out.println("Secret key of Bob " + kb);

}

}

7.

import java.security.\*;

public class Main

{

public static void main (String[]a)

{

try

{

MessageDigest md = MessageDigest.getInstance ("SHA1");

String input = "sjce";

md.update (input.getBytes());

byte[] output = md.digest();

System.out.println ("SHA1(\"" + input + "\") = " + bytesToHex (output));

}

catch (Exception e)

{

System.out.println ("Exception: " + e);

}

}

public static String bytesToHex (byte[]b)

{

char hexDigit[] =

{ '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D',

'E', 'F'

};

StringBuffer buf = new StringBuffer ();

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

{

buf.append (hexDigit[(b[j] >> 4) & 0x0f]);

buf.append (hexDigit[b[j] & 0x0f]);

}

return buf.toString ();

}

}