*Index*

|  |  |  |
| --- | --- | --- |
| *Sr No.* | *Practical No.* | *Date* |
| *1* | *Practical No.1* | *01-09-2022* |
| *2* | *Practical No.2* | *01-09-2022* |
| *3* | *Practical No.3* | *08-09-2022* |
| *4* | *Practical No.4* | *08-09-2022* |
| *5* | *Practical No.5* | *15-09-2022* |
| *6* | *Practical No.6* | *15-09-2022* |
| *7* | *Practical No.7* | *22-09-2022* |
| *8* | *Practical No.8* | *22-09-2022* |
| *9* | *Practical No.9* | *22-09-2022* |

*Practical No.1:*

*Aim:*

*Write programs to implement the following Substitution Cipher Techniques:*

1. *Caesar Cipher*
2. *Monoalphabetic Cipher*

*a) Caesar Cipher*

*Code:*

*dictionary = ['a', 'b', 'c', 'd',*

*'e', 'f', 'g', 'h',*

*'i', 'j', 'k', 'l',*

*'m', 'n', 'o', 'p',*

*'q', 'r', 's', 't',*

*'u', 'v', 'w', 'x',*

*'y', 'z', '0', '1',*

*'2', '3', '4', '5',*

*'6', '7', '8', '9',*

*'!', '@', '#', '$',*

*'%', '^', '\*']*

*plain\_text = input("Enter Plain Text: ").replace(" ", "").lower()*

*key = int(input("Key: "))*

*cipher\_text = ""*

*print("Plain Text: ", plain\_text)*

*for letter in plain\_text:*

*cipher\_text += dictionary[(dictionary.index(letter) + key) % len(dictionary)]*

*print("Cipher Text: ", cipher\_text)*

*b) Monoalphabetic Cipher*

*Code:*

*dictionary = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']*

*plain\_text = input("Enter Plain Text: ").replace(" ", "").lower()*

*cipher\_text = ""*

*for letter in plain\_text:*

*if letter in dictionary:*

*index = dictionary.index(letter)*

*cipher\_text += dictionary[-(index + 1)]*

*else:*

*# If the character is not in the dictionary, leave it as is.*

*cipher\_text += letter*

*print("Cipher Text: ", cipher\_text)*

*Practical No.2:*

*Aim:*

*Write programs to implement the following Substitution Cipher Techniques:*

1. *Vernam Cipher*
2. *PlayFair Cipher*

*a) Vernam Cipher*

*Code:*

*dictionary = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']*

*plain\_text = input("Enter Plain Text: ").replace(" ", "").lower()*

*# Taking input (until right)*

*key = ""*

*while True:*

*key = input("Enter key (text) (same length as plain text): ").replace(" ", "").lower()*

*if len(key) == len(plain\_text):*

*break*

*print("Plain Text: ", plain\_text)*

*cipher\_text = ""*

*# XOR operation function*

*def performXOR(binary\_letter1, binary\_letter2):*

*return bin(int(binary\_letter1, 2) ^ int(binary\_letter2, 2))[2:]*

*for i in range(len(plain\_text)):*

*# Converting plain text and key to binary*

*index\_plain = dictionary.index(plain\_text[i])*

*index\_key = dictionary.index(key[i])*

*binary\_letter1 = format(index\_plain, '05b') # 05b ensures 5-bit binary representation*

*binary\_letter2 = format(index\_key, '05b')*

*# Perform XOR on binary representations*

*binary\_output = performXOR(binary\_letter1, binary\_letter2)*

*# Convert binary to letter*

*cipher\_text += dictionary[int(binary\_output, 2) % len(dictionary)]*

*print("Cipher Text: ", cipher\_text)*

*b) Playfair Cipher*

*Code:*

*dictionary = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']*

*plain\_text = input("Enter Plain Text: ").replace(" ", "").lower()*

*# Taking input (until right)*

*key = ""*

*while True:*

*key = input("Enter key (text) (same length as plain text): ").replace(" ", "").lower()*

*if len(key) == len(plain\_text):*

*break*

*print("Plain Text: ", plain\_text)*

*cipher\_text = ""*

*# XOR operation function*

*def performXOR(binary\_letter1, binary\_letter2):*

*return bin(int(binary\_letter1, 2) ^ int(binary\_letter2, 2))[2:]*

*for i in range(len(plain\_text)):*

*# Converting plain text and key to binary*

*index\_plain = dictionary.index(plain\_text[i])*

*index\_key = dictionary.index(key[i])*

*binary\_letter1 = format(index\_plain, '05b') # 05b ensures 5-bit binary representation*

*binary\_letter2 = format(index\_key, '05b')*

*# Perform XOR on binary representations*

*binary\_output = performXOR(binary\_letter1, binary\_letter2)*

*# Convert binary to letter*

*cipher\_text += dictionary[int(binary\_output, 2) % len(dictionary)]*

*print("Cipher Text: ", cipher\_text)*

*Practical No.3:*

*Aim:*

*Write programs to implement the following Transposition Cipher Techniques:*

1. *Rail-Fence Cipher*
2. *Simple-Columnar Technique*

*a) Rail-Fence Cipher*

*Code:*

*package ins;*

*import java.util.Scanner;*

*class INS {*

*int depth;*

*String Encryption(String plainText, int depth) {*

*int r = depth, len = plainText.length();*

*int c = (int) Math.ceil((double) len / depth);*

*char mat[][] = new char[r][c];*

*int k = 0;*

*String cipherText = "";*

*for (int i = 0; i < c; i++) {*

*for (int j = 0; j < r; j++) {*

*if (k < len)*

*mat[j][i] = plainText.charAt(k++);*

*else*

*mat[j][i] = 'X';*

*}*

*}*

*for (int i = 0; i < r; i++) {*

*for (int j = 0; j < c; j++) {*

*cipherText += mat[i][j];*

*}*

*}*

*return cipherText;*

*}*

*String Decryption(String cipherText, int depth) {*

*int r = depth, len = cipherText.length();*

*int c = (int) Math.ceil((double) len / depth);*

*char mat[][] = new char[r][c];*

*int k = 0;*

*String plainText = "";*

*for (int i = 0; i < r; i++) {*

*for (int j = 0; j < c; j++) {*

*mat[i][j] = cipherText.charAt(k++);*

*}*

*}*

*for (int i = 0; i < c; i++) {*

*for (int j = 0; j < r; j++) {*

*plainText += mat[j][i];*

*}*

*}*

*return plainText;*

*}*

*}*

*public class Railfence {*

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

*INS rf = new INS();*

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

*int depth;*

*String plainText, cipherText, decryptedText;*

*System.out.println("Enter plain text:");*

*plainText = scn.nextLine();*

*System.out.println("Enter depth for Encryption:");*

*depth = scn.nextInt();*

*cipherText = rf.Encryption(plainText, depth);*

*System.out.println("Encrypted text is:\n" + cipherText);*

*decryptedText = rf.Decryption(cipherText, depth);*

*System.out.println("Decrypted text is:\n" + decryptedText);*

*}*

*}*

*b) Simple-Columnar Technique*

*Code:*

*import java.io.\*;*

*class SCT*

*{*

*public static void main(String args[])throws Exception*

*{*

*BufferedReader br=new BufferedReader(new InputStreamReader(System.in));*

*System.out.println("Enter your plain text");*

*String accept=br.readLine();*

*System.out.println("Enter the no of rows "); int r=Integer.parseInt(br.readLine()); System.out.println("Enter the no of cols"); int c=Integer.parseInt(br.readLine()); int count=0;*

*char cont[][]=new char[r][c];*

*for(int i=0;i<r;i++)*

*{*

*for(int j=0;j<c;j++)*

*{*

*if(count>=accept.length())*

*{*

*cont[i][j]=' '; count++;*

*}*

*else { cont[i][j]=accept.charAt(count); count++;*

*}*

*}*

*}*

*System.out.println("\nEnter the order of cols you want to view them in"); int choice[]=new int[c];*

*for(int k=0;k<c;k++)*

*{*

*System.out.println("Choice "+k+"-> ");*

*choice[k]=Integer.parseInt(br.readLine());*

*}*

*System.out.println("\nCipher text in matrix is ->");*

*String cipher="";*

*for(int j=0;j<c;j++)*

*{*

*int k=choice[j];*

*for(int i=0;i<r;i++)*

*{*

*cipher+=cont[i][k];*

*}*

*}*

*cipher=cipher.trim(); System.out.println(cipher);*

*}*

*}*

*Output:*

*Practical No.4:*

*Aim:*

*Write a program to encrypt and decrypt strings using*

1. *DES algorithm*
2. *AES algorithm*

*a) DES Algorithm*

*Code:*

*import java.io.FileInputStream; import java.io.FileOutputStream; import java.io.IOException; import java.io.InputStream; import java.io.OutputStream;*

*import java.security.InvalidAlgorithmParameterException; import java.security.InvalidKeyException; import java.security.NoSuchAlgorithmException; import java.security.spec.AlgorithmParameterSpec; import javax.crypto.Cipher; import javax.crypto.CipherInputStream; import javax.crypto.CipherOutputStream; import javax.crypto.KeyGenerator; import javax.crypto.NoSuchPaddingException; import javax.crypto.SecretKey; import javax.crypto.spec.IvParameterSpec;*

*public class DES*

*{*

*private static Cipher encrypt; private static Cipher decrypt;*

*private static final byte[] initialization\_vector = { 22, 33, 11, 44, 55, 99, 66, 77 }; public static void main(String[] args)*

*{*

*String textFile ="C:/Users/AGENT47/Desktop/DemoData.txt";*

*String encryptedData = "C:/Users/AGENT47/Desktop/encrypteddata.txt";*

*String decryptedData = "C:/Users/AGENT47/Desktop/decrypteddata.txt"; try {*

*SecretKey scrtkey = KeyGenerator.getInstance("DES").generateKey(); AlgorithmParameterSpec aps = new IvParameterSpec(initialization\_vector); encrypt = Cipher.getInstance("DES/CBC/PKCS5Padding"); encrypt.init(Cipher.ENCRYPT\_MODE, scrtkey, aps); decrypt = Cipher.getInstance("DES/CBC/PKCS5Padding"); decrypt.init(Cipher.DECRYPT\_MODE, scrtkey, aps); encryption(new FileInputStream(textFile), new FileOutputStream(encryptedData)); decryption(new FileInputStream(encryptedData), new FileOutputStream(decryptedData)); System.out.println("The encrypted and decrypted files have been created successfully.");*

*}*

*catch (NoSuchAlgorithmException | NoSuchPaddingException | InvalidKeyException | InvalidAlgorithmParameterException|IOException e)*

*{*

*e.printStackTrace();*

*}*

*}*

*private static void encryption(InputStream input, OutputStream output) throws IOException*

*{*

*output = new CipherOutputStream(output, encrypt);*

*writeBytes(input, output);*

*}*

*private static void decryption(InputStream input, OutputStream output)*

*throws IOException*

*{*

*input = new CipherInputStream(input, decrypt); writeBytes(input, output);*

*}*

*private static void writeBytes(InputStream input, OutputStream output) throws IOException*

*{*

*byte[] writeBuffer = new byte[512]; int readBytes = 0;*

*while ((readBytes = input.read(writeBuffer)) >= 0)*

*{*

*output.write(writeBuffer, 0, readBytes);*

*}*

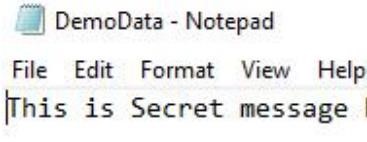
*output.close(); input.close();*

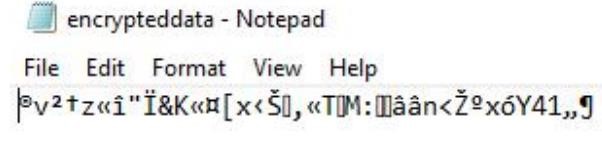
*}*

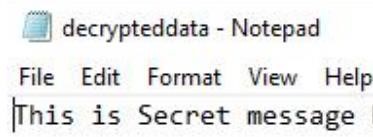
*}*

*Output:*

**

**

**

**

*b) AES Algorithm*

*Program Code:*

*import javax.crypto.Cipher; import javax.crypto.SecretKey; import javax.crypto.SecretKeyFactory; import javax.crypto.spec.IvParameterSpec; import javax.crypto.spec.PBEKeySpec; import javax.crypto.spec.SecretKeySpec; import java.nio.charset.StandardCharsets; import java.security.InvalidAlgorithmParameterException; import java.security.InvalidKeyException; import java.security.NoSuchAlgorithmException; import java.security.spec.InvalidKeySpecException; import java.security.spec.KeySpec; import java.util.Base64; import javax.crypto.BadPaddingException; import javax.crypto.IllegalBlockSizeException; import javax.crypto.NoSuchPaddingException;*

*public class AES*

*{*

*/\* Private variable declaration \*/ private static final String SECRET\_KEY = "123456789"; private static final String SALTVALUE = "abcdefg";*

*/\* Encryption Method \*/*

*public static String encrypt(String strToEncrypt)*

*{ try*

*{*

*/\* Declare a byte array. \*/*

*byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};*

*IvParameterSpec ivspec = new IvParameterSpec(iv);*

*/\* Create factory for secret keys. \*/*

*SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");*

*/\* PBEKeySpec class implements KeySpec interface. \*/*

*KeySpec spec = new PBEKeySpec(SECRET\_KEY.toCharArray(), SALTVALUE.getBytes(), 65536, 256);*

*SecretKey tmp = factory.generateSecret(spec);*

*SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES"); Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding"); cipher.init(Cipher.ENCRYPT\_MODE, secretKey, ivspec);*

*/\* Retruns encrypted value. \*/ return Base64.getEncoder()*

*.encodeToString(cipher.doFinal(strToEncrypt.getBytes(StandardCharsets.UTF\_8)));*

*}*

*catch (InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException | InvalidKeySpecException|BadPaddingException |*

*IllegalBlockSizeException | NoSuchPaddingException e)*

*{*

*System.out.println("Error occured during encryption: " + e.toString());*

*}*

*return null;*

*}*

*/\* Decryption Method \*/*

*public static String decrypt(String strToDecrypt)*

*{ try*

*{*

*/\* Declare a byte array. \*/*

*byte[] iv = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};*

*IvParameterSpec ivspec = new IvParameterSpec(iv);*

*/\* Create factory for secret keys. \*/*

*SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");*

*/\* PBEKeySpec class implements KeySpec interface. \*/*

*KeySpec spec = new PBEKeySpec(SECRET\_KEY.toCharArray(), SALTVALUE.getBytes(),*

*65536, 256);*

*SecretKey tmp = factory.generateSecret(spec);*

*SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES"); Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING"); cipher.init(Cipher.DECRYPT\_MODE, secretKey, ivspec);*

*/\* Retruns decrypted value. \*/*

*return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));*

*}*

*catch (InvalidAlgorithmParameterException | InvalidKeyException |*

*NoSuchAlgorithmException | InvalidKeySpecException|BadPaddingException | IllegalBlockSizeException | NoSuchPaddingException e)*

*{*

*System.out.println("Error occured during decryption: " + e.toString());*

*}*

*return null;*

*}*

*/\* Driver Code \*/*

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

*{*

*/\* Message to be encrypted. \*/*

*String originalval = "Hello There";*

*/\* Call the encrypt() method and store result of encryption. \*/*

*String encryptedval = encrypt(originalval);*

*/\* Call the decrypt() method and store result of decryption. \*/*

*String decryptedval = decrypt(encryptedval);*

*/\* Display the original message, encrypted message and decrypted message on the console. \*/*

*System.out.println("Original value: " + originalval);*

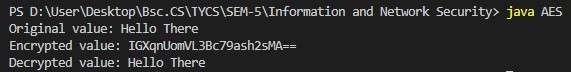
*System.out.println("Encrypted value: " + encryptedval);*

*System.out.println("Decrypted value: " + decryptedval);*

*}*

*}*

*Output:*

**

*Practical No.5:*

*Aim:*

*Write a program to implement RSA algorithm to perform encryption / decryption of a given string.*

*Code:*

*import java.math.BigInteger; import java.util.Random; import java.io.\*;*

*public class RSA { private BigInteger p; private BigInteger q; private BigInteger N; private BigInteger phi; private BigInteger e; private BigInteger d; private int bitlength = 1024; private int blocksize = 256; private Random r; public RSA() { r = new Random(); System.out.println("r"); System.out.println(r);*

1. *= BigInteger.probablePrime(bitlength, r);*

*System.out.println("p");*

*System.out.println(p);*

1. *= BigInteger.probablePrime(bitlength, r);*

*System.out.println("q");*

*System.out.println(q);*

*N = p.multiply(q);*

*System.out.println("N"); System.out.println(N);*

*phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));*

*System.out.println("phi"); System.out.println(phi);*

*e = BigInteger.probablePrime(bitlength/2, r);*

*System.out.println("e"); System.out.println(e);*

*while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0 ) { e.add(BigInteger.ONE);*

*}*

*d = e.modInverse(phi); System.out.println("d");*

*System.out.println(d);*

*}*

*public RSA(BigInteger e, BigInteger d, BigInteger N) { this.e = e; this.d = d;*

*this.N = N;*

*}*

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

*RSA rsa = new RSA();*

*DataInputStream in=new DataInputStream(System.in);*

*String teststring ;*

*System.out.println("Enter the plain text:");*

*teststring=in.readLine();*

*System.out.println("Encrypting String: " + teststring);*

*System.out.println("String in Bytes: " + bytesToString(teststring.getBytes()));*

*// encrypt*

*byte[] encrypted = rsa.encrypt(teststring.getBytes());*

*System.out.println("Encrypted String in Bytes: " + bytesToString(encrypted));*

*// decrypt*

*byte[] decrypted = rsa.decrypt(encrypted);*

*System.out.println("Decrypted String in Bytes: " + bytesToString(decrypted));*

*System.out.println("Decrypted String: " + new String(decrypted));*

*}*

*private static String bytesToString(byte[] encrypted) { String test = ""; for (byte b : encrypted) {*

*test += Byte.toString(b);*

*}*

*return test;*

*}*

*//Encrypt message*

*public byte[] encrypt(byte[] message) {*

*return (new BigInteger(message)).modPow(e, N).toByteArray();*

*}*

*// Decrypt message*

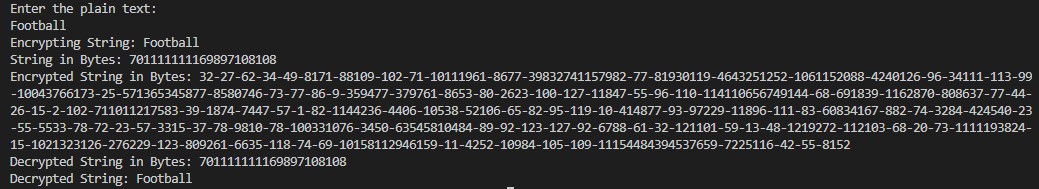
*public byte[] decrypt(byte[] message) {*

*return (new BigInteger(message)).modPow(d, N).toByteArray();*

*}*

*}*

*Output:*

**

*Practical No.6:*

*Aim:*

*Write a program to implement the Diffie-Hellman Key Agreement algorithm to generate symmetric keys.*

*Code:*

*import java.util.\*;*

*// create class DiffieHellman to calculate the key for two persons class DiffieHellman{ // main() method start*

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

*{*

*long P, G, x, a, y, b, ka, kb;*

*// create Scanner class object to take input from user*

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

*System.out.println("Both the users should be agreed upon the public keys G and P");*

*// take inputs for public keys from the user*

*System.out.println("Enter value for public key G:");*

*G = sc.nextLong();*

*System.out.println("Enter value for public key P:");*

*P = sc.nextLong();*

*// get input from user for private keys a and b selected by User1 and User2 System.out.println("Enter value for private key a selected by user1:"); a = sc.nextLong();*

*System.out.println("Enter value for private key b selected by user2:"); b = sc.nextLong();*

*// call calculatePower() method to generate x and y keys x = calculatePower(G, a, P); y = calculatePower(G, b, P);*

*// call calculatePower() method to generate ka and kb secret keys after the exchange of x and y keys*

*// calculate secret key for User1 ka = calculatePower(y, a, P); // calculate secret key for User2 kb = calculatePower(x, b, P);*

*// print secret keys of user1 and user2*

*System.out.println("Secret key for User1 is:" + ka);*

*System.out.println("Secret key for User2 is:" + kb);*

*}*

*// create calculatePower() method to find the value of x ^ y mod P private static long calculatePower(long x, long y, long P)*

*{*

*long result = 0; if (y == 1){*

*return x;*

*}*

*else{*

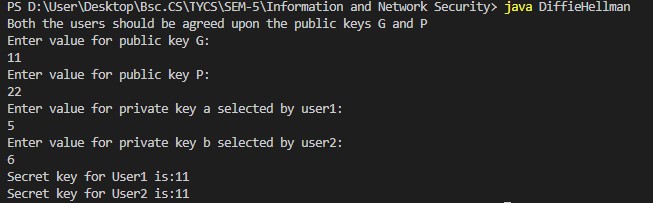
*result = ((long)Math.pow(x, y)) % P; return result;*

*}*

*}*

*}*

*Output:*

**

*Practical No.7:*

*Aim:*

*Write a program to implement the MD5 algorithm compute the message digest*

*Code:*

*import java.math.BigInteger; import java.security.MessageDigest; import java.security.NoSuchAlgorithmException; public class ins7 {*

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

*System.out.println("For null " + generateHash(""));*

*System.out.println("For simple text "+ generateHash("sies college."));*

*System.out.println("For simple text "+ generateHash("sies college"));*

*System.out.println("For simple numbers " + generateHash("12345"));*

*}*

*public static String generateHash(String input) { String md5 = null; if(null == input) return null;*

*try {*

*//Create MessageDigest object for MD5 or pass SHA-1*

*MessageDigest digest = MessageDigest.getInstance("MD5");*

*//Update input string in message digest*

*digest.update(input.getBytes(), 0, input.length()); //Converts message digest value in base 16 (hex) md5 = new BigInteger(1, digest.digest()).toString(16); } catch (NoSuchAlgorithmException e) { e.printStackTrace();*

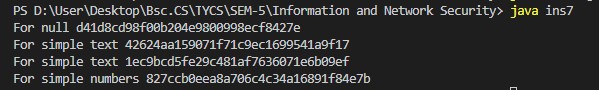
*}*

*return md5;*

*}*

*}*

*Output:*

**

*Practical No.8:*

*Aim:*

*Write a program to calculate HMAC-SHA1 Signature*

*Code:*

*import java.security.InvalidKeyException; import java.security.NoSuchAlgorithmException; import java.security.SignatureException; import java.util.Formatter; import javax.crypto.Mac;*

*import javax.crypto.spec.SecretKeySpec;*

*public class Ins8 {*

*private static final String HMAC\_SHA1\_ALGORITHM = "HmacSHA1";*

*private static String toHexString(byte[] bytes) {*

*Formatter formatter = new Formatter();*

*for (byte b : bytes) { formatter.format("%02x", b);*

*}*

*return formatter.toString();*

*}*

*public static String calculateRFC2104HMAC(String data, String key) throws SignatureException, NoSuchAlgorithmException, InvalidKeyException*

*{*

*SecretKeySpec signingKey = new SecretKeySpec(key.getBytes(), HMAC\_SHA1\_ALGORITHM);*

*Mac mac = Mac.getInstance(HMAC\_SHA1\_ALGORITHM); mac.init(signingKey);*

*return toHexString(mac.doFinal(data.getBytes()));*

*}*

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

*String hmac = calculateRFC2104HMAC("data", "key");*

*System.out.println(hmac);*

*assert hmac.equals("104152c5bfdca07bc633eebd46199f0255c9f49d");*

*}*

*}*

*Output:*

**

*Practical No.9:*

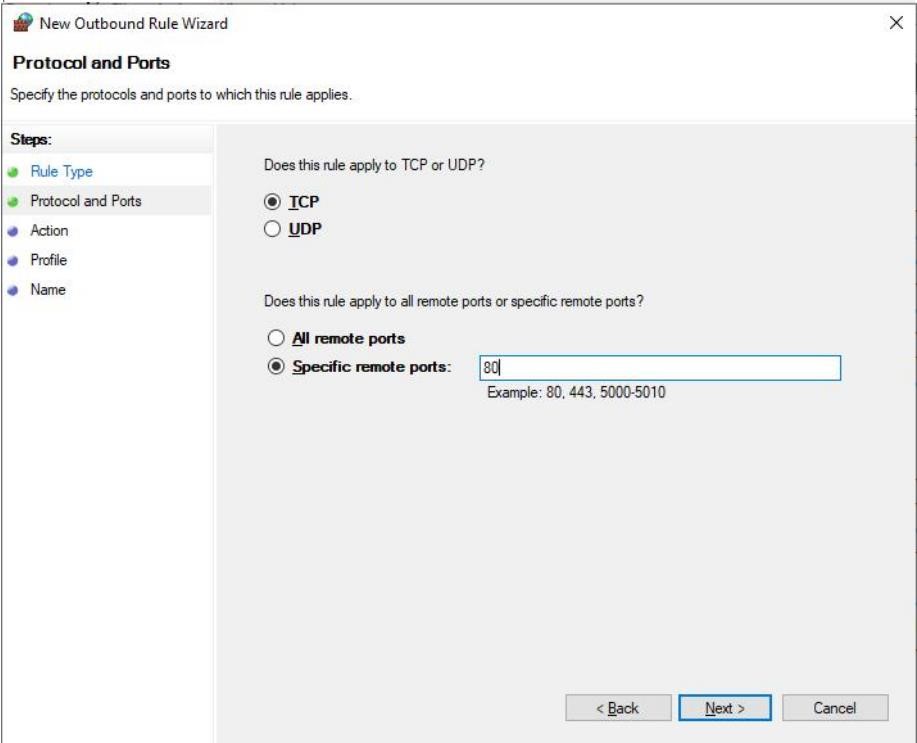
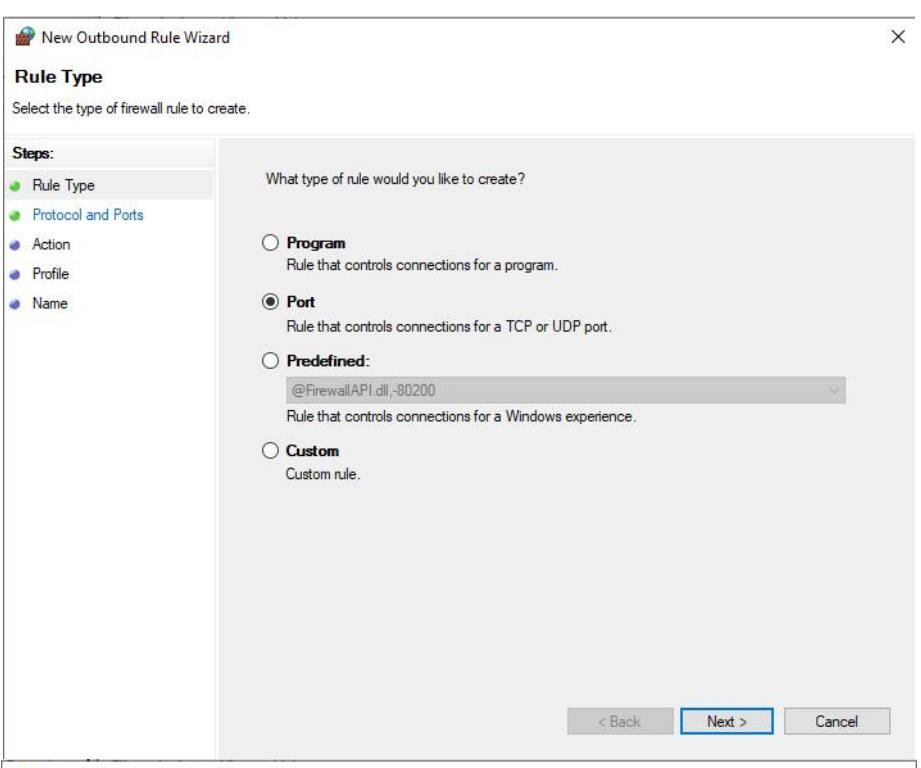
*Aim:*

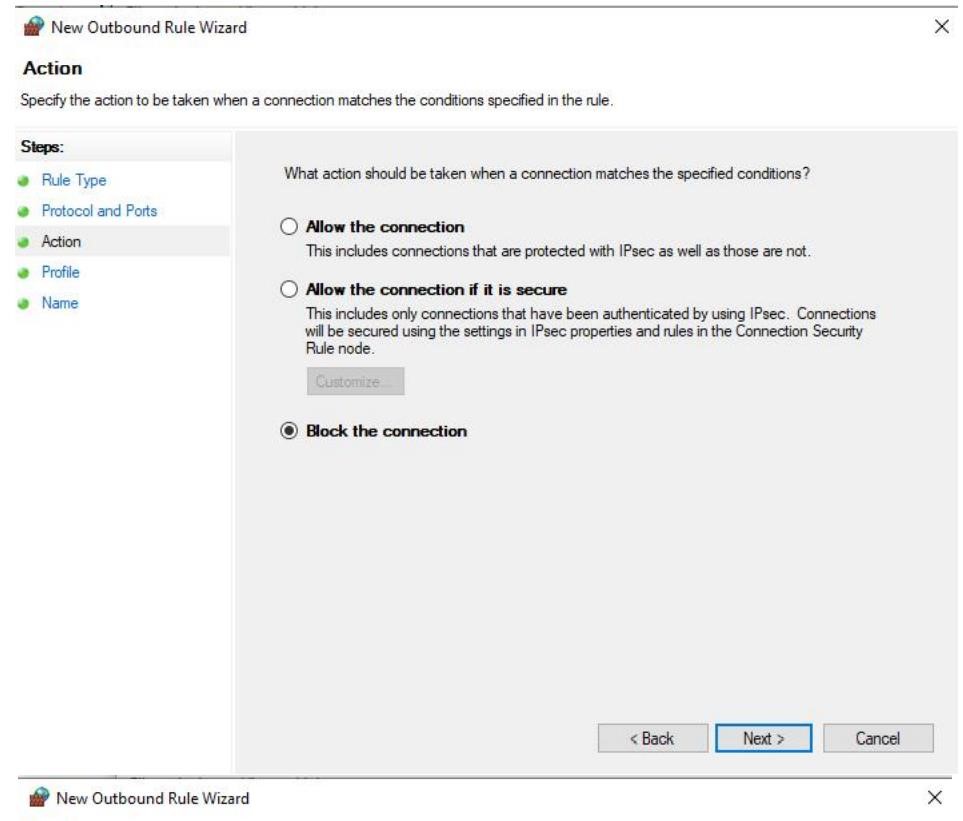
*Configure Windows Firewall to block*

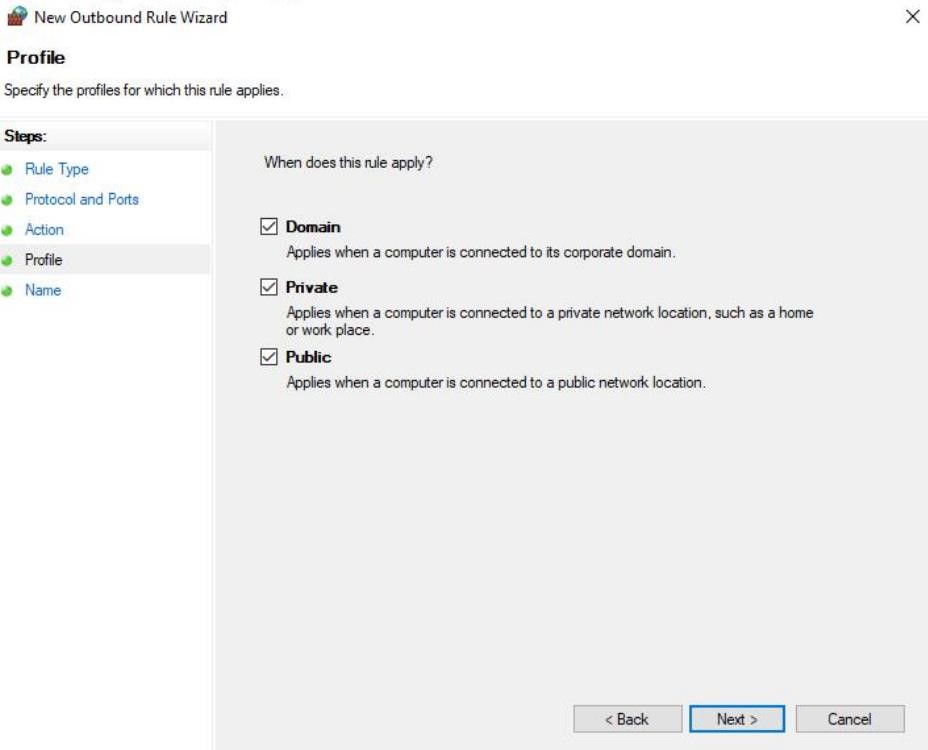
1. *A port*
2. *A program 3. A website*

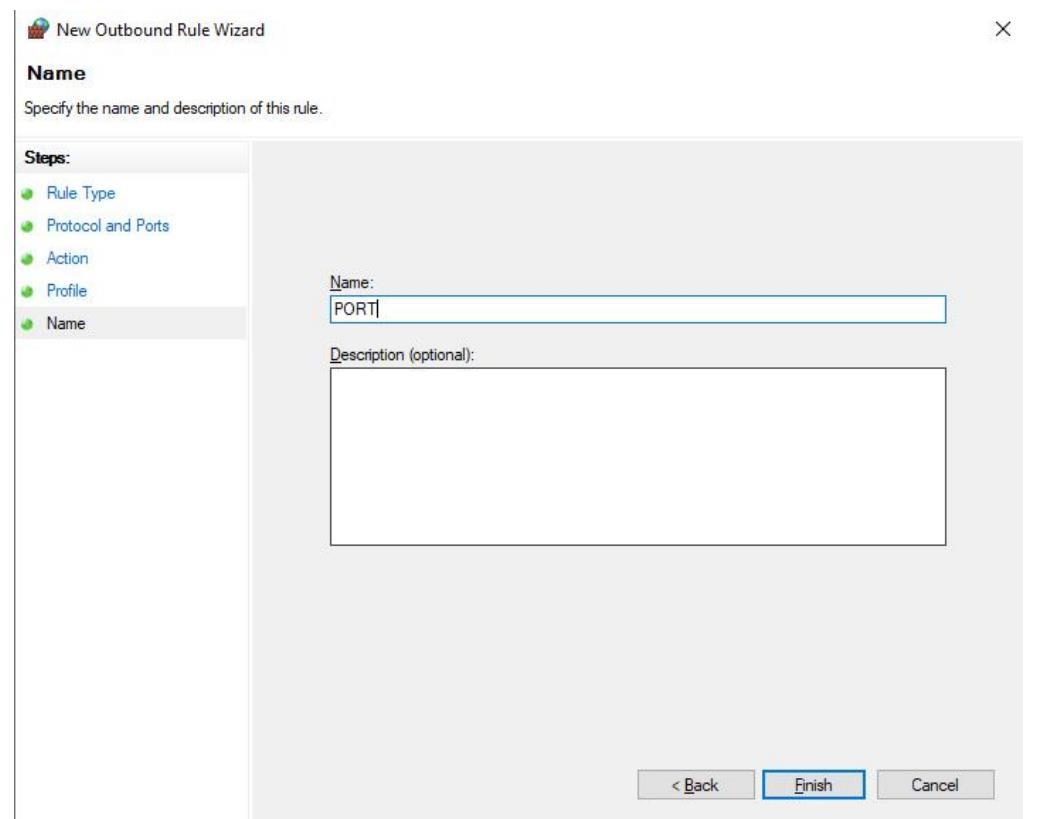
*Output:*

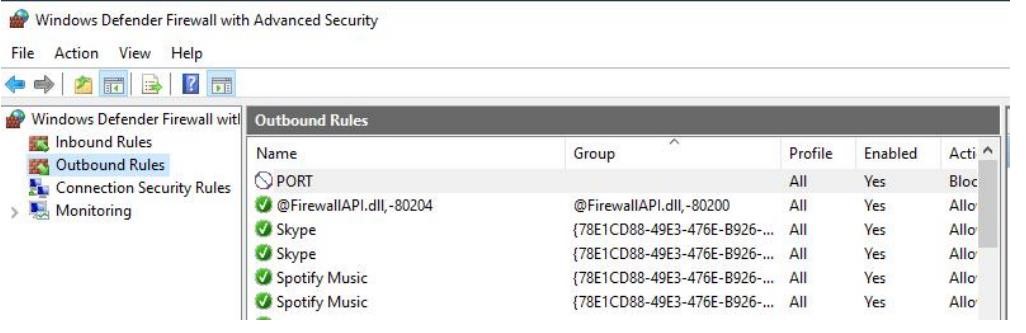
*A port:*

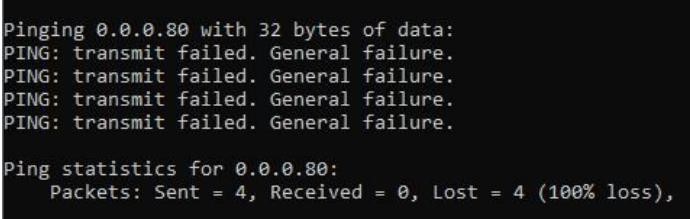


**

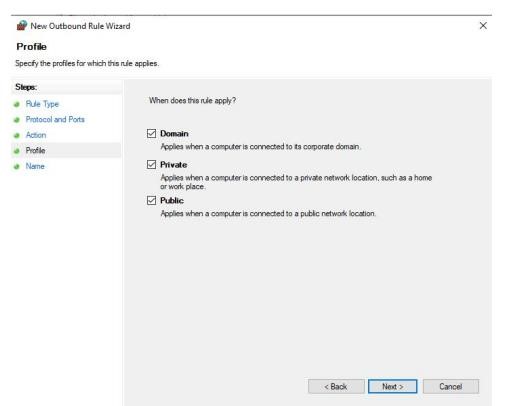
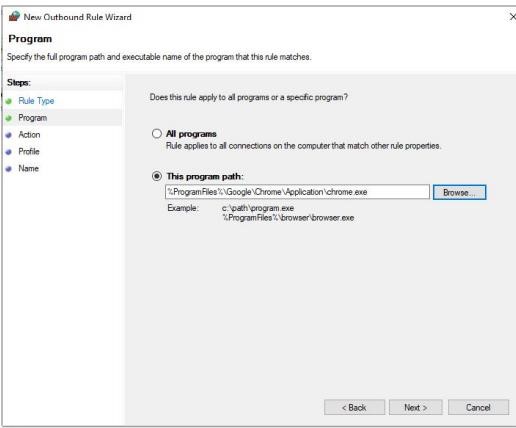
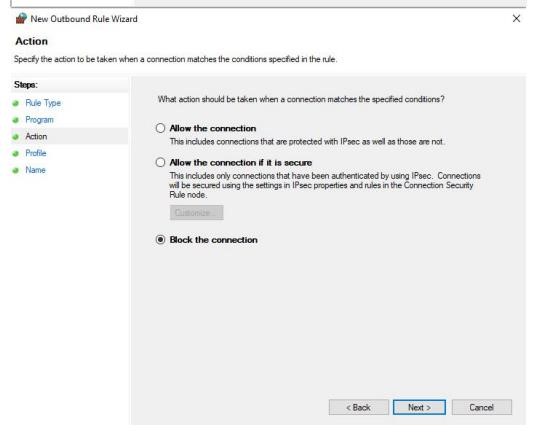
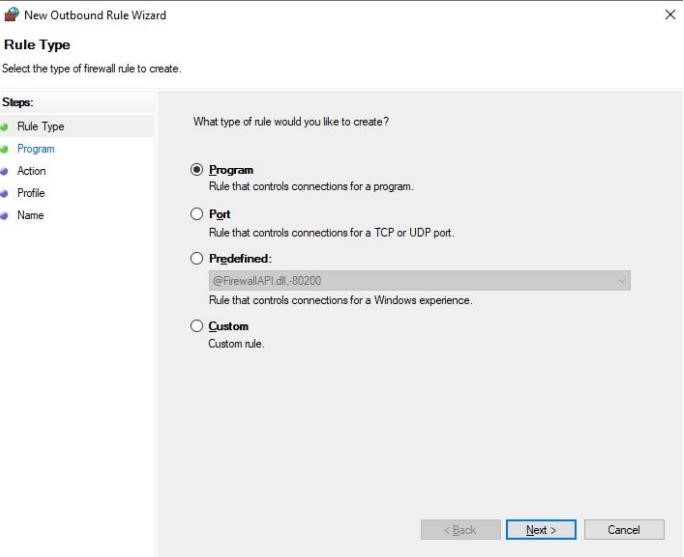
**

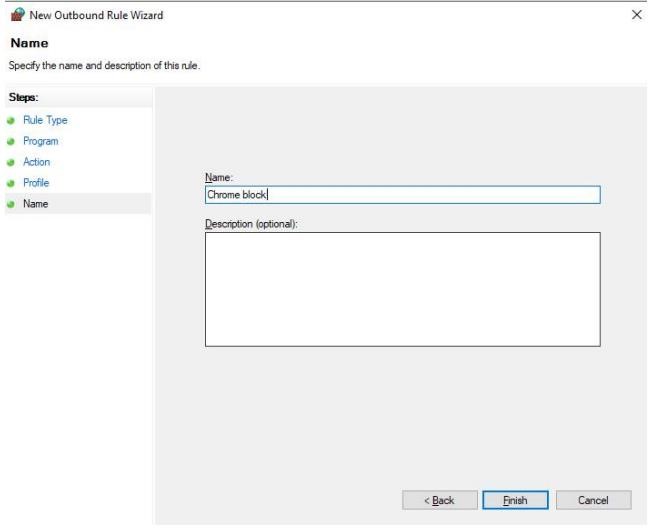
**

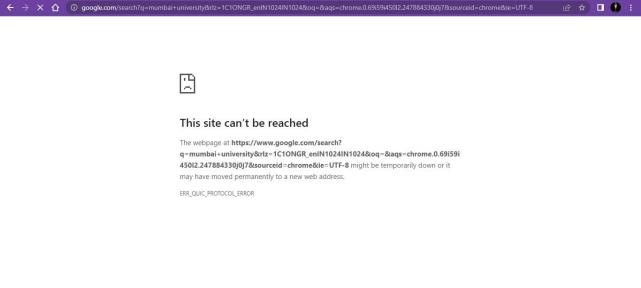
**

**

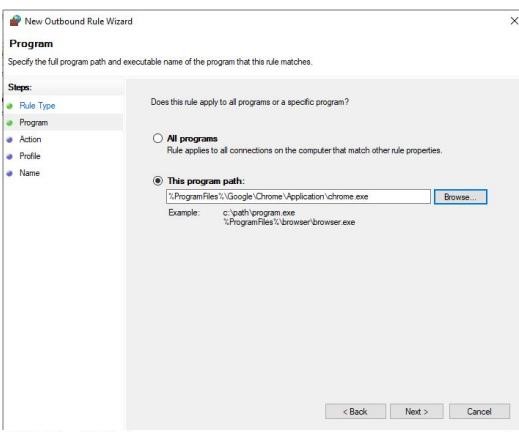
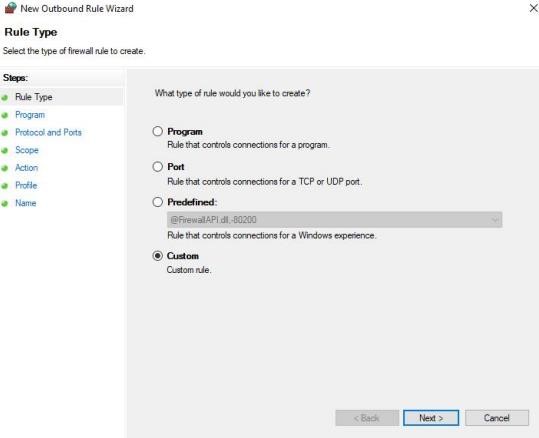
*A program:*

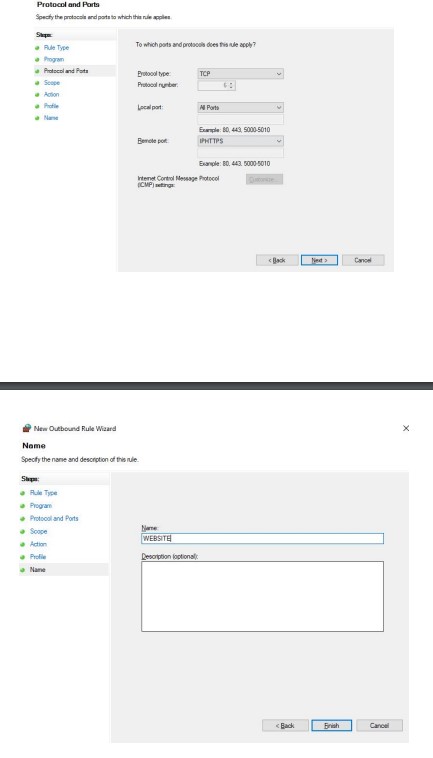


**

**

*A website:*



**

# *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*