Practical No – 01

Implementing Substitution and Transposition Ciphers

Aim : Design and implement algorithms to encrypt and decrypt messages using classical substitution and transposition techniques.

A. Caesar Cipher
B. Monoalphabetic Cipher
C. Rail Fence Cipher
D. Simple Columnar Technique
E. Vernam Cipher

Source Cod :

A. CaesarCipher.java

```
import java.util.Scanner;
class CeaserCipher {
   static String encryptCaeser(String message1, int key1){
      char ch;
      String encryptedMessage = " ";
      for ( int i = 0; i < message1.length();++i){
         ch = message1.charAt(i);
         if ( ch >= 'a' && ch <= 'z'){
            ch = (char) (ch + key1);
            if ( ch > 'z'){
               ch = (char) (ch - 'z' + 'a' - 1);
            }
            encryptedMessage += ch;
         } else if (ch >= 'A' && ch <= 'Z'){
            ch = (char) (ch + key1);
            if ( ch > 'Z'){
               ch = (char) ( ch - 'Z' + 'A' - 1);
            }
```

```
            encryptedMessage += ch;
        }else {
            encryptedMessage += ch;
        }
    }
    return encryptedMessage;
}
static String decryptCaesar(String message1, int key1){
    char ch;
    String decryptedMessage = " ";
    for ( int i = 0; i < message1.length(); ++i){
        ch = message1.charAt(i);
        if ( ch >= 'a' && ch <= 'z'){
            ch = (char) (ch - key1);
            if ( ch < 'a'){
                ch= (char) (ch + 'z' - 'a' + 1);
            }
            decryptedMessage += ch;
        } else if ( ch >= 'A' && ch <= 'Z'){
            ch = (char) (ch - key1);
            if ( ch < 'A'){
                ch = (char) ( ch + 'Z' - 'A' + 1);
            }
            decryptedMessage += ch;
        }
        else {
            decryptedMessage += ch;
        }
    }
    return decryptedMessage;
}
public static void main(String[] args) {
```

```
        String plainText;
        int key;
        String CipherText;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a message to encrypt: ");
        plainText = sc.nextLine();
        System.out.println("Enter key: ");
        key = sc.nextInt();
        CipherText = encryptCaeser(plainText, key);
        System.out.println("Cipher Text= " + CipherText);
        System.out.println("Original Text=" + decryptCaesar(CipherText, key));
    }
}
```

Output:

```
run:
Enter a message to encrypt:
roy
Enter key:
2
Cipher Text=  tqa
Original Text=  roy
BUILD SUCCESSFUL (total time: 7 seconds)
```

Source Code:

B. MonoalphabeticCipher.java

```
import java.util.Scanner;
public class MonoalphabetCipher {
   public static void main(String[] args) {
      final char RALPHABETS[] = {'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'};
      final char MALPHABETS[] = {'M','U','A','L','V','O','Z','K','R','N','J','X','Q','D','F','S','H','P','E','B','C','T','I','W','Y','G'};
     Scanner sc = new Scanner(System.in);
      String pltext;
      char citext[] = new char[20];
      char detext[] = new char[20];
      int i,l;
      System.out.print("Enter Plain Text: ");
      pltext = sc.nextLine();
      pltext = pltext.toLowerCase();
      l = (pltext.length());
      for ( i=0;i< l;i++){
         for (int j = 0; j < 26; j++){
            if (RALPHABETS[j] == pltext.charAt(i)){
               citext[i] = MALPHABETS[j];
               break;
            }
         }
      }
      System.out.print("Cipher Text : ");
      for ( i = 0; i < l;i++){
         System.out.print(citext[i]);
      }
      String b = new String(citext);
      for ( i = 0; i < l; i++){
         for (int  j = 0; j < 26; j++){
```

```
            if (MALPHABETS[j]== b.charAt(i)){
                detext[i] = RALPHABETS[j];
                break;
            }
        }
    }
    System.out.println("\nPlain Text :\n");
    for ( i = 0;i < l;i++){
        System.out.print(detext[i]);
    } }}
```

Output:

```
run:
Enter Plain Text: roy
Cipher Text : PFY
Plain Text:
roy
BUILD SUCCESSFUL (total time: 2 seconds)
```

Source Code:

C. Rail Fence Cipher

```
import java.util.*;
class RailFenceB {
   String Encryption (String plainText, int depth) throws Exception{
      int r = depth, len = plainText.length();
      int c = len/depth;
      c= c+1;
      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++);
               System.out.println("mat["+j+"]["+i+"]= "+mat[j][i]);
            }
         }
      }
      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) throws Exception{
      int r = depth, len = cipherText.length();
      int c = len/depth;
      char mat[][]= new char[r][c];
      int k = 0;
```

```
        String plaiText = "";
        for (int i = 0; i< r;i++){
            for(int j =0;j < c;j++){
                mat[i][j] = cipherText.charAt(k++);
                System.out.println("mat["+i+"]["+j+"]= "+mat[i][j]);
            }
        }
        for(int i = 0; i < c;i++){
            for(int j=0;j<r;j++){
                plaiText += mat[j][i];
            }
        }
        return plaiText;
    }
}
class Railfence {
    public static void main(String[] args) throws Exception{
        Scanner sc = new Scanner(System.in);
        int depth;

        String plainText, cipherText, decryptedText;

        System.out.println("Enter Plain text: ");
        plainText = sc.nextLine();

        System.out.println("Enter depth(No of Rails) for Encryption: ");
        depth = sc.nextInt();

        RailFenceB rf = new RailFenceB();
        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);
    }
}
```

Output:

```
run:
Enter Plain text:
hidethebox
Enter depth(No of Rails) for Encryption:
2
mat[0][0]= h
mat[1][0]= i
mat[0][1]= d
mat[1][1]= e
mat[0][2]= t
mat[1][2]= h
mat[0][3]= e
mat[1][3]= b
mat[0][4]= o
mat[1][4]= x
Decrypted Text Is :
hidethebox
BUILD SUCCESSFUL (total time: 11 seconds)
```

Source code:

D. Simple Columnar Technique

```
 import java.io.*;
import java.util.Scanner;
class SCT {
   public static void main(String[] args) throws Exception{
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter Your Plain Text: ");

      String accept = sc.nextLine();
      System.out.println("Enter the no of rows ");
      int r = Integer.parseInt(sc.nextLine());
      System.out.println("Enter the no of cols ");
      int c = Integer.parseInt(sc.nextLine());

      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]=' ';
            }else{
               cont[i][j]=accept.charAt(count);
                System.out.println("cont[][]:"+(accept.charAt(count)));
               count++;
            }
         }
      }
```

```
    for(int i=0; i< r; i++){
      for(int j = 0; j < c;j++){
        System.out.print("\t" + cont[i][j]);
      }
      System.out.print("\n");
    }

    System.out.println("\nEnter the order of cols you want to view there ");
    int choice[]= new int[c];
    for( int k=0;k<c;k++){
      System.out.println("Choice "+ k+ "->");
      choice[k] = Integer.parseInt(sc.nextLine());
    }

    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];
      }
    }
    System.out.println(cipher);
  }
}
```

Output:

```
run:
Enter Your Plain Text:
information
Enter the no of rows
4
Enter the no of cols
3
cont[][]:i
cont[][]:n
cont[][]:f
cont[][]:o
cont[][]:r
cont[][]:m
cont[][]:a
cont[][]:t
cont[][]:i
cont[][]:o
cont[][]:n
        i       n       f
        o       r       m
        a       t       i
        o       n
Enter the order of cols you want to view there
Choice 0->
2
Choice 1->
0
Choice 2->
1
Cipher text in matrix is ->
fmi ioaonrtn
BUILD SUCCESSFUL (total time: 52 seconds)
```

Source code:

E. Vernam Cipher

```
import java.lang.Math;
public class Vernam {
   public static void main(String[] args) {
      String Plaintext = new String("computercian");
      char[] arText = Plaintext.toCharArray();
      String key = new String("ABCDEFGXYZHIJ");
      char[] arKey = key.toCharArray();
      char[] Ciphertext = new char[13];
      System.out.println("Encoded"+ Plaintext + " to be... ");
       for(int i =0; i<arText.length;i++){
         Ciphertext[i] =(char) (arText[i] ^ arKey[i]);
         System.out.println(Ciphertext[i]);
      }
      System.out.println("Decoded to be... ");
      for(int i = 0; i<Ciphertext.length;i++){
         char temp = (char)(Ciphertext[i] ^ arKey[i]);
         System.out.print(temp);
      }
   }
}
```

Output:

```
run:
Encodedcomputercian to be...
"-.402"*:3)'Decoded to be...
computercianJBUILD SUCCESSFUL (total time: 0 seconds)
```

Practical No – 02
RSA Encryption and Decryption

Aim: Implement the RSA algorithm for public-key encryption and decryption, and explore its properties and security considerations.

Source code:

```
import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.*;
public class RSA {
   public static void main(String[] args) {
      int p,q,n,z,d = 0, e,i;
      double c;
      BigInteger msgback;
      p = 5;
      q = 11;
      int msg = 12;
      n= p * q;
      z= (p - 1) * (q - 1);
      System.out.println("The value of z = "+ z);
      for(e = 2;e < z; e++){
         if(gcd(e,z) == 1){
            break;
         }
      }
      System.out.println("The Value Of e ="+e);
      for(i = 0;i <= 9; i++){
         int x = 1+(i * z);
         if (x % e == 0){
            d = x/e;
            break;
```

```
        }
      }
      System.out.println("The value of d = "+ d);
      c = (Math.pow(msg, e)) % n;
      System.out.println("Encrypted message is : "+ c);

      BigInteger N = BigInteger.valueOf(n);
      BigInteger C = BigDecimal.valueOf(c).toBigInteger();
      msgback = (C.pow(d)).mod(N);
      System.out.println("decrypted message is :" + msgback);
   }
   static int gcd(int e, int z){
      if (e == 0){
         return z;
      } else{
         return gcd(z%e,e);
      }
   }
}
```

Output:

```
run:
The value of z = 40
The Value Of e =3
The value of d = 27
Encrypted message is : 23.0
decrypted message is :12
BUILD SUCCESSFUL (total time: 0 seconds)
```

Practical No – 03
Message Authentication Codes

Aim : Implement algorithms to generate and verify message authentication codes (MACs) for ensuring data integrity and authenticity.

Source code:

```
import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
public class MD5 {
      public static String toHexString(byte[] hash) {
      BigInteger number = new BigInteger(1, hash);
      StringBuilder hexString = new StringBuilder(number.toString(16));
      while (hexString.length() > 32) {
         hexString.insert(0, '0');
      }
      return hexString.toString(); }
   public static void main(String args[])throws NoSuchAlgorithmException
   {
      System.out.println("Hashcode Generated by MD5 for:");
         String s1= "Information and security";
         MessageDigest md=MessageDigest.getInstance("MD5");
         byte[] hash=md.digest(s1.getBytes(StandardCharsets.UTF_8));
         System.out.println("Message Digest: "+s1+":"+toHexString(hash));
    }}
```

Output**:**

```
run:
Hashcode Generated by MD5 for:
Message Digest: Information and security:c971095f58c8c7aa2aba10c9f61ebd82
BUILD SUCCESSFUL (total time: 0 seconds)
```

Source code:

```
import hashlib
result=hashlib.md5(b'good')
result=result.hexdigest()
print('Message Digest', result)
```

Output:

```
Message Digest 755f85c2723bb39381c7379a604160d8
```

Source code:

```
import hashlib
str = input('Enter String to encode :')
result = hashlib.sha1(str.encode())
result = result.hexdigest()
print("Output of SHA1 ", result)
```

Output:

```
Enter String to encode :hello
Output of SHA1  aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
```

Source code:

```
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA256
from Crypto.PublicKey import RSA
from Crypto import Random
def generate_signature(private_key,message):
    key = RSA.importKey(private_key)
    hashed_message = SHA256.new(message.encode('utf-8'))
    signer = PKCS1_v1_5.new(key)
    signature = signer.sign(hashed_message)
    return signature
def verify_signature(public_key,message,signature):
    key  = RSA.importKey(public_key)
    hashed_message = SHA256.new(message.encode('utf-8'))
    verifier = PKCS1_v1_5.new(key)
    return verifier.verify(hashed_message,signature)

random_generator = Random.new().read
key_pair = RSA.generate(2048,random_generator)
public_key = key_pair.publickey().export_key()
private_key = key_pair.export_key()
message = "Hello world!"
signature = generate_signature(private_key, message)
print("Generated Signature: ", signature)
is_valid = verify_signature(public_key, message, signature)
print("Signature Verification Result:", is_valid)
```

Output:

Generated Signature:
b"w6\xdf\x95\x18\xd9\x98\xf5)I\x9d\xc6\x0e&\n<\x0c\xa7O\xcd\xc1\x0f\x0c\xf7\x02\xa3\x17%)\x12\x96!Bn\xa2\x83\x88S\x02\xfd-
\xee\x1a|\xbc\xf5\xab\xfd\xcb\xe3$\xff\xa1\xea)\x84b\xe9\xf1!\r\xb7\x17\x15\xe9\xef\xe7\xd7\x048Y\xdbf\xda\xbe\x9aMt\xdf\xc5\x8d\xd5\xcc\x1d|[\xe2>'\xad\x82\xb1\xd3'\xc4\xabI\xbd9\x94\x03O\xe6\xffz\xfc\xe0\xb9\xdbV\x143\xc97^f,j\xc2\xbb`\x07\x99s{\xd4$\xa9&\x8c_U\x8e\xb3T4\xeb\xdf0\x99\x93\xbd\xc2\x98\x9b4\x17\x81\x99\xf9\x07\x89?\xaa\x99Z\n\xa8\xaa\x03X%\xbe\x8d\xee\xb1\xfbT\xdaD\x00\x86\xa2^\x8f\xecL!\xc9\xca<\x17\x1f\xe2\xd0\x8bd,i\xe3\x0c\xec\xcb\x11\x06\x92a\xa0\xe67\xfbLJ\x1c\xc1\xcb\x8a\xb8\xcck\x02W~\x9a\x9b*\xf7\xaf2\xa4\xee)2\xa4\xca\xb6P\x98\xe2q\x83\x84\xed\xc3k\xc5{h5:\xc0n\xb8\x9c\x1b\x00\x91\xf4\x03\xd0\xd1C\x86\xc05"

Signature Verification Result: True

Practical No – 04
Digital Signatures

Aim : Implement digital signature algorithms such as RSA-based signatures, and verify the integrity and authenticity of digitally signed messages.

Source Code:

```
import java.security.PrivateKey;
import java.security.*;
import java.util.Scanner;
import javax.xml.bind.DatatypeConverter;
public class Digital_Signature {
   private static final String SIGNING_ALGORITHM = "SHA256withRSA";
   private static final String RSA = "RSA";
   private static Scanner sc;
   public static byte [] Create_Digital_Signature(byte[] input,PrivateKey key) throws Exception{
      Signature signature = Signature.getInstance(SIGNING_ALGORITHM);
      signature.initSign(key);
      signature.update(input);
      return signature.sign();
   }
   public static KeyPair Generate_RSA_KeyPair() throws Exception{
      SecureRandom secureRandom = new SecureRandom();
      KeyPairGenerator keypairGenerator = KeyPairGenerator.getInstance(RSA);
      keypairGenerator.initialize(2048 , secureRandom);
      return keypairGenerator.generateKeyPair();
   }
   public static boolean Verify_Digital_Signature (byte[] input, byte[] signatureToVerify, PublicKey key)throws
Exception{
      Signature signature = Signature.getInstance(SIGNING_ALGORITHM);
      signature.initVerify(key);
      signature.update(input);
```

```
        return signature.verify(signatureToVerify);
    }
    public static void main(String[] args) throws Exception {
        String input = "Good Morning";
        KeyPair keypair = Generate_RSA_KeyPair();
        byte[] signature = Create_Digital_Signature(input.getBytes(),keypair.getPrivate());
        System.out.println("Signature Value:\n " + DatatypeConverter.printHexBinary(signature));
        System.out.println("Verification : " + Verify_Digital_Signature(input.getBytes(), signature, keypair.getPublic()));
    }
}
```

Output:

```
run:
Signature Value:

7535798513CAE97D7B25C04A5B669B602FB70E5721CBB1FD4723A4A4B9D7A556C6991F020A527B96041F1647F7A
053CD7DECFF5177BE0DE417E89500056B2B1E86C3567B2033DCBF86C9EFF885F8276998C2F5B7ADC8B7A6B781D0D
9DD1B5880D77831E2EAD9D1746878860C880B055CEA03A8183D5D50E6E09D6EF6B04879BE9089ED701769107955
2B5DAD90880302A4791F0F2D571C88542316A514860481F0A925C59416D5402D935E5E220345349973B7C53A714
A122AB4D03E5714D6C41130EC8FF4E8988B2777340E1BB384613DDA485ACAA7191B250D837B88A3EE6AB60F3663
A482B7DBBF3AA9DCE17311A8CC0C8F0A452E3949724E981C8619DACF

Verification : true

BUILD SUCCESSFUL (total time: 0 seconds)
```

Practical No – 05
Key Exchange using Diffie-Hellman

Aim : Implement the Diffie-Hellman key exchange algorithm to securely exchange keys between two entities over an insecure network.

Source code:

```
import java.util.*;

public class DH {

   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);

      System.out.println("Enter Prime Number 1 p: ");

      int p = sc.nextInt();

      System.out.println("Enter Prime Number 2 g: ");

      int g = sc.nextInt();

      System.out.println("Choose 1st secret number (Alice) 'a' : ");

      int a = sc.nextInt();

      System.out.println("Choose 2nd secret number (Bob) 'b' : ");

      int b = sc.nextInt();


      int A  = (int) Math.pow(g, a) % p;

      int B = (int) Math.pow(g, b) % p;

      System.out.println("Pubkic key of Alice:  " + A);

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


      int S_A = (int) Math.pow(B, a) % p;

      int S_B = (int) Math.pow(A, b) % p;


      System.out.println("Shared key of Alice S_A : "+ S_A);

      System.out.println("Shared key of Bob S_B: "+ S_B);


      if(S_A == S_B){
```

```
        System.out.println("Alice and Bob can communicate "+ "with each other");
        System.out.println("They share a secret no = "+ S_A);
    } else{
        System.out.println("Alice and Bob cannot"+ " communicate with each other!!!");
    }
}
}
```

Output:

```
run:
Enter Prime Number 1 p:
23
Enter Prime Number 2 g:
11
Choose 1st secret number (Alice) 'a' :
6
Choose 2nd secret number (Bob) 'b' :
5
Pubkic key of Alice:  9
Public key of Bob : 5
Shared key of Alice S_A : 8
Shared key of Bob S_B: 8
Alice and Bob can communicate with each other
They share a secret no = 8
BUILD SUCCESSFUL (total time: 29 seconds)
```

Practical No – 06
IP Security (IPsec) Configuration:

Aim : To Configure IPSec on network devices to provide secure communication and protect against unauthorized access and attacks.

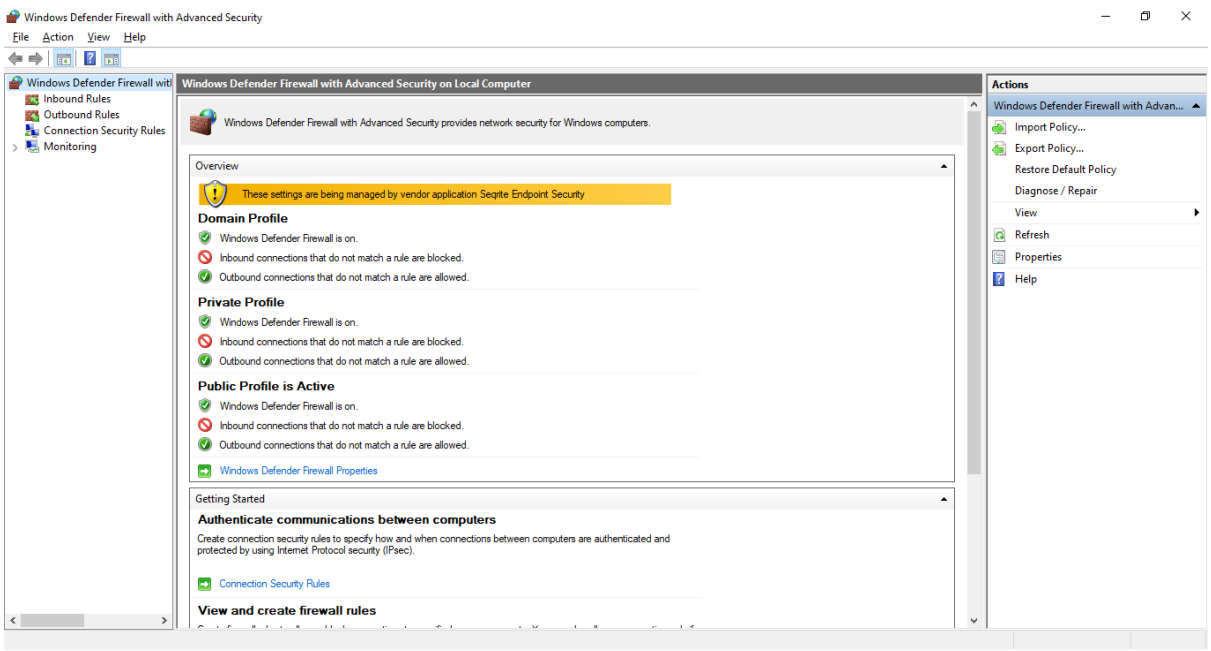
About this task

Historian supports encryption based on Internet Protocol Security to secure traffic between various Historian components and collectors without the need to use VPN or other security protocols.

Procedure

1. Run wf.msc.

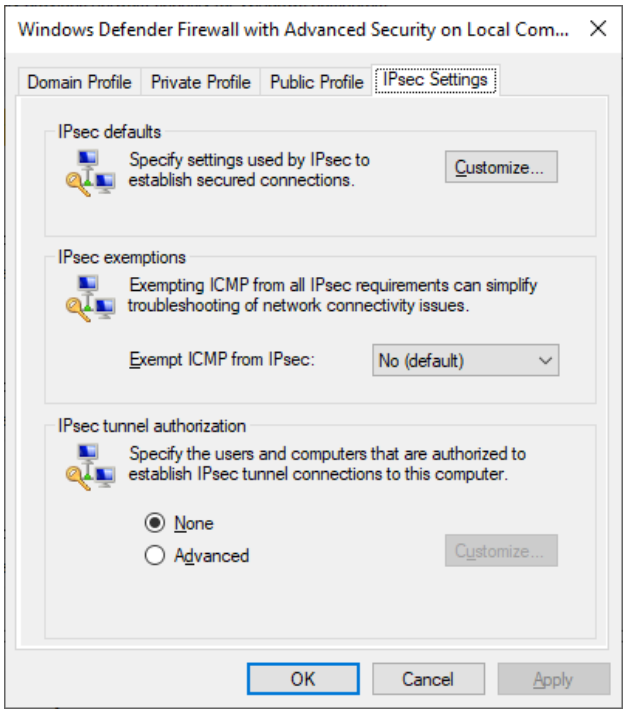
   The **Windows Defender Firewall with Advanced Security** window appears.



2. Create a security method:

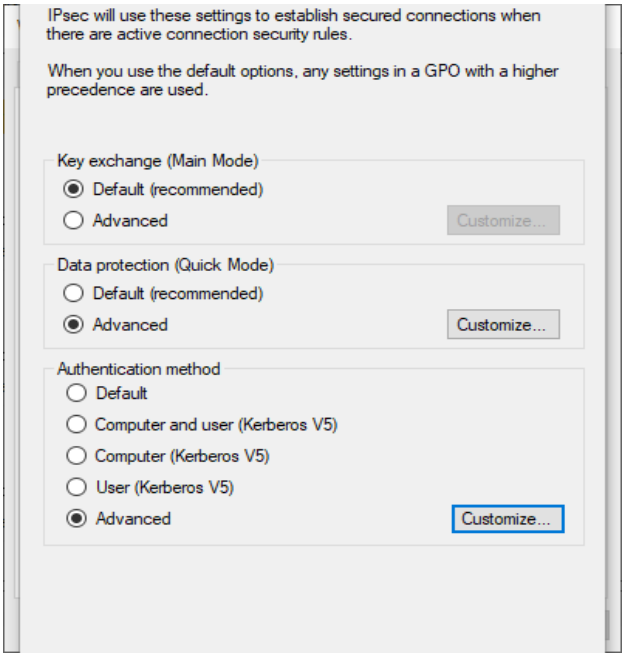
   a. Select **Actions** > **Properties**.

      The **Windows Defender Firewall with Advanced Security on Local Computer** window appears



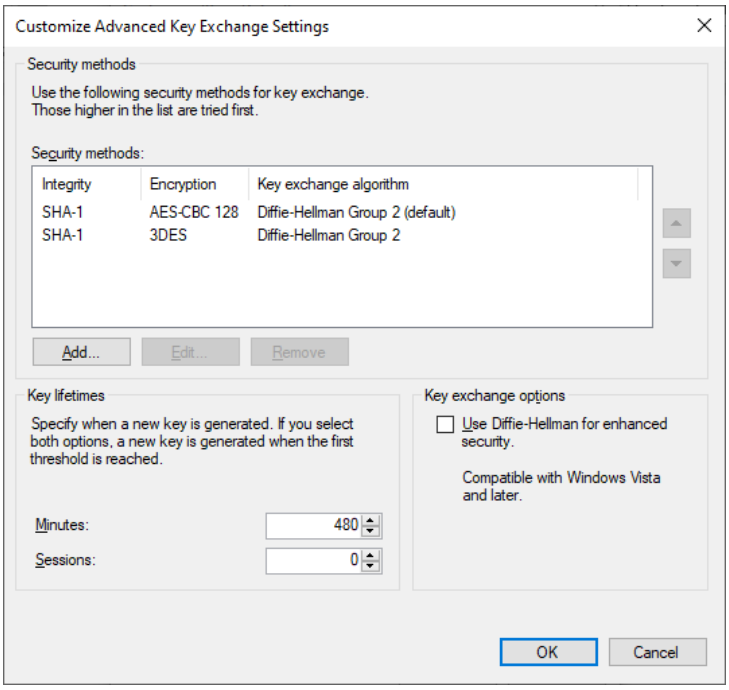
b. Select **IPsec Settings** > **Customize**.

The **IPsec Defaults** window appears.

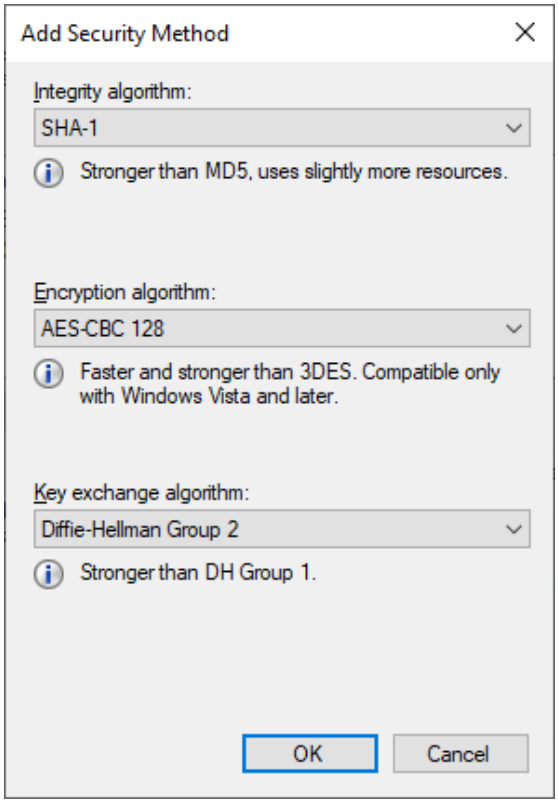


c. Under **Key exchange (Main Mode)**, select **Advanced** > **Customize**.

The **Customize Advanced Key Exchange Settings** window appears.

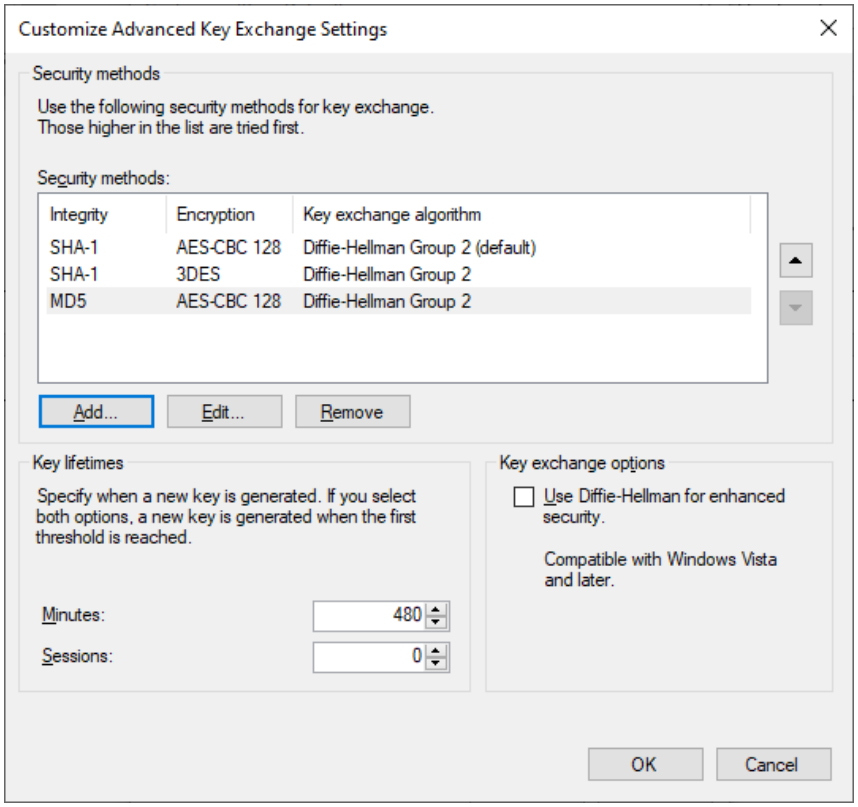


d. Select **Add**. The **Add Security Method** window appears.

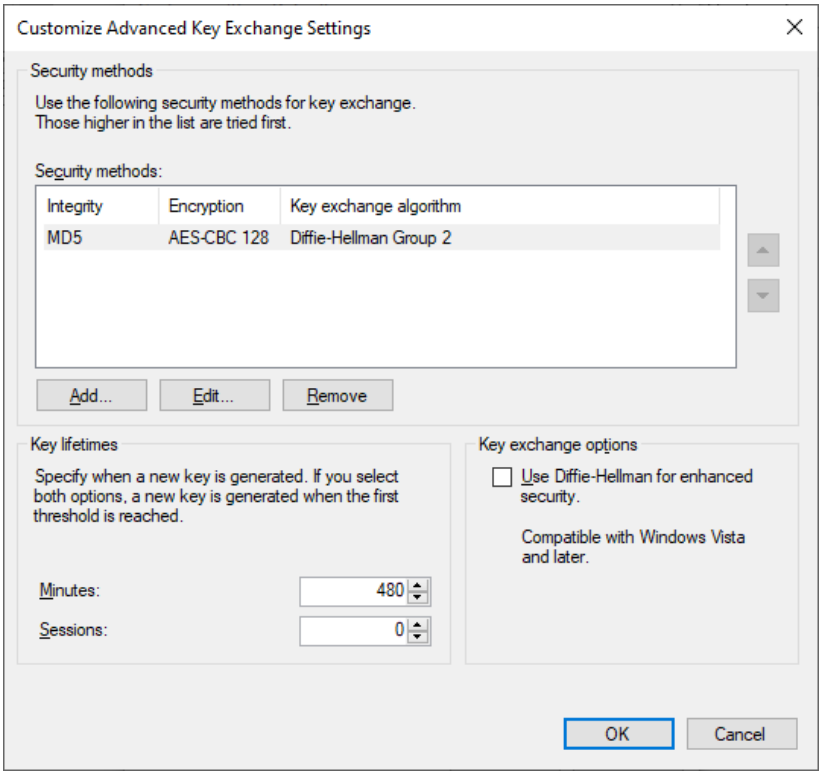


e. Select the algorithms that you want to use for each purpose.

The security method that you have added appears in the list



f. Move the security method that you have added to the top of the list. We recommend that you remove the other methods.

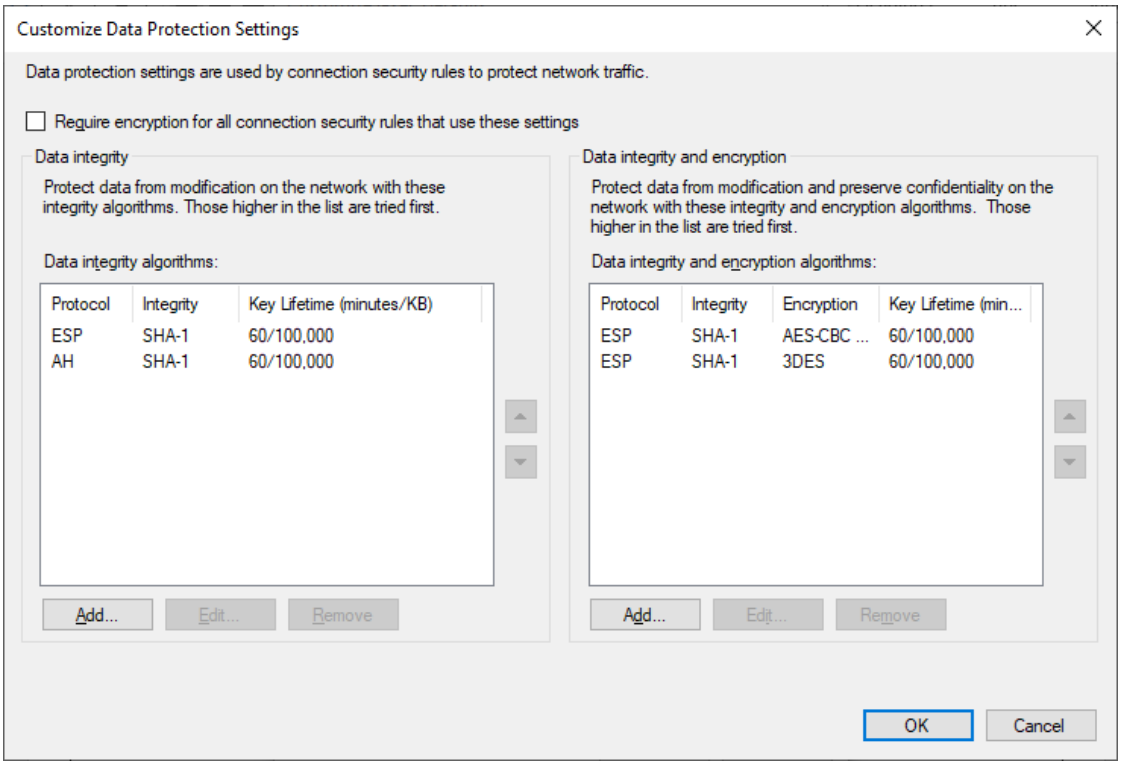


g. Select **OK**.

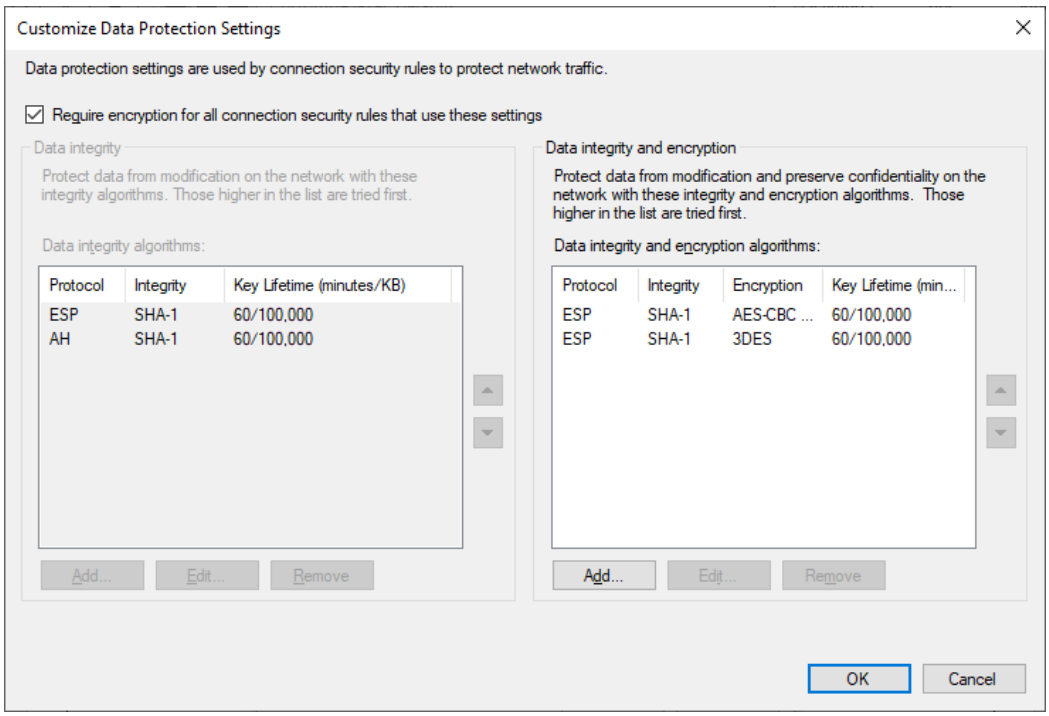
3. Add integrity and encryption algorithms:

   a. In the **Customize IPsec Defaults** window, under **Data protection (Quick Mode)**, select **Advanced** > **Customize**.

      The **Customize Data Protection Settings** window appears.

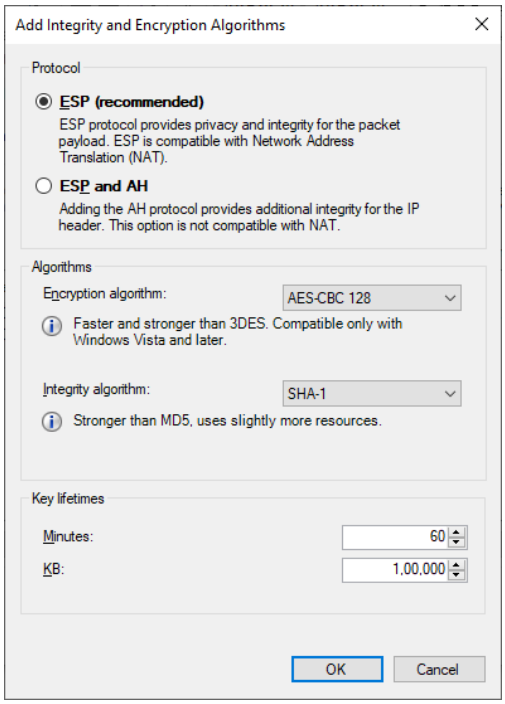


   b. Select the **Require encryption for all connection and security rules that use these settings** check box.



c. Under **Data integrity and encryption**, select **Add**.

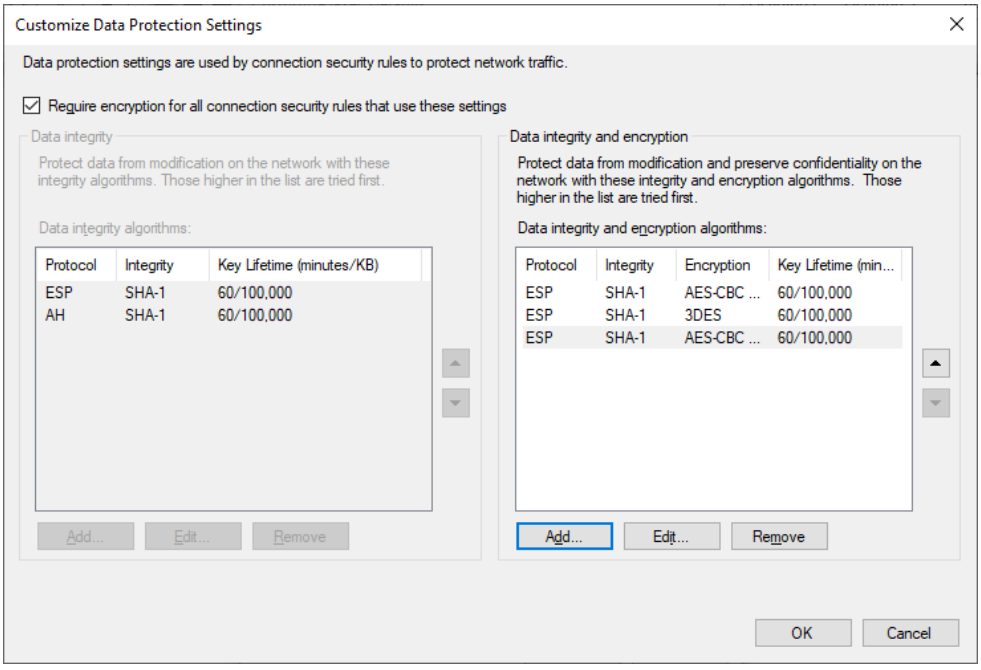
The **Add Integrity and Encryption Algorithms** window appears.



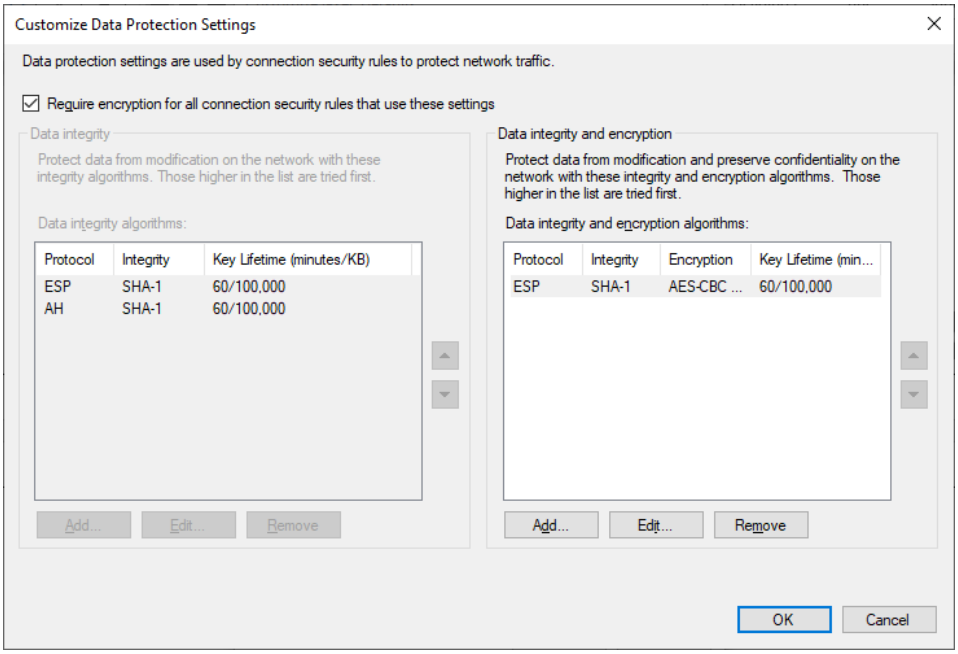
d. Under **Protocol**, ensure that **ESP** is selected.

e. Select the algorithms that you want to use for each purpose, and then select **OK**.

The algorithms that you have selected appear in the list.



f. Move the algorithms to the top of the list. We recommend that you remove the remaining items in the list.



g. Select **OK**.

Practical No – 07
Malware Analysis and Detection

Aim : Analyze and identify malware samples using antivirus tools, analyze their behavior, and develop countermeasures to mitigate their impact.