# Basic Theory of Cryptography

## 1.1 Introduction

Cryptography refers to the science and art of transforming messages to make them secure and immune to attacks. It is a method of storing and transmitting data in a particular form so that only those for whom it is intended can read and process it. Cryptography not only protects data from theft or alteration but can also be used for user authentication.

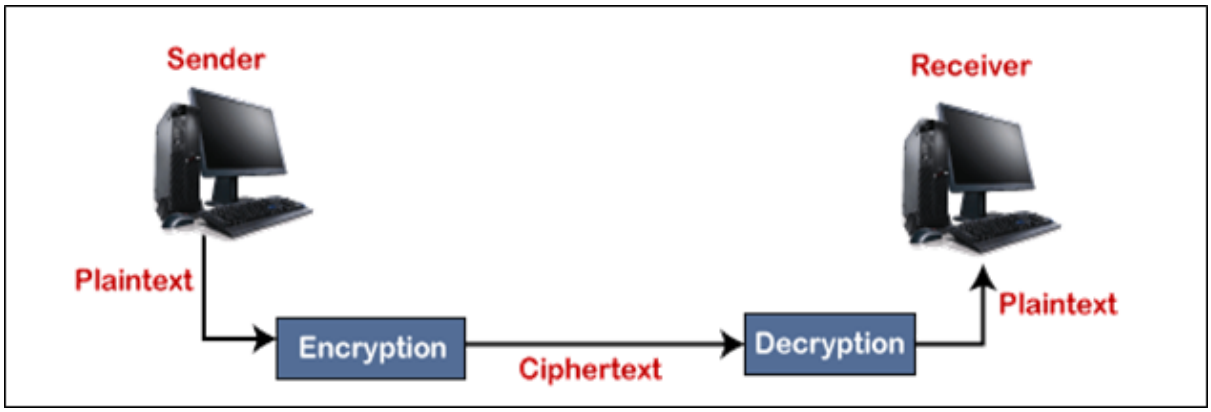



Fig.1.1: Cryptography process

## 1.2 Components

There are various components of cryptography which are as follows:

### Plaintext and Ciphertext

The original message, before being transformed, is called plaintext. After the message is transformed, it is called ciphertext. An encryption algorithm transforms the plaintext into ciphertext; a decryption algorithm transforms the ciphertext back into plaintext. The sender uses an encryption algorithm, and the receiver uses a decryption algorithm.

### Cipher

We refer to encryption and decryption algorithms as ciphers. The term cipher is also used to refer to different categories of algorithms in cryptography. This is not to say that every sender-receiver pair needs their very own unique cipher for secure communication. On the contrary, one cipher can serve millions of communicating pairs.

### Key

A key is a number (or a set of numbers) that the cipher, as an algorithm, operates on. To encrypt a message, we need an encryption algorithm, an encryption key, and plaintext. These create the ciphertext. To decrypt a message, we need a decryption algorithm, a decryption key, and the ciphertext. These reveal the original plaintext.

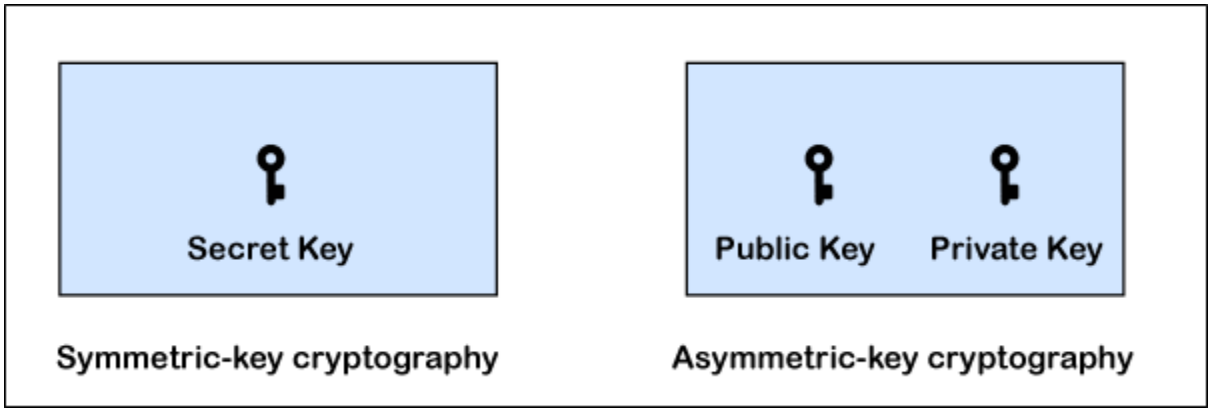



Fig.1.2: Types of cryptography key

## 1.3 There are two types of Cryptography

- Symmetric key cryptography
- Asymmetric key cryptography

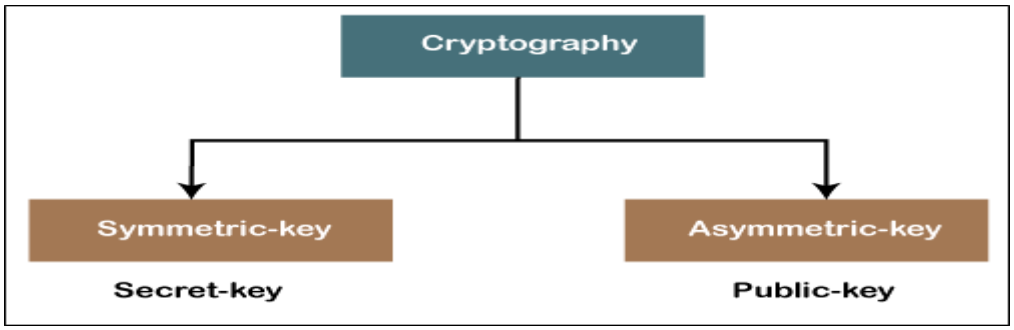



Fig.1.3: Types of Cryptography

### 1.3.1 Symmetric key cryptography

Symmetric key cryptography is that cryptography in which the same key (only one key) is used for encryption of plain text and decryption of ciphertext. Symmetric key cryptography is also known as secret-key cryptography.

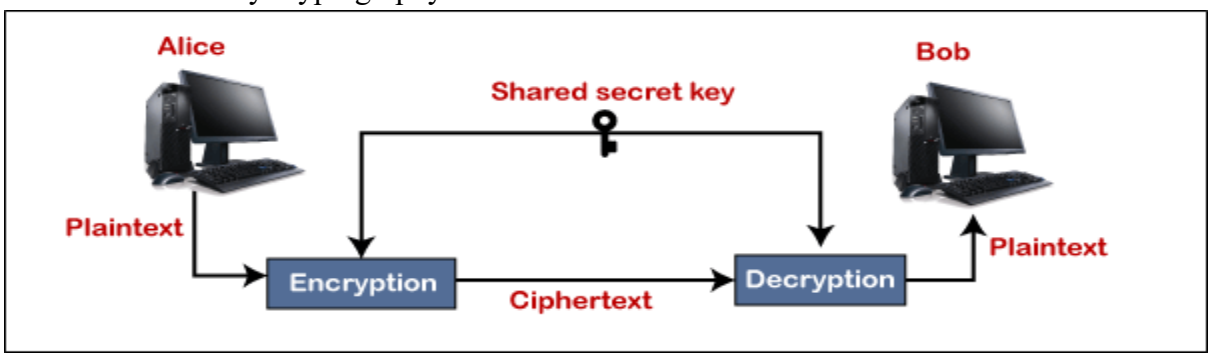



Fig.1.4: Symmetric key cryptography

### 1.3.2 Asymmetric key cryptography

Asymmetric key cryptography is that cryptography in which both encryption and decryption have different keys. In this, the public key is used to do encryption, and the private key is used to do the decryption. It is also called public-key cryptography.

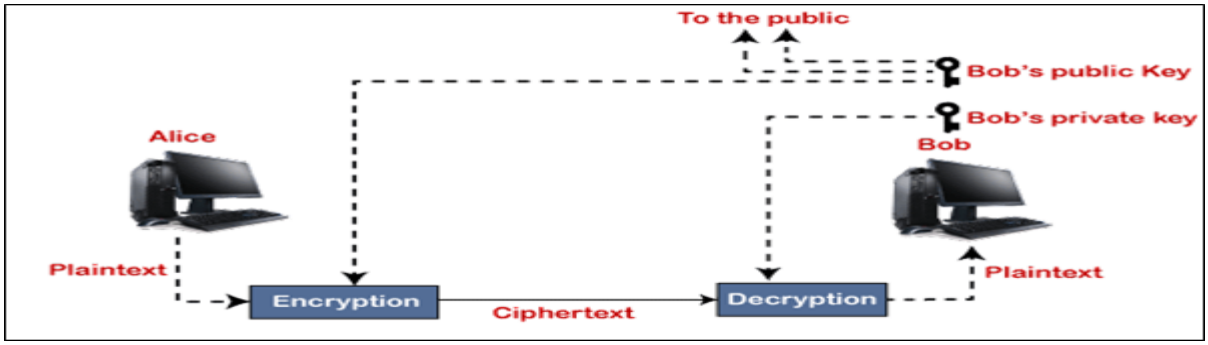


Fig.1.5: Asymmetric key cryptography

# Lab No.: 1 - Study of Classical Cipher

## 2.1 Caesor Cipher

The Caesar Cipher technique is one of the earliest and simplest method of encryption technique. It's is simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.

Thus, to cipher a given text we need an integer value, known as shift which indicates the number of positions each letter of the text has been moved down.

### 2.1.1 Algorithm for Caesar Cipher

**Input:**

- A String of lower-case letters, called Text.
- An Integer between 0-25 denoting the required shift.

**Procedure:**

- Traverse the given text one character at a time.
- For each character, transform the given character as per the rule, depending on whether we're encrypting or decrypting the text.
- Return the new string generated.

### 2.1.2 Code Implementation for Caesor Cipher

```
#Message to encrypt
message = list(input('message: '))
l=len(message)
for i in range(l):
      if ord(message[i])<ord('a') and
ord(message[i])>ord('z'):
            print('Out of range')
            exit()

p1= 'abcdefghijklmnopqrstuvwxyz'
     k=int(input('Enter value of key: '))

     #encryption
     newMessage = []
     letter=[]
     for i in range(l):
           c=(ord(message[i])-ord('a')+k)%26          #c=99-->99-97=2-->2+3=5
           letter=p1[c] #5-->f
           newMessage.append(letter)

     Ciphertext=' '.join(newMessage)
     print('Encrypted text: ',ciphertext)

     #decryption
     newMessage = []
     letter=[]
     for i in range(l):
           c=(ord(ciphertext[i])-ord('a')-k)%26
           letter=p1[c]
           newMessage.append(letter)

     Plaintext = ' '.join(newMessage)
     print('Decrypted text: ',plaintext)

```

**Output:**

```
message: haririjal
Enter value of key: 3
Encrypted text:  kdululmdo
Decrypted text:  haririjal
```

**Discussion:**

Caesor Cipher is a mono-alphabetic cipher wherein each letter of the plain text is substituted by another letter to form the cipher text. It is a simplest form of substitution cipher scheme. This cryptosystem is generally referred to as the Shift Cipher. The concept is to replace each alphabet by another alphabet which is 'shifted' by some fixed number between 0 and 25.

## 2.2 Substitution Cipher

In cryptography, a substitution cipher is a method of encoding by which units of plain text are replaced with cipher text, according to a regular system; the "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing an inverse substitution.

There are a number of different types of substitution cipher. If the cipher operates on single letters, it is termed a simple substitution cipher; a cipher that operates on larger groups of letters is termed poly-graphic. A mono-alphabetic cipher uses fixed substitution over the entire message, whereas a poly-alphabetic cipher uses a number of substitutions at different positions in the message, where a unit from the plain text is mapped to one of several possibilities in the cipher text and vice versa.

### 2.2.1 Algorithm for Substitution Cipher

**Input:**

- A String of both lower- and upper-case letters, called Plain Text.
- An Integer denoting the required key.

**Procedure:**

- Create a list of all the characters.
- Create a dictionary to store the substitution for all characters.
- For each character, transform the given character as per the rule, depending on whether we're encrypting or decrypting the text.
- Print the new string generated

### 2.2.2 Code Implementation for Substitution Cipher

```
#Message to encrypt
message = list(input('message: '))
l=len(message)
p1= 'abcdefghijklmnopqrstuvwxyz'
s1= 'kjpluvtrxysqcewnfbgdzmahio'

#encryption
newMessage = []
letter=[]

    for i in range(l):
        for j in range(26):
            if message[i] == p1[j]:
                letter = s1[j]
                newMessage.append(letter)
```

```
    ciphertext=''.join(newMessage)
    print('Encrypted text: ',ciphertext)


    #decryption

    newMessage = []
    letter=[]

    for i in range(l):
        for j in range(26):
            if ciphertext[i] == s1[j]:
                letter = p1[j]
                newMessage.append(letter)

    plaintext=''.join(newMessage)
    print('Decrypted text: ',plaintext)
```

**Output:**

```
message: bachelor
Encrypted text:  jkpruqwb
Decrypted text:  bachelor
```

**Discussion**

Simple substitution cipher is the most commonly used cipher and includes an algorithm of substituting every plain text character for every cipher text character. There are two table one is for plain text and another is for cipher text in this process, alphabets are jumbled in comparison with Caesar cipher algorithm

## 2.3 Simple XOR Cipher

XOR cipher employs the XOR logical operation in order to encrypt data. First, a random key is generated. Then, XOR operation is performed using the key so that an encrypted data is created. In order to decrypt, the same key should be used and XOR operation should be run again.

XOR operation uses the same key for both encryption and decryption. That is why it is known as a symmetric encryption. The concept of implementation is to first define XOR – encryption key and then to perform XOR operation of the characters in the String with this key which you want to encrypt. To decrypt the encrypted characters, we have to perform XOR operation again with the defined key.

### 2.3.1 Algorithm for Simple XOR Cipher:

**Input:**

- A String of both lower- and upper-case letters, called Plain Text.
- An encryption key.

**Procedure:**

- Bitwise XOR operation between plain text and key to get cipher text.
  i.e., plain text ⊕ key= cipher text
- Bitwise XOR operation between cipher text and key to get plain text.
  i.e., cipher text ⊕ key= plain text

## 2.3.2 Code Implementation for Simple XOR Cipher

```
1. # calculating XOR of two strings of binary number a and b
2. def xor(a, b):
3.       ans = ""
4.       for i in range(len(a)):
5.             if a[i] == b[i]:
6.                   ans = ans + "0"
7.             else:
8.                   ans = ans + "1"
9.       return ans
10.      #Plaintext
11.      a="10101101"
12.      #Key
13.      b="01010101"
14.      #Ciphertext
15.      c=xor(a,b)
16.      #Decrypted plaintext
17.      d=xor(c,b)
18.      print('plaintext: ',a)
19.      print('key : ',b)
20.      print('ciphertext: ',c)
21.      print('decrypted plaintext: ',d)
```

## Output:

```
plaintext:  10101101
key :  01010101
ciphertext:  11111000
decrypted plaintext:  10101101
```

**Discussion:**

XOR cipher is famous for being very resistant to brute force attacks where the attacker generates random keys and try them until the correct one is found.

In addition, the implementation of XOR is very easy. That is why XOR is used inside most encryption algorithms or used with various other encryption methods. Yet the most important feature of the XOR cipher is that it can be “the perfect cipher” with one time pad.

- One time pad refers to an encryption technique where the key is:
- Truly random,
- Kept secret,
- As long as (or longer than) the plain text,
- Never reused in part or in whole.

When XOR cipher has a random key that is as long as the message itself, it is impossible to crack it. In other words, it offers the highest level of security.

# Lab No.: 2 – Study of Classical Cipher

## 3.1 Transposition Cipher (Rail Fence Cipher)

In cryptography, a transposition cipher is a method of encryption by which the positions held by units of plain text (which are commonly characters or groups of characters) are shifted according to a regular system, so that the cipher text constitutes a permutation of the plain text. That is, the order of the units is changed (the plain text is reordered). Mathematically a bijective function, i.e., an inversible or one-to-one function, is used on the characters’ positions to encrypt and an inverse function to decrypt.

The rail fence cipher (also called a zigzag cipher) is a form of transposition cipher. It derives its name from the way in which it is encoded.

### 3.1.1 Algorithm for Transposition Cipher:

**Input:**

- A String of both lower- and upper-case letters, called Plain Text.
- A depth key.

**Procedure:**

- For encryption, in the rail fence cipher, the plain text is written downwards diagonally on successive "rails" of an imaginary fence, then moving up when the bottom rail is reached, down again when the top rail is reached, and so on until the whole plain text is written out. The cipher text is then read off in rows.

- For decryption, as we've seen earlier, the number of columns in rail fence cipher remains equal to the length of plain-text message. And the key corresponds to the number of rails. Hence, rail matrix can be constructed accordingly. Once we've got the matrix we can figure-out the spots where texts should be placed (using the same way of moving diagonally up and down alternatively). Then, we fill the cipher-text row wise. After filling it, we traverse the matrix in zig-zag manner to obtain the original text.

### 3.1.2 Code Implementation for Transposition Cipher

```
#    Transpossition Cipher (Rail Fence Technique)

def main():

    # get the plain text
    s = input("Enter the plaintext: ")
    # get the number of layers to rail encrypt
    n = int(input("Enter the rail fence depth: "))


     #Encryption of plaintext
         cipher_text = encrypt(s,n)
         print("Encrypted text: " + cipher_text)
     #Decryption of ciphertext
         plaintext=decrypt(cipher_text,n)
         print("Decrypted text: " + plaintext)


     #Encryption
     def encrypt(s,n):
       fence = [[] for i in range(n)]
       rail  = 0
       var   = 1

       for char in s:
         fence[rail].append(char)
         rail += var

         if rail == n-1 or rail == 0:
           var = -var
       res = ''
       for i in fence:
         for j in i:
      res += j
    return res
```

```
    #Decryption
    def decrypt(s,n):
        fence = [[] for i in range(n)]
        rail  = 0
        var   = 1

        for char in s:
            fence[rail].append(char)
            rail += var

            if rail == n-1 or rail == 0:
                var = -var

        rFence = [[] for i in range(n)]

        i = 0
        l = len(s)
        s = list(s)
        for r in fence:
            for j in range(len(r)):
                rFence[i].append(s[0])
                s.remove(s[0])
            i += 1

        rail = 0
        var  = 1
        r = ''
        for i in range(l):
            r += rFence[rail][0]
            rFence[rail].remove(rFence[rail][0])
            rail += var

            if rail == n-1 or rail == 0:
                var = -var

        return r

    if __name__ == '__main__':
        main()
```

**Output:**

```
Plaintext: asian school of management and technology
Railfence Depth= 5
Encrypted text=  ahmneysco aettcgisofnm  hoa loaeadnln gno
Decrypted text=  asian school of management and technology
```

**Discussion**

The term zigzag cipher may refer to the rail fence cipher as described above. However, it may also refer to a different type of cipher described by Fletcher Pratt in Secret and Urgent. It is "written by ruling a sheet of paper in vertical columns, with a letter at the head of each column. A dot is made for each letter of the message in the proper column, reading from top to bottom of the sheet. The letters at the head of the columns are then cut off, the ruling erased and the message of dots sent along to the recipient, who, knowing the width of the columns and the arrangement of the letters at the top, reconstitutes the diagram and reads what it has to say."

## 3.2 Playfair Cipher

In this scheme, pairs of letters are encrypted, instead of single letters as in the case of simple substitution cipher. The Playfair cipher was the first practical digraph substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher. In Playfair cipher unlike traditional cipher we encrypt a pair of alphabets (digraphs) instead of a single alphabet.

### 3.2.1 Algorithm for Playfair Cipher

**Input**

- A String of both lower- or upper-case letters, called Plain Text.
- A keyword.

**Procedure**

- Generate the key Square (5×5):
  - The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plain text. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plain text contains J, then it is replaced by I.
  - The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

- Split of message into pairs of two letters (digraphs)
  - The plain text is split into pairs of two letters (digraphs). If there is an odd number of letters, X is added to the last letter.
    - Pair cannot be made with same letter. Break the letter in single and add a bogus letter to the previous letter.
    - If the letter is standing alone in the process of pairing, then add an extra bogus letter with the alone letter

- **Rules for Encryption**
  - If both the letters are in the same column: Take the letter below each one (going back to the top if at the bottom).
  - If both the letters are in the same row: Take the letter to the right of each one (going back to the leftmost if at the rightmost position).
  - If neither of the above rules is true: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

- **Rules for Decryption**

- If both the letters are in the same column: Take the letter above each one (going back to the bottom if at the top).
- If both the letters are in the same row: Take the letter to the left of each one (going back to the rightmost if at the leftmost position).
- If neither of the above rules is true: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

### 3.2.2 Code Implementation for Playfair Cipher

```
#    Playfair Cipher

key = input("Enter key")
key = key.replace(" ", "")
key = key.upper()


def matrix(x, y, initial):
    return [[initial for i in range(x)] for j in range(y)]


    result = list()
    for c in key:  # storing key
        if c not in result:
            if c == 'J':
                result.append('I')
            else:
                result.append(c)
    flag = 0
    for i in range(65, 91):  # storing other character
        if chr(i) not in result:
            if i == 73 and chr(74) not in result:
                result.append("I")
                flag = 1
            elif flag == 0 and i == 73 or i == 74:
                pass
            else:
                result.append(chr(i))
    k = 0
    my_matrix = matrix(5, 5, 0)  # initialize matrix
    for i in range(0, 5):  # making matrix
        for j in range(0, 5):
            my_matrix[i][j] = result[k]
            k += 1


```

```
    def locindex(c):  # get location of each character
        loc = list()
        if c == 'J':
            c = 'I'
        for i, j in enumerate(my_matrix):
            for k, l in enumerate(j):
                if c == l:
                    loc.append(i)
                    loc.append(k)
                    return loc


    def encrypt():  # Encryption
        msg = str(input("ENTER MSG:"))
        msg = msg.upper()
        msg = msg.replace(" ", "")
        i = 0
        for s in range(0, len(msg) + 1, 2):
            if s < len(msg) - 1:
                if msg[s] == msg[s + 1]:
                    msg = msg[:s + 1] + 'X' + msg[s + 1:]
        if len(msg) % 2 != 0:
            msg = msg[:] + 'X'
        print("CIPHERTEXT:", end=' ')
        while i < len(msg):
            loc = list()
            loc = locindex(msg[i])
            loc1 = list()
            loc1 = locindex(msg[i + 1])
            if loc[1] == loc1[1]:
                print("{}{}".format(my_matrix[(loc[0] + 1)
% 5][loc[1]], my_matrix[(loc1[0] + 1) % 5][loc1[1]]), end='
')
            elif loc[0] == loc1[0]:

print("{}{}".format(my_matrix[loc[0]][(loc[1] + 1) % 5],
my_matrix[loc1[0]][(loc1[1] + 1) % 5]), end=' ')
            else:

print("{}{}".format(my_matrix[loc[0]][loc1[1]],
my_matrix[loc1[0]][loc[1]]), end=' ')
            i = i + 2


    def decrypt():  # decryption
        msg = str(input("ENTER CIPHERTEXT:"))
        msg = msg.upper()
        msg = msg.replace(" ", "")
```

```
79.           print("PLAIN TEXT:", end=' ')
80.          i = 0
81.          while i < len(msg):
82.              loc = list()
83.              loc = locindex(msg[i])
84.              loc1 = list()
85.              loc1 = locindex(msg[i + 1])
86.              if loc[1] == loc1[1]:
87.                  print("{}{}".format(my_matrix[(loc[0] - 1)
   % 5][loc[1]], my_matrix[(loc1[0] - 1) % 5][loc1[1]]), end='
   ')
88.              elif loc[0] == loc1[0]:
89.
   print("{}{}".format(my_matrix[loc[0]][(loc[1] - 1) % 5],
   my_matrix[loc1[0]][(loc1[1] - 1) % 5]), end=' ')
90.              else:
91.
   print("{}{}".format(my_matrix[loc[0]][loc1[1]],
   my_matrix[loc1[0]][loc[1]]), end=' ')
92.              i = i + 2
93.
94.
95.      while (1):
96.          choice = int(input("\n 1.Encryption \n
   2.Decryption: \n 3.EXIT"))
97.          if choice == 1:
98.              encrypt()
99.          elif choice == 2:
100.             decrypt()
101.         elif choice == 3:
102.             exit()
103.         else:
104.             print("Choose correct choice")
```

**Output**

```
Enter key: information

 1.Encryption
 2.Decryption:
 3.EXIT
 Choose value for the operation: 1
ENTER MSG:technology
CIPHERTEXT: AG BK FR SI HX
 1.Encryption
 2.Decryption:
 3.EXIT
 Choose value for the operation: 2
ENTER CIPHERTEXT:AG BK FR SI HX
PLAIN TEXT: TE CH NO LO GY
 1.Encryption
 2.Decryption:
 3.EXIT
 Choose value for the operation: 3
```

**Conclusion:**
Playfair Cipher is significantly harder to break since the frequency analysis technique used to break simple substitution ciphers is difficult but still can be used on (25*25) = 625 digraphs rather than 25 monographs which is difficult.

An interesting weakness is the fact that a digraph in the cipher text (AB) and it's reverse (BA) will have corresponding plain texts like UR and RU (and also cipher text UR and RU will correspond to plain text AB and BA, i.e., the substitution is self-inverse). That can easily be exploited with the aid of frequency analysis, if the language of the plain text is known. Another disadvantage is that Playfair cipher is a symmetric cipher thus same key is used for both encryption and decryption

# Lab No. 3 - Study of Asymmetric Cryptography

## 4.1 (RSA Algorithm)

RSA (Rivest–Shamir–Adleman) is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of the keys can be given to anyone. The other key must be kept private. The algorithm is based on the fact that finding the factors of a large composite number is difficult: when the factors are prime numbers, the problem is called prime factorization. It is also a key pair (public and private key) generator.

RSA involves a public key and private key. The public key can be known to everyone- it is used to encrypt messages. Messages encrypted using the public key can only be decrypted with the private key. The private key needs to be kept secret. Calculating the private key from the public key is very difficult.

### 4.1.1 Algorithm of RSA

**Procedure:**

- **Generating the keys**
  - Select two large prime numbers, $x$ and $y$. The prime numbers need to be large so that they will be difficult for someone to figure out.
  - Calculate $n = x * y$.
  - Calculate the *totient* function; $\phi(n)=(x-1)(y-1)$.
  - Select an integer $e$, such that $e$ is co-prime to $\phi(n)$ and $1 < e < \phi(n)$. The pair of numbers $(n,e)$ makes up the public key.
  - Calculate $d$ such that $e.d=1 \bmod \phi(n)$.
  - $d$ can be found using the extended euclidean algorithm. The pair $(n,d)$ makes up the private key.

- **Encryption**
  - Given a plain text $P$, represented as a number, the cipher text $C$ is calculated as: $C = P^e \bmod n$.

- **Decryption**
  - Using the private key $(n,d)$, the plain text can be found using: $P = C^d \bmod n$.

### 4.1.2 Code Implementation for RSA Algorithm

```
1.
2. #.....................................RSA
   Algorithm.....................................#
3.
4. from decimal import Decimal
5.
6. def gcd(a,b):
7.       if b==0:
8.             return a
9.       else:
10.                  return gcd(b,a%b)
11.      p = int(input('Enter the value of p = '))
12.      q = int(input('Enter the value of q = '))
13.      no = int(input('Enter the value of text = '))
14.      n = p*q
15.      phi = (p-1)*(q-1)
16.
17.      for e in range(2,phi):
18.            if gcd(e,phi)== 1:
19.                  break
20.
21.
22.      for i in range(1,10):
23.            x = 1 + i*phi
24.            if x % e == 0:
25.                  d = int(x/e)
26.                  break
27.      ctt = Decimal(0)
28.      ctt =pow(no,e)
29.      ct = ctt % n
30.
31.      dtt = Decimal(0)
32.      dtt = pow(ct,d)
33.      dt = dtt % n
34.
35.      print('n = '+str(n)+' e = '+str(e)+' phi = '+str(phi)+'
   d = '+str(d)+' cipher text = '+str(ct)+' decrypted text =
   '+str(dt))
36.
```

**output:**

```
Enter the value of p = 13
Enter the value of q = 11
Enter the value of text = 88
n = 143 e = 7 phi = 120 d = 103 cipher text = 88 decrypted text = 88
```

**Discussion**

In RSA cryptography, both the public and the private keys can encrypt a message; the opposite key from the one used to encrypt a message is used to decrypt it. This attribute is one reason why RSA has become the most widely used asymmetric algorithm: It provides a method to assure the confidentiality, integrity, authenticity, and non-repudiation of electronic communications and data storage.

RSA derives its security from the difficulty of factoring large integers that are the product of two large prime numbers. Multiplying these two numbers is easy, but determining the original prime numbers from the total or factoring, is considered infeasible due to the time it would take using even today's supercomputers

# Lab No. 4: Study of Symmetric Cryptography
# 5.1 (Data Encryption Standard – DES)

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).
DES was finally published as FIPS 46 in the Federal Register in January 1977. NIST, however, defines DES as the standard for use in unclassified applications. DES has been the most widely used symmetric-key block cipher since its publication. NIST later issued a new standard (FIPS 46-3) that recommends the use of triple DES (repeated DES cipher three times) for future applications.

## 5.1.1 Overview

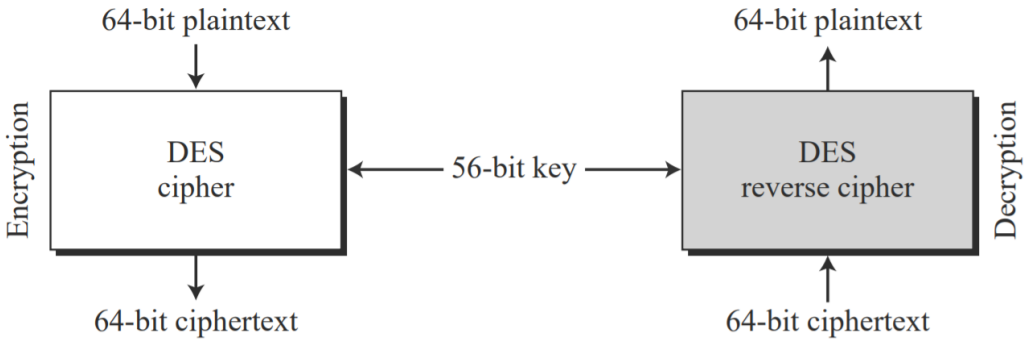



Fig. 2.1: Encryption and decryption with DES

At the encryption site, DES takes a 64-bit plaintext and creates a 64-bit ciphertext; at the decryption site, DES takes a 64-bit ciphertext and creates a 64-bit block of plaintext. The same 56-bit cipher key is used for both encryption and decryption.

## 5.1.2 DES Structure

DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only).

The encryption process is made of two permutations (P-boxes), which we call initial and final permutations, and sixteen Feistel rounds. Each round uses a different 48-bit round key generated from the cipher key according to a predefined algorithm

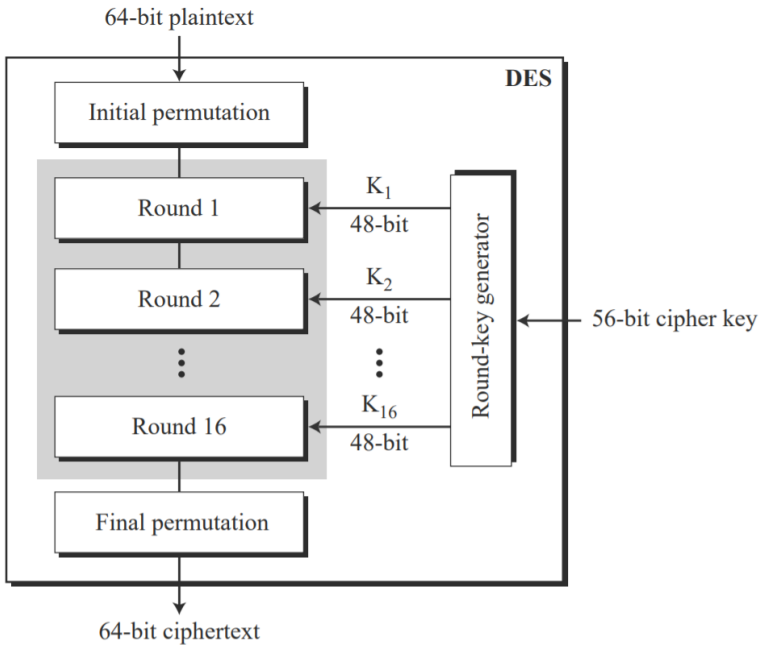



Fig. 2.2: General Structure of DES

### 5.1.3 Code Implementation for DES

```
#-*- Data Encryption Standard (DES) -*-
def main():
    key = "secret_k"
    text= input("Enter Plaintext: ")
    d = des()
    r = d.encrypt(key,text)
    r2 = d.decrypt(key,r)
    print("Ciphered Text: ",r)
    print("Deciphered Text: ",r2)

#Initial permut matrix for the datas
PI = [58, 50, 42, 34, 26, 18, 10, 2,
      60, 52, 44, 36, 28, 20, 12, 4,
      62, 54, 46, 38, 30, 22, 14, 6,
      64, 56, 48, 40, 32, 24, 16, 8,
      57, 49, 41, 33, 25, 17, 9, 1,
      59, 51, 43, 35, 27, 19, 11, 3,
      61, 53, 45, 37, 29, 21, 13, 5,
      63, 55, 47, 39, 31, 23, 15, 7]

#Initial permut made on the key
CP_1 = [57, 49, 41, 33, 25, 17, 9,
        1, 58, 50, 42, 34, 26, 18,
        10, 2, 59, 51, 43, 35, 27,
        19, 11, 3, 60, 52, 44, 36,
        63, 55, 47, 39, 31, 23, 15,
        7, 62, 54, 46, 38, 30, 22,
```

```
14, 6, 61, 53, 45, 37, 29,
        21, 13, 5, 28, 20, 12, 4]

#Permut applied on shifted key to get Ki+1
CP_2 = [14, 17, 11, 24, 1, 5, 3, 28,
        15, 6, 21, 10, 23, 19, 12, 4,
        26, 8, 16, 7, 27, 20, 13, 2,
        41, 52, 31, 37, 47, 55, 30, 40,
        51, 45, 33, 48, 44, 49, 39, 56,
        34, 53, 46, 42, 50, 36, 29, 32]

#Expand matrix to get a 48bits matrix of datas to apply the xor with Ki
E = [32, 1, 2, 3, 4, 5,
     4, 5, 6, 7, 8, 9,
     8, 9, 10, 11, 12, 13,
     12, 13, 14, 15, 16, 17,
     16, 17, 18, 19, 20, 21,
     20, 21, 22, 23, 24, 25,
     24, 25, 26, 27, 28, 29,
     28, 29, 30, 31, 32, 1]

#SBOX
S_BOX = [

[[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7],
 [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8],
 [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0],
 [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13],
],

[[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10],
 [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5],
 [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15],
 [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9],
],

[[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8],
 [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1],
 [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7],
 [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12],
],

[[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15],
 [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9],
 [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4],
 [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14],
],
```

```
[[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9],
 [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6],
 [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14],
 [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3],
],

[[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11],
 [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8],
 [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6],
 [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13],
],

[[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1],
 [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6],
 [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2],
 [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12],
],

[[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7],
 [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2],
 [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8],
 [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11],
]
]

#Permut made after each SBox substitution for each round
P = [16, 7, 20, 21, 29, 12, 28, 17,
     1, 15, 23, 26, 5, 18, 31, 10,
     2, 8, 24, 14, 32, 27, 3, 9,
     19, 13, 30, 6, 22, 11, 4, 25]

#Final permut for datas after the 16 rounds
PI_1 = [40, 8, 48, 16, 56, 24, 64, 32,
        39, 7, 47, 15, 55, 23, 63, 31,
        38, 6, 46, 14, 54, 22, 62, 30,
        37, 5, 45, 13, 53, 21, 61, 29,
        36, 4, 44, 12, 52, 20, 60, 28,
        35, 3, 43, 11, 51, 19, 59, 27,
        34, 2, 42, 10, 50, 18, 58, 26,
        33, 1, 41, 9, 49, 17, 57, 25]

#Matrix that determine the shift for each round of keys
SHIFT = [1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1]

def string_to_bit_array(text):
    array = list()
    for char in text:
        binval = binvalue(char, 8)
```

```
array.extend([int(x) for x in list(binval)])
    return array

def bit_array_to_string(array):
    res = ''.join([chr(int(y,2)) for y in [''.join([str(x) for x in _bytes]
) for _bytes in  nsplit(array,8)]])
    return res

def binvalue(val, bitsize):
    binval = bin(val)[2:] if isinstance(val, int) else bin(ord(val))[2:]
    if len(binval) > bitsize:
        raise "binary value larger than the expected size"
    while len(binval) < bitsize:
        binval = "0"+binval
    return binval

def nsplit(s, n):
    return [s[k:k+n] for k in range(0, len(s), n)]

ENCRYPT=1
DECRYPT=0

class des():
    def __init__(self):
        self.password = None
        self.text = None
        self.keys = list()

    def run(self, key, text, action=ENCRYPT, padding=False):
        if len(key) < 8:
            raise "Key Should be 8 bytes long"
        elif len(key) > 8:
            key = key[:8]

        self.password = key
        self.text = text

        if padding and action==ENCRYPT:
            self.addPadding()
        elif len(self.text) % 8 != 0:
            raise "Data size should be multiple of 8"

self.generatekeys()
        text_blocks = nsplit(self.text, 8)
        result = list()
        for block in text_blocks:
            block = string_to_bit_array(block)
            block = self.permut(block,PI)
```

```
g, d = nsplit(block, 32)
            tmp = None
            for i in range(16):
                d_e = self.expand(d, E)
                if action == ENCRYPT:
                    tmp = self.xor(self.keys[i], d_e)
                else:
                    tmp = self.xor(self.keys[15-i], d_e)
                tmp = self.substitute(tmp)
                tmp = self.permut(tmp, P)
                tmp = self.xor(g, tmp)
                g = d
                d = tmp
            result += self.permut(d+g, PI_1)
        final_res = bit_array_to_string(result)
        if padding and action==DECRYPT:
            return self.removePadding(final_res)
        else:
            return final_res

    def substitute(self, d_e):
        subblocks = nsplit(d_e, 6)
        result = list()
        for i in range(len(subblocks)):
            block = subblocks[i]
            row = int(str(block[0])+str(block[5]),2)
            column = int(''.join([str(x) for x in block[1:][:-1]]),2)
            val = S_BOX[i][row][column]
            bin = binvalue(val, 4)
            result += [int(x) for x in bin]
        return result

    def permut(self, block, table):
        return [block[x-1] for x in table]

    def expand(self, block, table):
        return [block[x-1] for x in table]

    def xor(self, t1, t2):
        return [x^y for x,y in zip(t1,t2)]

    def generatekeys(self):
        self.keys = []
        key = string_to_bit_array(self.password)
        key = self.permut(key, CP_1)
        g, d = nsplit(key, 28)
        for i in range(16):
            g, d = self.shift(g, d, SHIFT[i])
```

```
tmp = g + d
            self.keys.append(self.permut(tmp, CP_2))

    def shift(self, g, d, n):
        return g[n:] + g[:n], d[n:] + d[:n]

    def addPadding(self):
        pad_len = 8 - (len(self.text) % 8)
        self.text += pad_len * chr(pad_len)

    def removePadding(self, data):
        pad_len = ord(data[-1])
        return data[:-pad_len]

    def encrypt(self, key, text, padding=True):
        return self.run(key, text, ENCRYPT, padding)

    def decrypt(self, key, text, padding=True):
        return self.run(key, text, DECRYPT, padding)

if __name__ == '__main__':
    main()
```

**Output:**

```
Enter Plaintext: tribhuvan university
Ciphered Text:  Ö²Ôèüçç]6´►r/TZ·rZîê¨t
Deciphered Text:  tribhuvan university
```

**Discussion:**

The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.

- **Avalanche effect** − A small change in plain text results in the very great change in the cipher text.
- **Completeness** − Each bit of cipher text depends on many bits of plain text.

During the last few years, cryptanalysis have found some weaknesses in DES when key selected are weak keys. These keys shall be avoided. DES has proved to be a very well-designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

# Lab No.: 5 - Configuration of Firewall

## 6.1 What is Firewall Configuration?

A firewall plays a viral role in network security and needs to be properly configured to keep organizations protected from data leakage and cyberattacks.

This is possible by configuring domain names and Internet Protocol (IP) addresses to keep the firewall secure. Firewall policy configuration is based on network type, such as public or private, and can be set up with security rules that block or allow access to prevent potential attacks form hackers or malware.

### 6.1.1 Implementation of Firewall

In this lab, the implementation of a firewall with the following configuration was done.

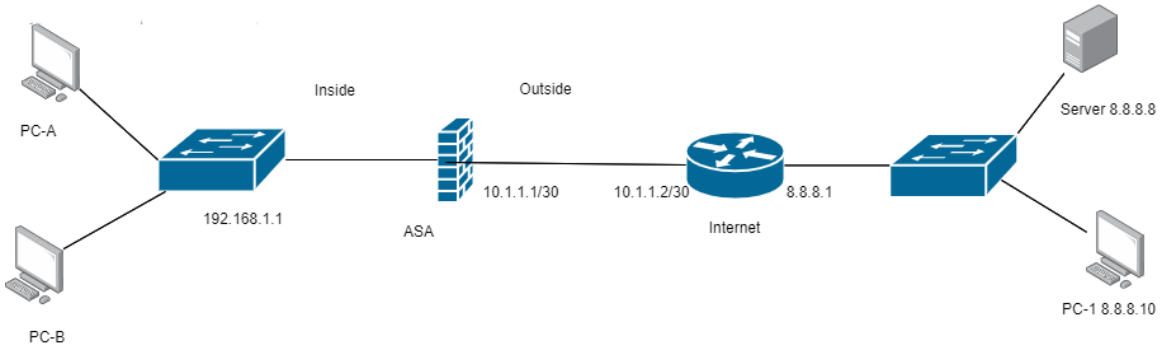
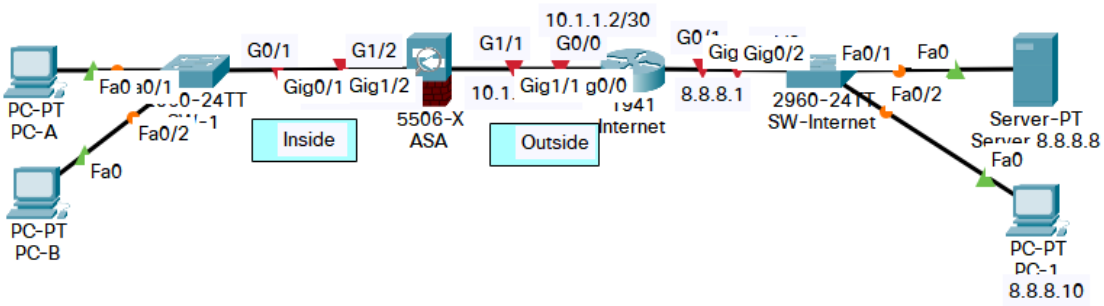



Fig. 3.1: Firewall Topology

**Topology:**



**Router Configuration:**

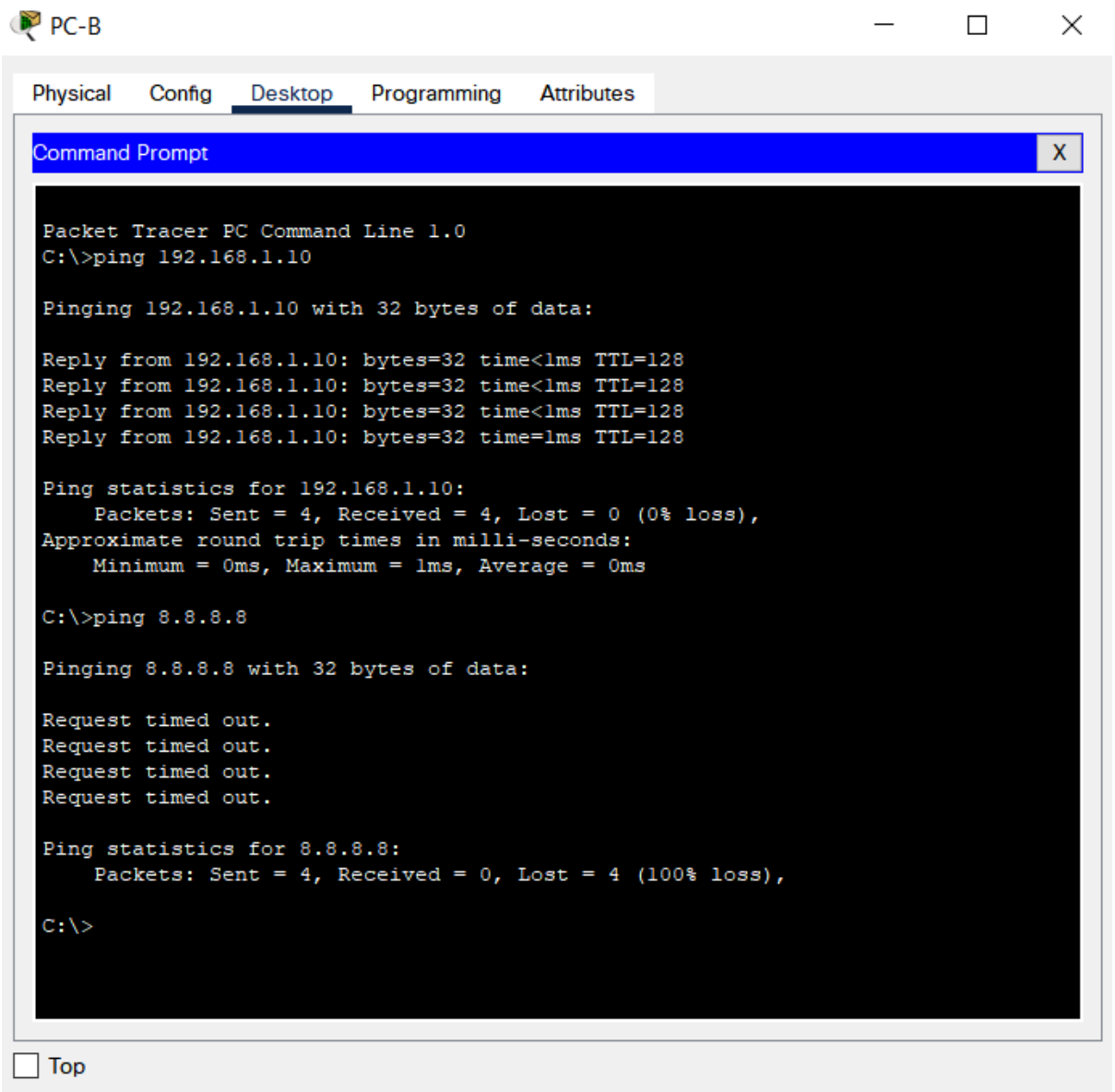
```
Router>en
Router#conf t
Router(config)#hostname Internet
Internet(config)#int g0/1
Internet(config-if)#ip address 8.8.8.1 255.255.255.0
Internet(config-if)#no shut
Internet(config-if)#int g0/0
Internet(config-if)#ip address 10.1.1.2 255.255.255.252
Internet(config-if)#no shut
Internet(config-if)#exit
```

**Firewall ASA Configuration:**

```
ciscoasa>en
Password:
ciscoasa#conf t
ciscoasa(config)#hostname ASA
ASA(config)#enable password cisco
ASA(config)#show run
ASA(config)#int g1/1
ASA(config-if)#no ip address
ASA(config-if)#no nameif
ASA(config-if)#no security-level
ASA(config-if)#int g1/2
ASA(config-if)#no nameif
ASA(config-if)#no security-level
ASA(config-if)#no ip address dhcp
ASA(config-if)#exit
ASA(config)# show run
ASA(config)#int g1/1
ASA(config-if)#ip address 10.1.1.1 255.255.255.252
ASA(config-if)#nameif outside
ASA(config-if)# security-level 0
ASA(config-if)#no shut
ASA(config-if)#int g1/2
ASA(config-if)#ip address 192.168.1.1 255.255.255.0
ASA(config-if)#nameif inside
ASA(config-if)# security-level 100
ASA(config-if)#no shut
ASA(config-if)#exit
ASA(config)# show in ip brief
```

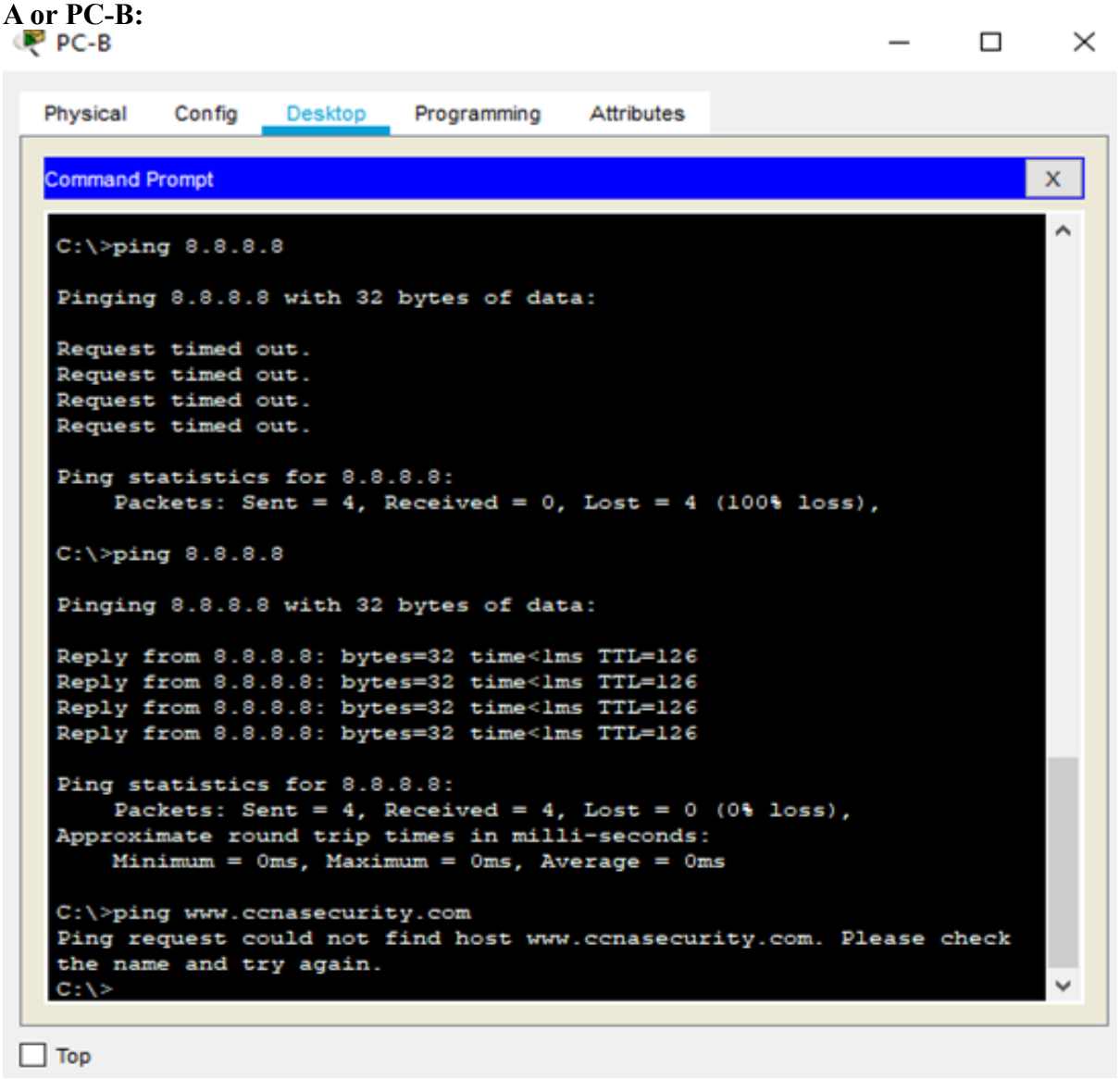
```
ASA(config)#dhcp address 192.168.1.10-192.168.1.20 inside
ASA(config)#dhcp dns 8.8.8.8
ASA(config)#dhcp option 3 ip 192.168.1.1
ASA(config)#dhcp enable inside
ASA(config)#route outside 0.0.0.0 0.0.0.0 10.1.1.2
ASA(config)#object network INSIDE-NET
ASA(config-network-object)#subnet 192.168.1.0 255.255.255.0
ASA(config-network-object)#nat (inside,outside) dynamic interface
ASA(config-network-object)#exit
```

**Test: Ping PC-B from PC-A and server 8.8.8.8 from PC-A or PC-B:**

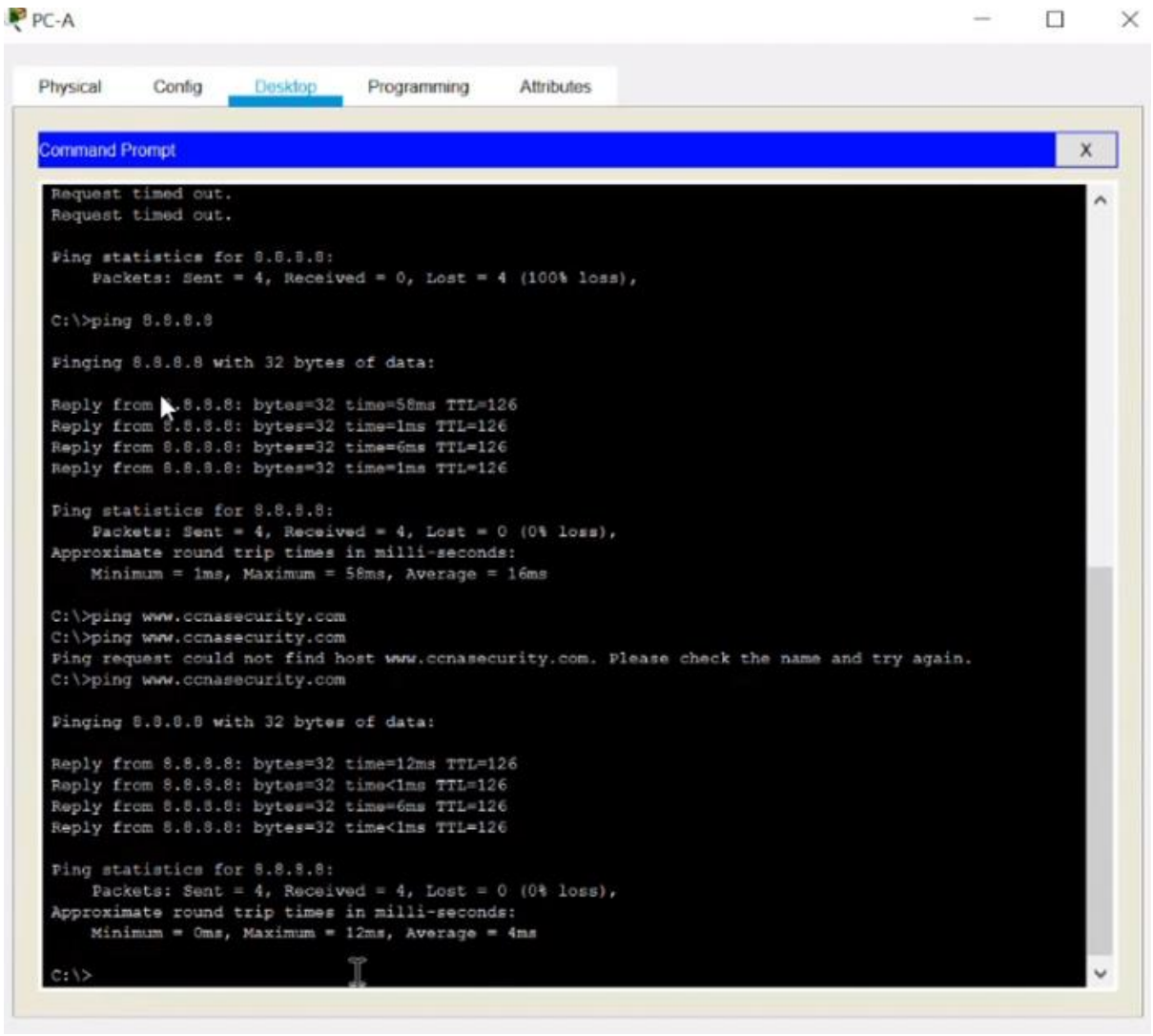


```
ASA#conf t
ASA(config)#class-map inspection_default
ASA(config-cmap)#match default-inspection-traffic
ASA(config-cmap)#exit
ASA(config)#policy-map global_policy
ASA(config-cmap)#class?
ASA(config-cmap)#class inspection_default
ASA(config-cmap-c)#inspect icmp
ASA(config-cmap-c)#exit
ASA(config)#service-policy global_policy global
ASA(config)#show run
ASA(config) #policy-map global_policy
ASA(config-pmap)#class inspection_default
ASA(config-pmap-c)#inspect http
ASA(config-pmap-c)#exit
ASA(config)#show run
```

**Test: Ping server 8.8.8.8 from PC-A or PC-B and ping www.ccnasecurity.com from PC-A or PC-B:**



```
ASA(config)#policy-map global_policy
ASA(config-pmap)#class inspection_default
ASA(config-pmap-c)#?
ASA(config-pmap-c)#inspect dns
ASA(config-pmap-c)#exit
```



# Lab No.: 6 - Configuration of IPSec VPN

## 7.1 What is IPsec VPN?

A Virtual Private Network (VPN) is an essential technology for securing data that is going over the Internet. By creating a secure tunnel, it ensures data is not exposed to bad actors (hacker, surveillance) over the public network.

Internet Protocol Security (IPsec) is a VPN standard that provides Layer 3 security. It's a suite of protocols that provides confidentiality, integrity and authentication to date.

As shown in the topology below, we will setup a VPN between the Internet Service Provider (ISP) and customer networks.

### 7.1.1 Implementation of IPSec VPN

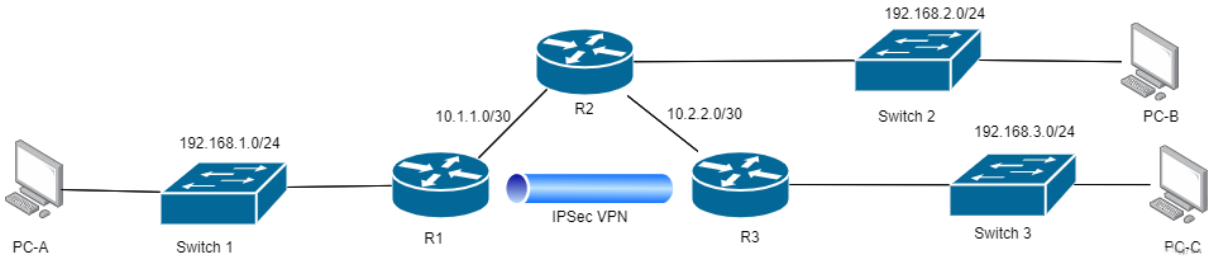
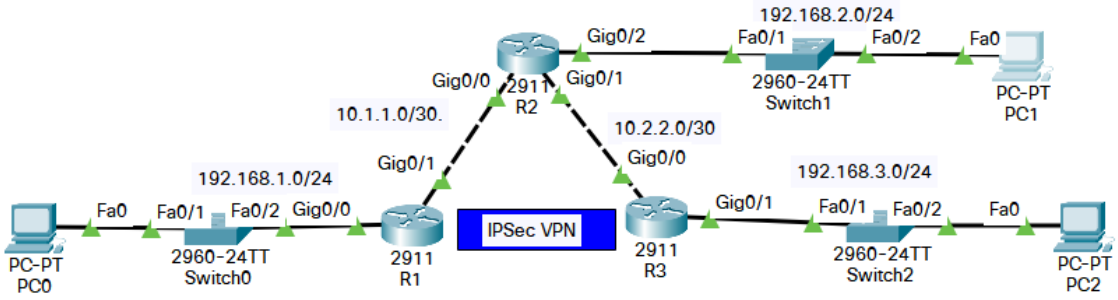



Fig. 3.2: IPSec VPN Topology

## Topology



**Configuration of R1:**

```
Router>enable
Router#configure terminal
Router(config)#hostname R1
R1(config)#interface g0/0
R1(config-if)#ip address 192.168.1.1 255.255.255.0
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#interface gigabitEthernet 0/1
R1(config-if)#ip address 10.1.1.2 255.255.255.252
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#router rip
R1(config-router)#version 2
R1(config-router)#network 192.168.1.0
R1(config-router)#network 10.1.1.0
R1(config-router)#no auto-summary
R1(config-router)#end
R1#write
```

**Configuration of R2:**

```
Router>enable
Router#configure t
Router(config)#hostname R2
R2(config)#interface gigabitEthernet 0/2
R2(config-if)#ip address 192.168.2.1 255.255.255.0
R2(config-if)#no shutdown
R2(config-if)#exit
R2(config)#interface gigabitEthernet 0/0
R2(config-if)#ip address 10.1.1.1 255.255.255.252
R2(config-if)#no shutdown
R2(config-if)#exit
R2(config)#interface gigabitEthernet 0/1
R2(config-if)#ip address 10.2.2.1 255.255.255.252
R2(config-if)#no shutdown
R2(config-if)#exit
R2(config)#router rip
R2(config-router)#version 2
R2(config-router)#network 192.168.2.0
R2(config-router)#network 10.1.1.0
R2(config-router)#network 10.2.2.0
R2(config-router)#no auto-summary
R2(config-router)#end
R2#wr
```

**Configuration of R3:**

```
Router>enable
Router#configure t
Router(config)#hostname R3
R3(config)#interface gigabitEthernet 0/1
R3(config-if)#ip address 192.168.3.1 255.255.255.0
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#interface gigabitEthernet 0/0
R3(config-if)#ip address 10.2.2.2 255.255.255.252
R3(config-if)#no shutdown
R3(config-if)#exit
R3(config)#router rip
R3(config-router)#version 2
R3(config-router)#network 192.168.3.0
R3(config-router)#network 10.2.2.0
R3(config-router)#no auto-summary
R3(config-router)#end
R3#wr
```

**Enabling security in R1:**

```
R1(config)#license boot module c2900 technology-package securityk9
R1#copy running-config startup-config
R1#reload
R1>show version
-----------------------------------------------------------------
Technology    Technology-package          Technology-package
Current         Type          Next reboot
-----------------------------------------------------------------
ipbase        ipbasek9      Permanent     ipbasek9
security       securityk9    Evaluation    securityk9
uc            disable       None           None
data          disable       None          None
```

**Enabling security in R3:**

```
R3(config)#license boot module c2900 technology-package securityk9
R3#copy running-config startup-config
R3#reload
R3>show version
-----------------------------------------------------------------
Technology    Technology-package          Technology-package
Current         Type          Next reboot
-----------------------------------------------------------------
ipbase        ipbasek9      Permanent     ipbasek9
security       securityk9    Evaluation    securityk9
uc            disable       None           None
data          disable       None          None
```

**Configuration of R1:**

```
R1(config)#access-list 110 permit ip 192.168.1.0 0.0.0.255 192.168.3.0 0.0.0.255
R1(config)#crypto isakmp policy 10
R1(config-isakmp)#encryption aes
R1(config-isakmp)#authentication pre-share
R1(config-isakmp)#group 2
R1(config-isakmp)#exit
R1(config)#crypto isakmp key cisco address 10.2.2.2
R1(config)#crypto ipsec transform-set VPN-SET esp-3des esp-sha-hmac
R1(config)#crypto map VPN-MAP 10 ipsec-isakmp
R1(config-crypto-map)#description VPN connection to R3
R1(config-crypto-map)#set peer 10.2.2.2
R1(config-crypto-map)#set transform-set VPN-SET
R1(config-crypto-map)#match address 110
R1(config-crypto-map)#exit
R1(config)#int g0/1
R1(config-if)#crypto map VPN-MAP
*Jan 3 07:16:26.785: %CRYPTO-6-ISAKMP_ON_OFF: ISAKMP is ON
```

**Configuration of R3:**

```
R3(config)#access-list 110 permit ip 192.168.3.0 0.0.0.255 192.168.1.0 0.0.0.255
R3(config)#crypto isakmp policy 10
R3(config-isakmp)#encryption aes
R3(config-isakmp)#authentication pre-share
R3(config-isakmp)#group 2
R3(config-isakmp)#exit
R3(config)#crypto isakmp key cisco address 10.1.1.2
R3(config)#crypto ipsec transform-set VPN-SET esp-3des esp-sha-hmac
R3(config)#crypto map VPN-MAP 10 ipsec-isakmp
R3(config-crypto-map)#description VPN connection to R1
R3(config-crypto-map)#set peer 10.1.1.2
R3(config-crypto-map)#set transform-set VPN-SET
R3(config-crypto-map)#match address 110
R3(config-crypto-map)#exit
R3(config)#interface g0/0
R3(config-if)#crypto map VPN-MAP
*Jan 3 07:16:26.785: %CRYPTO-6-ISAKMP_ON_OFF: ISAKMP is ON
```

**Verify IPSec Tunnel on R1:**

```
R1#show crypto ipsec sa

interface: GigabitEthernet0/1
Crypto map tag: VPN-MAP, local addr 10.1.1.2

protected vrf: (none)
local ident (addr/mask/prot/port): (192.168.1.0/255.255.255.0/0/0)
remote ident (addr/mask/prot/port): (192.168.3.0/255.255.255.0/0/0)
current_peer 10.2.2.2 port 500
PERMIT, flags={origin_is_acl,}
#pkts encaps: 0, #pkts encrypt: 0, #pkts digest: 0
#pkts decaps: 0, #pkts decrypt: 0, #pkts verify: 0
#pkts compressed: 0, #pkts decompressed: 0
#pkts not compressed: 0, #pkts compr. failed: 0
#pkts not decompressed: 0, #pkts decompress failed: 0
#send errors 0, #recv errors 0
```

***Now Ping PC2 with PC0***

**Check IPSec in R1:**

```
R1#show crypto ipsec sa

interface: GigabitEthernet0/1
Crypto map tag: VPN-MAP, local addr 10.1.1.2

protected vrf: (none)
local ident (addr/mask/prot/port): (192.168.1.0/255.255.255.0/0/0)
remote ident (addr/mask/prot/port): (192.168.3.0/255.255.255.0/0/0)
current_peer 10.2.2.2 port 500
PERMIT, flags={origin_is_acl,}
#pkts encaps: 3, #pkts encrypt: 3, #pkts digest: 0
#pkts decaps: 2, #pkts decrypt: 2, #pkts verify: 0
#pkts compressed: 0, #pkts decompressed: 0
#pkts not compressed: 0, #pkts compr. failed: 0
#pkts not decompressed: 0, #pkts decompress failed: 0
#send errors 1, #recv errors 0
```

## 8.1 Conclusion

This report is very useful for all those who want to learn fundamental knowledge about cryptography, configuration of firewall and configuration of VPN IPSec in computer security. This report helped me to know about the problems difficulties during configuration of network like firewall and VPN IPSec. I learned lots of things during the preparation of this lab report. While preparing this report I felt many problems, from those problem I gained the problem-solving skill. This report will more advantages for those who want to learn about different types of cryptography, configuration of firewall for the secure connection between local devices and server also advantage for learning VPN IPSec configuration.

# References

*Visual paradigm online - suite of powerful tools*. Visual Paradigm Online - Suite of Powerful Tools. (n.d.). https://online.visual-paradigm.com/.

What is cryptography in computer network? (n.d.). https://www.tutorialspoint.com/what-is-cryptography-in-computer-network.

Admin. (2020, October 10). *Cryptography in computer network*. Tutorial And Example. https://www.tutorialandexample.com/cryptography-in-computer-network/.

Data encryption standard. (n.d.). https://www.tutorialspoint.com/cryptography/data_encryption_standard.htm.