EXPERIMENT 1 AND 2

Name: Abhishek Chopra **UID: 2019130009**

Batch A **Subject: CSS**

Aim:

1)To implement Substitution, ROT 13, Transposition, Double Transposition, and Vernam Cipher in Scilab/C/Python/R.

2) Implement Diffie Hellman key exchange algorithm in Scilab/C/Python/R.

Theory:

Cryptography: In cryptography, encryption is the process of transforming information (referred to as plaintext) using an algorithm (called cipher) to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. The result of the process is encrypted information (in cryptography, referred to as cipher text). In many contexts, the word encryption also implicitly refers to the reverse process, decryption (e.g. “software for encryption” can typically also perform decryption), to make the encrypted information readable again (i.e. to make it unencrypted). Encryption is used to protect data in transit, for example data being transferred via networks (e.g. the Internet, e-commerce), mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines. There have been numerous reports of data in transit being intercepted in recent years/ Encrypting data in transit also helps to secure it as it is often difficult to physically secure all access to networks.

Substitution Technique: In cryptography, a substitution cipher is a method of encryption by which units of plaintext are replaced with ciphertext 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 polygraphic. A monoalphabetic cipher uses fixed substitution over the entire message, whereas a polyalphabetic cipher uses a number of substitutions at different times in the message, where a unit from the plaintext is mapped to one of several possibilities in the ciphertext and vice-versa.

Transposition Technique: In cryptography, a transposition cipher is a method of encryption by which the positions held by units of plaintext (which are commonly characters or groups of characters) are shifted according to a regular system, so that the ciphertext constitutes a permutation of the plaintext. That is, the order of the units is changed. Mathematically a bijective function is used on the characters' positions to encrypt and an inverse function to decrypt. In a columnar transposition, the message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled

order. Both the width of the rows and the permutation of the columns are usually defined by a keyword. For Traditional Crypto Methods and Key exchange/PV example, the word ZEBRAS is of length 6 (so the rows are of length 6), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be "6 3 2 4 1 5". In a regular columnar transposition cipher, any spare spaces are filled with nulls; in an irregular columnar transposition cipher, the spaces are left blank. Finally, the message is read off in columns, in the order specified by the keyword.

Double Transposition: A single columnar transposition could be attacked by guessing possible column lengths, writing the message out in its columns (but in the wrong order, as the key is not yet known), and then looking for possible anagrams. Thus to make it stronger, a double transposition was often used. This is simply a columnar transposition applied twice. The same key can be used for both transpositions, or two different keys can be used.

Vernam cipher: In modern terminology, a Vernam cipher is a symmetrical stream cipher in which the plaintext is XORed with a random or pseudo random stream of data (the "keystream") of the same length to generate the ciphertext. If the keystream is truly random and used only once, this is effectively a one-time pad. Substituting pseudorandom data generated by a cryptographically secure pseudorandom number generator is a common and effective construction for a stream cipher.

Diffie –Hellman Key exchange algorithm: The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher. Although Diffie–Hellman key agreement itself is an anonymous (non-authenticated) keyagreement protocol, it provides the basis for a variety of authenticated protocols, and is used to provide perfect forward secrecy in Transport Layer Security's ephemeral modes (referred to as EDH or DHE depending on the cipher suite).

Code:

```
print('Crypto Method Menu')

print('1 Substitution')

print('2 ROT 13')

print('3 Transpose')

print('4 Double Transposition')

print('5 Vernam Cipher')

print('6 Diffie – Hellman')

print()
```

```
a =(int)(input('Enter the Crypto Method: '))
print()

def encrypt(b, key):
  matrix = createMatrix(len(key), b)
  sequence = getSequence(key)
  ans = ""
  for num in range(len(sequence)):
    pos = sequence.index(num+1)
    for row in range(len(matrix)):
      if len(matrix[row]) > pos:
        ans += matrix[row][pos]
  return ans

def createMatrix(width, b):
  r = 0
  c = 0
  matrix = [[]]
  for pos, ch in enumerate(b):
    matrix[r].append(ch)
    c += 1
    if c >= width:
      c = 0
      r += 1
      matrix.append([])

  return matrix

def getSequence(key):
```

```
  sequence = []
  for pos, ch in enumerate(key):
    previousLetters = key[:pos]
    newNumber = 1
    for previousPos, previousCh in enumerate(previousLetters):
      if previousCh > ch:
        sequence[previousPos] += 1
      else:
        newNumber += 1
    sequence.append(newNumber)
  return sequence

def decrypt(message, keyword):
  matrix = createDecrMatrix(getSequence(keyword), message)

  plaintext = ""
  for r in range(len(matrix)):
    for c in range (len(matrix[r])):
      plaintext += matrix[r][c]
  return plaintext



def createDecrMatrix(keywordSequence, message):
  width = len(keywordSequence)
  height = len(message) // width
  if height * width < len(message):
    height += 1

  matrix = createEmptyMatrix(width, height, len(message))
```

```
  pos = 0
  for num in range(len(keywordSequence)):
    column = keywordSequence.index(num+1)

    r = 0
    while (r < len(matrix)) and (len(matrix[r]) > column):
      matrix[r][column] = message[pos]
      r += 1
      pos += 1

  return matrix



def createEmptyMatrix(width, height, length):
  matrix = []
  totalAdded = 0
  for r in range(height):
    matrix.append([])
    for c in range(width):
      if totalAdded >= length:
        return matrix
      matrix[r].append('')
      totalAdded += 1
  return matrix



def encryptS(b,k):
    newS = ''
```

```
    for i in range(len(b)):
        val = ord(b[i])
        dup = k
        d = val + k
        if val >= 97 and val <= 122:
            if d > 122:
                k -= (122 - val)
                k = k % 26
                newS += chr(96 + k)
            else:
                newS += chr(d)
            k = dup
        elif val >= 65 and val <= 90:
            if d > 90:
                k -= (90 - val)
                k = k % 26
                newS += chr(64 + k)
            else:
                newS += chr(d)
            k = dup
        else:
            newS += chr(d)
    return newS


def decryptS(b,k):
    s = ''
    for i in range(len(b)):
        val = ord(b[i])
        dup = k
```

```
        d = val-k
        if val >= 97 and val <= 122:
            if d < 97:
                k += (122-val)
                k = k%26
                s += chr(122 - k)
            else:
                s += chr(d)
            k = dup
        elif val >= 65 and val <= 90:
            if d < 65:
                k += (90 - val)
                k = k%26
                s += chr(90 - k)
            else:
                s += chr(d)
            k = dup
        else:
            s += chr(d)
    return s

def generateKey(string, key):
    key = list(key)
    if len(string) == len(key):
        return(key)
    else:
        for i in range(len(string) -
                       len(key)):
            key.append(key[i % len(key)])
```

```
    return("" . join(key))


def encryptVernam(b,final_key):
    b_list = []
    k_list = []
    final_l = []
    for c in b:
        b_list.append(ord(c) - 65)
    for d in final_key:
        k_list.append(ord(d) - 65)
    for i in range(0, len_b):
        final_l.append((b_list[i] + k_list[i]) % 26)
    ans = ''
    for i in range(0, len_b):
        ans = ans + chr(final_l[i] + 65)
    return ans



def decryptVername(encrypted,key):
    orig_text = []
    for i in range(len(encrypted)):
        d = (ord(encrypted[i])-ord(key[i])+26)%26
        d += ord('A')
        orig_text.append(chr(d))
    return ("".join(orig_text))

if a==1:
    b = input('Enter message: ')
    k = int(input('Number of positions to be shifted: '))
```

```
    s = encryptS(b,k)
    print('Encrypted Message: ',s)
    print('Decrypted Message: ',decryptS(s,k))
elif a==2:
    b = input('Enter message: ')
    newS = ''
    s = encryptS(b, 13)
    print('Encrypted Message: ', s)
    print('Decrypted Message: ', decryptS(s, 13))



elif a==3:
    b = input('Enter message: ')
    key = input('Enter key: ')
    encypted = (encrypt(b,key))
    print('Encrypted Message: ',encypted)
    decrypted = decrypt(encypted,key)

    print('Decrypted Message: ',decrypted)
elif a==4:
    b = input('Enter message: ')
    key = input('Enter key: ')
    encrypted = encrypt(encrypt(b,key),key)
    print(encrypted)
    decrypted = decrypt(decrypt(encrypted,key),key)

    print(decrypted)
elif a==5:
    b = input('Enter message: ').replace(" ", "")
```

```
    b.upper()
    len_b = len(b)
    key = input('Enter key: ')
    key.upper()
    final_key = generateKey(b,key)
    encrypted = (encryptVernam(b,final_key))
    decrypted = decryptVername(encrypted,final_key)
    print(encrypted)
    print(decrypted)
elif a==6:
    #P = 23
    #G = 9
    P = int(input('Enter Prime Number P: '))
    G = int(input('Enter Prime Number G: '))
    #print('The Value of P is :%d'%(P))
    #print('The Value of G is :%d'%(G))

    a = int(input('Enter Private KeyA for Alice: '))
    x = int(pow(G,a,P))

    b = int(input('Enter Private KeyB for Bob: '))
    y = int(pow(G,b,P))

    ka = int(pow(y,a,P))

    kb = int(pow(x,b,P))

    print('Secret key for the Alice is : %d'%(ka))
    print('Secret Key for the Bob is : %d'%(kb))
```

Output:

```
C:\Users\Abhishek\Documents\CSS\Exp 1>python exp1.py
Crypto Method Menu
1 Substitution
2 ROT 13
3 Transpose
4 Double Transposition
5 Vernam Cipher
6 Diffie - Hellman

Enter the Crypto Method: 1

Enter message: I am the Finance Secretary of the college
Number of positions to be shifted: 7
Encrypted Message:  P'ht'aol'Mpuhujl'Zljylahyf'vm'aol'jvsslnl
Decrypted Message:  I am the Finance Secretary of the college
```

```
C:\Users\Abhishek\Documents\CSS\Exp 1>python exp1.py
Crypto Method Menu
1 Substitution
2 ROT 13
3 Transpose
4 Double Transposition
5 Vernam Cipher
6 Diffie - Hellman

Enter the Crypto Method: 2

Enter message: I am responsible for the finances of the college and maintain statements
Encrypted Message:  V-nz-erfcbafvoyr-sbe-gur-svanaprf-bs-gur-pbyyrtr-naq-znvagnva-fgngrzragf
Decrypted Message:  I am responsible for the finances of the college and maintain statements

C:\Users\Abhishek\Documents\CSS\Exp 1>python exp1.py
Crypto Method Menu
1 Substitution
2 ROT 13
3 Transpose
4 Double Transposition
5 Vernam Cipher
6 Diffie - Hellman

Enter the Crypto Method: 3

Enter message: I am also a part of the Students Council
Enter key: great
Encrypted Message:  moaf eCiaspoed cIaattttu l  husn  r Snol
Decrypted Message:  I am also a part of the Students Council
```

```
C:\Users\Abhishek\Documents\CSS\Exp 1>python exp1.py
Crypto Method Menu
1 Substitution
2 ROT 13
3 Transpose
4 Double Transposition
5 Vernam Cipher
6 Diffie - Hellman

Enter the Crypto Method: 4

Enter message: It is my responsibilty to convey the greivances of the students to the college authorities
Enter key: excellent
trir tehcn oo nfvhcevipmightegteet nyeutroyssie ss  eoo tacso oayti sll uti  bthsIteed nel
It is my responsibilty to convey the greivances of the students to the college authorities

C:\Users\Abhishek\Documents\CSS\Exp 1>python exp1.py
Crypto Method Menu
1 Substitution
2 ROT 13
3 Transpose
4 Double Transposition
5 Vernam Cipher
6 Diffie - Hellman

Enter the Crypto Method: 5

Enter message: I AM AN INTERFACE BETWEEN THE TWO
Enter key: WOW
EOIWBEJHANTWYSXAHSASJPVAPKK
IAMANINTERFACEBETWEENTHETWO
```

```
C:\Users\Abhishek\Documents\CSS\Exp 1>python exp1.py
Crypto Method Menu
1 Substitution
2 ROT 13
3 Transpose
4 Double Transposition
5 Vernam Cipher
6 Diffie - Hellman

Enter the Crypto Method: 6

Enter Prime Number P: 23
Enter Prime Number G: 7
Enter Private KeyA for Alice: 13
Enter Private KeyB for Bob: 17
Secret key for the Alice is : 7
Secret Key for the Bob is : 7

C:\Users\Abhishek\Documents\CSS\Exp 1>
```

Github Link:

https://github.com/AbhishekC20001/CSS-Lab-2019130009

Conclusion:

In the case of substitution cipher the key space is only 26 for which the attacker might take only a short amount of time to break or to try all the possible combinations of the shift variable. Hence it is not at all a reliable method/algorithm to transfer and protect data.

The case of ROT13 is also similar to the above substitution cipher just that the shift variable takes the value of 13 and letters of the alphabet are offset 13 places. Hence this type of algorithm though a standard one yet is not secure at all.

The transposition here implemented by me is the columnar one where the message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order. Both the length of the rows and the permutation of the columns are usually defined by a keyword variable. The columnar transposition technique of encryption is easy to understand and implement but still complex to break by brutal force attack or cryptanalysis. The hacker or the intermediate person cannot break this code unless otherwise he knows the method. Hence this algorithm is definitely better than the above two but also has some disadvantages that it is prone to slowness of encryption and also sensitive to lossy data or increased characters in the plain text.

Similar to the above columnar transposition the double transposition has also been implemented just that the same steps have been repeated twice for more security and hence protection. This algorithm is way better to break than the above ones.

Lastly, in the Vernam Cipher method since its key-exchange procedure uses real random number generation and secure key distribution, the Vernam Cipher with a one-time pad is said to be an unbreakable symmetric encryption technique. Each plaintext character from a message is mixed with one character from a key stream, according to the Vernam Cipher. If we utilise a completely random key stream, we'll get a truly random ciphertext that has no relation to the plaintext. Hence it proves to be a good encryption technique.

Diffie Hellman is a key exchange method that allows 2 parties to jointly establish a mutual secret key over a public channel without it being transmitted over the Internet, hence it is a fundamental part of securely exchanging data online. I found that as long as it is implemented alongside an appropriate authentication method and the numbers have been selected

properly, it is not considered vulnerable to attack. Hence if the value of ‘the first prime or p’ is taken to be a safe prime and the value of ‘the second prime or g’ is taken to be a primitive root mod p then the chances of the leaking of message is close to impossible hence the most security. Hence I infer that while it is really tough for someone snooping the network to decrypt the data and get the keys, it is still possible if the numbers generated are not entirely random.