Practical 1:

Implement Caesar cipher encryption-decryption.

Code:

```
plaintext = input("Enter the value of Plaintext : ")
key = int(input("Enter the value of key : "))
# Encryption of Plaintext to Cipher text
def encrypt(p,k):
    cipher=""
    for i in p:
        if i.isupper():
            cipher += chr((((ord(i)+k)-65)% 26)+65)
        else:
            cipher += chr((((ord(i)+k)-97)% 26)+97)
    return cipher
ciphertext = encrypt(plaintext,key)
print("Ciphertext for the plain text :",plaintext," is ",ciphertext)
# Decryption of Cipher text to Plain text
def decrypt(c,k):
    plain=""
    for i in c:
        if i.isupper():
            plain += chr((((ord(i)-k)-65)% 26)+65)
        else:
            plain += chr((((ord(i)-k)-97)% 26)+97)
    return plain
decrypted_plain = decrypt(ciphertext,key)
print("Plaintext for the cipher text :",ciphertext," is ",decrypted_plain)
# CryptAnalysis(Brute Force Attack) of caesar cipher to get key from
thecipher text
def crypt_analysis(c,p):
    for j in range(1,26):
        plain=""
        for i in c:
            if i.isupper():
```

```
                plain += chr((((ord(i)-j)-65)% 26)+65)
            else:
                plain += chr((((ord(i)-j)-97)% 26)+97)
        if(plain == p):
            break
    return j
k = crypt_analysis(ciphertext,plaintext)
print("Key after crypt_analysis : ",k)
```

Output:

Enter the value of Plaintext: security

Enter the value of key: 2

Ciphertext for the plain tex: security is ugewtkva

Plaintext for the cipher text: ugewtkva is security

Key after crypt_analysis: 2

Practical 2

Implement Monoalphabetic cipher encryption-decryption.

Code:

```
s=input("enter the string:")
key=['d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u',
'v','w','x','y','z','a','b','c']
def encrypt(s):
    result=""
    for i in range(len(s)):
        char=s[i]
        r=(ord(char)-97)
        result+=key[r]
    return result
print(encrypt(s))
m=encrypt(s)
def decrypt(m):
    result=""
    for j in range(len(m)):
        for i in range(len(key)):
            char=m[j]
            if(key[i]==m[j]):
                r=(i+97)
                result+=chr(r)
    return result
print(decrypt(m))
```

Output:

enter the string: abcd

defg

abcd

Practical 3

Implement Playfair cipher encryption-decryption.

Code:

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

p=input("Enter the text: ")

k=0

i=0

plain=""

while (k <len(p)-1 ):

        if(p[k]==p[k+1] ):

            plain+=p[k]

            plain+='x'

            k=k+1

        else:

            plain+=p[k]

            plain+=p[k+1]

            k=k+2

if (p[len(p)-1]!= plain[len(plain)-1] or  len(p)%2==0):

    plain+=p[k]

if (len(plain)%2!=0):

    plain+='x'

p=plain

print("Intermediate text:",p)

cipher=""
```

```
for k in range(0,len(p),2):
    for i in range (0,5):
        for j in range(0,5):
            if (play[i][j]==p[k] or (p[k]=='j'and play[i][j]=='i')):
                row1=i
                column1=j
            if (play[i][j]==p[k+1]or(p[k]=='j'and play[i][j]=='i')):
                row2=i
                column2=j
    if (row1==row2):
        cipher+=(play[row1][(column1+1)%5])
        cipher+=(play[row2][(column2+1)%5])
    elif(column1==column2):
        cipher+=(play[(row1+1)%5][column1])
        cipher+=(play[(row2+1)%5][column1])
    else:
        cipher+=play[row1][column2]
        cipher+=play[row2][column1]
print("The cipher is :",cipher)
```

Output:

Enter the text: hello

Intermediate text: helxlo

The cipher is: kcnvmp

Practical 4

Implement Polyalphabetic cipher encryption-decryption.

Code:

```
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 cipherText(string, key):
    cipher_text = []
    for i in range(len(string)):
        x = (ord(string[i]) +
            ord(key[i])) % 26
        x += ord('A')
        cipher_text.append(chr(x))
    return("" . join(cipher_text))
def originalText(cipher_text, key):
    orig_text = []
    for i in range(len(cipher_text)):
        x = (ord(cipher_text[i]) -
            ord(key[i]) + 26) % 26
        x += ord('A')
        orig_text.append(chr(x))
    return("" . join(orig_text))
if __name__ == "__main__":
    string =input("enter the string:")
    keyword = input("enter the keyword:")
    key = generateKey(string, keyword)
    cipher_text = cipherText(string,key)
    print("Ciphertext :", cipher_text)
```

```
    print("Original/Decrypted Text :",
        originalText(cipher_text, key))
```

Output:

enter the string: INFORMATION

enter the keyword: ONE

Ciphertext: WAJCEQOGMCA

Original/Decrypted Text: INFORMATION

Practical 5

Implement Hill cipher encryption-decryption.

Code:

```
#ENCRYPTION
keyMatrix = [[0] * 3 for i in range(3)]
messageVector = [[0] for i in range(3)]
cipherMatrix = [[0] for i in range(3)]
def getKeyMatrix(key):
    k = 0
    for i in range(3):
        for j in range(3):
         keyMatrix[i][j] = ord(key[k]) % 65
         k += 1
def encrypt(messageVector):
    for i in range(3):
        for j in range(1):
         cipherMatrix[i][j] = 0
         for x in range(3):
            cipherMatrix[i][j] += (keyMatrix[i][x] * messageVector[x][j])
         cipherMatrix[i][j] = cipherMatrix[i][j] % 26
def HillCipher(message, key):
    getKeyMatrix(key)
    for i in range(3):
        messageVector[i][0] = ord(message[i]) % 65
    encrypt(messageVector)
    CipherText = []
    for i in range(3):
        CipherText.append(chr(cipherMatrix[i][0] + 65))
    print("Plaintext: ",message)
    print("Ciphertext: ", "".join(CipherText))
def main():
    message = "EAT"
    key = "GYBNQKURP"
```

```
    HillCipher(message, key)
if __name__ == "__main__":
    main()
```

Output:

Plaintext: EAT

Ciphertext: RIB

Code:

```
#DECRYPTION
import numpy
def create_matrix_from(key):
    m=[[0] * 3 for i in range(3)]
    for i in range(3):
        for j in range(3):
         m[i][j] = ord(key[3*i+j]) % 65
    return m
def encrypt(P, K):
    C=[0,0,0]
    C[0] = (K[0][0]*P[0] + K[1][0]*P[1] + K[2][0]*P[2]) % 26
    C[1] = (K[0][1]*P[0] + K[1][1]*P[1] + K[2][1]*P[2]) % 26
    C[2] = (K[0][2]*P[0] + K[1][2]*P[1] + K[2][2]*P[2]) % 26
    return C
def Hill(message, K):
    cipher_text = []
    for i in range(0,len(message), 3):
        P=[0, 0, 0]
        for j in range(3):
         P[j] = ord(message[i+j]) % 65
        C = encrypt(P,K)
        for j in range(3):
         cipher_text.append(chr(C[j] + 65))
    return "".join(cipher_text)
def MatrixInverse(K):
```

```
    det = int(numpy.linalg.det(K))
    det_multiplicative_inverse = pow(det, -1, 26)
    K_inv = [[0] * 3 for i in range(3)]
    for i in range(3):
        for j in range(3):
         Dji = K
         Dji = numpy.delete(Dji, (j), axis=0)
         Dji = numpy.delete(Dji, (i), axis=1)
         det = Dji[0][0]*Dji[1][1] - Dji[0][1]*Dji[1][0]
         K_inv[i][j] = (det_multiplicative_inverse * pow(-1,i+j) * det) % 26
    return K_inv
if __name__ == "__main__":
    message = "MYSECRETMESSAGE"
    key = "RRFVSVCCT"
    K = create_matrix_from(key)
    cipher_text = Hill(message, K)
    print ('Cipher text: ', cipher_text)
    K_inv = MatrixInverse(K)

    plain_text = Hill(cipher_text, K_inv)
    print ('Plain text: ', plain_text)
```

Output:

Cipher text: QWWOIVXSXOMMEMU

Plain text: MYSECRETMESSAGE

Practical 6

To implement Simple DES or AES.

Code:

```
def hex2bin(s):
    mp = {'0': "0000",
        '1': "0001",
        '2': "0010",
        '3': "0011",
        '4': "0100",
        '5': "0101",
        '6': "0110",
        '7': "0111",
        '8': "1000",
        '9': "1001",
        'A': "1010",
        'B': "1011",
        'C': "1100",
        'D': "1101",
        'E': "1110",
        'F': "1111"}
    bin = ""
    for i in range(len(s)):
        bin = bin + mp[s[i]]
    return bin
# Binary to hexadecimal conversion
def bin2hex(s):
    mp = {"0000": '0',
        "0001": '1',
        "0010": '2',
        "0011": '3',
        "0100": '4',
        "0101": '5',
        "0110": '6',
```

```
        "0111": '7',
        "1000": '8',
        "1001": '9',
        "1010": 'A',
        "1011": 'B',
        "1100": 'C',
        "1101": 'D',
        "1110": 'E',
        "1111": 'F'}
    hex = ""
    for i in range(0, len(s), 4):
        ch = ""
        ch = ch + s[i]
        ch = ch + s[i + 1]
        ch = ch + s[i + 2]
        ch = ch + s[i + 3]
        hex = hex + mp[ch]
    return hex
# Binary to decimal conversion
def bin2dec(binary):
    binary1 = binary
    decimal, i, n = 0, 0, 0
    while(binary != 0):
        dec = binary % 10
        decimal = decimal + dec * pow(2, i)
        binary = binary//10
        i += 1
    return decimal
# Decimal to binary conversion
def dec2bin(num):
    res = bin(num).replace("0b", "")
    if(len(res) % 4 != 0):
        div = len(res) / 4
        div = int(div)
```

```
        counter = (4 * (div + 1)) - len(res)
        for i in range(0, counter):
            res = '0' + res
    return res
# Permute function to rearrange the bits
def permute(k, arr, n):
    permutation = ""
    for i in range(0, n):
        permutation = permutation + k[arr[i] - 1]
    return permutation
# shifting the bits towards left by nth shifts
def shift_left(k, nth_shifts):
    s = ""
    for i in range(nth_shifts):
        for j in range(1, len(k)):
            s = s + k[j]
        s = s + k[0]
        k = s
        s = ""
    return k
# calculating xow of two strings of binary number a and b
def xor(a, b):
    ans = ""
    for i in range(len(a)):
            if a[i] == b[i]:
                ans = ans + "0"
            else:
                ans = ans + "1"
    return ans
# Table of Position of 64 bits at initial level: Initial Permutation Table
initial_perm = [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]
# Expansion D-box Table
exp_d = [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]
# Straight Permutation Table
per = [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]
# S-box Table
sbox = [[[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]]]
# Final Permutation Table
final_perm = [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]
def encrypt(pt, rkb, rk):
    pt = hex2bin(pt)
# Initial Permutation
    pt = permute(pt, initial_perm, 64)
    print("After initial permutation", bin2hex(pt))
# Splitting
    left = pt[0:32]
    right = pt[32:64]
    for i in range(0, 16):
        # Expansion D-box: Expanding the 32 bits data into 48 bits
        right_expanded = permute(right, exp_d, 48)
        # XOR RoundKey[i] and right_expanded
        xor_x = xor(right_expanded, rkb[i])
        # S-boxex: substituting the value from s-box table by calculating row
and column
        sbox_str = ""
        for j in range(0, 8):
            row = bin2dec(int(xor_x[j * 6] + xor_x[j * 6 + 5]))
            col = bin2dec(
                int(xor_x[j * 6 + 1] + xor_x[j * 6 + 2] + xor_x[j * 6 + 3] +
xor_x[j * 6 + 4]))
            val = sbox[j][row][col]
            sbox_str = sbox_str + dec2bin(val)
        # Straight D-box: After substituting rearranging the bits
        sbox_str = permute(sbox_str, per, 32)
        # XOR left and sbox_str
        result = xor(left, sbox_str)
        left = result
        # Swapper
        if(i != 15):
            left, right = right, left
        print("Round ", i + 1, " ", bin2hex(left),
            " ", bin2hex(right), " ", rk[i])
# Combination
```

```
    combine = left + right
# Final permutation: final rearranging of bits to get cipher text
    cipher_text = permute(combine, final_perm, 64)
    return cipher_text
pt = "123456ABCD132536"
key = "AABB09182736CCDD"
# Key generation
# --hex to binary
key = hex2bin(key)
# --parity bit drop table
keyp = [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]
# getting 56 bit key from 64 bit using the parity bits
key = permute(key, keyp, 56)
# Number of bit shifts
shift_table = [1, 1, 2, 2,
            2, 2, 2, 2,
            1, 2, 2, 2,
            2, 2, 2, 1]
# Key- Compression Table : Compression of key from 56 bits to 48 bits
key_comp = [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]
```

```
# Splitting
left = key[0:28] # rkb for RoundKeys in binary
right = key[28:56] # rk for RoundKeys in hexadecimal
rkb = []
rk = []
for i in range(0, 16):
# Shifting the bits by nth shifts by checking from shift table
    left = shift_left(left, shift_table[i])
    right = shift_left(right, shift_table[i])
# Combination of left and right string
    combine_str = left + right
# Compression of key from 56 to 48 bits
    round_key = permute(combine_str, key_comp, 48)
    rkb.append(round_key)
    rk.append(bin2hex(round_key))
print("Encryption")
cipher_text = bin2hex(encrypt(pt, rkb, rk))
print("Cipher Text : ", cipher_text)
print("Decryption")
rkb_rev = rkb[::-1]
rk_rev = rk[::-1]
text = bin2hex(encrypt(cipher_text, rkb_rev, rk_rev))
print("Plain Text : ", text)
```

Output:

Encryption

After initial permutation 14A7D67818CA18AD

Round 1 18CA18AD 5A78E394 194CD072DE8C

Round 2 5A78E394 4A1210F6 4568581ABCCE

Round 3 4A1210F6 B8089591 06EDA4ACF5B5

Round 4 B8089591 236779C2 DA2D032B6EE3

Round 5 236779C2 A15A4B87 69A629FEC913

Round 6 A15A4B87 2E8F9C65 C1948E87475E

Round 7 2E8F9C65 A9FC20A3 708AD2DDB3C0

Round 8 A9FC20A3 308BEE97 34F822F0C66D

Round 9 308BEE97 10AF9D37 84BB4473DCCC

Round 10 10AF9D37 6CA6CB20 02765708B5BF

Round 11 6CA6CB20 FF3C485F 6D5560AF7CA5

Round 12 FF3C485F 22A5963B C2C1E96A4BF3

Round 13 22A5963B 387CCDAA 99C31397C91F

Round 14 387CCDAA BD2DD2AB 251B8BC717D0

Round 15 BD2DD2AB CF26B472 3330C5D9A36D

Round 16 19BA9212 CF26B472 181C5D75C66D

Cipher Text : C0B7A8D05F3A829C

Decryption

After initial permutation 19BA9212CF26B472

Round 1 CF26B472 BD2DD2AB 181C5D75C66D

Round 2 BD2DD2AB 387CCDAA 3330C5D9A36D

Round 3 387CCDAA 22A5963B 251B8BC717D0

Round 4 22A5963B FF3C485F 99C31397C91F

Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3

Round 6 6CA6CB20 10AF9D37 6D5560AF7CA5

Round 7 10AF9D37 308BEE97 02765708B5BF

Round 8 308BEE97 A9FC20A3 84BB4473DCCC

Round 9 A9FC20A3 2E8F9C65 34F822F0C66D

Round 10 2E8F9C65 A15A4B87 708AD2DDB3C0

Round 11 A15A4B87 236779C2 C1948E87475E

Round 12 236779C2 B8089591 69A629FEC913

Round 13 B8089591 4A1210F6 DA2D032B6EE3

Round 14 4A1210F6 5A78E394 06EDA4ACF5B5

Round 15 5A78E394 18CA18AD 4568581ABCCE

Round 16 14A7D678 18CA18AD 194CD072DE8C

Plain Text : 123456ABCD132536

Practical 7

Implement Diffi-Hellmen key exchange Method.

Code:

```
from random import randint
if __name__ == '__main__':
          P = 23
          G = 9
          print('The Value of P is :%d'%(P))
          print('The Value of G is :%d'%(G))
          a = 4
          print('The Private Key a for Alice is :%d'%(a))

x = int(pow(G,a,P))
          b = 3
          print('The Private Key b for Bob is :%d'%(b))
          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:

The Value of P is :23

The Value of G is :9

The Private Key a for Alice is :4

The Private Key b for Bob is :3

Secret key for the Alice is : 9

Secret Key for the Bob is : 9

Practical 8

Implement RSA encryption-decryption algorithm.

Code:

```
from decimal import Decimal
  def gcd(k,totient):
    while totient!=0:
        c=k%totient
        k=totient
        totient=c
    return k
#input variables
d=0
p = eval(input("enter value of p:"))
q = eval(input("enter value of q:"))
message = eval(input("message"))
#calculate n
n = p*q
#calculate totient
totient = (p-1)*(q-1)
#calculate K
for k in range(2,totient):
    if gcd(k,totient)== 1:
        break
for i in range(1,10):
    x = 1 + i*totient
    if x % k == 0:
        d = int(x/k)
        break
local_cipher = Decimal(0)
local_cipher =pow(message,k)
cipher_text = local_cipher % n
decrypt_t = Decimal(0)
decrypt_t= pow(cipher_text,d)
```

```
decrpyted_text = decrypt_t % n
  print('n = '+str(n))
print('k = '+str(k))
print('totient = '+str(totient))
print('d = '+str(d))
print('cipher text = '+str(cipher_text))
print('decrypted text = '+str(decrpyted_text))
```

Output:

enter value of p:78

enter value of q:89

message44

n = 6942

k = 3

totient = 6776

d = 2259

cipher text = 1880

decrypted text = 44

Practical 9

Write a program to generate SHA-1 hash.

Code:

```
import hashlib

str ="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

result = hashlib.sha1(str.encode())

print("The hexadecimal equivalent of SHA1 is : ",result.hexdigest())
```

Output:

The hexadecimal equivalent of SHA1 is : 80256f39a9d308650ac90d9be9a72a9562454574

Practical 10

Implement a digital signature algorithm.

Code:

```
# Function to find gcd
# of two numbers
def euclid(m, n):
    if n == 0:
        return m
    else:
        r = m % n
        return euclid(n, r)
# Program to find
# Multiplicative inverse
def exteuclid(a, b):
    r1 = a
    r2 = b
    s1 = int(1)
    s2 = int(0)
    t1 = int(0)
    t2 = int(1)
    while r2 > 0:
        q = r1//r2
        r = r1-q * r2
        r1 = r2
        r2 = r
        s = s1-q * s2
        s1 = s2
        s2 = s
        t = t1-q * t2
        t1 = t2
        t2 = t
    if t1 < 0:
        t1 = t1 % a
```

```
    return (r1, t1)
# Enter two large prime
# numbers p and q
p = 283
q = 47
n = p * q
Pn = (p-1)*(q-1)
# Generate encryption key
# in range 1<e<Pn
key = []
for i in range(2, Pn):
    gcd = euclid(Pn, i)
    if gcd == 1:
        key.append(i)
# Select an encryption key
# from the above list
e = int(313)

# Obtain inverse of
# encryption key in Z_Pn
r, d = exteuclid(Pn, e)
if r == 1:
    d = int(d)
    print("decryption key is: ", d)
else:
    print("Multiplicative inverse for\
    the given encryption key does not \
    exist. Choose a different encryption key ")
# Enter the message to be sent
M = 19070
# Signature is created by Alice
S = (M**d) % n
# Alice sends M and S both to Bob
# Bob generates message M1 using the
```

```
# signature S, Alice's public key e
# and product n.
M1 = (S**e) % n
# If M = M1 only then Bob accepts
# the message sent by Alice.
if M == M1:
    print("As M = M1, Accept the\
    message sent by Alice")
else:
    print("As M not equal to M1,\
    Do not accept the message\
    sent by Alice ")
```

Output:

decryption key is: 373

As M not equal to M1, Do not accept the message sent by Alice

Practical 11

Perform various encryption-decryption techniques with cryptool.

What is Crypt-tool?

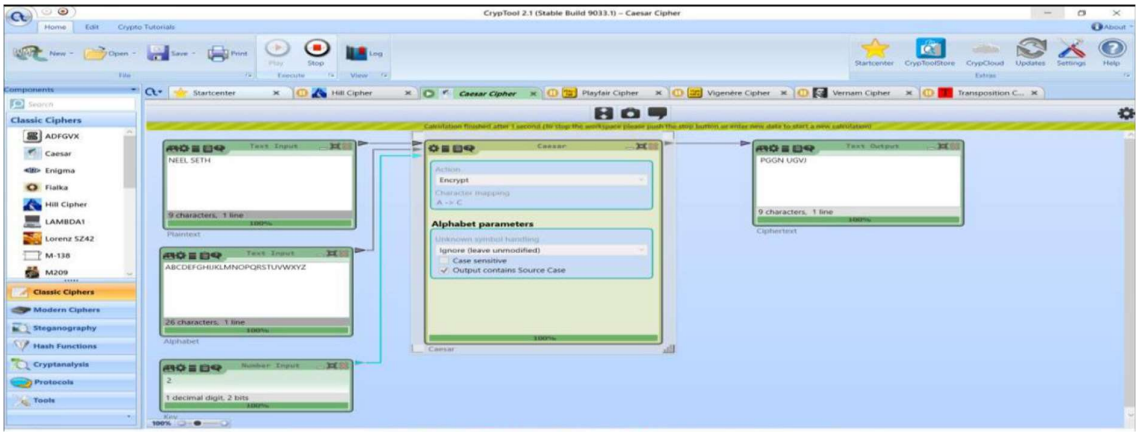
- Crypt-Tool is an open-source project that focuses on free e-learning software.
- A freeware program with graphical user interface (GUI).
- A tool for applying and analysing cryptographic algorithms.
- With extensive online help, it's understandable without deep crypto knowledge.
- Contains nearly all state-of-the-art crypto algorithms.
- “Playful” introduction to modern and classical cryptography.
- Not a “hacker" tool.

Here are some examples:-

Caesar Cipher:

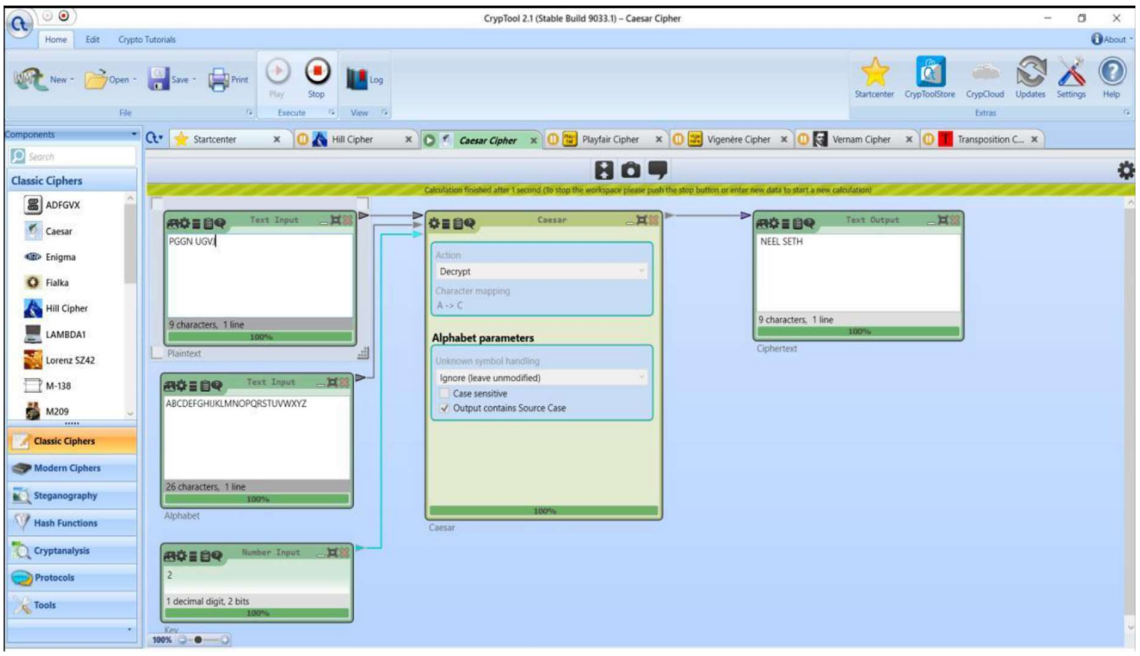
The Caesar Cipher is a very basic substitution cipher – each letter is replaced by another. It also includes an offset that determines how many letters away from the original the substitutedletter would be. So, for instance, if the offset were 2, A would become C and W would become Y.

Encryption:



Information Security (3170720)

Decryption:

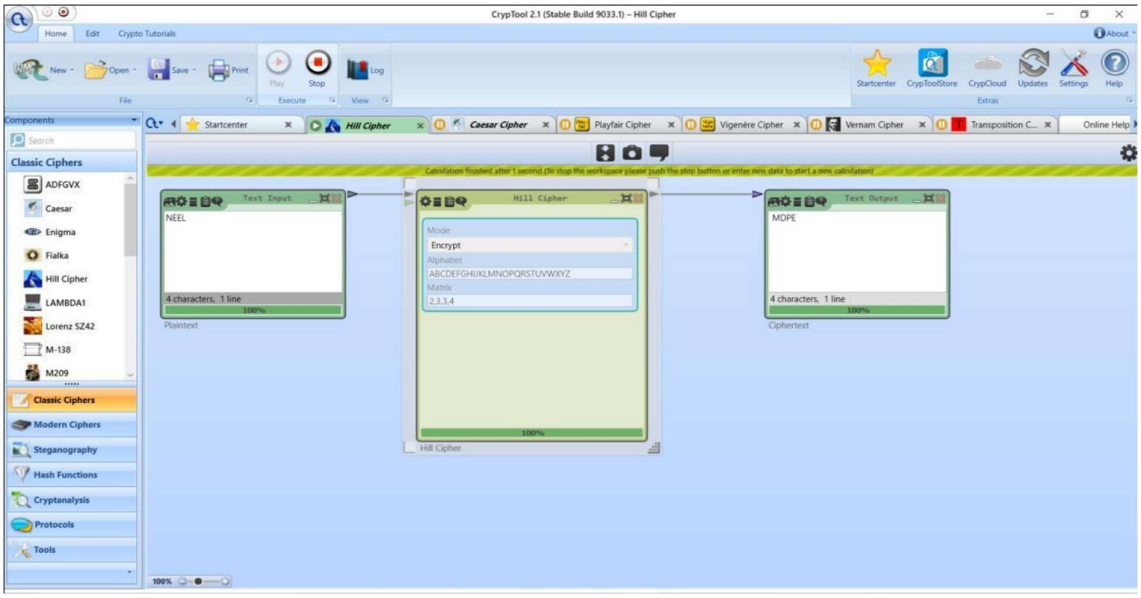


Hill Cipher:

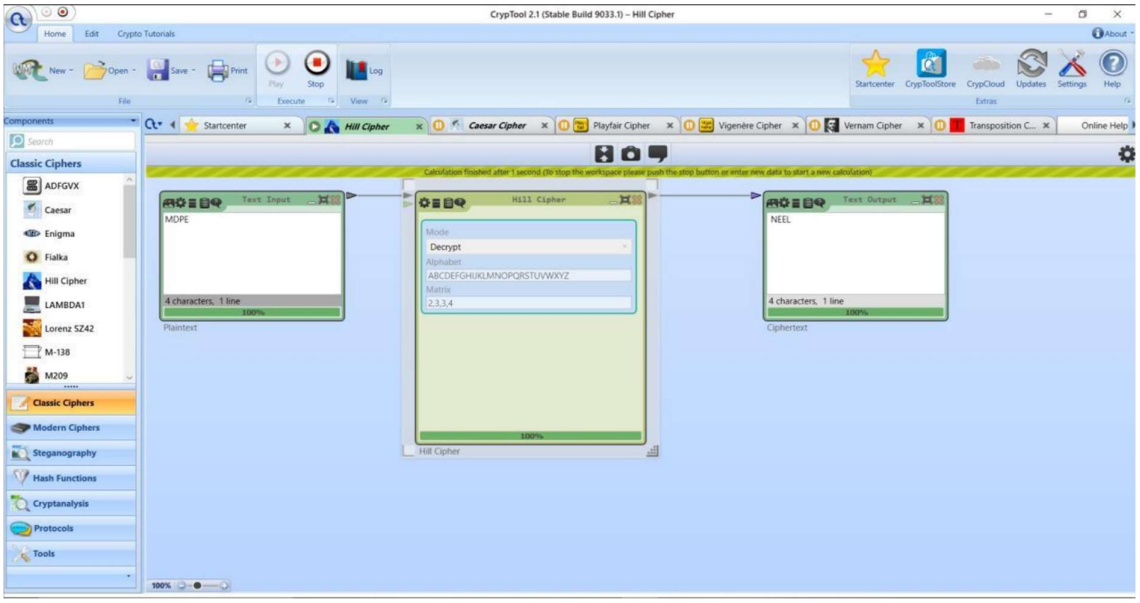
- Hill cipher is a polygraphic substitution cipher based on linear algebra. Each letter is represented by a number modulo 26.
- Often the simple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher.
- To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26.
- To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.
- The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26).

Information Security (3170720)

Encryption:



Decryption:



Information Security (3170720)

Playfair Cipher:

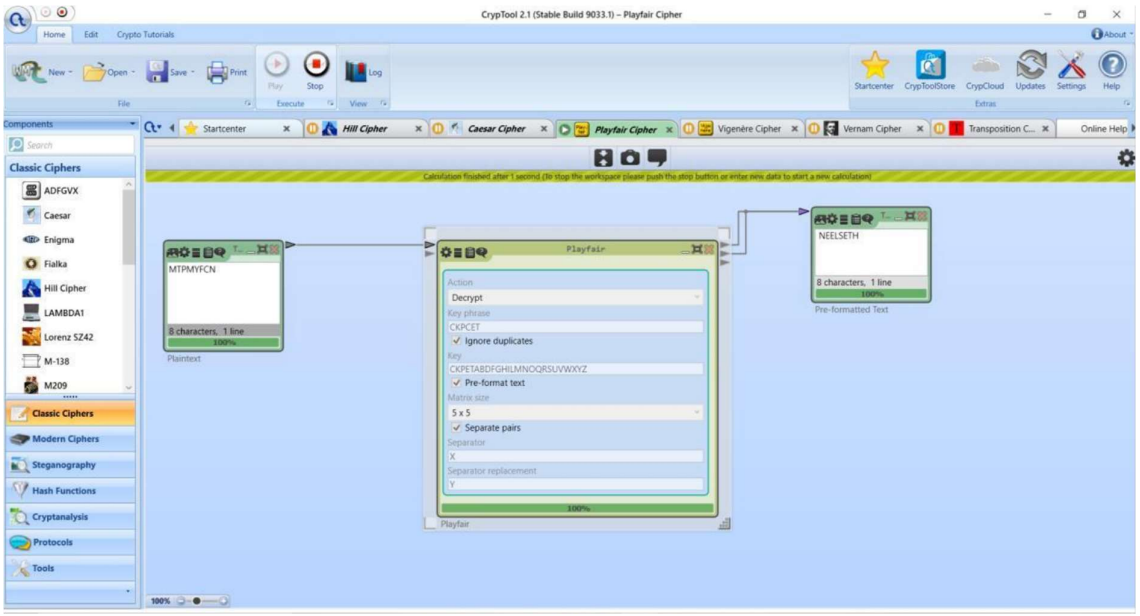
• The Playfair cipher was the first practical digraph substitution cipher.

• The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfairwho promoted the use of the cipher.

• In playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead ofa single alphabet.

Encryption:

Decryption:



Information Security (3170720)

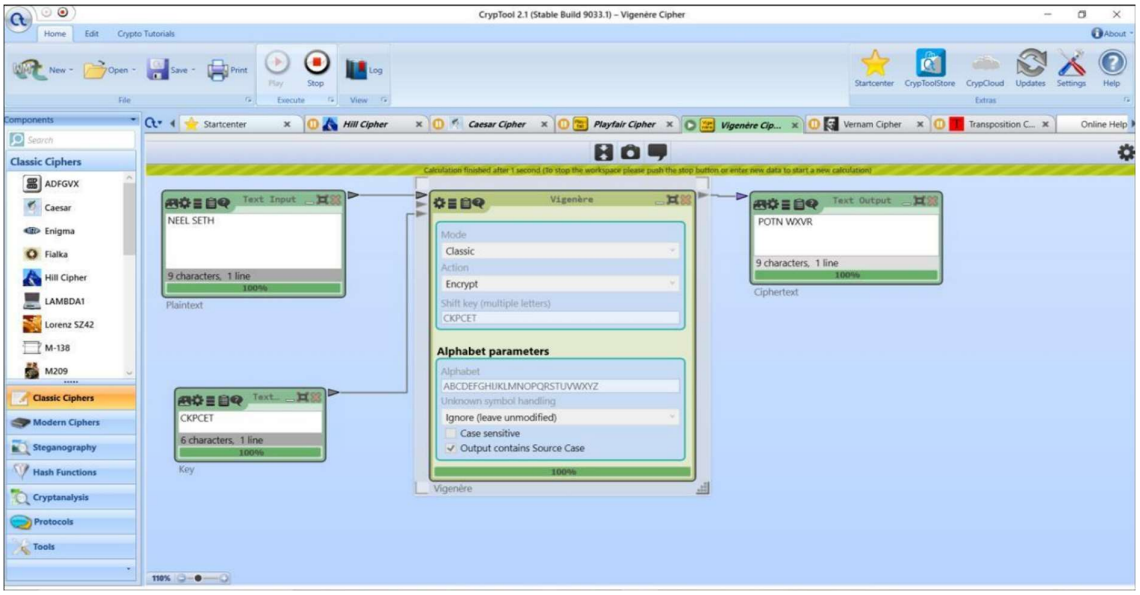
Vigenere Cipher:

• Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets .The encryption of the original text is done using the Vigenère square or Vigenère table.

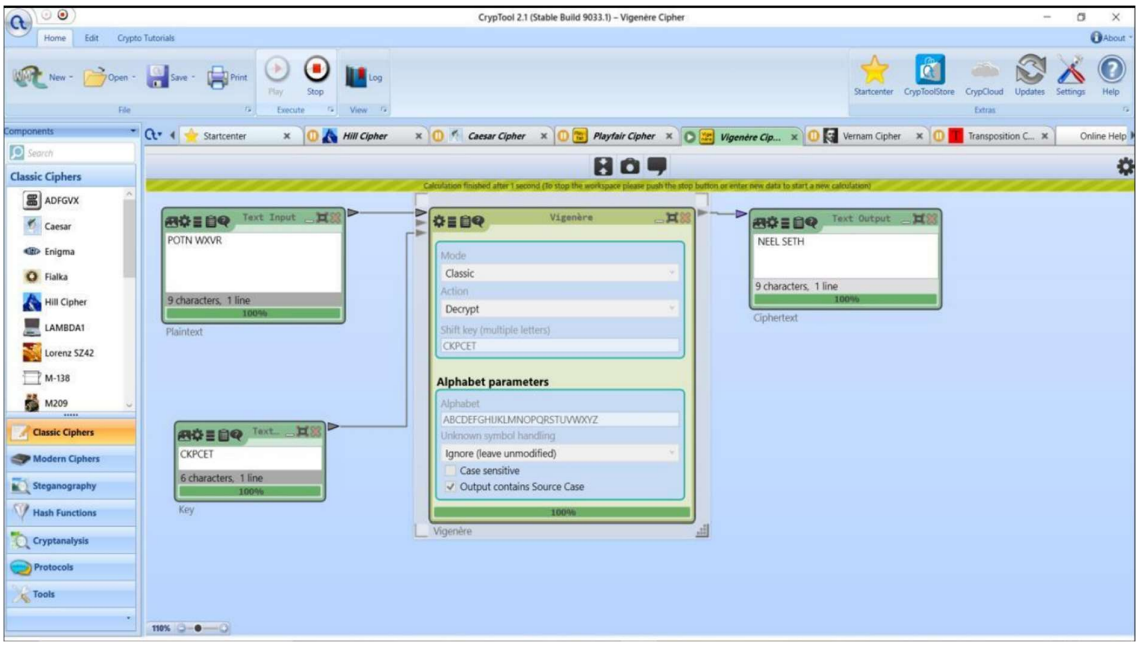
• The table consists of the alphabets written out 26 times in different rows, each alphabet shiftedcyclically to the left compared to the previous alphabet, corresponding to the 26 possible CaesarCiphers.

• At different points in the encryption process, the cipher uses a different alphabet from one ofhe rows.

Encryption:



Decryption:



Practical 12

Study and use the Wireshark for the various network protocols.

History of Wireshark:

Wireshark was started with the intention of developing a tool for closely analysing network packets. It was started by Gerald Combez in 1997. Its initial name was Ethereal. It was initially released in July 1998 as version 0.2.0. Due to the support it got from the developer community, it grew rapidly and was released as version 1.0 in 2008, almost two years after it was renamed to Wireshark

What is Wireshark?

• Wireshark is an open-source packet analyzer, which is used for education, analysis, software development, communication protocol development, and network troubleshooting.

• It is used to track the packets so that each one is filtered to meet our specific needs. It is commonly called a sniffer, network protocol analyzer, and network analyzer. It is also used by network security engineers to examine security problems.

• Wireshark is a free to use application which is used to apprehend the data back and forth. It is often called a free packet sniffer computer application. It puts the network card into an unselective mode, i.e., to accept all the packets which it receives.

Uses of Wireshark:

Wireshark can be used in the following ways:

1. It is used by network security engineers to examine security problems.

2. It allows the users to watch all the traffic being passed over the network.

3. It is used by network engineers to troubleshoot network issues.

4. It also helps to troubleshoot latency issues and malicious activities on your network.

5. It can also analyze dropped packets.

6. It helps us to know how all the devices like laptop, mobile phones, desktop, switch, routers, etc., communicate in a local network or the rest of the world.

Uses of Wireshark:

• Wireshark is similar to tcpdump in networking. Tcpdump is a common packet analyzer which allows the user to display other packets and TCP/IP packets, being transmitted and received over a network attached to the computer. It has a graphic end and some sorting and filtering functions. Wireshark users can see all the traffic passing through the network.

• Wireshark can also monitor the unicast traffic which is not sent to the network's MAC address interface. But the switch does not pass all the traffic to the port. Hence, the promiscuous mode is not sufficient to see all the traffic. The various network taps or port mirroring is used to extend capture at any point.

• Port mirroring is a method to monitor network traffic. When it is enabled, the switch sends the copies of all the network packets present at one port to another port.

Features of Wireshark

• It is multi-platform software, i.e., it can run on Linux, Windows, OS X, FreeBSD, NetBSD, etc.

• It is a standard three-pane packet browser.

• It performs deep inspection of hundreds of protocols.

• It often involves live analysis, i.e., from the different types of the network like the Ethernet, loopback, etc., we can read live data.

• It has sort and filter options which makes it easy for the user to view the data.

• It is also useful in VoIP analysis.

• It can also capture raw USB traffic.

• Various settings, like timers and filters, can be used to filter the output. o It can only capture packet on the PCAP (an application programming interface used to capture the network) supported networks.

• Wireshark supports a variety of well-documented capture file formats such as the PcapNg and Libpcap. These formats are used for storing the captured data.

• It is the no.1 piece of software for its purpose. It has countless applications ranging from the tracing down, unauthorized traffic, firewall settings, etc.

Information Security (3170720)

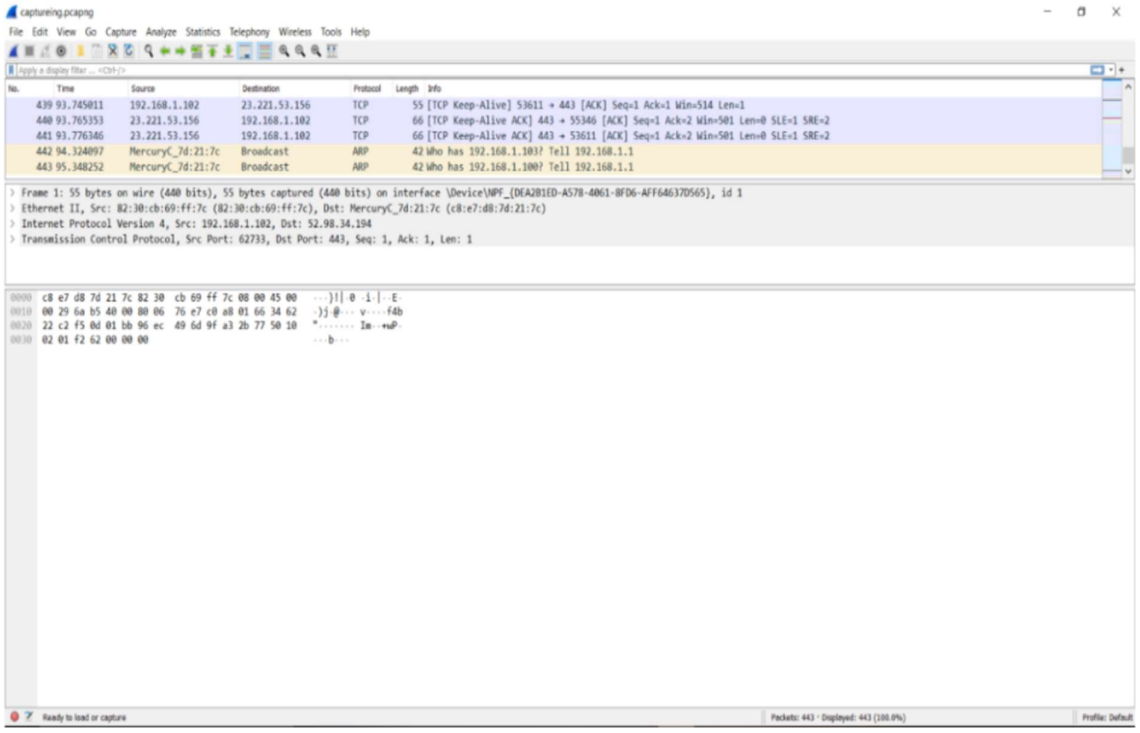
How to view and analyze packet contents

The captured data interface contains three main sections:

• The packet list pane (the top section)

• The packet details pane (the middle section)

• The packet bytes pane (the bottom section)

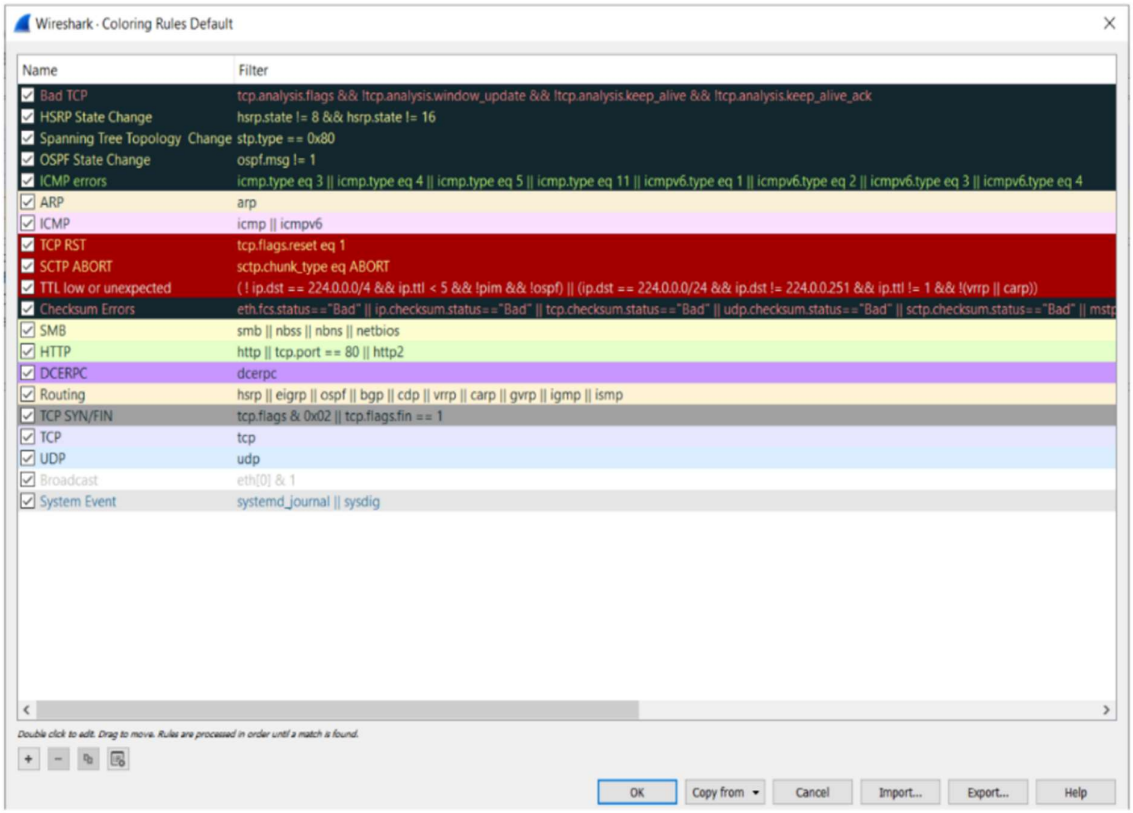


Wireshark color rules

• While Wireshark's capture and display filters limit which packets are recorded or shown on the screen, its colorization function takes things a step further: It can distinguish between different packet types based on their individual hue. This quickly locates certain packets within a saved set by their row colour in the packet list pane.

• Wireshark comes with about 20 default coloring rules, each can be edited, disabled, or deleted. Select View > Coloring Rules for an

• overview of what each color means.



Statistics in Wireshark

• Other useful metrics are available through the Statistics drop-down menu. These include size and timing information about the capture file, along with dozens of charts and graphs ranging in topic from packet conversation breakdowns to load distribution of HTTP requests.

• Display filters can be applied to many of these statistics via their interfaces, and the results can be exported to common file formats, including CSV, XML, and TXT.