

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering) Shahbad
Daulatpur, Bawana Road, Delhi 110042

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



INFORMATION AND NETWORK SECURITY (CO - 427) LAB FILE

November, 2022

SUBMITTED BY:

NAME: ZISHNENDU SARKER

ROLL: 2K19/CO/450

7th Semester, 2022

SUBMITTED TO:

Prof. SHAILENDER KUMAR

**INFORMATION AND NETWORK SECURITY
DELHI TECHNOLOGICAL UNIVERSITY**

INDEX

SL.NO	NAME OF THE EXPERIMENT	DATE
01	Write a program to implement Caesar Cipher	
02	Write a program to implement of Vignere Cipher	
03	Write a program to implement Playfair Cipher	
04	Write a program to implement Hill Cipher	
05	Write a program to implement Diffie Hellman Exchange algorithm.	
06	Write a program to implement S-DES sub-key generation.	
07	Write a program to generate SHA-1 hash.	
08	Write a program to implement a signature scheme - Digital Signature Standard.	
09	Study and use Wireshark in various protocols	

EXPERIMENT NO. 01

Aim:

To implement a program to show encryption and decryption through the Caesar cipher

Theory for Ceaser Cipher:

The Caesar cipher is the simplest and oldest method of cryptography. The Caesar cipher method is based on a mono-alphabetic cipher and is also called a shift cipher or additive cipher. Julius Caesar used the shift cipher (additive cipher) technique to communicate with his officers. For this reason, the shift cipher technique is called the Caesar cipher. The Caesar cipher is a kind of replacement (substitution) cipher, where all letters of plain text is replaced by another letter.

Let's take an example to understand the Caesar cipher. Suppose we are shifting with 1, then A will be replaced by B, B will be replaced by C, C will be replaced by D, D will be replaced by E, and this process continues until the entire plain text is finished.

Caesar ciphers is a weak method of cryptography. It can be easily hacked. It means the message encrypted by this method can be easily decrypted.

Plaintext: It is a simple message written by the user.

Ciphertext: It is an encrypted message after applying some techniques.

The formula of encryption is: $E_n(x) = (x + n) \bmod 26$.

The formula of decryption is: $D_n(x) = (x_i - n) \bmod 26$

If any case (D_n) value becomes negative (-ve), in this case, we will add 26 in the negative value.
Where,

E denotes the encryption

D denotes the decryption

x denotes the letters value

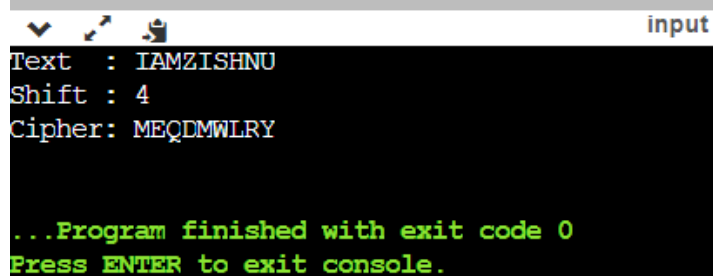
n denotes the key value (shift value)

Table for encryption and decryption for caesar cipher:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Code & Output:

```
1 def encrypt(text,s):
2     result = ""
3     for i in range(len(text)):
4         char = text[i]
5         if (char.isupper()):
6             result += chr((ord(char) + s-65) % 26 + 65)
7         else:
8             result += chr((ord(char) + s - 97) % 26 + 97)
9
10    return result
11 text = "IAMZISHNU"
12 s = 4
13 print ("Text : " + text)
14 print ("Shift : " + str(s))
15 print ("Cipher: " + encrypt(text,s))
```



```
Text : IAMZISHNU
Shift : 4
Cipher: MEQDMWLRV

...Program finished with exit code 0
Press ENTER to exit console.
```

Learning :

Here we have learned about caesar cipher where we implemented the encryption process in python. The reverse way is used to decrypt the ciphertexts as here the key remains the same.

EXPERIMENT -02

AIM: It is required to implement vignere cipher.

Theory for Vigenere Cipher:

The vignere cipher is an algorithm that is used to encrypting and decrypting the text. The Vigenere cypher is a method that combines a number of interconnected Caesar cyphers to encrypt an alphabetic text. It is based on the letters in a keyword. This polyalphabetic substitution cypher serves as an illustration. This algorithm is simple to comprehend and use. The 26-by-26 Vigenère cypher employs a table with the row and column headings A to Z. This table is also known as the Vigenere Table or Vigenere Square. We'll be using the Vigenere Table. The 26 English letters are included in the first row of this table. The letters are cyclically moved to the left one place in each row starting with the second row. For instance, the letter A goes to the end when B is moved to the first place on the second row.

- The table consists of the alphabet written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers.
- At different points in the encryption process, the cipher uses a different alphabet from one of the rows.
- The alphabet used at each point depends on a repeating keyword.

When the vignere table is not given, the encryption and decryption are done by Vigenar Vigenere algebraic formula in this method (convert the letters (A-Z) into the numbers (0-25)).

Formula of encryption is, $E_i = (P_i + K_i) \bmod 26$

Formula of decryption is, $D_i = (E_i - K_i) \bmod 26$

If any case (D_i) value becomes negative (-ve), in this case, we will add 26 in the negative value

The group of 26 alphabets are like below

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Code & Output:

```
1 def generateKey(string, key):
2     key = list(key)
3     if len(string) == len(key):
4         return(key)
5     else:
6         for i in range(len(string) -
7                         len(key)):
8             key.append(key[i % len(key)])
9     return("".join(key))
10
11 def cipherText(string, key):
12     cipher_text = []
13     for i in range(len(string)):
14         x = (ord(string[i]) +
15             ord(key[i])) % 26
16         x += ord('A')
17         cipher_text.append(chr(x))
18     return("".join(cipher_text))
19
20 def originalText(cipher_text, key):
21     orig_text = []
22     for i in range(len(cipher_text)):
23         x = (ord(cipher_text[i]) -
24             ord(key[i]) + 26) % 26
25         x += ord('A')
26         orig_text.append(chr(x))
27     return("".join(orig_text))
28
29 if __name__ == "__main__":
30     string = "CITYOFINDIA"
31     keyword = "ZISHNU"
32     key = generateKey(string, keyword)
33     cipher_text = cipherText(string, key)
34     print("Ciphertext :", cipher_text)
35     print("Original/Decrypted Text :",
36           originalText(cipher_text, key))
```

```
input
Ciphertext : BQLFBZHVVEN
Original/Decrypted Text : CITYOFINDIA
```

Learning Outcome :

Here we have learned about caesar cipher and vigenere cipher where we implemented the encryption process in python. The reverse way is used to decrypt the ciphertexts as here the key remains the same.

EXPERIMENT -03

AIM: It is required to implement playfair cipher.

Aim:

To implement a program to show encryption and decryption through the playfair Cipher

Theory:

The first useful digraph substitution cypher was the Playfair cypher. The plan was created in 1854 by Charles Wheatstone, but it was given the name Lord Playfair in honor of the person who encouraged the employment of the cypher. Unlike typical cyphers, the playfair cipher encrypts two alphabets (or digraphs) rather than just one.

British forces employed it for tactical objectives during the Second Boer War and World War I, while Australian forces utilized it for similar reasons during World War II. This was due to the fact that Playfair is easy to use and doesn't need any specialized equipment.

Algorithm of Playfair cipher:

1. Pair cannot be made with same letter. Break the letter in single and add a bogus letter to the previous letter.

Plain Text: "hello"

After Split: 'he' 'lx' 'lo'

Here 'x' is the bogus letter.

2. If the letter is standing alone in the process of pairing, then add an extra bogus letter with the alone letter

Plain Text: "helloe"

AfterSplit: 'he' 'lx' 'lo' 'ez'

Here 'z' is the bogus letter.

- 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.

Code & Output:

```
1 def toLowerCase(text):
2     return text.lower()
3
4 def removeSpaces(text):
5     newText = ""
6     for i in text:
7         if i == " ":
8             continue
9         else:
10            newText = newText + i
11    return newText
12
13 def Diagraph(text):
14     Diagraph = []
15     group = 0
16     for i in range(2, len(text), 2):
17         Diagraph.append(text[group:i])
18
19         group = i
20     Diagraph.append(text[group:])
21     return Diagraph
22
23 def FillerLetter(text):
24     k = len(text)
25     if k % 2 == 0:
26         for i in range(0, k, 2):
27             if text[i] == text[i+1]:
28                 new_word = text[0:i+1] + str('x') + text[i+1:]
29                 new_word = FillerLetter(new_word)
30                 break
31             else:
32                 new_word = text
33     else:
34         for i in range(0, k-1, 2):
35             if text[i] == text[i+1]:
36                 new_word = text[0:i+1] + str('x') + text[i+1:]
37                 new_word = FillerLetter(new_word)
38                 break
39             else:
40                 new_word = text
41     return new_word
42
43 list1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm',
44         'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

```

46 def generateKeyTable(word, list1):
47     key_letters = []
48     for i in word:
49         if i not in key_letters:
50             key_letters.append(i)
51
52     compElements = []
53     for i in key_letters:
54         if i not in compElements:
55             compElements.append(i)
56     for i in list1:
57         if i not in compElements:
58             compElements.append(i)
59
60     matrix = []
61     while compElements != []:
62         matrix.append(compElements[:5])
63         compElements = compElements[5:]
64
65     return matrix
66
67
68 def search(mat, element):
69     for i in range(5):
70         for j in range(5):
71             if(mat[i][j] == element):
72                 return i, j
73
74
75 def encrypt_RowRule(matr, e1r, e1c, e2r, e2c):
76     char1 = ''
77     if e1c == 4:
78         char1 = matr[e1r][0]
79     else:
80         char1 = matr[e1r][e1c+1]
81
82     char2 = ''
83     if e2c == 4:
84         char2 = matr[e2r][0]
85     else:
86         char2 = matr[e2r][e2c+1]
87
88     return char1, char2
89

```

```

def encrypt_ColumnRule(matr, e1r, e1c, e2r, e2c):
    char1 = ''
    if e1r == 4:
        char1 = matr[0][e1c]
    else:
        char1 = matr[e1r+1][e1c]

    char2 = ''
    if e2r == 4:
        char2 = matr[0][e2c]
    else:
        char2 = matr[e2r+1][e2c]

    return char1, char2

def encrypt_RectangleRule(matr, e1r, e1c, e2r, e2c):
    char1 = ''
    char1 = matr[e1r][e2c]

    char2 = ''
    char2 = matr[e2r][e1c]

    return char1, char2

def encryptByPlayfairCipher(Matrix, plainList):
    CipherText = []
    for i in range(0, len(plainList)):
        c1 = 0
        c2 = 0
        ele1_x, ele1_y = search(Matrix, plainList[i][0])
        ele2_x, ele2_y = search(Matrix, plainList[i][1])

        if ele1_x == ele2_x:
            c1, c2 = encrypt_RowRule(Matrix, ele1_x, ele1_y, ele2_x, ele2_y)
        elif ele1_y == ele2_y:
            c1, c2 = encrypt_ColumnRule(Matrix, ele1_x, ele1_y, ele2_x, ele2_y)
        else:
            c1, c2 = encrypt_RectangleRule(
                Matrix, ele1_x, ele1_y, ele2_x, ele2_y)

```

```

132         cipher = c1 + c2
133         CipherText.append(cipher)
134     return CipherText
135
136
137
138 text_Plain = 'Technological'
139 text_Plain = removeSpaces(toLowerCase(text_Plain))
140 PlainTextList = Diagraph(FillerLetter(text_Plain))
141 if len(PlainTextList[-1]) != 2:
142     PlainTextList[-1] = PlainTextList[-1]+'z'
143
144 key = "Medalist"
145 print("Key text:", key)
146 key = toLowerCase(key)
147 Matrix = generateKeyTable(key, list1)
148
149 print("Plain Text:", text_Plain)
150 CipherList = encryptByPlayfairCipher(Matrix, PlainTextList)
151
152 CipherText = ""
153 for i in CipherList:
154     CipherText += i
155 print("CipherText:", CipherText)
156

```

input

```

Key text: Medalist
Plain Text: technological
CipherText: sdtmfumufsbllcl

```

Learning Outcomes:

Here, we learned about the encryption process of playfair cipher. The reverse process is used to decrypt the cipher text into the plain text.

EXPERIMENT -04

AIM: It is required to implement hill cipher.

THEORY:

A polygraphic substitution cipher built on linear algebra is the Hill cipher. A number modulo 26 represents each letter. It is common to employ the straightforward formula $A = 0, B = 1, \dots, Z = 25$, however this is not a necessary component of the encryption. Each block of n letters, which is thought of as an n -component vector, is multiplied by an invertible $n \times n$ matrix against modulus 26 to encrypt a message. Each block is multiplied by the inverse of the encryption matrix to decrypt the message.

The set of invertible $n \times n$ matrices should contain the cypher key, which is the matrix used for encryption (modulo 26).

Here, polygraphic substitution cipher defines that Hill Cipher can work seamlessly with digraphs (two-letter blocks), trigraphs (three-letter blocks), or any multiple-sized blocks for building a uniform cipher.

Hill Cipher is based on a particular mathematical topic of linear Algebra and the sophisticated use of matrices in general, as well as rules for modulo arithmetic. As a prerequisite, it would be better for learners and professionals to have a sound understanding of both linear Algebra and Matrices. Thus, most of the problems and solutions for Hill Ciphers are mathematical, making it easy to withhold or hide letters precisely.

We will study Hill Cipher encryption and decryption procedures in solving 2×2 and 3×3 matrices. Although it can be used for higher matrices (4×4 , 5×5 , or 6×6), it also requires a higher and advanced level of mathematics, adding more complexity. Here, we have used simple examples that provide in-depth knowledge on this topic.

Code & Output :

```

1 keyMatrix = [[0] * 3 for i in range(3)]
2
3 messageVector = [[0] for i in range(3)]
4
5 cipherMatrix = [[0] for i in range(3)]
6
7 def getKeyMatrix(key):
8     k = 0
9     for i in range(3):
10        for j in range(3):
11            keyMatrix[i][j] = ord(key[k]) % 65
12            k += 1
13
14 def encrypt(messageVector):
15     for i in range(3):
16         for j in range(1):
17             cipherMatrix[i][j] = 0
18             for x in range(3):
19                 cipherMatrix[i][j] += (keyMatrix[i][x] *
20                                         messageVector[x][j])
21             cipherMatrix[i][j] = cipherMatrix[i][j] % 26
22
23 def HillCipher(message, key):
24     getKeyMatrix(key)
25
26     for i in range(3):
27         messageVector[i][0] = ord(message[i]) % 65
28
29     encrypt(messageVector)
30
31     CipherText = []
32     for i in range(3):
33         CipherText.append(chr(cipherMatrix[i][0] + 65))
34
35

```

```

35
36     print("Ciphertext: ", "".join(CipherText))
37
38 def main():
39
40     message = "nitiwit"
41     key = "GYBNQKURP"
42
43     HillCipher(message, key)
44
45 if __name__ == "__main__":
46     main()
47

```

input

Ciphertext: HTF

LEARNING OUTCOMES:

Here, we have learned about hill cipher, how to encrypt using hill cipher and how efficient it is. The reverse way will introduce the decryption part

EXPERIMENT -05

Aim:

It is required to implement the Diffie-Hellman Exchange algorithm.

THEORY:

Using the elliptic curve to produce points and the parameters to obtain the secret key, the Diffie-Hellman method is being utilized to create a shared secret that may be used for secret conversations while transferring data over a public network.

For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b .

P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

Diffie-Hellman algorithm:

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b .

P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

Code & Output:

```
1 from random import randint
2
3 if __name__ == '__main__':
4
5     P = 20
6
7     G = 8
8
9
10    print('The Value of P is :%d'%(P))
11    print('The Value of G is :%d'%(G))
12
13    a = 4
14    print('The Private Key a for Alice is :%d'%(a))
15
16    x = int(pow(G,a,P))
17
18    b = 3
19    print('The Private Key b for Bob is :%d'%(b))
20
21    y = int(pow(G,b,P))
22
23    ka = int(pow(y,a,P))
24
25    kb = int(pow(x,b,P))
26
27    print('Secret key for the Alice is : %d'%(ka))
28    print('Secret Key for the Bob is : %d'%(kb))
```

input

```
The Private Key a for Alice is :4
The Private Key b for Bob is :3
Secret key for the Alice is : 16
Secret Key for the Bob is : 16
```

LEARNING OUTCOMES:

Here, we have learned about Diffie-Hellman exchange algorithm, how to encrypt using it and how efficient it is. The reverse way will introduce the decryption part.

EXPERIMENT -06

AIM: It is required to implement S-DES sub-key generation.

Theory: The general design of the streamlined DES. An 8-bit block of plaintext (for example, 10111101) and a 10-bit key are the inputs for the S-DES encryption technique, which yields an 8-bit block of ciphertext as the output. The 10-bit key that was used to encrypt the initial 8-bit block of plaintext is entered into the S-DES decryption method together with an 8-bit block of ciphertext.

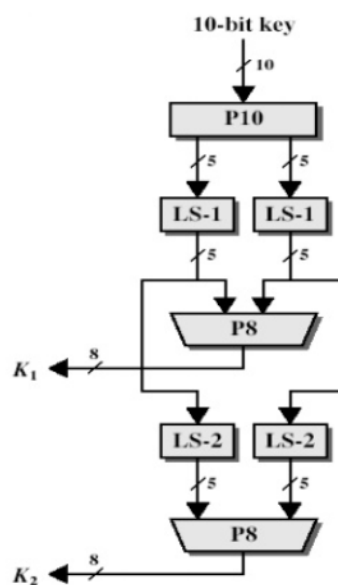


Figure: key generation for S-DES

S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption algorithm. First, permute the key in the following fashion. Let the 10-bit key be designated as $(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10})$. Then the permutation P10 is defined as:

$P10(k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}) = (k_3, k_5, k_2, k_7, k_4, k_{10}, k_1, k_9, k_8, k_6)$ P10 can be concise.

Code & Output :

```
1 import numpy as np
2
3 def table_shift(array, table_array):
4     array_shifted = np.zeros(table_array.shape[0], dtype='int')
5     for index, value in enumerate(table_array): array_shifted[index] = array[value - 1]
6     return array_shifted
7
8 def array_split(array):
9     left_split = array[:int(len(array) / 2)]
10    right_split = array[int(len(array) / 2):]
11    return left_split, right_split
12
13 def shifting_LtoR(array):
14    temp = array[0]
15    for index in range(1, len(array)): array[index - 1] = array[index]
16    array[len(array) - 1] = temp
17    return array
18
19 table_p_10 = np.array([3, 5, 2, 7, 4, 10, 1, 9, 8, 6])
20 table_p_08 = np.array([6, 3, 7, 4, 8, 5, 10, 9])
21
22 key = list('0001011011')
23
24 def split_and_merge(key):
25     left_split, right_split = array_split(key)
26     return np.concatenate((shifting_LtoR(left_split), shifting_LtoR(right_split)))
27
28 def key_generation_1(key, table):
29     k = table_shift(key, table)
30     key_merge = split_and_merge(k)
31     return table_shift(key_merge, table)
32
33 def key_generation_2(key, table): return split_and_merge(key)
34
35 print("Name: Zishnendu Sarker ")
36 print("Lab : 05 ")
37 key_1 = key_generation_1(key, table_p_10)
38 print("".join([str(elem) for elem in key_1])) #1000111010
39
40 key_2 = key_generation_2(key_1, table_p_08)
41 print("".join([str(elem) for elem in key_2])) #0001110101
42
```

input

```
Name: Zishnendu Sarker
Lab : 05
1001110100
0011101001
```

Learning Outcome:

The process of encrypting a plaintext message into an encrypted message with the use of S-DES has been divided into multi-steps which may help you to understand it as easily as possible. Points should be remembered: It is a block cipher, It has 8-bit block size of plain text or cipher text, it uses 10-bit key size for encryption, it is a symmetric cipher, it has Two Rounds.

EXPERIMENT -07

AIM: It is required to implement generate SHA-1 hash.

Theory: A hash function called SHA-1 (Secure Hash Algorithm 1) creates a 160-bit hash code (message digest) corresponding to an input. The American National Security Agency began the SHA-1 in 1993. The security programs and protocols TLS, SSL, PGP, SSH, IPsec, and S/MIME all use it extensively.

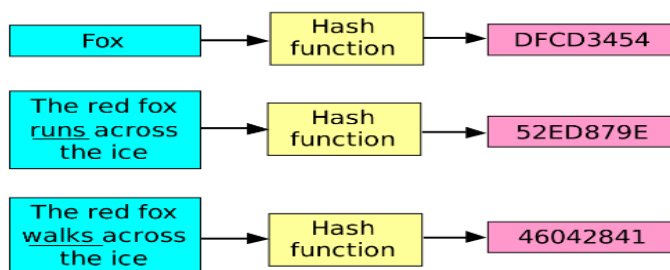
Compared to MD5, the SHA-1 hashing method is more secure. It is key to securing confidential data, such as passwords and credit card details.

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographically broken but still widely used hash function that takes an input and produces a 160-bit (20-byte) hash value known as a message digest – typically rendered as 40 hexadecimal digits.

SHA-1 produces a 160-bit hash value or message digests from the inputted data (data that requires encryption), which resembles the hash value of the MD5 algorithm. It uses 80 rounds of cryptographic operations to encrypt and secure a data object. Some of the protocols that use SHA-1 include:

- Transport Layer Security (TLS)
- Secure Sockets Layer (SSL)
- Pretty Good Privacy (PGP)
- Secure Shell (SSH)
- Secure/Multipurpose Internet Mail Extensions (S/MIME)
- Internet Protocol Security (IPSec)

SHA-1 is commonly used in cryptographic applications and environments where the need for data integrity is high. It is also used to index hash functions and identify data corruption and checksum errors.



Code & Output:

```
1 import hashlib
2
3 # initializing string
4 str = "sristi"
5
6
7 result = hashlib.sha256(str.encode())
8
9 # printing the equivalent hexadecimal value.
10 print("The hexadecimal equivalent of SHA256 is : ")
11 print(result.hexdigest())
12
13 print ("\r")
14
15 # initializing string
16 str = "sristi"
17
18
19 result = hashlib.sha384(str.encode())
20
21 # printing the equivalent hexadecimal value.
22 print("The hexadecimal equivalent of SHA384 is : ")
23 print(result.hexdigest())
24
25 print ("\r")
26
27 # initializing string
28 str = "sristi"
29
30
31 result = hashlib.sha224(str.encode())
32
33 # printing the equivalent hexadecimal value.
34 print("The hexadecimal equivalent of SHA224 is : ")
35 print(result.hexdigest())
```

```

36
37 print ("\r")
38
39 # initializing string
40 str = "srusti"
41
42
43 result = hashlib.sha512(str.encode())
44
45 # printing the equivalent hexadecimal value.
46 print("The hexadecimal equivalent of SHA512 is : ")
47 print(result.hexdigest())
48
49 print ("\r")
50
51 # initializing string
52 str = "srusti"
53
54
55 result = hashlib.sha1(str.encode())
56
57 # printing the equivalent hexadecimal value.
58 print("The hexadecimal equivalent of SHA1 is : ")
59 print(result.hexdigest())

```

input

```

The hexadecimal equivalent of SHA384 is :
e7ce5a962d28de95592c9a5867939a53fa38ad65c76b996bb7874074de11fa32e845a3bd10f9a034819179de0a
366e9d

The hexadecimal equivalent of SHA224 is :
f49442ca0043a9b7c3d7a7ed6ebd240349c89cbe77047537d0a810c7

The hexadecimal equivalent of SHA512 is :
6a9fd6f6bf6ebac32e5e76f0591e20ce213dbcb11464b70dcdb228d78068650b2e37b3918c449dacb62247fbba
942c05328c1bc600a38a885aeade160377db1a

The hexadecimal equivalent of SHA1 is :
6ba3d3754f5de7d5c6ec71e20e6164daf7f056ed

```

Learning Outcome:

The four round constants k are 230 times the square roots of 2, 3, 5, and 10. However, they were incorrectly rounded to the nearest integer instead of being rounded to the nearest odd integer, with equilibrated proportions of zero and one bit. As well, choosing the square root of 10 (which is not a prime) made it a common factor for the two other chosen square roots of primes 2 and 5, with possibly usable arithmetic properties across successive rounds, reducing the strength of the algorithm against finding collisions on some bits. The first four starting values for h_0 through h_3 are the same with the MD5 algorithm, and the fifth (for h_4) is similar. However, they were not properly verified for being resistant against inversion of the few first rounds to infer possible collisions on some bits, usable by multiblock differential attacks.

properties across successive rounds, reducing the strength of the algorithm against finding collisions on some bits.

The first four starting values for h_0 through h_3 are the same with the MD5 algorithm, and the fifth (for h_4) is similar. However, they were not properly verified for being resistant against inversion of the few first rounds to infer possible collisions on some bits, usable by multiblock differential attacks.

EXPERIMENT -08

AIM: It is required to implement a signature scheme - Digital Signature Standard.

Theory:

When sending an electronic message, the authenticity of the message is checked using digital signatures. A public key system is used by an algorithm for digital signatures. The intended receiver confirms the message with the intended transmitter's public key after the intended transmitter signs it with his or her private key. Message authentication, message integrity, and non-repudiation services can all be offered by a digital signature.

Digital Signature Scheme: In RSA, d is private; e and n are public.

- Alice creates her digital signature using $S=M^d \bmod n$ where M is the message
- Alice sends Message M and Signature S to Bob
- Bob computes $M1=S^e \bmod n$
- If $M1=M$ then Bob accepts the data sent by Alice.

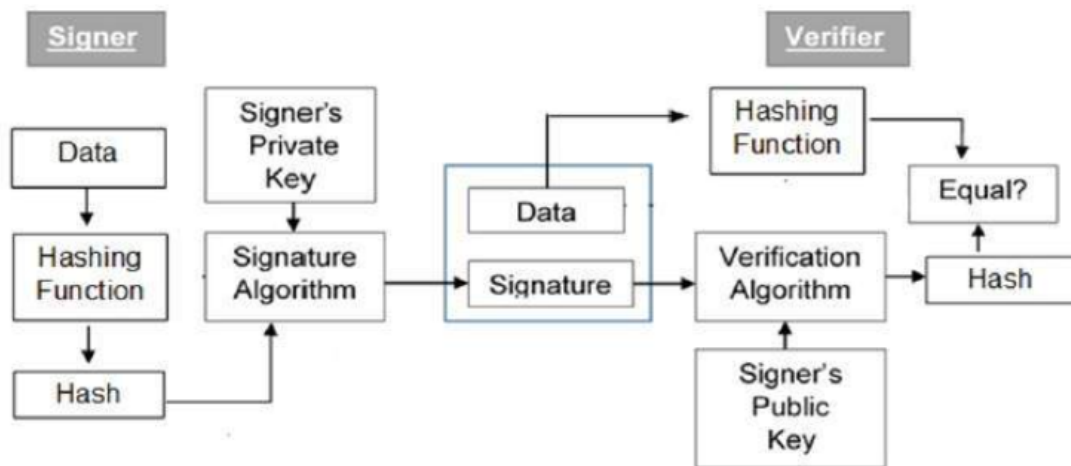
A hash code is generated from the message and given as an input to the signature function on the sender side. The other inputs to a signature function include a unique random number k for the signature, the private key of sender $PR(a)$, and global public key i.e., $PU(g)$.

The output of the signature function consists of two components: s & r , which is concatenated with the input message and then sent to the receiver.

Signature = $\{s, r\}$.

At the receiver side, the hash code for the message sent is generated by the receiver by applying a hash function. The verification function is used for verifying the message and signature sent by the sender. The verification function takes hash code generated, signature components s and r , the public key of the sender ($PU(a)$), and global public key.

The signature function is compared with the output of the verification function and if both the values match, the signature is valid because A valid signature can only be generated by the sender using its private key.



Code & Output:

```

1  # Function to find gcd
2  # of two numbers
3  def euclid(m, n):
4
5      if n == 0:
6          return m
7      else:
8          r = m % n
9          return euclid(n, r)
10
11
12  # Program to find
13  # Multiplicative inverse
14  def exteuclid(a, b):
15
16      r1 = a
17      r2 = b
18      s1 = int(1)
19      s2 = int(0)
20      t1 = int(0)
21      t2 = int(1)
22
23      while r2 > 0:
24
25          q = r1//r2
26          r = r1-q * r2
27          r1 = r2
28          r2 = r
29          s = s1-q * s2
30          s1 = s2
31          s2 = s
32          t = t1-q * t2
33          t1 = t2
34          t2 = t
35
36      if t1 < 0:
37          t1 = t1 % a
38
39      return (r1, t1)
40
41  # Enter two large prime
42  # numbers p and q
43  p = 823
44  q = 953
45  n = p * q
46  Pn = (p-1)*(q-1)
47  # Generate encryption key
48  # in range 1<e<Pn
49  key = []
50  for i in range(2, Pn):
51
52      gcd = euclid(Pn, i)
53
  
```

```

53
54     if gcd == 1:
55         key.append(i)
56
57     # Select an encryption key
58     # from the above list
59     e = int(313)
60     # Obtain inverse of
61     # encryption key in  $Z_{Pn}$ 
62     r, d = exteuclid(Pn, e)
63     if r == 1:
64         d = int(d)
65         print("decryption key is: ", d)
66
67     else:
68         print("Multiplicative inverse for\
69         the given encryption key does not \
70         exist. Choose a different encryption key ")
71
72     # Enter the message to be sent
73     M = 19070
74     # Signature is created
75     S = (M**d) % n
76
77
78     M1 = (S**e) % n
79
80
81
82     if M == M1:
83         print("As M = M1, Accept the\
84         message sent by Zishnu")
85     else:
86         print("As M not equal to M1,\
87         Do not accept the message\
88         sent by Zishnu ")

```

input

```

decryption key is: 160009
As M = M1, Accept the message sent by Zishnu

```

Learning Outcome:

From the experiment, we have learned that asymmetric cryptographic methods like digital signatures are employed to confirm the legitimacy of digital messages and documents. It makes use of the idea of public/private key pairs, where the two keys are mathematically connected and offer security benefits above and above those of handwritten signatures. A digital signature is encrypted using a person's private key, and only that person's public key may be used to decrypt the signature.

EXPERIMENT -09

AIM: Study and use the Wireshark for various network protocols.

THEORY:

The most famous and commonly used network protocol analyzer in the world is called Wireshark. It is the de facto (and frequently de jure) standard across many commercial and non-profit firms, governmental organisations, and educational institutions because it enables you to observe what's occurring on your network at a microscopic level. The continued development of Gerald Combs' 1998 project, Wireshark, is made possible by the voluntary contributions of networking specialists from all around the world.

Wireshark has a rich feature set which includes the following:

Thorough examination of hundreds of procedures, with new ones being added on a regular basis. offline analysis and real-time capture. standard packet browser with three panes. Multi-platform: runs on several operating systems, including Windows, Linux, macOS, Solaris, FreeBSD, and NetBSD. A GUI or the TTY-mode TShark programme can be used to browse network data that has been captured. The industry's strongest display filters. Detailed VoIP analysis several various capture file types can be read/writable. Gzip-compressed capture files may instantly be uncompressed. Many protocols, including IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP, and WPA/WPA2, are supported in terms of decryption. The packet list may be analysed quickly and easily by using colouring rules. Export options for output include XML, PostScript®, CSV, and plain text.

Implementation and Output:

Capturing from Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

No.	Time	Source	Destination	Protocol	Length	Info
134	7.451366	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34
135	7.456964	142.250.182.174	192.168.1.36	UDP	68	443 → 56385 Len=26
136	7.666687	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34
137	7.672550	142.250.182.174	192.168.1.36	UDP	68	443 → 56385 Len=26
138	7.904589	192.168.1.36	142.250.206.174	UDP	1009	57628 → 443 Len=967
139	7.909497	142.250.206.174	192.168.1.36	UDP	72	443 → 57628 Len=30
140	7.914435	192.168.1.36	142.250.206.174	UDP	75	57628 → 443 Len=33
141	7.990472	142.250.206.174	192.168.1.36	UDP	129	443 → 57628 Len=87
142	7.990651	192.168.1.36	142.250.206.174	UDP	79	57628 → 443 Len=37
143	7.991287	142.250.206.174	192.168.1.36	UDP	68	443 → 57628 Len=26
144	7.994373	192.168.1.36	142.250.206.174	UDP	75	57628 → 443 Len=33
145	7.997099	142.250.206.174	192.168.1.36	UDP	68	443 → 57628 Len=26
146	8.086882	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34
147	8.093861	142.250.182.174	192.168.1.36	UDP	68	443 → 56385 Len=26
148	8.133299	192.168.1.36	8.241.134.254	ICMP	106	Echo (ping) request id=0x0001, seq=61536, ttl=7 (no response found!)
149	8.617473	192.168.1.36	172.217.166.10	UDP	75	62852 → 443 Len=33
150	8.623789	172.217.166.10	192.168.1.36	UDP	67	443 → 62852 Len=25
151	8.900838	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34
152	8.906193	142.250.182.174	192.168.1.36	UDP	68	443 → 56385 Len=26
153	8.934475	209.197.3.8	192.168.1.36	TCP	54	80 → 58121 [ACK] Seq=1 Ack=1 Win=62 Len=0
154	8.934498	192.168.1.36	209.197.3.8	TCP	54	[TCP ACKed unseen segment] 58121 → 80 [ACK] Seq=1 Ack=2 Win=514 Len=0
155	9.681627	192.168.1.36	52.189.124.115	TCP	54	49702 → 443 [RST, ACK] Seq=2 Ack=1 Win=0 Len=0
156	9.681628	192.168.1.36	52.184.81.210	TCP	54	49720 → 443 [RST, ACK] Seq=2 Ack=1 Win=0 Len=0
157	9.775840	142.250.182.174	192.168.1.36	UDP	80	443 → 56385 Len=38
158	9.778279	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34
159	9.990741	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34
160	9.997802	142.250.182.174	192.168.1.36	UDP	68	443 → 56385 Len=26
161	10.208414	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34
162	10.214177	142.250.182.174	192.168.1.36	UDP	68	443 → 56385 Len=26
163	10.425594	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34

> Frame 1: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface \Device\NPF_{612C7F1C-DF68-4376-86F7-11E3394C26C2}, id 0

> Ethernet II, Src: CloudNet_7d:e8:1d (f8:89:d2:7d:e8:1d), Dst: optilink_70:b3:30 (b4:f9:49:70:b3:30)

> Internet Protocol Version 4, Src: 192.168.1.36, Dst: 52.189.124.115

> Transmission Control Protocol, Src Port: 49702, Dst Port: 443, Seq: 1, Ack: 1, Len: 0

Source Port: 49702

Destination Port: 443

[Stream index: 0]

[Conversation completeness: Incomplete (20)]

[TCP Segment Len: 0]

Sequence Number: 1 (relative sequence number)

Sequence Number (raw): 3334412646

[Next Sequence Number: 2 (relative sequence number)]

Acknowledgment Number: 1 (relative ack number)

Acknowledgment number (raw): 3304658056

0101 = Header Length: 20 bytes (5)

> Flags: 0x011 (FIN, ACK)

Window: 512

[Calculated window size: 512]

[Window size scaling factor: -1 (unknown)]

Checksum: 0x4831 [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

> [Timestamps]

0000 b4 f9 49 70 b3 30 f8 89 d2 7d e8 1d 08 00 45 00 ..Ip.θ...}....E-

0010 00 28 3a b8 40 00 80 06 4d 6b c0 a8 01 24 34 6d .{.:θ...Mk...\$m

0020 7c 73 c2 26 01 bb c6 bf 19 66 c1 65 8d 88 50 11 |s&....-f.e..P-

0030 02 00 48 31 00 00 ..H.L..

Packets: 163 · Displayed: 163 (100.0%) Profile: Def

Fig: for TCP protocol

Capturing from Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

15

No.	Time	Source	Destination	Protocol	Length	Info
1455	52.716226	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34
1456	52.722664	142.250.182.174	192.168.1.36	UDP	68	443 → 56385 Len=26
1457	52.931844	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34
1458	52.938527	142.250.182.174	192.168.1.36	UDP	68	443 → 56385 Len=26
1459	53.073946	192.168.1.36	20.84.22.197	TCP	338	[TCP Retransmission] 58158 → 443 [PSH, ACK] Seq=2 Ack=1 Win=515 Len=284
1460	53.151793	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34
1461	53.156948	142.250.182.174	192.168.1.36	UDP	68	443 → 56385 Len=26
1462	53.574263	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34
1463	53.580159	142.250.182.174	192.168.1.36	UDP	68	443 → 56385 Len=26
1464	54.389265	192.168.1.36	142.250.182.174	UDP	76	56385 → 443 Len=34
1465	54.394434	142.250.182.174	192.168.1.36	UDP	68	443 → 56385 Len=26
1466	54.559900	192.168.1.36	20.84.22.197	TCP	338	[TCP Retransmission] 58158 → 443 [PSH, ACK] Seq=2 Ack=1 Win=515 Len=284
1467	54.701185	192.168.1.36	142.250.193.228	QUIC	1292	Initial, DCID=fasfb0a1a7b15c8b, PKN: 1, CRYPTO, CRYPTO, PADDING, PING, CRYPTO, PADDING, PING, PADDING, CRYPTO, PADDING, CRYPTO, CRYPTO, PADDING, CRYPTO, PADDING, CRYPTO, PING, PADDING, CRYPTO, PING
1468	54.702148	192.168.1.36	142.250.193.228	QUIC	122	0-RTT, DCID=fasfb0a1a7b15c8b
1469	54.733631	142.250.193.228	192.168.1.36	QUIC	1292	Initial, SCID=fasfb0a1a7b15c8b, PKN: 1, ACK, PADDING
1470	54.733812	192.168.1.36	142.250.193.228	QUIC	1292	Initial, DCID=fasfb0a1a7b15c8b, PKN: 4, PADDING, PING, PADDING
1471	54.740159	142.250.193.228	192.168.1.36	QUIC	1292	Initial, SCID=fasfb0a1a7b15c8b, PKN: 2, ACK, PADDING
1472	54.777554	142.250.193.228	192.168.1.36	QUIC	1292	Protected Payload (KP0)
1473	54.777554	142.250.193.228	192.168.1.36	QUIC	826	Protected Payload (KP0)
1474	54.777554	142.250.193.228	192.168.1.36	QUIC	222	Protected Payload (KP0)
1475	54.777554	142.250.193.228	192.168.1.36	QUIC	67	Protected Payload (KP0)
1476	54.777941	192.168.1.36	142.250.193.228	QUIC	120	Handshake, DCID=fasfb0a1a7b15c8b
1477	54.778074	192.168.1.36	142.250.193.228	QUIC	75	Protected Payload (KP0), DCID=fasfb0a1a7b15c8b
1478	54.782842	142.250.193.228	192.168.1.36	QUIC	162	Protected Payload (KP0)
1479	54.782970	192.168.1.36	142.250.193.228	QUIC	75	Protected Payload (KP0), DCID=fasfb0a1a7b15c8b
1480	55.165989	192.168.1.36	209.197.3.8	TCP	54	58121 → 80 [FIN, ACK] Seq=1 Ack=2 Win=514 Len=0
1481	55.166116	192.168.1.36	209.197.3.8	TCP	54	58188 → 80 [FIN, ACK] Seq=1 Ack=2 Win=514 Len=0
1482	55.204775	209.197.3.8	192.168.1.36	TCP	54	[TCP Previous segment not captured] 80 → 58121 [ACK] Seq=2 Ack=2 Win=62 Len=0
1483	55.204775	209.197.3.8	192.168.1.36	TCP	54	[TCP Previous segment not captured] 80 → 58188 [ACK] Seq=2 Ack=2 Win=62 Len=0
1484	55.240655	209.197.3.8	192.168.1.36	TCP	54	80 → 58121 [FIN, ACK] Seq=2 Ack=2 Win=62 Len=0

Frame 1108: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface \Device\NPF_{612C7F1C-DF68-4376-86F7-11E3394C26C2}, id 0
 Ethernet II, Src: optLink_70:b3:30 (b4:f9:89:70:b3:30), Dst: CloudNet_7d:e8:1d (f8:89:d2:7d:e8:1d)
 Internet Protocol Version 4, Src: 172.217.166.10, Dst: 192.168.1.36
 User Datagram Protocol, Src Port: 443, Dst Port: 62852
 Source Port: 443
 Destination Port: 62852
 Length: 33
 Checksum: 0xf6e9 [unverified]
 [Checksum Status: Unverified]
 [Stream index: 0]
 [Timestamps]
 UDP payload (25 bytes)
 Data (25 bytes)

0000 f8 89 d2 7d e8 1d b4 f9 49 70 b3 30 08 00 45 00 ...}....Ip 0:E-
 0010 00 35 00 00 40 00 3c 11 29 08 ac d9 a6 0a c0 a8 -5-@<.).....
 0020 01 24 01 b6 f5 84 00 21 f6 e9 42 4c 43 e7 92 47 \$.....!..BLC-G
 0030 12 c3 23 30 5d 17 fc 0a f4 64 50 0e 10 18 d7 d8 ..#0}...dP.....
 0040 e3 5c 45 ..E

Fig: for UDP protocol

Learning Outcome:

From here , we learn how to sync with network layer, and how to identify the network protocol.

Network protocol and their usage and identification