SARASWATI Education Society's

SARASWATI College of Engineering

Learn Live Achieve and Contribute

Kharghar, Navi Mumbai - 410 210.

NAAC Accredited

Department of Computer Engineering

Name : Shaikh Adnan Shaukat Ali

Roll No : 76

Class/Sem : TE/VI

Branch : Computer

Year : 2021-22

Subject : Cryptography and System Security(CSS)

INDEX

Expt. No.	Date	Name of the Experiment
1	27/1/2022	Find the multiplicative inverse of any number Zn using Extended Euclidean algorithm. Also find the gcd and t values. Show the output of each step.
2	27/1/2022	Implementation of Euler's Totient Function.
3	8/2/2022	To implement Caeser cipher (Encryption and Decryption).
4	24/3/2022	To implement a program in python for password cracking using Brute Force Method. Password contains only lower alphabet characters. Length of password is 4.
5	24/3/2022	Implementation of RSA Cryptosystem, which includes: a. Implementation of RSA Key Generation Algorithm. b. Encryption and decryption using RSA algorithm.
6	24/3/2022	For varying message sizes, generate message digest using MD5 algorithm and check the integrity of message.
7	24/3/2022	For varying message sizes, generate message digest using SHA-1 algorithm and check the integrity of message.
8	4/4/2022	Implementation of Diffie Hellman Algorithm in python. Also, perform Diffie Hellman Practical using Cryptography lab of Virtual labs of IIT Bombay.
9	5/5/2022	Study of packet sniffer tool "wireshark": a. Download and install wirshark and capture icmp,tcp,and http packets in promiscuous mode. b. Explore how the packets can be traced based on different filters.
10	13/4/2022	Study the use of network reconnaissance tools like WHOIS, dig, tracert, nslookup.
11	13/4/2022	Perform Digital Signature Scheme Experiment using Cryptography lab of Virtual labs of IIT Bombay.

Prof. Ritu Jain — **Subject I/C**

Prof. Sujata Bhairnallykar — **H.O.D**

SARASWATI Education Society's
SARASWATI College of Engineering
Learn Live Achieve and Contribute
Kharghar, Navi Mumbai - 410 210.
NAAC Accredited

INDEX

Assign. No.	Date	Name of the Experiment
1	17/3/2022	Assignment 1
2	15/4/2022	Assignment 2

Prof. Ritu Jain
Subject I/C

Prof. Sujata Bhairnallykar
H.O.D

Experiment No. 1

Aim: To find the multiplicative inverse of any number Zn using Extended Euclidean algorithm.

Theory: Extended Euclidean algorithm can be used to find the multiplicative inverse of number on modulus operation if exist. Relation can be derived as follows:

$$b(t) \equiv 1(mod n)$$

i.e. $t = b^{-1}$

This can be written as:

$$b(t) = n(q) + 1$$

$$b(t) + n(-q) = 1$$

Comparing it with: $b(x) + a(y) = GCD(b, a)$

We inferred that inverse of 'b' (t) exist or $b(t) \equiv 1(mod n)$ only holds, when $GCD(b, n) = 1$ and $b^{-1} = t = x$ in Extended Euclidean equation.

Above equation also clear that we can use Extended Euclidean algorithm to find multiplicative inverse of a number. Let us look at algorithm and example:

Extended Euclidean algorithm:

Initialize: $r1 = b, r2 = n, t1 = 0 \; and \; t2 = 1$

Repeat following steps till$(r2 > 0)$:

1. $q = r1//r2$ -------- {Where, // refers to integer division}
2. $r = r1 - q * r2$
3. $r1, r2 = r2, r$ ------- {Interchange r1 and r2 with r2 and r}

4. $t = t1 - q * t2$

5. $t1, t2 = t2, t$ ------- {Interchange t1 and t2 with t2 and t}

After completion of iteration if $r1 = 1 \Rightarrow GCD(b,n) = 1$ then inverse exist and it is stored in $t1(b^{-1})$ variable.

Example:

Let b= 420, n = 69

q	r1	r2	r	t1	t2	t
6	420	69	6	0	0	-6
11	69	6	3	1	-6	67
2	2	6	0	-6	67	-140
	3	0		67	-140	

Since, $r1 != 1$ multiplicative inverse of 420 doesn't exist when mod with 69.

Implementation:

```
import pandas as pd

def multiplicative_inverse(b,n):
    r1,r2,t1,t2 = b,n,0,1
    arrays = [[] for _ in range(7)]

    while(r2>0):
        q = r1//r2
        arrays[0].append(q), arrays[1].append(r1), arrays[2].append(r2)

        r = r1 - q*r2
        r1,r2 = r2,r
        arrays[3].append(r), arrays[4].append(t1), arrays[5].append(t2)

        t = t1 - q*t2
        t1,t2 = t2,t
        arrays[6].append(t)
```

```
    b_inverse = t1 if r1 == 1 else False

    arrays[0].append(None), arrays[1].append(r1), arrays[2].append(r2)
    arrays[3].append(None), arrays[4].append(t1), arrays[5].append(t2)
    arrays[6].append(None)

    table = pd.DataFrame({
        "q": arrays[0],
        "r1": arrays[1],
        "r2": arrays[2],
        "r": arrays[3],
        "t1": arrays[4],
        "t2": arrays[5],
        "t": arrays[6]
    })

    return b_inverse,table

b,n = map(int,input("Please enter the value of two numbers to find their m
ultiplicative inverse: ").strip().split(" "))

b_inverse,table = multiplicative_inverse(b,n)

if not b_inverse:
    print("Inverse doesn't exist")
else:
    print(f"Inverse of b = {b_inverse+n if b_inverse<0 else b_inverse}")

table
```

Output:

```
Please enter the value of two numbers to find their multiplicative inverse: 95 77
Inverse of b = 40
```

	q	r1	r2	r	t1	t2	t
0	1.0	95	77	18.0	0	1	-1.0
1	4.0	77	18	5.0	1	-1	5.0
2	3.0	18	5	3.0	-1	5	-16.0
3	1.0	5	3	2.0	5	-16	21.0
4	1.0	3	2	1.0	-16	21	-37.0
5	2.0	2	1	0.0	21	-37	95.0
6	NaN	1	0	NaN	-37	95	NaN

```
Please enter the value of two numbers to find their multiplicative inverse: 7000 85
Inverse doesn't exist
```

	q	r1	r2	r	t1	t2	t
0	82.0	7000	85	30.0	0	1	-82.0
1	2.0	85	30	25.0	1	-82	165.0
2	1.0	30	25	5.0	-82	165	-247.0
3	5.0	25	5	0.0	165	-247	1400.0
4	NaN	5	0	NaN	-247	1400	NaN

Experiment No. 2

Aim: To implement Euler's Totient Function

Theory: Euler's totient function, written $\boldsymbol{\varphi(n)}$, and defined as the number of positive integers less than n and relatively prime to n. i.e. $\boldsymbol{|GCD(n,x) = 1|\ where\ x\ \epsilon\ Z_n\ (Residues\ of\ n)}$.

$e.g.\ \varphi(25) = 20$. Listing all relatively prime to 25 between [1, 24]: {1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 16, 17, 18, 19, 21, 22, 23, 24}.

It can be easily inferred that, for any prime p.

$\boldsymbol{\varphi(p) = p - 1}$. $e.g.\ \varphi(17) = 16$

It also have important properties for prime factorization:

1. $\boldsymbol{\varphi(n) = \varphi(p \times q) = \varphi(p) \times \varphi(q) = (p-1) \times (q-1)}$ ---- {where, p&q are distinct prime}. $e.g.\varphi(35) = \varphi(7 \times 5) = \varphi(7) \times \varphi(5) = (7-1) \times (5-1) = 24$.

2. $\boldsymbol{\varphi(n) = \varphi(p^k) = (p^k - p^{k-1})}$ --- {where, p is prime}.

$e.g.\varphi(441) = \varphi(7^2 \times 3^2) = (7^2 - 7^1) \times (3^2 - 3^1) = 252$.

Implementation:

```
def gcd(a,b):
    if not b:
        return a
    return gcd(b,a%b)

def coprime(a,b):
    return gcd(a,b) == 1

def totient(n):
    assert n>0, "Number should be greater than 0"
    totient_set ={1}
    i = n-1
    while(i>1):
        if coprime(n,i):
            totient_set.add(i)
        i -= 1
```

```
    return totient_set, len(totient_set)

totient_set, totient_value = totient(int(input("Enter number to find its t
otient value: ")))

print(f"totient_value = {totient_value}")
```

Output:

```
Enter number to find its totient value: 441
totient_value = 252
```

```
Enter number to find its totient value: 67
totient_value = 66
```

```
Enter number to find its totient value: 999
totient_value = 648
```

```
Enter number to find its totient value: 89
totient_value = 88
```

```
Enter number to find its totient value: 44
totient_value = 20
```

```
Enter number to find its totient value: 848
totient_value = 416
```

Experiment No. 3

Aim: To implement Additive Cipher (Caesar Cipher).

Theory: Additive Cipher also known as **Caesar Cipher** is an example of **Substitution Cipher** in which each character of Plain text is replaced by some other character present in its domain using key in the range of size of domain, in Additive Cipher size of domain is equal to total alphabets i.e. 26 (count starts from 0 but we exclude 0 because it gives same cipher text as plain text).

Additive Cipher key range**:** [1,25]

Encipher Function: $E(P,k) = (P+k)(mod26)$ – {Where, P is the numeric value of alphabet in string and k is the key agreed by sender and receiver}.

Decipher Function: $D(C,k) = (C-k)(mod26)$ – {Where, C is the numeric value of alphabet in string and k is the same key as in Encipher Function}.

We mod by 26 because the numeric values of our domain can never exceed 25. Cipher text is always in capital case.

Alphabets and their numeric values: Values are same for both Cipher text and Plain text.

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
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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

E.g. Plain text: Python is awesome, Key = 5

$$P -> 15 -> E(15,5) = (15+5)(mod26) = 20 -> U$$

$$y -> 25 -> E(25,5) = (24+5)(mod26) = 3 -> D$$

Similarly, doing for each this for each alphabet in Plain text we get Cipher text as: UDYMTSNXFBJXTRJ. We removed spaces in Plain text we can also keep it as it is.

Cipher text: UDYMTSNXFBJXTRJ, Key = 5

$$U \rightarrow 20 \rightarrow D(20,5) = (20-5)(mod26) = 15 \rightarrow p$$

$$D \rightarrow 3 \rightarrow D(3,5) = (3-5)(mod26) = 24 \rightarrow y$$

Similarly, doing this for each alphabet in Cipher text we get Plain text as: pythonisawesome. It can be seen that it is same as above Plain text (spaces removed).

By above example it is clear that $E(P,k) \Leftrightarrow D^{-1}(P,k)$ and $D(C,k) \Leftrightarrow E^{-1}(C,k)$. Since, both function are invertible they form one-to-one correspondence with each other.

Implementation:

```
import numpy as np
import string

class AdditiveCipher:

    def __init__(self, key = np.random.randint(1,26)):
        assert 1<=key<= 25, "Key should be in between [1,25]"
        self.domain = 26
        self.key = key
        self.map = {key:value for key,value in zip(list(string.ascii_uppercase),range(0,26))}
        self.reverse_map = {value:key for key,value in self.map.items()}

    def encode(self, plain_txt):
        plain_txt = plain_txt.replace(" ","")
        assert plain_txt.isalpha() or plain_txt, "Plain text should only contain alphabetic character"
        cipher_txt = ""

        for txt in plain_txt:
            cipher_txt += self.reverse_map[(self.map[txt.upper()]+self.key)%self.domain]

        return cipher_txt

    def decode(self, cipher_txt):
        assert cipher_txt.isupper() and cipher_txt.isalpha(), "Cipher text should only contain uppercase alphabetic character"
```

```
        plain_txt = ""

        for txt in cipher_txt:
            plain_txt += self.reverse_map[(self.map[txt]-self.key)%self.domain].lower()

        return plain_txt

additive = AdditiveCipher(int(input("Enter key in between [1,26] for encipherment and decipherment: ")))
print()

z = additive.encode(input("Enter a text [a-z]|[A-Z] to encode it:"))
y = additive.decode(z)
print(f"Plain text = {y}, Cipher text = {z}\n\n")

z = additive.decode(input("Enter a text [A-Z] to decode it:"))
y = additive.encode(z)
print(f"Plain text = {z}, Cipher text = {y}")
```

Output:

```
Enter key in between [1,26] for encipherment and decipherment: 6

Enter a text [a-z]|[A-Z] to encode it:Its me Mario
Plain text = itsmemario, Cipher text = OZYSKSGXOU


Enter a text [A-Z] to decode it:OZYSKSGXOU
Plain text = itsmemario, Cipher text = OZYSKSGXOU

Enter key in between [1,26] for encipherment and decipherment: 25

Enter a text [a-z]|[A-Z] to encode it:ZaAWurdo
Plain text = zaawurdo, Cipher text = YZZVTQCN


Enter a text [A-Z] to decode it:YZZVTQCN
Plain text = zaawurdo, Cipher text = YZZVTQCN


Enter key in between [1,26] for encipherment and decipherment: 10

Enter a text [a-z]|[A-Z] to encode it:Area of accuracy between Frequency and Time remain constant in Wavelet transform
Plain text = areaofaccuracybetweenfrequencyandtimeremainconstantinwavelettransform, Cipher text = KBOKYPKMMEBKMILODGOOXPBOAEOXM
IKXNDSWOBOWKSXMYXCDKXDSXGKFOVODDBKXCPYBW


Enter a text [A-Z] to decode it:KBOKYPKMMEBKMILODGOOXPBOAEOXMIKXNDSWOBOWKSXMYXCDKXDSXGKFOVODDBKXCPYBW
Plain text = areaofaccuracybetweenfrequencyandtimeremainconstantinwavelettransform, Cipher text = KBOKYPKMMEBKMILODGOOXPBOAEOXM
IKXNDSWOBOWKSXMYXCDKXDSXGKFOVODDBKXCPYBW
```

Experiment No. 4

Aim: To implement a program in python for password cracking using Brute Force Method. Password contains only lower alphabet characters. Length of password is 4

Theory:

A brute-force attack is an attempt to discover a password by systematically trying every possible combination of letters, numbers, and symbols until you discover the one correct combination that works. If your web site requires user authentication, you are a good target for a brute-force attack.

In this Brute Force attack we will try to crack the password of length 4 containing lower case alphabets by permuting over different combination of length 4 lowercase alphabets.

Steps in Brute Force:

1. Check if password is in correct format i.e. it should be a String, it should be of length 4 and it should contain lower case alphabets. If all these conditions are satisfied program move forward.

2. It checks for each character in password the matching character in alphabets and keep the account of number of iteration. If the character of password matches the character of alphabets it then move forwards to next character of password. This step is repeated until the last character of password.

3. Return the guess password and number of iteration take to guess it.

Implementation:

```
import string
class CrackPassword:
    def __init__(self, password = None):
```

```
        self.check_password(password)
        self.password = password

    def check_password(self,password):
        assert type(password) == str, "Password can only be of type string"
        assert len(password) == 4, 'Password should be of length 4'
        assert password.isalpha() and password.islower(),"Password should only contain lower case alphabets"

    def change_password(self,password = None):
        self.check_password(password)
        self.password = password

    def crack_password(self):
        self.check_password(self.password)
        iterated = 0
        guess = ""

        for x in self.password:
            for y in string.ascii_lowercase:
                iterated += 1
                if x == y:
                    guess += y
                    break

        return (guess,iterated)
```

Output:

```
pass1 = CrackPassword("hill")
guess, iteration = pass1.crack_password()
print(f"Guess = {guess}\nno. of iteration = {iteration}")
```

```
Guess = hill
no. of iteration = 41
```

```
pass1 = CrackPassword("crak")
guess, iteration = pass1.crack_password()
print(f"Guess = {guess}\nno. of iteration = {iteration}")
```

```
Guess = crak
no. of iteration = 33
```

```
pass1 = CrackPassword(1234)
guess, iteration = pass1.crack_password()
print(f"Guess = {guess}\nno. of iteration = {iteration}")
```

```
AssertionError: Password can only be of type string
```

Experiment No. 5

Aim: Implementation of RSA Cryptosystem:

1. Implementation of RSA Key Generation Algorithm.

2. Encryption and decryption using RSA algorithm.

Theory:

The **RSA algorithm** is an asymmetric cryptography algorithm; this means that it uses a *public* key and a *private* key (i.e. two different, mathematically linked keys). As their names suggest, a public key is shared publicly, while a private key is secret and must not be shared with anyone.

The RSA algorithm is named after those who invented it in 1978: Ron Rivest, Adi Shamir, and Leonard Adleman.

The following illustration highlights how asymmetric cryptography works:

How it works

The RSA algorithm ensures that the keys, in the above illustration, are as secure as possible. The following steps highlight how it works:

1. Generating the keys

1. Select two large prime numbers, p and q. The prime numbers need to be large so that they will be difficult for someone to figure out.
2. Calculate $n = p \times q$.
3. Calculate the ***totient*** function; $\varphi(n) = (p-1)(q-1)$.
4. Select an integer e, such that e is ***co-prime*** to $\varphi(n)$ and $1 < e < \varphi(n)$. The pair of numbers (n, e) makes up the public key.

Note: Two integers are co-prime if the only positive integer that divides them is 1.

Calculate $d => e.d \equiv 1 mod \varphi(n)$

d can be found using the ***Extended Euclidean algorithm***. The pair (n, d) makes up the private key.

2. Encryption

Given a plaintext P, represented as a number, the cipher text C is calculated as:

$C = P^e \equiv n.$

3. Decryption

Using the private key (n, d), the plaintext can be found using:

$$P = C^d \equiv n$$

Example:

Assuming receiver selects $p = 193 \; and \; q = 103$

$n = p \times q = 193 \times 103 = 19879$

$\varphi(n) = (p-1) \times (q-1) = 192 \times 102 = 19584$

Considering, e = 1901 which is indeed coprime to $\varphi(n)$

$d = e^{-1} \equiv \varphi(n) = 3173$

If sender wants to send the Plain text $P = 997$ using public key e it will encrypted as follows:

$C = P^e \equiv n = 997^{1901} \equiv 19879 = 7915$

This Cipher text $C = 7915$ will be received at receiver side and it will use its own private key d to decrypt it:

$$P = C^d \equiv n = 7915^{3173} \equiv 19879 = 997$$

Implementation:

```
import math
import random

#implementation of RSA key generation algorithm and encryption, decryption using same key
def gcd(a,b):
    if not b:
        return a
    return gcd(b,a%b)

class RSA:
  def __init__(self):
    self.primes = []
    self.generate_totient()
    self.generate_keys()

  def generate_random_prime(self):
    self.primes.append(2)
    for i in range(3,200):
      if not i%2:
        continue
      Flag = True
      for j in range(2,i):
        if not i%j:
          Flag = False
          break
      if Flag:
        self.primes.append(i)
```

```
def select_primes(self):
    self.generate_random_prime()
    p = random.sample(self.primes, 1)[0]
    q = p
    while(q == p):
        q = random.sample(self.primes, 1)[0]
    self.p = p
    self.q = q

def generate_totient(self):
    self.select_primes()
    self.n = self.p * self.q
    self.totient_n = (self.p-1)*(self.q-1)

def multiplicative_inverse(self):

    a,m,x,y = self.e, self.totient_n,1,0
    while (a > 1):

        q = a // m
        t = m
        m = a % m
        a = t
        t = y
        y = x - q * y
        x = t

    self.e_inverse = x

def generate_keys(self):
    ls = [x for x in range(2,self.totient_n)]
```

```
        e = None
        ls = random.sample(ls,len(ls))
        for x in ls:
            k = gcd(self.totient_n,x)
            if k == 1:
                e = x
                break
        self.e = e
        self.multiplicative_inverse()
        self.generate_private_key()

    def generate_private_key(self):
        self.d = self.e_inverse % self.totient_n

    def fast_expo(self,txt,key):
        return (txt**key)%self.n

    def encryption(self,plain_txt):
        return self.fast_expo(plain_txt,self.e)

    def decryption(self, cipher_txt):
        return self.fast_expo(cipher_txt, self.d)

g = RSA()
print(f"p = {g.p}, q = {g.q}, n = {g.n}, totient_n = {g.totient_n} \
public_key = {g.e} public_key_inverse = {g.e_inverse} private_key = {g.d}\n")

encrypted = g.encryption(int(input(f"Enter value in [1,{g.n}] to encrypt it: ")))
print(f"Encrypted value = {encrypted}\n")
decrypted = g.decryption(int(input(f"Enter value for n = {g.n} to decrypt it: ")))
print(f"Decrypted value = {decrypted}")
```

Output:

```
p = 149, q = 151, n = 22499, totient_n = 22200 public_key = 911 public_key_inverse = -6409 private_key = 15791

Enter value in [1,22498] to encrypt it: 22495
Encrypted value = 476

Enter value for n = 22499 to decrypt it: 476
Decrypted value = 22495
```

```
p = 67, q = 137, n = 9179, totient_n = 8976 public_key = 7835 public_key_inverse = -4429 private_key = 4547

Enter value in [1,9178] to encrypt it: 9000
Encrypted value = 8437

Enter value for n = 9179 to decrypt it: 8437
Decrypted value = 9000
```

```
p = 179, q = 113, n = 20227, totient_n = 19936 public_key = 15445 public_key_inverse = -7107 private_key = 12829

Enter value in [1,20226] to encrypt it: 19999
Encrypted value = 13568

Enter value for n = 20227 to decrypt it: 13568
Decrypted value = 19999
```

```
p = 53, q = 139, n = 7367, totient_n = 7176 public_key = 509 public_key_inverse = -1579 private_key = 5597

Enter value in [1,7366] to encrypt it: 1
Encrypted value = 1

Enter value for n = 7367 to decrypt it: 1
Decrypted value = 1
```

```
p = 101, q = 173, n = 17473, totient_n = 17200 public_key = 1739 public_key_inverse = -5341 private_key = 11859

Enter value in [1,17472] to encrypt it: 6942
Encrypted value = 3891

Enter value for n = 17473 to decrypt it: 3891
Decrypted value = 6942
```

Experiment No. 6

Aim: For varying message sizes, generate message digest using MD5 algorithm and check the integrity of message

Theory:

There are two main kinds of encryption algorithms: Symmetric Key Algorithm (SKA) and Asymmetric Key Algorithm (AKA). In additional, there is also another assistant algorithm called: Hash, which compresses message of any length to certain fixed length (message-digest). This process is irreversible. Hash function can be used in many fields such as: digital signature, message integrity test, and message originality etc. Hash algorithm mainly includes: MD×(Message－Digest Algorithm), SHA×(Secure Hash Algorithms), N-Hash, RIPE-MD, and HAVAL etc.

MD5 Algorithm:

MD5 message-digest algorithm is the 5th version of the Message-Digest Algorithm developed by Ron Rivest to produce a 128-bit message digest. MD5 is quite fast than other versions of the message digest, which takes the plain text of 512-bit blocks, which is further divided into 16 blocks, each of 32 bit and produces the 128-bit message digest, which is a set of four blocks, each of 32 bits. MD5 produces the message digest through five steps, i.e. padding, append length, dividing the input into 512-bit blocks, initialising chaining variables a process blocks and 4 rounds, and using different constant it in each iteration.

Use of MD5 Algorithm

It was developed with the main motive of security as it takes an input of any size and produces an output if a 128-bit hash value. To be considered cryptographically secure, MD5 should meet two requirements:

1. It is impossible to generate two inputs that cannot produce the same hash function.

2. It is impossible to generate a message having the same hash value.

Initially, MD5 was developed to store one way hash of a password, and some file servers also provide pre-computed MD5 checksum of a file so that the user can compare the checksum of the downloaded file to it. Most Unix based Operating Systems include MD5 checksum utilities in their distribution packages.

MD5 produces an output of 128-bit hash value. This encryption of input of any size into hash values undergoes 5 steps, and each step has its predefined task.

Step1: Append Padding Bits

- Padding means adding extra bits to the original message. So in MD5 original message is padded such that its length in bits is congruent to 448 modulo 512. Padding is done such that the total bits are 64 less, being a multiple of 512 bits length.
- Padding is done even if the length of the original message is already congruent to 448 modulo 512. In padding bits, the only first bit is 1, and the rest of the bits are 0.

Step 2: Append Length

After padding, 64 bits are inserted at the end, which is used to record the original input length. Modulo 2^64. At this point, the resulting message has a length multiple of 512 bits.

Step 3: Initialize MD buffer.

A four-word buffer (A, B, C, D) is used to compute the values for the message digest. Here A, B, C, D are 32- bit registers and are initialized in the following way

Word A	01	23	45

Word B	89	Ab	Cd
Word C	Fe	Dc	Ba
Word D	76	54	32

Step 4: Processing message in 16-word block

MD5 uses the auxiliary functions, which take the input as three 32-bit numbers and produce 32-bit output. These functions use logical operators like OR, XOR, NOR.

F(X, Y, Z)	XY v not (X)Z
G(X, Y, Z)	XZ v Y not (Z)
H(X, Y, Z)	X xor Y xor Z
I(X, Y, Z)	Y xor (X v not (Z))

The content of four buffers are mixed with the input using this auxiliary buffer, and 16 rounds are performed using 16 basic operations.

Output-

After all, rounds have performed, the buffer A, B, C, D contains the MD5 output starting with lower bit A and ending with higher bit D.

Below are the advantages and disadvantages:

- MD5 Algorithms are useful because it is easier to compare and store these smaller hashes than store a large variable length text. It is a widely used algorithm for one-way hashes used to verify without necessarily giving the original value. Unix systems use the MD5 Algorithm to store the passwords of the user in a 128-bit encrypted format. MD5 algorithms are widely used to check the integrity of the files.
- Moreover, it is very easy to generate a message digest of the original message using this algorithm. It can perform the message digest of a message having any number of bits; it is not limited to a message in the multiples of 8, unlike MD5sum, which is limited to octets.
- But for many years, MD5 has prone to hash collision weakness, i.e. it is possible to create the same hash function for two different inputs. MD5 provides no security over these collision attacks. Instead of MD5, SHA (Secure Hash Algorithm, which produces 160-bit message digest and designed by NSA to be a part of digital signature algorithm) is now acceptable in the cryptographic field for generating the hash function as it is not easy to produce SHA-I collision and till now no collision has been produced yet.
- Moreover, it is quite slow then the optimized SHA algorithm. SHA is much secure than the MD5 algorithm, and moreover, it can be implemented in existing technology with exceeding rates, unlike MD5. Nowadays, new hashing algorithms are coming up in the market, keeping in mind higher security of data like SHA256 (which generates 256 bits of signature of a text).

Implementation:

```
import hashlib
class MD5:
    def __init__(self):
        self.__keys = ["Adnan is a human", "Humans are mamals", "Adnan is a mamal"]
        self.digest_keys = [self.digest(key) for key in self.__keys]
        for key, digest_key in zip(self.__keys,self.digest_keys):
            print(f"key = {key}\nEquivalent MD5 hexadecimal digest = {digest_key}\n")
    def digest(self,string):
        return hashlib.md5(string.encode()).hexdigest()
    def check_in_digest_keys(self,string):
        return self.digest(string) in self.digest_keys
    def digest_strings(self,strings):
        for x in strings:
            print(f"Given string: {x}\nMD5 hexadecimal digest value of given string: {self.digest(x)}")
            print(f"present in digest_keys {self.check_in_digest_keys(x)}\n")
obj = MD5()
given_strings = [
                 "Adnan is a human", "Adnan is not a human",
                 "Humans are reptile", "Humans are mamals",
                 "Adnan is a mamal", "Adnan is a reptile"
                ]
obj.digest_strings(given_strings)
```

Output:

```
key = Adnan is a human
Equivalent MD5 hexadecimal digest = a235b8e8f838ea6096318e4e3cdfb920

key = Humans are mamals
Equivalent MD5 hexadecimal digest = 5a55ebe77084688bf175b64fe537334c

key = Adnan is a mamal
Equivalent MD5 hexadecimal digest = 2dddc6e1eec9674281626983e03798dd

Given string: Adnan is a human
MD5 hexadecimal digest value of given string: a235b8e8f838ea6096318e4e3cdfb920
present in digest_keys True

Given string: Adnan is not a human
MD5 hexadecimal digest value of given string: 959be596779c379e1116378ac2820539
present in digest_keys False

Given string: Humans are reptile
MD5 hexadecimal digest value of given string: b55d5122a5da1821b8d3fbb0e01bfec0
present in digest_keys False

Given string: Humans are mamals
MD5 hexadecimal digest value of given string: 5a55ebe77084688bf175b64fe537334c
present in digest_keys True

Given string: Adnan is a mamal
MD5 hexadecimal digest value of given string: 2dddc6e1eec9674281626983e03798dd
present in digest_keys True

Given string: Adnan is a reptile
MD5 hexadecimal digest value of given string: b11741565fcdde9ae5286280a9f355a8
present in digest_keys False
```

Experiment No. 7

Aim: For varying message sizes, generate message digest using SHA-1 algorithm and check the integrity of message

Theory:

There are two main kinds of encryption algorithms: Symmetric Key Algorithm (SKA) and Asymmetric Key Algorithm (AKA). In additional, there is also another assistant algorithm called: Hash, which compresses message of any length to certain fixed length (message-digest). This process is irreversible. Hash function can be used in many fields such as: digital signature, message integrity test, and message originality etc. Hash algorithm mainly includes: MD×(Message－Digest Algorithm), SHA×(Secure Hash Algorithms), N-Hash, RIPE-MD, and HAVAL etc.

SHA1 Algorithm:

SHA1 also is another main hash algorithm, which is primarily based on MD4 principle. Because it produces 160-bit output, so SHA1 needs five 32-bit registers. But the method of message digest and data filling works just like MD5 algorithm. SHA has 4 main rounds iterative, and each round has 20 steps operations. There are only some minute differences including: rotate left, addition constant, and non-linear function. Here are the five initialized variables:

$$H0 = 0x67452301$$

$$H1 = 0xEFCDAB89$$

$$H2 = 0x98BADCFE$$

$$H3 = 0x10325476$$

$$H4 = 0xC3D2E1F0$$

The constants used in process are stated as follows:

$$k_t = \begin{cases} 5a827999 & 0 \le t \le 19 \\ 6ed9eba1 & 20 \le t \le 39 \\ 8f1bbcdc & 40 \le t \le 59 \\ Ca62c1d6 & 60 \le t \le 79 \end{cases}$$

We can use these constants to judge whether one procedure has adopted SHA1 algorithm. After finishing initialization, we then start the main rotation of algorithm.

SHA1 algorithm consists of 6 tasks:

Task 1. Appending Padding Bits. The original message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. The padding rules are:

- The original message is always padded with one bit "1" first.
- Then zero or more bits "0" are padded to bring the length of the message up to 64 bits less than a multiple of 512.

Task 2. Appending Length. 64 bits are appended to the end of the padded message to indicate the length of the original message in bytes. The rules of appending length are:

- The length of the original message in bytes is converted to its binary format of 64 bits. If overflow happens, only the low-order 64 bits are used.
- Break the 64-bit length into 2 words (32 bits each).
- The low-order word is appended first and followed by the high-order word.

Task 3. Preparing Processing Functions. SHA1 requires 80 processing functions defined as:

$$f(t;B,C,D) = \begin{cases} (BandC)or((notB)andD) & 0 \le t \le 19 \\ BxorCxorD & 20 \le t \le 39 \\ (BandC)or(BandD)or(CandD) & 40 \le t \le 59 \\ BxorCxorD & 60 \le t \le 79 \end{cases}$$

Task 6. Processing Message in 512-bit Blocks. This is the main task of SHA1 algorithm, which loops through the padded and appended message in blocks of 512 bits each. For each input block, a number of operations are performed. This task can be described in the following pseudo code slightly modified from the RFC 3174's method 1:

Input and predefined functions:

M[1, 2, ..., N]: Blocks of the padded and appended message

f(0;B,C,D), f(1,B,C,D), ..., f(79,B,C,D): Defined as above

K(0), K(1), ..., K(79): Defined as above

H0, H1, H2, H3, H4: Word buffers with initial values

Algorithm:

```
For loop on k = 1 to N
  (W(0),W(1),...,W(15)) = M[k] /* Divide M[k] into 16 words */
  For t = 16 to 79 do:
    W(t) = (W(t-3) XOR W(t-8) XOR W(t-14) XOR W(t-16)) <<< 1
  A = H0, B = H1, C = H2, D = H3, E = H4
  For t = 0 to 79 do:
    TEMP = A<<<5 + f(t;B,C,D) + E + W(t) + K(t)
    E = D, D = C, C = B<<<30, B = A, A = TEMP
  End of for loop
```

H0 = H0 + A, H1 = H1 + B, H2 = H2 + C, H3 = H3 + D, H4 = H4 + E

End of for loop

Output:

H0, H1, H2, H3, H4: Word buffers with final message digest

Implementation:

```
import hashlib

class SHA1:
    def __init__(self):
        self.__keys = ["Adnan is a human", "Humans are mamals", "Adnan is a mamal"]
        self.digest_keys = [self.digest(key) for key in self.__keys]
        for key, digest_key in zip(self.__keys,self.digest_keys):
            print(f"key = {key}\nEquivalent Sha1 hexadecimal digest = {digest_key}\n")
    def digest(self,string):
        return hashlib.sha1(string.encode()).hexdigest()
    def check_in_digest_keys(self,string):
        return self.digest(string) in self.digest_keys
    def digest_strings(self,strings):
        for x in strings:
            print(f"Given string: {x}\nSha1 hexadecimal digest value of given string: {self.digest(x)}")
```

```
        print(f"present in digest_keys {self.check_in_digest_keys(x)}\n")

obj = SHA1()

given_strings = [

            "Adnan is a human", "Adnan is not a human",

            "Humans are reptile", "Humans are mamals",

            "Adnan is a mamal", "Adnan is a reptile"

           ]

obj.digest_strings(given_strings)
```

Output:

```
key = Adnan is a human
Equivalent Sha1 hexadecimal digest = b534ccf13fdcdf68ef0ba785c891f3ce6b082ab6

key = Humans are mamals
Equivalent Sha1 hexadecimal digest = 01692710f02ef3368ff45b21c96975ac492dc059

key = Adnan is a mamal
Equivalent Sha1 hexadecimal digest = ecda2d5fcb7338c1418adef04862a16c05e70f27

Given string: Adnan is a human
Sha1 hexadecimal digest value of given string: b534ccf13fdcdf68ef0ba785c891f3ce6b082ab6
present in digest_keys True

Given string: Adnan is not a human
Sha1 hexadecimal digest value of given string: 220c34a00d4e3010da7de455d40682fc94cb549d
present in digest_keys False

Given string: Humans are reptile
Sha1 hexadecimal digest value of given string: 461b008a39d7f033f1f001851ee0bd1ae1c448f3
present in digest_keys False

Given string: Humans are mamals
Sha1 hexadecimal digest value of given string: 01692710f02ef3368ff45b21c96975ac492dc059
present in digest_keys True

Given string: Adnan is a mamal
Sha1 hexadecimal digest value of given string: ecda2d5fcb7338c1418adef04862a16c05e70f27
present in digest_keys True

Given string: Adnan is a reptile
Sha1 hexadecimal digest value of given string: aaa8c10433d9b74a0f64c66a4445be451f459574
present in digest_keys False
```

Experiment No. 8

Aim: To implement Diffie Hellman Algorithm in python and virtual lab.

Theory:

Whitefield Diffie and Martin Hellman develop Diffie Hellman key exchange Algorithms in 1976 to overcome the problem of key agreement and exchange. It enables the two parties who want to communicate with each other to agree on a symmetric key, a key that can be used for encrypting and decryption; Diffie Hellman key exchange algorithm can be used for only key exchange, not for encryption and decryption process. The algorithm is based on mathematical principles.

Diffie Hellman Algorithm

1. Shared values

- p: p is a prime number
- g: $g <$ p and $g \in < Z_p^*, * >$ is a generator.

2. Key generation for user A

- Select a Private key X_A Here, $X_A < g$

Now, Calculation of Public key $R_A = g^{X_A} mod\ p$

3. Key generation for user B

- Select a Private key X_B Here, $X_B < g$
- Now, Calculation of Public key $R_B = g^{X_B} mod\ p$

4. Calculation of Secret Key by A

- $key_A = R_B^{X_A} \, mod \, p$

5. Calculation of Secret Key by B

- $key_B = R_A^{X_B} \, mod \, p$

$$key = key_A = key_B = g^{X_A X_B} mod \, p$$

Example

$p = 61, g = 45, X_A = 36, X_B = 29$

$$R_A = 45^{36} mod 61 = 20$$

$$R_B = 45^{29} mod 61 = 19$$

$$key = 45^{29*36} \, mod 61 = 58$$

<u>Implementation</u>:

```
import random
import math

def check_multiplicative_inverse(x,y):
    if not y:
        return x

    return check_multiplicative_inverse(y,x%y)

class DH:

    def __init__(self,prime):
        assert self.check_prime(prime), "Number is not prime"
        self.prime = prime

        self.relative_primes = []
        for x in range(2,self.prime):
```

```
            if check_multiplicative_inverse(self.prime,x) == 1:
                self.relative_primes.append(x)
        self.generator = random.choice(self.relative_primes)

        self.alice_R_one()
        self.bob_R_two()
        self.alice_secret_key()
        self.bob_secret_key()

    def check_prime(self,prime):

        if prime == 1:
            return False
        if prime == 2:
            return True
        if not prime%2:
            return False

        for x in range(3,math.ceil(math.sqrt(prime))):
            if not prime%x:
                return False
        return True

    def alice_R_one(self):
        self.x = random.randint(0,self.prime)
        self.R_one = pow(self.generator,self.x,self.prime)

    def bob_R_two(self):
        self.y = random.randint(0,self.prime)
        self.R_two = pow(self.generator,self.y,self.prime)

    def alice_secret_key(self):
        self.key_A = pow(self.R_two,self.x,self.prime)


    def bob_secret_key(self):
        self.key_B = pow(self.R_one,self.y,self.prime)

obj = DH(int(input("Please Enter a prime Number: ")))
print(f"prime number: {obj.prime},  generator: {obj.generator}\n\
A secret key: {obj.x},  B secret key: {obj.y}\n\
A public key: {obj.R_one},  B public key: {obj.R_two}\n\
Secret key calculated by A: {obj.key_A},  Secret key calculated by B: {obj.key_B}")
```

Output:

```
Please Enter a prime Number: 67
prime number: 67,  generator: 19
A secret key: 2,  B secret key: 36
A public key: 26,  B public key: 25
Secret key calculated by A: 22,  Secret key calculated by B: 22
```

```
Please Enter a prime Number: 15

AssertionError: Number is not prime
```

```
Please Enter a prime Number: 83
prime number: 83,  generator: 8
A secret key: 9,  B secret key: 32
A public key: 5,  B public key: 33
Secret key calculated by A: 75,  Secret key calculated by B: 75
```

Vlab Output:

Experiment No 9

Aim: Study of packet sniffer tool "wireshark":

a) Download and install wireshark and capture icmp, tcp, and http packets in promiscuous mode.

b) Explore how the packets can be traced based on different filters.

Requirements: Compatible version of Wireshark.

Theory:

Wireshark is a network packet analyzer. A network packet analyzer presents captured packet data in as much detail as possible.

You could think of a network packet analyzer as a measuring device for examining what's happening inside a network cable, just like an electrician uses a voltmeter for examining what's happening inside an electric cable (but at a higher level, of course).

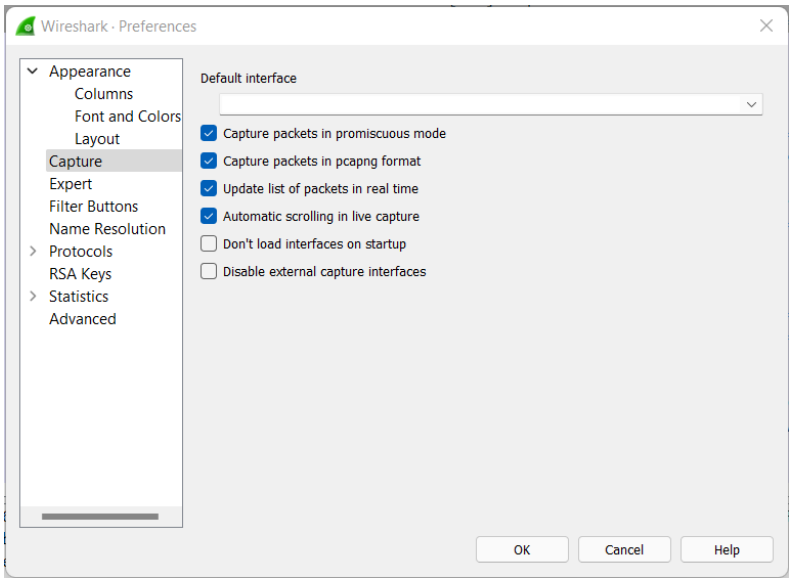
In the past, such tools were either very expensive, proprietary, or both. However, with the advent of Wireshark, that has changed. Wireshark is available for free, is open source, and is one of the best packet analyzers available today.

Intended purposes

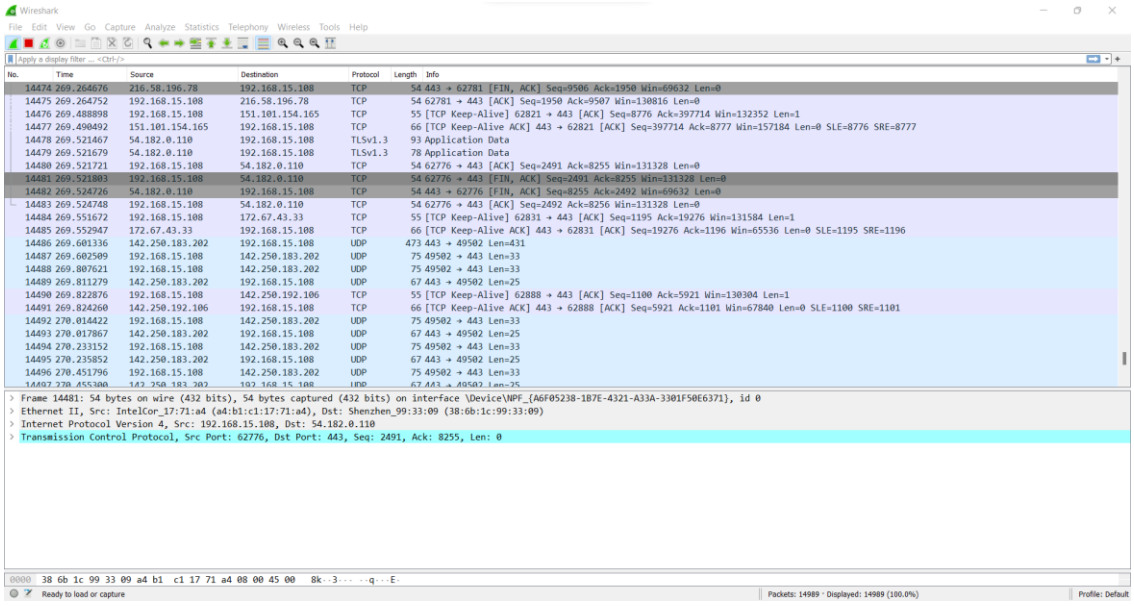
- Network administrators use it to *troubleshoot network problems*
- Network security engineers use it to *examine security problems*
- QA engineers use it to *verify network applications*
- Developers use it to *debug protocol implementations*
- People use it to *learn network protocol* internals

Output:

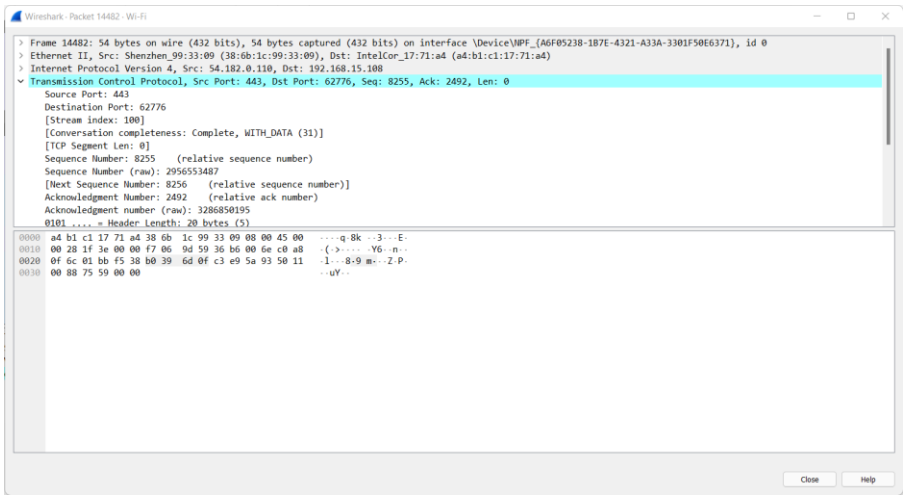
Turning on promiscuous mode



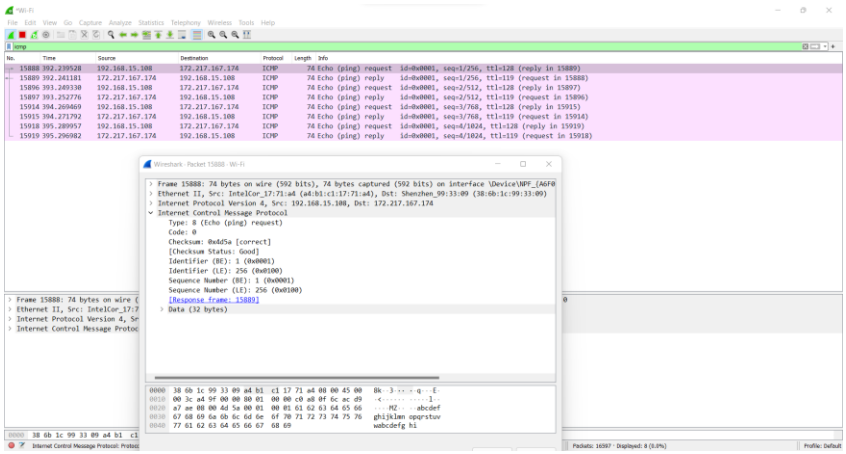
Monitoring all packets routing through Wi-Fi



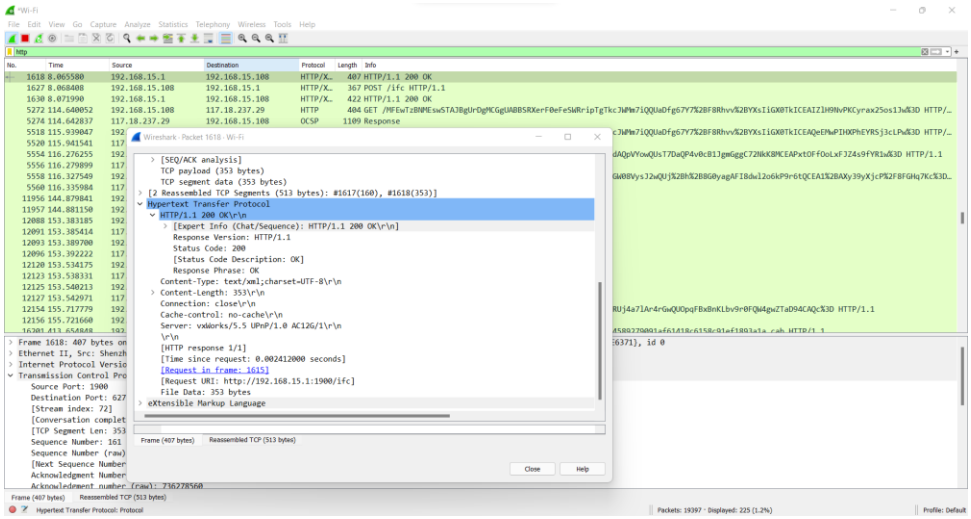
TCP packet



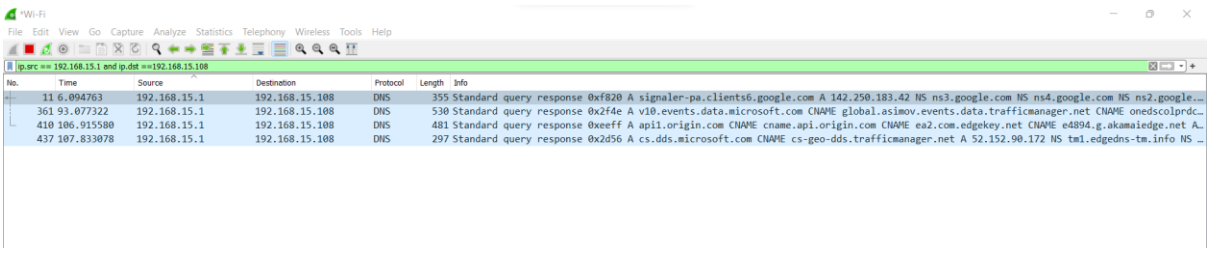
Filter ICMP packet



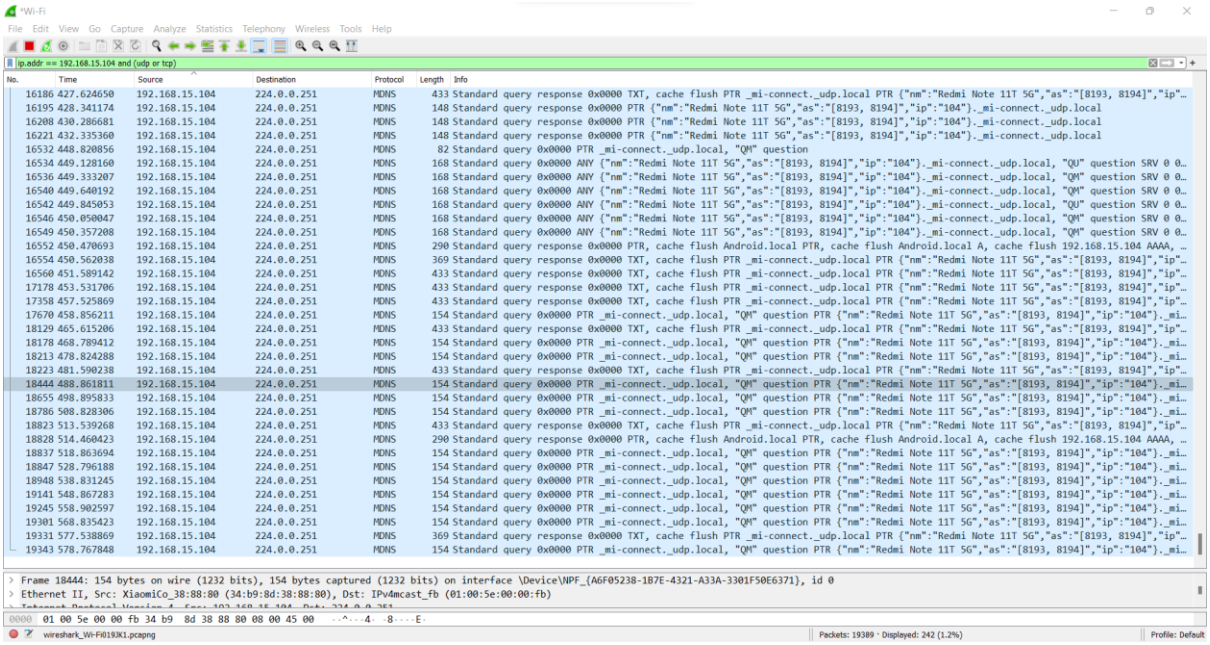
HTTP packet



Filter source and destination IP



Filter IP and protocols



Experiment No 10

Aim: To Study the use of network reconnaissance tools like WHOIS, dig, tracert, nslookup.

Theory:

WHOIS

WHOIS is a TCP-based query and response protocol that is commonly used to provide information services to Internet users. It returns information about the registered Domain Names, an IP address block, Name Servers and a much wider range of information services.

To use WHOIS in windows we need to install WhoIs from Microsoft site and extract its folder and to switch to its directory in cmd.

Dig

Dig (Domain Information Groper) is a powerful command-line tool for querying DNS name servers.

The dig command, allows you to query information about various DNS records, including host addresses, mail exchanges, and name servers. It is the most commonly used tool among system administrators for troubleshooting DNS problems because of its flexibility and ease of use.

To use dig in windows we need to install Bind and set path for it. Bind also provides other commands support such as tracert and nslookup.

Tracert

This diagnostic tool determines the path taken to a destination by sending Internet Control Message Protocol (ICMP) echo Request or ICMPv6 messages to the destination with incrementally increasing time to live (TTL) field values. Each router along the path is required to decrement the TTL in an IP packet by at least 1 before forwarding it. Effectively, the TTL

is a maximum link counter. When the TTL on a packet reaches 0, the router is expected to return an ICMP time Exceeded message to the source computer.

This command determines the path by sending the first echo Request message with a TTL of 1 and incrementing the TTL by 1 on each subsequent transmission until the target responds or the maximum number of hops is reached. The maximum number of hops is 30 by default and can be specified using the **-h** parameter.

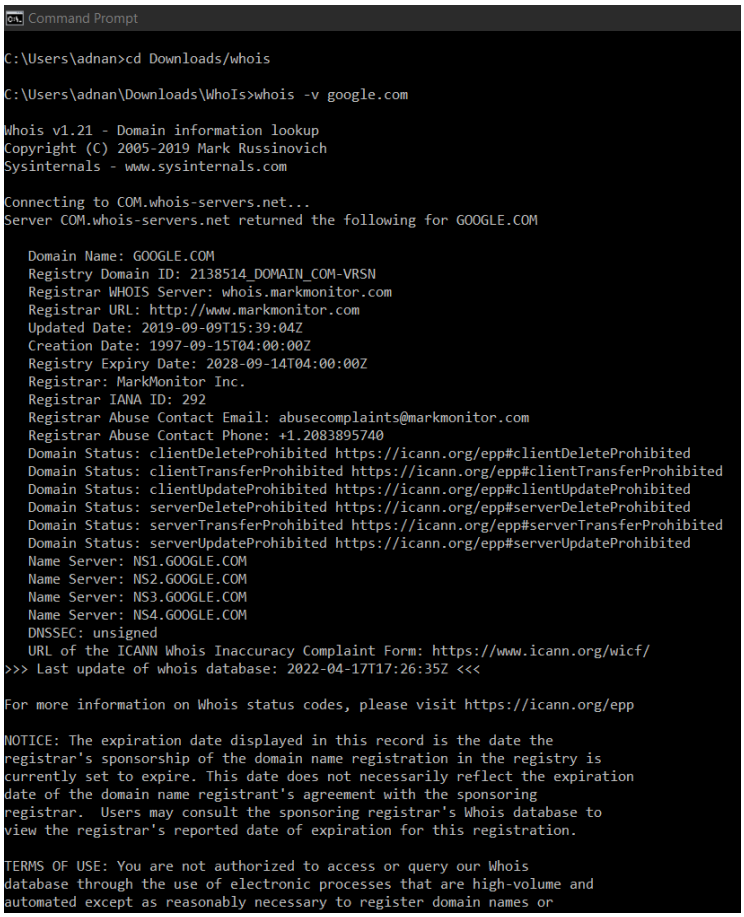
Nslookup

The nslookup command queries internet domain name servers in two modes. Interactive mode allows you to query name servers for information about various hosts and domains, or to print a list of the hosts in a domain. In noninteractive mode, the names and requested information are printed for a specified host or domain.

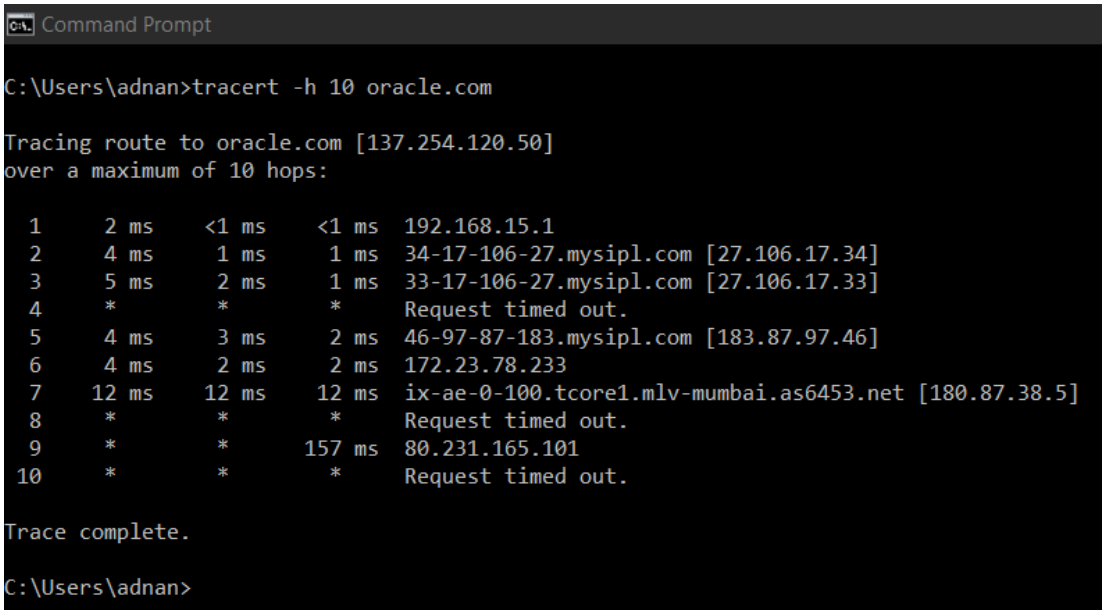
The nslookup command enters interactive mode when no arguments are given, or when the first argument is a - (minus sign) and the second argument is the host name or internet address of a name server. When no arguments are given, the command queries the default name server. The nslookup command enters non-interactive mode when you give the name or internet address of the host to be looked up as the first argument. The optional second argument specifies the host name or address of a name server.

Output:

Whois



Tracert



Dig

```
Command Prompt

C:\Users\adnan>dig yahoo.com trace

; <<>> DiG 9.16.27 <<>> yahoo.com trace
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4583
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 5, ADDITIONAL: 10

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;yahoo.com.                     IN      A

;; ANSWER SECTION:
yahoo.com.              1237    IN      A       74.6.143.25
yahoo.com.              1237    IN      A       74.6.231.20
yahoo.com.              1237    IN      A       98.137.11.163
yahoo.com.              1237    IN      A       74.6.231.21
yahoo.com.              1237    IN      A       74.6.143.26
yahoo.com.              1237    IN      A       98.137.11.164

;; AUTHORITY SECTION:
yahoo.com.              51952   IN      NS      ns4.yahoo.com.
yahoo.com.              51952   IN      NS      ns3.yahoo.com.
yahoo.com.              51952   IN      NS      ns2.yahoo.com.
yahoo.com.              51952   IN      NS      ns1.yahoo.com.
yahoo.com.              51952   IN      NS      ns5.yahoo.com.

;; ADDITIONAL SECTION:
ns4.yahoo.com.          312358  IN      A       98.138.11.157
ns1.yahoo.com.          312358  IN      A       68.180.131.16
ns1.yahoo.com.          53595   IN      AAAA    2001:4998:1b0::7961:686f:6f21
ns5.yahoo.com.          53166   IN      A       202.165.97.53
ns5.yahoo.com.          53595   IN      AAAA    2406:2000:1d0::7961:686f:6f21
ns2.yahoo.com.          312358  IN      A       68.142.255.16
ns2.yahoo.com.          53595   IN      AAAA    2001:4998:1c0::7961:686f:6f21
ns3.yahoo.com.          1217    IN      A       27.123.42.42
ns3.yahoo.com.          745     IN      AAAA    2406:8600:f03f:1f8::1003

;; Query time: 15 msec
;; SERVER: 192.168.15.1#53(192.168.15.1)
;; WHEN: Sun Apr 17 23:11:01 India Standard Time 2022
;; MSG SIZE  rcvd: 416

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 43794
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1
```

Nslookup

```
Command Prompt

C:\Users\adnan>nslookup -type=ns redhat.com
Server:  UnKnown
Address:  192.168.15.1

Non-authoritative answer:
redhat.com      nameserver = a9-65.akam.net
redhat.com      nameserver = a10-65.akam.net
redhat.com      nameserver = a28-64.akam.net
redhat.com      nameserver = a13-66.akam.net
redhat.com      nameserver = a16-67.akam.net
redhat.com      nameserver = a1-68.akam.net

a13-66.akam.net internet address = 2.22.230.66
a13-66.akam.net AAAA IPv6 address = 2600:1480:800::42
a9-65.akam.net  internet address = 184.85.248.65
a9-65.akam.net  AAAA IPv6 address = 2a02:26f0:117::41
a10-65.akam.net internet address = 96.7.50.65
a16-67.akam.net internet address = 23.211.132.67
a16-67.akam.net AAAA IPv6 address = 2600:1406:1b::43
a28-64.akam.net internet address = 95.100.173.64
```

```
C:\Users\adnan>nslookup -type=any google.com
Server:  UnKnown
Address:  192.168.15.1

Non-authoritative answer:
google.com      AAAA IPv6 address = 2404:6800:4009:810::200e
google.com
        primary name server = ns1.google.com
        responsible mail addr = dns-admin.google.com
        serial  = 442195542
        refresh = 900 (15 mins)
        retry   = 900 (15 mins)
        expire  = 1800 (30 mins)
        default TTL = 60 (1 min)
google.com      internet address = 142.250.183.110
google.com      nameserver = ns1.google.com
google.com      nameserver = ns4.google.com
google.com      nameserver = ns2.google.com
google.com      nameserver = ns3.google.com

google.com      nameserver = ns3.google.com
google.com      nameserver = ns2.google.com
google.com      nameserver = ns4.google.com
google.com      nameserver = ns1.google.com
ns4.google.com  internet address = 216.239.38.10
ns4.google.com  AAAA IPv6 address = 2001:4860:4802:38::a
ns3.google.com  internet address = 216.239.36.10
ns3.google.com  AAAA IPv6 address = 2001:4860:4802:36::a
ns1.google.com  internet address = 216.239.32.10
ns1.google.com  AAAA IPv6 address = 2001:4860:4802:32::a
ns2.google.com  internet address = 216.239.34.10
ns2.google.com  AAAA IPv6 address = 2001:4860:4802:34::a
```

```
C:\Users\adnan>nslookup -type=mx facebook.com
Server:  UnKnown
Address:  192.168.15.1

Non-authoritative answer:
facebook.com    MX preference = 10, mail exchanger = smtpin.vvv.facebook.com

facebook.com    nameserver = b.ns.facebook.com
facebook.com    nameserver = c.ns.facebook.com
facebook.com    nameserver = d.ns.facebook.com
facebook.com    nameserver = a.ns.facebook.com
a.ns.facebook.com       internet address = 129.134.30.12
a.ns.facebook.com       AAAA IPv6 address = 2a03:2880:f0fc:c:face:b00c:0:35
b.ns.facebook.com       internet address = 129.134.31.12
b.ns.facebook.com       AAAA IPv6 address = 2a03:2880:f0fd:c:face:b00c:0:35
c.ns.facebook.com       internet address = 185.89.218.12
c.ns.facebook.com       AAAA IPv6 address = 2a03:2880:f1fc:c:face:b00c:0:35
d.ns.facebook.com       internet address = 185.89.219.12
d.ns.facebook.com       AAAA IPv6 address = 2a03:2880:f1fd:c:face:b00c:0:35
```

Experiment No. 11

Aim: To perform Digital Signature Scheme Experiment using Virtual lab.

Theory:

What Are Digital Signatures?

The objective of digital signatures is to authenticate and verify documents and data. This is necessary to avoid tampering and digital modification or forgery during the transmission of official documents.

With one exception, they work on the public key cryptography architecture. Typically, an asymmetric key system encrypts using a public key and decrypts with a private key. For digital signatures, however, the reverse is true. The signature is encrypted using the private key and decrypted with the public key. Because the keys are linked, decoding it with the public key verifies that the proper private key was used to sign the document, thereby verifying the signature's provenance.

RSA Signatures

The RSA public-key cryptosystem provides a digital signature scheme (sign + verify), based on the math of the modular exponentiations and discrete logarithms and the computational difficulty of the RSA problem (and its related integer factorization problem).

Key Generation

The RSA algorithm uses **keys** of size 1024, 2048, 4096, ..., 16384 bits. RSA supports also longer keys (e.g. 65536 bits), but the performance is too slow for practical use (some operations may take several minutes or even hours). For 128-bit security level, a 3072-bit key is required.

The **RSA key-pair** consists of:

- public key {$\boldsymbol{n}$, $\boldsymbol{e}$}
- private key {$\boldsymbol{n}$, $\boldsymbol{d}$}

The numbers $\boldsymbol{n}$ and $\boldsymbol{d}$ are typically big integers (e.g. 3072 bits), while $\boldsymbol{e}$ is small, typically 65537.

By definition, the RSA key-pairs has the following property:

$$(m^e)^d \equiv (m^d)^e \equiv m(mod n)$$

for all $\boldsymbol{m}$ in the range [0...$\boldsymbol{n}$)

RSA Sign

Signing a message ***msg*** with the private key exponent $\boldsymbol{d}$:

1. Calculate the message hash: $\boldsymbol{h} = hash(\boldsymbol{msg})$
2. Encrypt $\boldsymbol{h}$ to calculate the signature: $s = h^d(mod n)$

The hash $\boldsymbol{h}$ should be in the range [0...$\boldsymbol{n}$). The obtained **signature** $\boldsymbol{s}$ is an integer in the range [0...$\boldsymbol{n}$).

RSA Verify Signature

Verifying a signature $\boldsymbol{s}$ for the message ***msg*** with the public key exponent $\boldsymbol{e}$:

Calculate the message hash: $\boldsymbol{h} = hash(\boldsymbol{msg})$

Decrypt the signature: $h' = s^e(mod n)$

Compare $\boldsymbol{h}$ with $\boldsymbol{h'}$ to find whether the signature is valid or not

If the signature is correct, then the following will be true:

$h' = s^e(mod n) = (h^d)^e(mod n) = h$

Vlab output:

Digitally sign the plaintext with Hashed RSA.

Plaintext (string):

testing SHA-1

Hash output(hex):

dc724af18fbdd4e59189f5fe768a5f8311527050

Input to RSA(hex):

dc724af18fbdd4e59189f5fe768a5f8311527050 Apply RSA

Digital Signature(hex):

```
4e29ebb817be5d8fc591a631626dcd19fd8ef343cc42e6b6e875397394a1b170
2ece10d16d571833be50b3cbc3dd785b8ea3c0fa36057e0637210a0e4f89f8b5
29eee4bd6943f30858cd6c6f64f8227926eb6dc8b4fc151d1380ae9640fabdda
f535d0b1f0766c2556619bdb0b3e9ab159f3b8d106f559c1a0ba9d8164ca6e78
```

Digital Signature(base64):

```
TinruBe+XY/FkaYxYm3NGf2O80PMQua26HU5c5ShsXAuzhDRbVcYM75Qs8vD3Xhb
jqPA+jYFfgY3IQoOT4n4tSnu5L1pQ/MIWM1sb2T4Inkm623ItPwVHROArpZA+r3a
9TXQsfB2bCVWYZvbCz6asVnzuNEG9VnBoLqdgWTKbng=
```

Status:

Time: 5ms

RSA public key

Public exponent (hex, F4=0x10001):

10001

Modulus (hex):

```
a5261939975948bb7a58dffe5ff54e65f0498f9175f5a09288810b8975871e99
af3b5dd94057b0fc07535f5f97444504fa35169d461d0d30cf0192e307727c06
5168c788771c561a9400fb49175e9e6aa4e23fe11af69e9412dd23b0cb6684c4
c2429bce139e848ab26d0829073351f4acd36074eafd036a5eb83359d2a698d3
```

1024 bit | 1024 bit (e=3) | 512 bit | 512 bit (e=3)

Assignment 1

Q.1) Encrypt "The key is hidden under the door" using playfair ciphers with key word "domestic"

So Ans)

D	O	M	E	S
T	I	C	A	B
F	G	H	J	K
L	N	P	Q	R
U	V	W	X	Y/Z

Plain text: ["TH", "EK", "EY", "IS", "HI" "DX", "DE", "NX"]

↑ dogus character

ENCRYPTION: "TH" → "CF"

"EK" → "SJ"; "EY" → "SX"

"IS" → "BO"; ~~"HI" → "CG"~~

"HI" → "GC"; "DX" → "EU"

~~"DE" → "EU"; "NX" → "QV"~~

~~Encypted text: CFSJSX BOGCEUEUQV~~

"DE" → "OS"; "NX" → "QV"

Cipher text: CFSJSXBOGCEUOSQV

Q.2) Encrypt the given message using Autokey cipher, Key = 7 & the Message is: "The house is being sold tonight".

Ans)

Plain text	T	H	E	H	O	U	S	E	T	S	B	E	I	N	G
~~Key Stream~~															
P's values	19	07	04	07	14	20	18	04	08	18	01	04	08	13	6
Key Stream	07	19	07	04	07	14	20	18	04	08	18	01	04	08	13
C's values	00	00	11	11	21	08	12	22	12	00	19	05	12	21	20
Cipher text	A	A	L	L	V	I	M	W	M	A	T	F	M	V	U

Plain text	S	O	L	D	T	O	N	I	G	H	T
P's values	18	14	11	03	19	14	13	08	06	07	19
Key Stream	26	18	14	11	03	19	14	13	08	06	07
Cipher text Values	24	6	25	14	22	07	01	21	14	13	00
Cipher text	Y	G	Z	O	W	H	B	V	O	N	A

Cipher text: AALLVIMWMATFMVTYGZO
WHBVONA

Q.3) Use the Playfair cipher with the keyword: "HEALTH" to encipher the message "Life is full of Surprises".

H	E	A	L	T
B	C	D	F	G
I	J/K	M	N	O
P	Q	R	S	U
V	W	X	Y	Z

Plain text = {"Li", "fe", "is", "fu", "ll"}

Plain text = ["LI", "FE", "IS", "FO", "X", "LO", "FS", "UR", "PR", "IS", "ES"]

Encryption: "LI" → "HN", "FE" → "CL"
"IS" → "NP", "FO" → "GS"
"LX" → "AY", "LO" → "IN"
"FS" → "NY", "UR" → "PS"
"RR" → "QS", "IS" → "NP"
"ES" → "LQ"

Cipher text: HNCLNPGSAYINNYPSQSNP
LQ.

Q.4) Encrypt the plain text message "SECURITY" using affine ciphex with the key pair (3, 7). Decrypt to get back original plaintext.

Ans) $K_1 = 3$; $K_2 = 7$ $\Rightarrow C(K_1, K_2) = (PK_1 + K_2) \bmod 26$

multiplicative, Additive

$$\boxed{K_2^{-1} = -K_2 = -7}$$

$$K_1 \cdot K_1^{-1} \equiv 1 \pmod{26}$$

i.e $K_1 \cdot (K_1^{-1}) + n(-9) = 1$

$\Rightarrow$ G.C.D $(K_1, n) = 1$

G.C.D $(3, 26) \Rightarrow$ $26 = 3(8) + 2$

$3 = 2(1) + 1$

$2 = 1(2)$

$\therefore$ G.C.D $(3, 26)$

$26 + 3(-8) = 2$

$\therefore 3 + 2(-1) = 1$

$3 + (-1)[26 + 3[-8]] = 1$

$26[-1] + 3[+9] = 1$

$\therefore \boxed{K_1^{-1} = +9}$; $\boxed{K_2^{-1} = -7}$

$$P(K_1^{-1}, K_2^{-1}) = [(C + K_2^{-1}) K_1^{-1}] \pmod{26}$$

Plain text = "SECURITY"

Encryption:

S → 18 → C(3,7)

∴ S ⇒ 18 ⇒ (18×3 + 7)(mod 26) = 9 ⇒ J

E ⇒ 04 ⇒ (04×3+7)(mod26) = 19 ⇒ T

C ⇒ 02 ⇒ (02×3+7)(mod26) = 13 ⇒ N

U ⇒ 20 ⇒ (20×3+7)(mod26) = 15 ⇒ P

R ⇒ 17 ⇒ (17×3+7)(mod26) = 6 ⇒ G

I ⇒ 08 ⇒ (08×3+7)(mod26) = 5 ⇒ F

T ⇒ 19 ⇒ (19×3+7)(mod26) = 12 ⇒ M

Y ⇒ 24 ⇒ (24×3+7)(mod26) = 01 ⇒ B

Cipher text: JTNPGFMB

Decryption:

Character → Value → P(9,7)...

J ⇒ 9 ⇒ [(9-7)×9](mod26) = 18 ⇒ S

T ⇒ 19 ⇒ [(19-7)×9](mod26) = 04 ⇒ E

N ⇒ 13 ⇒ [(13-7)×9](mod26) = 02 ⇒ C

P ⇒ 15 ⇒ [(15-7)×9](mod26) = 20 ⇒ U

G ⇒ 6 ⇒ [(6-7)×9](mod26) = 17 ⇒ R

F ⇒ 5 ⇒ [(5-7)×9](mod26) = 08 ⇒ I

M ⇒ 12 ⇒ [(12-7)×9](mod26) = 19 ⇒ T

B ⇒ 01 ⇒ [(1-7)×9](mod26) = 24 ⇒ Y

Plain text: SECURITY.

Q.5) Use hill cipher to encrypt the text "short" the key to be used is "hill"

Ans) Key = [[K1 = h = 07 , K2 = i = 08] [K3 = i = 11, K4 = i = 11]]

$$K = \begin{bmatrix} 7 & 8 \\ 11 & 11 \end{bmatrix} ; |K| = 77 - 88 = -11$$

Plain text = [[P1 = s = 18 , P2 = h = 07]
[P3 = o = 14 , P4 = r = 17]
[P5 = t = 19 , P6 = x = 23]]
↑ Bogus character.

$$\therefore P = \begin{bmatrix} 18 & 7 \\ 14 & 17 \\ 19 & 23 \end{bmatrix}$$

Encryption:

$$C = (P \cdot K) \bmod 26 = \left(\begin{bmatrix} 18 & 7 \\ 14 & 17 \\ 19 & 23 \end{bmatrix} \begin{bmatrix} 7 & 8 \\ 11 & 11 \end{bmatrix} \right) \bmod 26$$

$$C = \begin{bmatrix} 18\times7 + 7\times11 & 18\times8 + 7\times11 \\ 14\times7 + 17\times11 & 14\times8 + 17\times11 \\ 19\times7 + 23\times11 & 19\times8 + 23\times11 \end{bmatrix} \bmod 26$$

$$C = \begin{bmatrix} 21 & 13 \\ 25 & 13 \\ 22 & 15 \end{bmatrix} \Rightarrow \begin{bmatrix} [21 \Rightarrow V, & 13 \Rightarrow N] \\ [25 \Rightarrow Z, & 13 \Rightarrow N] \\ [22 \Rightarrow W, & 15 \Rightarrow P] \end{bmatrix}$$

∴ Ciphere text = VNZNWP.

Q.6) In an RSA System the Public Key(e,n) of user A is defined as (7,119). Calculate totient(n) and Private key d. What is the cipher text when you encrypt message m=10 using the Public key.

Ans) $\phi(n) = \phi(119)$

Prime factorization of $119 = 7^1 \times 17^1$

$\therefore \phi(119) = \phi(7^1 \times 17^1) \Leftrightarrow \phi(7) \times \phi(17)$

$$\boxed{\phi(119) = 6 \times 16 = 96}$$

$e \cdot d \equiv 1 \pmod{\phi(n)}$

$\therefore e \cdot (d) + \phi(n)(-9) = 1$

$\therefore GCD(e, \phi(n)) = GCD(7, 96)$

$96 = 7(13) + 5 \Leftrightarrow 5 = 96 + 7(-13)$
$7 = 5(1) + 2 \Leftrightarrow 2 = 7 + 5(-1)$
$5 = 2(2) + 1 \Leftrightarrow 1 = 5 + 2(-2)$
$2 = 1(2)$
$\therefore GCD(7, 96) = 1$

$\therefore 2 = 7 + (-1)[96 + 7(-13)]$
$2 = 96(-1) + 7(14)$

$1 = [96 + 7(-13)] + (-2)[96(-1) + 7(14)]$
$\therefore 1 = 96(+3) + 7(-41)$

$d = -41 \pmod{96} \Leftrightarrow 55$

$m = 10$

Encrypted message $= m^e \pmod{n}$

$C = 10^7 \pmod{119}$

$C \equiv [(10^3)^2 \cdot 10] \pmod{119}$

$C = [(48)^2 \cdot 10] \pmod{119}$

$C = [48 \cdot 480] \pmod{119}$

$C = [48 \cdot 4] \pmod{119}$

$C = 73$

Q.7) If A and B wish to use RSA to communicate securly. A chooses public key (e,n) as $(7, 247)$ and B chooses Public key (e,n) as $(5, 221)$

i) Calculate A's Private key:

A of $(e,n) = (7, 247)$

$\therefore \phi(n) = \phi(247)$

$247 = 13^1 \times 19^1$

$\therefore \phi(247) = \phi(13 \times 19) = \phi(13) \times \phi(19)$

$\phi(247) = 12 \times 18 = 216$

$\therefore d = -41 \pmod{96} = 55$

$d = e^{-1} \bmod \phi(n)$

$e \cdot (e^{-1}) + \phi(n)(-q) = 1$

$\therefore G.C.D(e, \phi(n)) \Rightarrow G.C.D(7, 216)$

$216 = 7(30) + 6 \Rightarrow 6 = 216 + 7(-30)$

$7 = 6(1) + 1 \Rightarrow 1 = 7 + 6(-1)$

$6 = 1(6) + 0$

$\therefore 1 = 7 + (-1)[216 + 7(-30)]$

$7(31) + 216(-1) = 1$

$\therefore e^{-1} = 31$

$d = 31 \bmod 216 = 31$

$\therefore \boxed{d = 31}$

ii) Calculate Private key of B:

B of $(e, n) = (5, 221)$

$\phi(221) = \phi(13 \times 17) = \phi(13) \times \phi(17)$

$\therefore \phi(221) = 12 \times 16 = 192$

$d = e^{-1} \bmod \phi(n)$

$e \cdot (e^{-1}) + \phi(n)(-q) = 1$

G.C.D(5, 192) ⇒

$$192 = 5(38) + 2$$
$$5 = 2(2) + 1$$
$$2 = 1(2) + 0$$

$$2 = 192 + 5(-38)$$

$$1 = 5 + 2(-2)$$

$$\therefore 1 = 5 + (-2)[192 + 5(-38)]$$

$$192(-2) + 5(77) = 1$$

$$\therefore d = e^{-1} 77 \pmod{192} = \boxed{77}$$

$$\therefore \boxed{d = 77}$$

(iii) What will be the cipher text sent by to B, if A wishes to send M = 8 to B

Ans) If A wish to send some text to B
A will use publickey of B (e, n)
i.e. (5, 221)
M = 8.

$$\therefore \text{Ciphertext} \Rightarrow C = M^e \pmod{n}$$

$$\therefore C = 8^5 \pmod{221} = 3125 \pmod{221}$$

$$\therefore \boxed{C = 31}$$

Date: 15-04-2022

CSS Assignment 2

Q1) Write a short note on TCP/IP Vulnerabilities (layer wise).

Identifying Possible Network Interface Layer Attacks

At the Network Interface layer, the packet of information that is placed on the wire is known as a frame. The packet is comprised of three areas: the header, the payload, and the FCS. Because the Network Interface layer is used for communications on a local network, the attacks that occur at this level would be carried out on local networks. Some of the ways the network layer can be exploited to compromise the C-I-A triad include the following:

- MAC address spoofing: The header contains the MAC address of the source and destination computers and is required to successfully send a directed message from a source computer to a destination computer. Attackers can easily spoof the MAC address of another computer. Any security mechanism based on MAC addresses is vulnerable to this type of attack.
- Denial of service (DoS): A DoS attack overloads a single system so that it cannot provide the service it is configured to provide. An ARP protocol attack could be launched against a computer to overwhelm it, which would make it unavailable to support the C-I-A triad.
- ARP cache poisoning: The ARP cache stores MAC addresses of computers on the local network that have been contacted within a certain amount of time in memory. If incorrect, or spoofed, entries were added to the ARP cache, then the computer is not able to send information to the correct destination.

Identifying Possible Internet Layer Attacks

At the Internet layer, IP datagrams are formed. The packet is comprised of two areas: the header and the payload. Some of the ways the Internet layer can be exploited to compromise the C-I-A triad include the following:

- IP address spoofing: If the IP header fields and lengths are known, the IP address in the IP datagram can be easily discovered and spoofed. Any security mechanism based on the source IP address is vulnerable to this attack.
- Man-in-the-middle attacks: This attack occurs when a hacker places himself or herself between the source and destination computer in such a way that neither notices his or her existence. Meanwhile, the attacker can modify packets or simply view their contents.
- DoS: With a DoS attack at this level, simple IP-level protocols and utilities can be exploited to overload a computer, thus breaking the C-I-A triad.
- Incorrect reassembly of fragmented datagrams: For fragmented datagrams, the Offset field is used with packet reassembly. If the offset is changed, the datagram is reformed incorrectly. This could allow a datagram that would typically not pass through a firewall to gain access to your internal network, and could disrupt the C-I-A triad.
- Corrupting packets: Because IP datagrams can pass through several computers between the source and destination, the information in the IP header fields is read and sometimes modified, such as when the information reaches a router. If the packet is intercepted, the information in the header can be modified, corrupting the IP datagram. This could cause the datagram to never reach the destination computer, or it could change the protocols and payload information in the datagram.

Identifying Possible Transport Layer Attacks

At the Transport layer, either a UDP header is added to the message or a TCP header is added. The application that is requesting the service determines what protocol will be used. Some of the ways the Transport layer can be exploited to compromise the C-I-A triad include the following:

- Manipulation of the UDP or TCP ports: By knowing the UDP and TCP header fields and lengths, the ports that are used for communications between a source and destination computer can be identified, and that information can be corrupted or exploited.
- DoS: With a DoS attack at this level, simple IP-level protocols and utilities can be exploited to overload a computer, thus breaking the C-I-A triad. For instance, by knowing the steps involved in a three-way TCP handshake, a hacker or cracker might send the packets in the incorrect order and disrupt the availability of one of your servers. An example of this is a SYN flood, where a hacker sends a large number of SYN packets to a server and leaves the session half open. The server leaves these sessions half-open for a prescribed amount of time. If the hacker is successful in opening all available sessions, legitimate traffic will be unable to reach the server.
- Session hijacking: This kind of attack occurs after a source and destination computer have established a communications link. A third computer disables the ability of one the computers to communicate, and then imitates that computer. Because the connection has already been established, the third computer can disrupt your C-I-A triad.

Identifying Possible Application Layer Attacks

Application layer attacks can be some of the most difficult to protect against because they take advantage of vulnerabilities in applications and lack of end-user knowledge of computer security. Some of the ways the Application layer can be exploited to compromise the C-I-A triad include the following:

- E-mail application exploits: Attachments can be added to e-mail messages and delivered to a user's inbox. The user can open the e-mail message and run the application. The attachment might do immediate damage, or might lay dormant and be used later. Similarly, hackers often embed malicious code in Hypertext Markup Language (HTML) formatted messages. Exploits of this nature might take advantage of vulnerability in the client's e-mail application or a lack of user knowledge about e-mail security concerns.
- Web browser exploits: When a client computer uses a Web browser to connect to a Web server and download a Web page, the content of the Web page can be active. That is, the content is not just static information, but can be executable code. If the code is malicious, it can be used to disrupt the C-I-A triad.
- FTP client exploits: File Transfer Protocol (FTP) is used to transfer files from one computer to another. When a client has to provide a user name and password for authentication, that information can be sent across the Internet using plain text. The information can be captured at any point along the way. If the client uses the same user name and password as they use to attach to your corporate servers, that information could be obtained by a hacker or cracker and used to access your company's information.

Q2) Write a short note on firewall and its types.

A firewall is a network security perimeter device that inspects traffic entering and leaving the network. Depending on the security rules assigned specifically to it, the firewall either permits safe traffic or denies traffic it deems as dangerous.

A firewall's main objective is to establish a barrier (or "wall") that separates an internal network from incoming external traffic (such as the internet) for the purpose of blocking malicious network packets like malware and hacking.

How does firewall technology work?

Firewalls carefully analyze incoming traffic arriving on a computer's entry point, called a port, which determines how external devices communicate with each other and exchange information.

Firewalls operate using specific firewall rules. A firewall rule will typically include a source address, a protocol, a port number and a destination address.

Here's an analogy to explain the components of a firewall rule. Instead of protecting a network, think of a giant castle. The source address represents a person wishing to enter the castle. The port represents a room in the castle. The protocol represents a mode of transportation, and the destination address represents the castle.

Only trusted people (source addresses) may enter the castle (destination address) at all. Or perhaps only people that arrive on foot (protocol). Once inside, only people within the house are permitted to enter certain rooms (destination ports), depending on who they are. The king may be allowed in any room (any port), while guests and servants may only access a certain number of rooms (specific ports).

Types of firewalls

First, firewalls are classified by what they are and where they reside. For example, firewalls can either be hardware or software, cloud-based or on-premises.

A software firewall resides on an endpoint (like a computer or mobile device) and regulates traffic directly from that device. Hardware firewalls are physical pieces of equipment that reside between your gateway and network. Cloud-based firewalls, also known as Firewall-as-a-service (FaaS), act like any other internet-based SaaS solutions, performing their work in the cloud.

The most common firewall types based on methods of operation are:

- Packet-filtering firewalls
- Proxy firewalls
- NAT firewalls
- Web application firewalls
- Next-gen firewalls (NGFW)

Packet-filtering firewalls

Packet-filtering firewalls, the most basic firewall type, examine packets and prevent them from moving on if the specific security rule is not met. This firewall's function is to perform a simple check of all data packets arriving from the network router and inspecting the specifics like source and destination IP address, port number, protocol, and other surface-level data.

Packet filtering firewalls don’t open data packets to inspect their contents. Any data packet that fails the simple inspection is dropped.

These firewalls are not resource-intensive and have a low impact on system performance. Their main drawback is that they provide only basic protection and are therefore more vulnerable to being bypassed.

Packet-filtering firewalls can either be stateful and stateless. Stateless firewalls only analyze each packet individually, whereas stateful firewalls — the more secure option — take previously inspected packets into consideration.

Proxy firewalls

Proxy firewalls, also known as application-level firewalls, filter network traffic at the application layer of the OSI network model. As an intermediary between two systems, proxy firewalls monitor traffic at the application layer (protocols at this layer include HTTP and FTP). To detect malicious traffic, both stateful and deep packet inspection are leveraged.

Proxy firewalls typically operate in the cloud or through another proxy device. Instead of allowing traffic to connect directly, a connection to the traffic's source is established and the data packet is inspected.

Speed can be a key weakness of proxy firewalls, as the transfer process creates extra steps that may slow things down.

NAT firewalls

Network address translation (NAT) firewalls work by assigning a public address to a group of devices inside a private network. With NAT, individual IP addresses are hidden. Therefore, attackers scanning for IP addresses on a network are prevented from discovering specific details.

NAT firewalls and proxy firewalls both act as a go-between connecting groups of devices with outside traffic.

Web application firewalls

Web application firewalls (WAF) are responsible for filtering, monitoring, and blocking data packets as they travel in and out of websites or web applications. A WAF can either reside on the network, at the host or in the cloud and is typically placed in front of one or many websites or applications. WAFs are available as server plugins, cloud services, or network appliances.

A WAF is most similar to the proxy firewall, but has a more specific focus on defending against application layer web-based attackers.

NGFW firewalls

As the threat landscape intensifies, the Next-generation firewall (NGFW) is the most popular firewall type available today.

Thanks to the major improvements in storage space, memory, and processing speeds, NGFWs build upon traditional firewalls' features and add other critical security functions like intrusion prevention, VPN, anti-malware, and even encrypted traffic inspection. NGFW’s ability to handle deep packet inspection means that the firewall can unpack the packet's data to prevent any packets with malicious data from moving forward.

Q3) A and B decide to use Diffie Hellman algorithm to share a key. They chose $p = 23$ and $g = 5$ as the public parameters. Their secret keys are 6 and 15 respectively. Compute the secret key that they share.

Solution:

$prime\ number = \ p = 23, \qquad Genrater = \ g = 5$

$Secret\ key\ of\ A = \ X_A = 6, \qquad Secret\ key\ of\ B = X_B = 15$

$Public\ key\ of\ A = \ R_A = g^{X_A}(modp) \ = 5^6(mod23) = \ (5^2)^3(mod23)$

$R_A = \ 2^3(mod23) = 8$ Send to B

$Public\ key\ of\ B = \ R_B = g^{X_B}(modp) \ = 5^{15}(mod23) = \ (5^2)^7 \times 5(mod23)$

$R_B = 2^7 \times 5(mod23) = 13 \times 5(mod23) = 72\ (mod23) = 19$ Send to A

$Secret\ Key\ calculated\ by\ A = key = \ (R_B)^{X_A} = 19^6(mod23) = 2$

$Secret\ Key\ calculated\ by\ B = key = \ (R_A)^{X_B} = 8^{15}(mod23) = 2$