PRACTICAL 1

Aim: Implement Caesar cipher encryption-decryption.

Input:

```
def caesar_cipher(text, shift, mode='encrypt'):
    result = ''
    if mode == 'decrypt':
        shift = -shift
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base + shift) % 26 + base)
        else:
            result += char
    return result

# Example usage

message = "CKPITHAWALA"
shift= 3
encrypted = caesar_cipher(message, shift, 'encrypt')
decrypted = caesar_cipher(encrypted, shift, 'decrypt')

print("Encrypted:", encrypted)
print("Decrypted:", decrypted)
```

Output:

```
Encrypted: FNSLWKDZDOD
Decrypted: CKPITHAWALA
```

PRACTICAL 2

Aim: Implement Monoalphabetic cipher encryption-decryption.

Input:

```
import string

def monoalphabetic_cipher(text, key, mode='encrypt'):
    alphabet = string.ascii_lowercase
    key_map = dict(zip(alphabet, key.lower())) if mode == 'encrypt' else
dict(zip(key.lower(), alphabet))
     result = ''
    for char in text:
        if char.isalpha():
            is_upper = char.isupper()
            mapped = key_map[char.lower()]
            result += mapped.upper() if is_upper else mapped
        else:
            result += char
    return result

key = "phqgiumeaylnofdxjkrcvstzwb"

message = "CKPITHAWALA"
encrypted = monoalphabetic_cipher(message, key, 'encrypt')
decrypted = monoalphabetic_cipher(encrypted, key, 'decrypt')

print("Encrypted:", encrypted)
print("Decrypted:", decrypted)
```

Output:

```
Encrypted: QLXACEPTPNP
Decrypted: CKPITHAWALA
```

PRACTICAL 3

Aim: Implement Playfair cipher encryption-decryption.

Input:

```
import string

def generate_key_matrix(key):
    key = key.lower().replace("j", "i")
    alphabet = string.ascii_lowercase.replace("j", "")
    matrix_key = "".join(dict.fromkeys(key + alphabet))  # remove duplicates
    return [list(matrix_key[i:i+5]) for i in range(0, 25, 5)]

def find_pos(matrix, char):
    for i, row in enumerate(matrix):
        if char in row:
            return i, row.index(char)
    return None

def playfair_cipher(text, key, mode='encrypt'):
    matrix = generate_key_matrix(key)
    text = text.lower().replace("j", "i").replace(" ", "")
    pairs, i = [], 0

    while i < len(text):
        a = text[i]
        b = text[i+1] if i+1 < len(text) else 'x'
        if a == b:
            pairs.append(a + 'x')
            i += 1
        else:
            pairs.append(a + b)
            i += 2

    result = ''
    for a, b in pairs:
        r1, c1 = find_pos(matrix, a)
        r2, c2 = find_pos(matrix, b)

        if r1 == r2:  # same row
            result += matrix[r1][(c1 + (1 if mode == 'encrypt' else -1)) % 5]
            result += matrix[r2][(c2 + (1 if mode == 'encrypt' else -1)) % 5]
        elif c1 == c2:  # same column
            result += matrix[(r1 + (1 if mode == 'encrypt' else -1)) % 5][c1]
```

```
            result += matrix[(r2 + (1 if mode == 'encrypt' else -1)) % 5][c2]
        else:  # rectangle swap
            result += matrix[r1][c2] + matrix[r2][c1]

    return result.upper()

# Example
key = "devansh kapadia"
message = "Hide"
encrypted = playfair_cipher(message, key, 'encrypt')
decrypted = playfair_cipher(encrypted, key, 'decrypt')

print("Encrypted:", encrypted)
print("Decrypted:", decrypted)
```

Output:

```
Encrypted: SBEV
Decrypted: HIDE
```

PRACTICAL 4

Aim: Implement Polyalphabetic cipher encryption-decryption.

Input:

```
def vigenere_cipher(text, key, mode='encrypt'):
    result = ''
    key = key.lower()
    klen = len(key)

    for i, char in enumerate(text):
        if char.isalpha():
            shift = ord(key[i % klen]) - ord('a')
            if mode == 'decrypt':
                shift = -shift
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base + shift) % 26 + base)
        else:
            result += char
    return result

# Example usage
message = "CKPITHAWALA" key
= "key"

encrypted = vigenere_cipher(message, key, 'encrypt')
decrypted = vigenere_cipher(encrypted, key, 'decrypt')
print("Encrypted:", encrypted)
print("Decrypted:", decrypted)
```

Output:

```
Encrypted: MONSXFKAYVE
Decrypted: CKPITHAWALA
```

PRACTICAL 5

Aim: Implement Hill cipher encryption-decryption.

Input:

```
import numpy as np

def hill_cipher(text, key, mode='encrypt'):
    text = text.replace(" ", "").lower()
    n = key.shape[0]

    # pad text if needed
    if len(text) % n != 0:
        text += 'x' * (n - len(text) % n)

    # convert text to numbers
    nums = [ord(c) - ord('a') for c in text]
    nums = np.array(nums).reshape(-1, n)

    if mode == 'decrypt':
        det = int(round(np.linalg.det(key)))
        det_inv = pow(det % 26, -1, 26)
        key_inv = (det_inv * np.round(det * np.linalg.inv(key)).astype(int)) % 26
        key = key_inv
    result = (nums.dot(key) % 26).flatten()
    return ''.join(chr(num + ord('a')) for num in result)

# Example usage
key = np.array([[3, 3],

                [2, 5]])

message = " CKPCET"
encrypted = hill_cipher(message, key, 'encrypt')
decrypted = hill_cipher(encrypted, key, 'decrypt')

print("Encrypted:", encrypted)
print("Decrypted:", decrypted)
```

Output:

```
Encrypted: aexdyd
Decrypted: ckpcet
```

PRACTICAL 6

Aim: To implement Simple DES or AES.

Input:

```python
import os
from cryptography.hazmat.primitives.ciphers.aead import AESGCM


def aes_encrypt(plaintext: str, key: bytes | None = None):
    key = key or AESGCM.generate_key(bit_length=256)
    nonce = os.urandom(12)
    ct = AESGCM(key).encrypt(nonce, plaintext.encode(), None)
    return nonce, ct, key


def aes_decrypt(nonce: bytes, ciphertext: bytes, key: bytes) -> str:
    return AESGCM(key).decrypt(nonce, ciphertext, None).decode()


# Example
msg = "hello everyone!"
nonce, ct, key = aes_encrypt(msg)
pt = aes_decrypt(nonce, ct, key)
print("Ciphertext (hex):", ct.hex())
print("Decrypted:", pt)
```

Output:

```
Ciphertext (hex): 1fd64f55e77f6d1890c7ae98b6378ff6ab225acecc4db7a645eff74b0c592a
Decrypted: hello everyone!
```

PRACTICAL 7

Aim: Implement Diffie-Hellman Key exchange Method.

Input:

```
import random
from math import gcd

# Generate keys
def generate_keys(p, q):
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 65537
    while gcd(e, phi) != 1:
        e = random.randrange(2, phi)
    d = pow(e, -1, phi)
    return (e, n), (d, n)

# Encrypt
def encrypt(msg, pub):
    e, n = pub
    return [pow(ord(c), e, n) for c in msg]

# Decrypt
def decrypt(cipher, priv):
    d, n = priv
    return ''.join(chr(pow(c, d, n)) for c in cipher)

# Example
p, q = 61, 53
public_key, private_key = generate_keys(p, q)
message = "CKPITHAWALA"
cipher = encrypt(message, public_key)
plain = decrypt(cipher, private_key)

print("Encrypted:", cipher)
print("Decrypted:", plain)
```

Output:

```
Encrypted: [641, 597, 2933, 1486, 2159, 3000, 2790, 604, 2790, 2726, 2790]
Decrypted: CKPITHAWALA
```

PRACTICAL 8

Aim: Implement RSA encryption-decryption algorithm.

Input:

```
import random

from math import gcd
# Generate keys
def generate_keys(p, q):
    n = p * q
    phi = (p-1)*(q-1)
    e = 65537
    while gcd(e, phi) != 1:
        e = random.randrange(2, phi)
    d = pow(e, -1, phi)
    return (e, n), (d, n)

# Encrypt
def encrypt(msg, pub):
    e, n = pub
    return [pow(ord(c), e, n) for c in msg]

# Decrypt
def decrypt(cipher, priv):
    d, n = priv
    return ''.join(chr(pow(c, d, n)) for c in cipher)

# Example
p, q = 61, 53
public_key, private_key = generate_keys(p, q)

message = "CKPITHAWALA"
cipher = encrypt(message, public_key)
plain = decrypt(cipher, private_key)

print("Encrypted:", cipher)
print("Decrypted:", plain)
```

Output:

```
Encrypted: [641, 597, 2933, 1486, 2159, 3000, 2790, 604, 2790, 2726, 2790]
Decrypted: CKPITHAWALA
```

PRACTICAL 9

Aim: Write a program to generate SHA-1 hash.

Input:

```
import hashlib
def sha1_hash(text: str) -> str:
    return hashlib.sha1(text.encode()).hexdigest()

# Example
msg = "Devansh Kapadia"
print("SHA-1 Hash:", sha1_hash(msg))
```

Output:

```
SHA-1 Hash: 17f3e2c23bc78e446c54cd287a53c0e36cb9acee
```

PRACTICAL 10

Aim: Implement a digital signature algorithm.

Input:

```python
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives.asymmetric.utils import Prehashed

# Generate private & public key
private_key = ec.generate_private_key(ec.SECP256R1())
public_key = private_key.public_key()

# Message
message = b"Sent by Devansh Kapadia"

# Sign
signature = private_key.sign(message, ec.ECDSA(hashes.SHA256()))

# Verify
try:
    public_key.verify(signature, message, ec.ECDSA(hashes.SHA256()))
    print("Signature verified ✅")
except:
    print("Signature verification failed ❌")
```

Output:

```
PS C:\Users\Devansh Kapadia\OneDrive\Desktop\information Security> python test.py
Signature verified ✅
```

PRACTICAL 11

Aim: Perform various encryption-decryption techniques with crypt-tool.

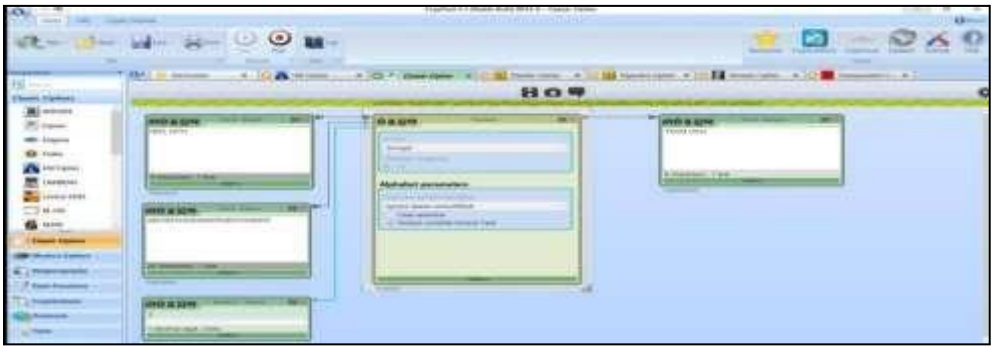
What is Crypt-tool?

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

Caesar Cipher:

The Caesar Cipher is a very basic substitution cipher – each letter is replaced by another. It also includes an offset that determines how many letters away from the original the substituted letter would be.

Encryption:



Decryption:

Hill Cipher:

- Hill cipher is a polygraphic substitution cipher based on linear algebra. Each letter is represented by a number modulo 26.
- To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26.
- To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.
- The matrix used for encryption is the cipher key.

Encryption:

Decryption:

Playfair Cipher:

- The Playfair cipher was the first practical digraph substitution cipher.
- The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher.
- In playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet.

Encryption:

Decryption:

PRACTICAL 12

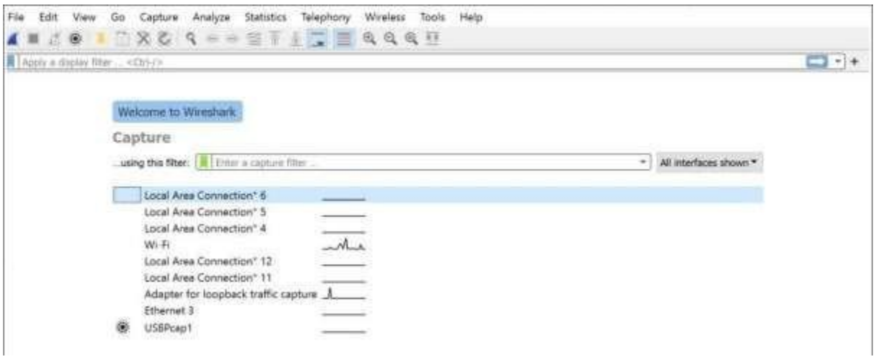
Aim: Study and use the Wireshark for the various network protocols.

History of Wireshark:

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

What is Wireshark?

- Wireshark is an open-source packet analyzer, which is used for education, analysis, software development, communication protocol development, and network troubleshooting.
- It is used to track the packets so that each one is filtered to meet our specific needs. It is commonly called a sniffer, network protocol analyzer, and network analyzer.
- It is also used by network security engineers to examine security problems.
- Wireshark is a free to use application which is used to apprehend the data back and forth.
- It is often called a free packet sniffer computer application. It puts the network card into an unselective mode, i.e., to accept all the packets which it receives.



Uses of Wireshark:

Wireshark can be used in the following ways:

1. It is used by network security engineers to examine security problems.
2. It allows the users to watch all the traffic being passed over the network.
3. It is used by network engineers to troubleshoot network issues.
4. It also helps to troubleshoot latency issues and malicious activities on your network.
5. It can also analyze dropped packets.
6. It helps us to know how all the devices like laptop, mobile phones, desktop, switch, routers, etc., communicate in a local network or the rest of the world.

Functionality of Wireshark:

- Wireshark is similar to tcpdump in networking.
- Tcpdump is a common packet analyzer which allows the user to display other packets and TCP/IP packets, being transmitted and received over a network attached to the computer.
- It has a graphic end and some sorting and filtering functions.
- Wireshark users can see all the traffic passing through the network.
- Wireshark can also monitor the unicast traffic which is not sent to the network's MAC address interface. But the switch does not pass all the traffic to the port. Hence, the promiscuous mode is not sufficient to see all the traffic. The various network taps or port mirroring is used to extend capture at any point.

Features of Wireshark:

- It is multi-platform software, i.e., it can run on Linux, Windows, OS X, FreeBSD, NetBSD, etc.
- It is a standard three-pane packet browser.
- It performs deep inspection of hundreds of protocols.
- It often involves live analysis, i.e., from the different types of the network like the Ethernet, loopback, etc., we can read live data.
- It has sort and filter options which makes it easy for the user to view the data.
- It is also useful in VoIP analysis.
- It can also capture raw USB traffic.
- Various settings, like timers and filters, can be used to filter the output. o It can only capture packet on the PCAP (an application programming interface used to capture the network) supported networks.
- Wireshark supports a variety of well-documented capture file formats such as the PcapNg and Libpcap. These formats are used for storing the captured data.
- It is the no.1 piece of software for its purpose. It has countless applications ranging from the tracing down, unauthorized traffic, firewall settings, etc.

How to view and analyze packet contents:

The captured data interface contains three main sections:

- The packet list pane (the top section)
- The packet details pane (the middle section)
- The packet bytes pane (the bottom section)

Wireshark color rules:

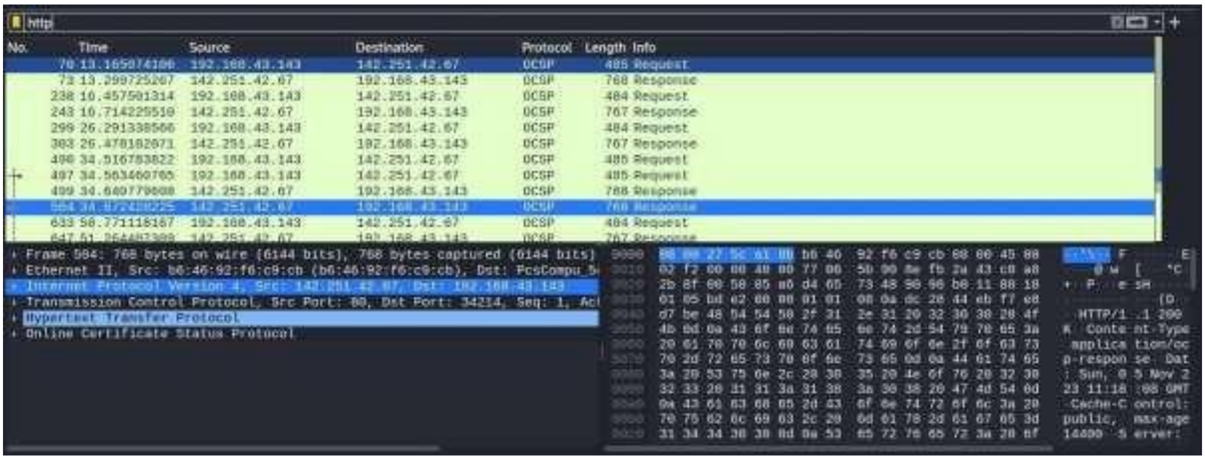
- While Wireshark's capture and display filters limit which packets are recorded or shown on the screen, its colorization function takes things a step further: It can distinguish between different packet types based on their individual hue. This quickly locates certain packets within a saved set by their row colour in the packet list pane.
- Wireshark comes with about 20 default coloring rules, each can be edited, disabled, or deleted. Select View > Coloring Rules for an
- overview of what each color means.

Statistics in Wireshark:

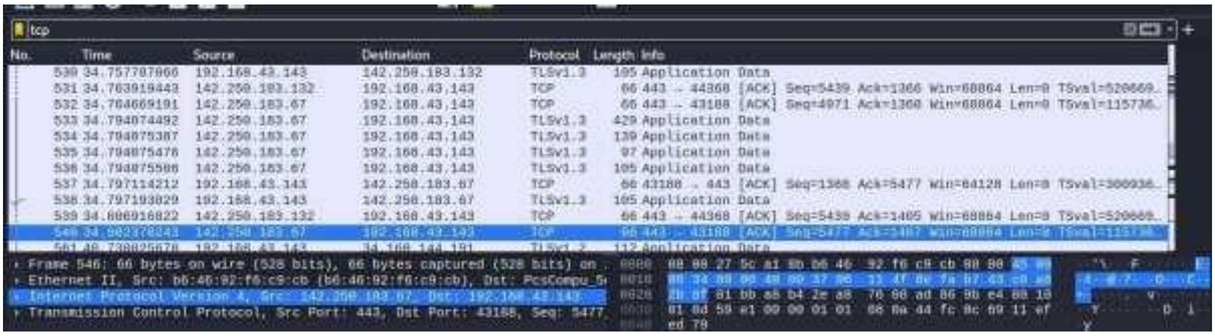
- Other useful metrics are available through the Statistics drop-down menu. These include size and timing information about the capture file, along with dozens of charts and graphs ranging in topic from packet conversation breakdowns to load distribution of HTTP requests.
- Display filters can be applied to many of these statistics via their interfaces, and the results can be exported to common file formats, including CSV, XML, and TEXT.

Network Protocols in Wireshark are:

1. HTTP: HTTP means Hypertext Transfer Protocol. HTTP packet has 4 layers. OCSP is a Hypertext Transfer Protocol (HTTP) used for obtaining the revocation status of an X. 509 digital certificate.



2. TCP: TCP means Transmission Control Protocol. TCP packet has 3 layer.



3. ICMP: ICMP means internet Control Message Protocol. ICMP packets has 2 layers.