**A. D. Patel Institute of Technology**

**Department of Information Technology**

Lab Manual

Subject: Information and Network Security

(202046708)

CLASS: IT- A

A.Y. 2025 (Semester-VI)

**Name: Astha Dhanani Enrolment No:12202080601006 Semester:6th**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr**  **No.** | **Experiment** | **Date**  **Of Exp.** | **Date of**  **Submi.** | **Page**  **No.** | **Sign** |
| **1** | To implement Caesar cipher encryption-decryption. |  |  |  |  |
| **2** | To implement Monoalphabetic cipher encryption-decryption |  |  |  |  |
| **3** | To implement Playfair cipher encryption-decryption. |  |  |  |  |
| **4** | To implement Polyalphabetic cipher encryption-decryption. |  |  |  |  |
| **5** | To implement Hill cipher encryption-decryption. |  |  |  |  |
| **6** | To implement Rail Fence and Columnar transposition cipher encryption-decryption |  |  |  |  |
| **7** | To implement Simplified Data Encryption Standard. |  |  |  |  |
| **8** | To implement Diffi-Hellman Key Exchange method. |  |  |  |  |
| **9** | To implement RSA encryption-decryption algorithm |  |  |  |  |
| **10** | Demonstrate and perform various encryption-decryption techniques with cryptool |  |  |  |  |
| **11** | Study and use open-source packet analyzer-Wireshark to understand security mechanism of various network protocols. |  |  |  |  |
| **12** | Detail Case study: Real world implementation of Network Security Algorithm/Concept. |  |  |  |  |

**PRACTICLE- 1**

**AIM:** To implement Caesar cipher encryption-decryption.

**Code**-

def cc(text, s):

result = ""

for i in range(len(text)):

char = text[i]

if char.isupper():

result += chr((ord(char) + s-65) % 26 + 65)

else:

result += chr((ord(char) + s-97) % 26 + 97)

return result

text = "CVMUADITITAD"

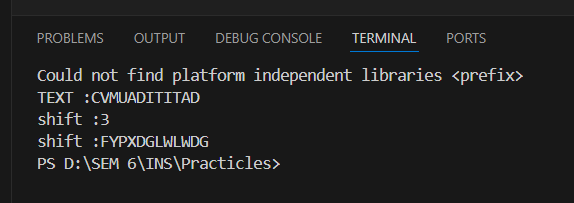
s = 3

print("TEXT :" + text)

print("shift :" + str(s))

print("shift :" + cc(text, s))

**Output-**



**Both encryption and decryption-**

def caesar\_cipher(text, shift, mode='encrypt'):

    if mode == 'decrypt':

        shift = -shift

    result = ""

    for char in text:

        if char.isalpha():

            start = ord('A') if char.isupper() else ord('a')

            result += chr((ord(char) - start + shift) % 26 + start)

        else:

            result += char

    return result

plaintext = "Hello, World!"

shift\_value = 3

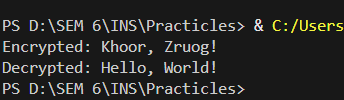
encrypted\_text = caesar\_cipher(plaintext, shift\_value, mode='encrypt')

print(f"Encrypted: {encrypted\_text}")

decrypted\_text = caesar\_cipher(encrypted\_text, shift\_value, mode='decrypt')

print(f"Decrypted: {decrypted\_text}")

**Output-**



**PRACTICLE- 2**

**AIM:** To implement Monoalphabetic cipher encryption-decryption

**Code-**

import string

def generate\_key():

    from random import shuffle

    letters = list(string.ascii\_uppercase)

    shuffle(letters)

    return ''.join(letters)

def encrypt(plaintext, key):

    alphabet = string.ascii\_uppercase

    encryption\_table = str.maketrans(alphabet, key)

    return plaintext.upper().translate(encryption\_table)

def decrypt(ciphertext, key):

    alphabet = string.ascii\_uppercase

    decryption\_table = str.maketrans(key, alphabet)

    return ciphertext.upper().translate(decryption\_table)

if \_\_name\_\_ == "\_\_main\_\_":

    key = generate\_key()

    print(f"Generated Key: {key}")

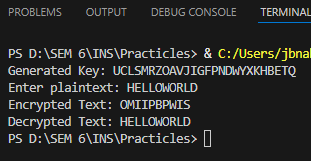
    plaintext = input("Enter plaintext: ")

    encrypted\_text = encrypt(plaintext, key)

    print(f"Encrypted Text: {encrypted\_text}")

    decrypted\_text = decrypt(encrypted\_text, key)

    print(f"Decrypted Text: {decrypted\_text}")

**Output-**

**PRACTICLE- 3**

**AIM:** To implement Playfair cipher encryption-decryption.

**Code-**

def prepare\_text(text, filler='X'):

    text = text.upper().replace(" ", "").replace("J", "I")

    result = ""

    i = 0

    while i < len(text):

        result += text[i]

        if i + 1 < len(text) and text[i] == text[i + 1]:

            result += filler

        elif i + 1 < len(text):

            result += text[i + 1]

            i += 1

        i += 1

    return result + filler if len(result) % 2 else result

def create\_matrix(keyword):

    matrix = "".join(dict.fromkeys(keyword.upper().replace("J", "I") + "ABCDEFGHIKLMNOPQRSTUVWXYZ"))

    return [matrix[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)

def process\_pair(pair, matrix, encrypt=True):

    r1, c1 = find\_pos(matrix, pair[0])

    r2, c2 = find\_pos(matrix, pair[1])

    if r1 == r2: return matrix[r1][(c1 + (1 if encrypt else -1)) % 5] + matrix[r2][(c2 + (1 if encrypt else -1)) % 5]

    if c1 == c2: return matrix[(r1 + (1 if encrypt else -1)) % 5][c1] + matrix[(r2 + (1 if encrypt else -1)) % 5][c2]

    return matrix[r1][c2] + matrix[r2][c1]

def playfair(text, keyword, encrypt=True):

    matrix = create\_matrix(keyword)

    text = prepare\_text(text)

    return "".join(process\_pair(text[i:i+2], matrix, encrypt) for i in range(0, len(text), 2))

keyword = input("Enter keyword: ")

text = input("Enter text: ")

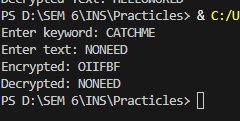
encrypted = playfair(text, keyword, True)

print(f"Encrypted: {encrypted}")

decrypted = playfair(encrypted, keyword, False)

print(f"Decrypted: {decrypted}")

**Output-**



**PRACTICLE- 4**

**AIM:** To implement Polyalphabetic cipher encryption-decryption. ( Vigenère Cipher)

**Code-**

def vigenere\_encrypt(text, key):

"""Encrypts the text using the Vigenère cipher."""

text = text.upper()

key = key.upper()

encrypted = []

key\_index = 0

for char in text:

if char.isalpha():

shift = ord(key[key\_index]) - ord('A')

encrypted\_char = chr((ord(char) - ord('A') + shift) % 26 + ord('A'))

encrypted.append(encrypted\_char)

key\_index = (key\_index + 1) % len(key)

else:

encrypted.append(char)

return ''.join(encrypted)

def vigenere\_decrypt(ciphertext, key):

"""Decrypts the ciphertext using the Vigenère cipher."""

ciphertext = ciphertext.upper()

key = key.upper()

decrypted = []

key\_index = 0

for char in ciphertext:

if char.isalpha():

shift = ord(key[key\_index]) - ord('A')

decrypted\_char = chr((ord(char) - ord('A') - shift) % 26 + ord('A'))

decrypted.append(decrypted\_char)

key\_index = (key\_index + 1) % len(key)

else:

decrypted.append(char)

return ''.join(decrypted)

if \_\_name\_\_ == "\_\_main\_\_":

key = input("Enter the key: ")

plaintext = input("Enter plaintext: ")

encrypted = vigenere\_encrypt(plaintext, key)

print(f"Encrypted Text: {encrypted}")

decrypted = vigenere\_decrypt(encrypted, key)

print(f"Decrypted Text: {decrypted}")

**Output**-

A screenshot of a computer screen

Description automatically generated

**PRACTICLE- 5**

**AIM:** To implement Hill cipher encryption-decryption.

**Code-**

import numpy as np

def text\_to\_numbers(text):

    return [ord(char) - ord('A') for char in text.upper().replace(" ", "")]

def numbers\_to\_text(numbers):

    return ''.join(chr((num % 26) + ord('A')) for num in numbers)

def prepare\_text(text, block\_size):

    text = text.upper().replace(" ", "")

    while len(text) % block\_size != 0:

        text += "X"

    return text

def hill\_cipher(text, key\_matrix, mode="encrypt"):

    block\_size = key\_matrix.shape[0]

    text = prepare\_text(text, block\_size)

    numbers = text\_to\_numbers(text)

    matrix = np.array(numbers).reshape(-1, block\_size).T

    if mode == "decrypt":

        det = int(round(np.linalg.det(key\_matrix))) % 26

        det\_inv = pow(det, -1, 26)

        adj = np.round(det \* np.linalg.inv(key\_matrix)).astype(int) % 26

        key\_matrix = (det\_inv \* adj) % 26

    result = np.dot(key\_matrix, matrix) % 26

    return numbers\_to\_text(result.T.flatten())

if \_\_name\_\_ == "\_\_main\_\_":

    key = np.array([[6, 24, 1], [13, 16, 10], [20, 17, 15]])  # Example key

    plaintext = input("Enter plaintext: ")

    encrypted = hill\_cipher(plaintext, key, mode="encrypt")

    print(f"Encrypted: {encrypted}")

    decrypted = hill\_cipher(encrypted, key, mode="decrypt")

    print(f"Decrypted: {decrypted}")

**Output-**

**A screen shot of a computer code

Description automatically generated**

**PRACTICLE- 6**

**AIM:** To implement Rail Fence and Columnar transposition cipher encryption-decryption

**Code-**

**Rail Fence-**

# Rail Fence Cipher Encryption

def rail\_fence\_encrypt(text, key):

    rail = [['\n' for \_ in range(len(text))] for \_ in range(key)]

    dir\_down = False

    row, col = 0, 0

    for char in text:

        if row == 0 or row == key - 1:

            dir\_down = not dir\_down

        rail[row][col] = char

        col += 1

        row += 1 if dir\_down else -1

    result = ''.join(''.join(rail[i]) for i in range(key))

    return result.replace('\n', '')

# Rail Fence Cipher Decryption

def rail\_fence\_decrypt(cipher, key):

    rail = [['\n' for \_ in range(len(cipher))] for \_ in range(key)]

    dir\_down = None

    row, col = 0, 0

    for i in range(len(cipher)):

        if row == 0:

            dir\_down = True

        if row == key - 1:

            dir\_down = False

        rail[row][col] = '\*'

        col += 1

        row += 1 if dir\_down else -1

    index = 0

    for i in range(key):

        for j in range(len(cipher)):

            if rail[i][j] == '\*' and index < len(cipher):

                rail[i][j] = cipher[index]

                index += 1

    result = []

    row, col = 0, 0

    for i in range(len(cipher)):

        if row == 0:

            dir\_down = True

        if row == key - 1:

            dir\_down = False

        result.append(rail[row][col])

        col += 1

        row += 1 if dir\_down else -1

    return ''.join(result)

text = "HELLO WORLD"

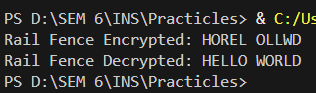
key = 3

cipher = rail\_fence\_encrypt(text, key)

print("Rail Fence Encrypted:", cipher)

print("Rail Fence Decrypted:", rail\_fence\_decrypt(cipher, key))

**Output-**



**Columnar Transposition Cipher**

import math

# Columnar Transposition Encryption

def columnar\_encrypt(text, key):

    col = len(key)

    row = math.ceil(len(text) / col)

    matrix = [['' for \_ in range(col)] for \_ in range(row)]

    index = 0

    for r in range(row):

        for c in range(col):

            if index < len(text):

                matrix[r][c] = text[index]

                index += 1

    sorted\_key = sorted(list(key))

    cipher = ''

    for char in sorted\_key:

        col\_index = key.index(char)

        for r in range(row):

            cipher += matrix[r][col\_index]

    return cipher

# Columnar Transposition Decryption

def columnar\_decrypt(cipher, key):

    col = len(key)

    row = math.ceil(len(cipher) / col)

    matrix = [['' for \_ in range(col)] for \_ in range(row)]

    sorted\_key = sorted(list(key))

    index = 0

    for char in sorted\_key:

        col\_index = key.index(char)

        for r in range(row):

            if index < len(cipher):

                matrix[r][col\_index] = cipher[index]

                index += 1

    text = ''

    for r in range(row):

        for c in range(col):

            text += matrix[r][c]

    return text

# Example Usage

text = "HELLO WORLD"

key = "3142"

cipher = columnar\_encrypt(text, key)

print("Columnar Encrypted:", cipher)

print("Columnar Decrypted:", columnar\_decrypt(cipher, key))

**Output-**

**A black screen with white text

Description automatically generated**

**PRACTICLE- 7**

**AIM:** To implement Simplified Data Encryption Standard.

**Code-**

# Permutation and shifting helper functions

def permute(bits, table):

    return ''.join(bits[i] for i in table)

def shift\_left(bits, shifts):

    return bits[shifts:] + bits[:shifts]

# S-Boxes (fixed substitution tables for S-DES)

S0 = [[1, 0, 3, 2], [3, 2, 1, 0], [0, 2, 1, 3], [3, 1, 3, 2]]

S1 = [[0, 1, 2, 3], [2, 0, 1, 3], [3, 0, 1, 0], [2, 1, 0, 3]]

# Generate the two keys (K1 and K2) from the original key

def generate\_keys(key):

    p10\_table = [2, 4, 1, 6, 3, 9, 0, 8, 7, 5]

    p8\_table = [5, 2, 6, 3, 7, 4, 9, 8]

    key = permute(key, p10\_table)

    left, right = key[:5], key[5:]

    left, right = shift\_left(left, 1), shift\_left(right, 1)

    k1 = permute(left + right, p8\_table)

    left, right = shift\_left(left, 2), shift\_left(right, 2)

    k2 = permute(left + right, p8\_table)

    return k1, k2

# F function (expansion, substitution, and permutation)

def f\_function(bits, key):

    ep\_table = [3, 0, 1, 2, 1, 2, 3, 0]

    p4\_table = [1, 3, 2, 0]

    bits = permute(bits, ep\_table)

    xor\_result = ''.join(str(int(bits[i]) ^ int(key[i])) for i in range(len(bits)))

    left, right = xor\_result[:4], xor\_result[4:]

    row\_l, col\_l = int(left[0] + left[3], 2), int(left[1] + left[2], 2)

    row\_r, col\_r = int(right[0] + right[3], 2), int(right[1] + right[2], 2)

    sbox\_output = f"{S0[row\_l][col\_l]:02b}{S1[row\_r][col\_r]:02b}"

    return permute(sbox\_output, p4\_table)

# Encryption/Decryption process

def sdes\_encrypt\_decrypt(plain\_text, keys, encrypt=True):

    ip\_table = [1, 5, 2, 0, 3, 7, 4, 6]

    ip\_inv\_table = [3, 0, 2, 4, 6, 1, 7, 5]

    keys = keys if encrypt else keys[::-1]

    bits = permute(plain\_text, ip\_table)

    left, right = bits[:4], bits[4:]

    left = ''.join(str(int(left[i]) ^ int(f\_function(right, keys[0])[i])) for i in range(4))

    left, right = right, left

    left = ''.join(str(int(left[i]) ^ int(f\_function(right, keys[1])[i])) for i in range(4))

    return permute(left + right, ip\_inv\_table)

# Main function to demonstrate S-DES

def sdes\_demo():

    key = "1010000010"  # 10-bit key

    plaintext = "10111101"  # 8-bit plaintext

    k1, k2 = generate\_keys(key)

    print("Key 1:", k1)

    print("Key 2:", k2)

    cipher = sdes\_encrypt\_decrypt(plaintext, (k1, k2), encrypt=True)

    print("Encrypted:", cipher)

    decrypted = sdes\_encrypt\_decrypt(cipher, (k1, k2), encrypt=False)

    print("Decrypted:", decrypted)

sdes\_demo()

**Output-**

**A screen shot of a computer

Description automatically generated**

**PRACTICLE- 8**

**AIM:** To implement Diffi-Hellman Key Exchange method.

**Code-**

# Diffie-Hellman Key Exchange

def diffie\_hellman(p, g, private\_a, private\_b):

    # Step 1: Public keys

    public\_a = (g \*\* private\_a) % p

    public\_b = (g \*\* private\_b) % p

    # Step 2: Shared secret keys

    shared\_key\_a = (public\_b \*\* private\_a) % p

    shared\_key\_b = (public\_a \*\* private\_b) % p

    return public\_a, public\_b, shared\_key\_a, shared\_key\_b

p = 23  # A prime number (shared publicly)

g = 5   # A primitive root modulo p (shared publicly)

private\_a = 6  # Alice's private key (secret)

private\_b = 15 # Bob's private key (secret)

public\_a, public\_b, shared\_key\_a, shared\_key\_b = diffie\_hellman(p, g, private\_a, private\_b)

print("Public Key (x):", public\_a)

print("Public Key (y):", public\_b)

print("Shared Key (x):", shared\_key\_a)

print("Shared Key (y):", shared\_key\_b)

assert shared\_key\_a == shared\_key\_b

print("Shared Secret Key:", shared\_key\_a)

**Output-**

**A screenshot of a computer program

Description automatically generated**

**PRACTICLE- 9**

**AIM:** To implement RSA encryption-decryption algorithm

**Code-**

import math

# Helper function to calculate GCD

def gcd(a, b):

    while b:

        a, b = b, a % b

    return a

# Helper function for modular exponentiation

def mod\_exp(base, exp, mod):

    result = 1

    while exp > 0:

        if exp % 2 == 1:  # If exp is odd

            result = (result \* base) % mod

        base = (base \* base) % mod

        exp //= 2

    return result

# RSA Key Generation

def generate\_keys(p, q):

    n = p \* q

    phi = (p - 1) \* (q - 1)

    e = 2

    while e < phi and gcd(e, phi) != 1:

        e += 1

    d = pow(e, -1, phi)

    return (e, n), (d, n)  # Public key (e, n), Private key (d, n)

# RSA Encryption

def encrypt(plain\_text, public\_key):

    e, n = public\_key

    return [mod\_exp(ord(char), e, n) for char in plain\_text]

# RSA Decryption

def decrypt(cipher\_text, private\_key):

d, n = private\_key

    return ''.join([chr(mod\_exp(char, d, n)) for char in cipher\_text])

# Example Usage

p = 61  # Prime number 1

q = 53  # Prime number 2

# Key generation

public\_key, private\_key = generate\_keys(p, q)

print("Public Key:", public\_key)

print("Private Key:", private\_key)

# Encrypt and decrypt

plain\_text = "HELLO"

print("Original Text:", plain\_text)

cipher\_text = encrypt(plain\_text, public\_key)

print("Encrypted Text:", cipher\_text)

decrypted\_text = decrypt(cipher\_text, private\_key)

print("Decrypted Text:", decrypted\_text)

**Output-**

**A screen shot of a computer

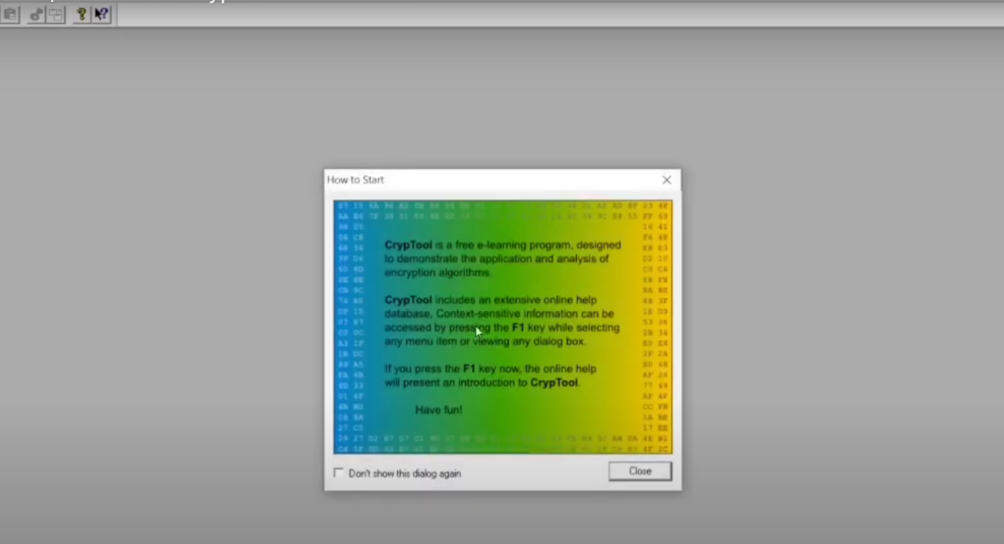
Description automatically generated**

**PRACTICLE- 10**

**AIM:** Demonstrate and perform various encryption-decryption techniques with cryptool

**Tools Required:**

* Install CrypTool 1 (for beginners, CT1 is recommended)
  + Download & install from official site.

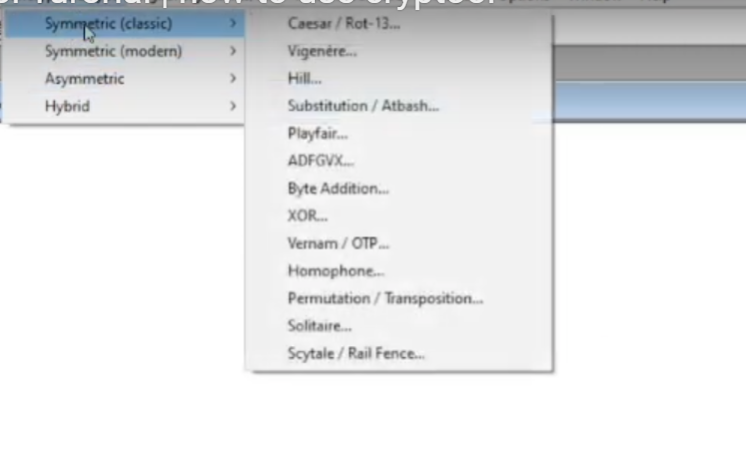
****

S**teps to Perform in CrypTool:**

A. Symmetric Encryption (e.g., AES, DES):

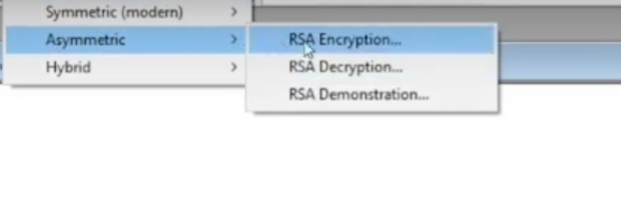
1. Open CrypTool.
2. Go to Encrypt/Decrypt → Symmetric (modern) → AES.
3. Type any sample text like: "Hello, this is test data"
4. Choose a key (e.g., 12345678) and encrypt the text.
5. Copy the cipher text.
6. Now go to Encrypt/Decrypt → Symmetric (modern) → AES again.
7. Paste the cipher text and decrypt using the same key.
8. Text should match the original.

**A screenshot of a computer

AI-generated content may be incorrect.**

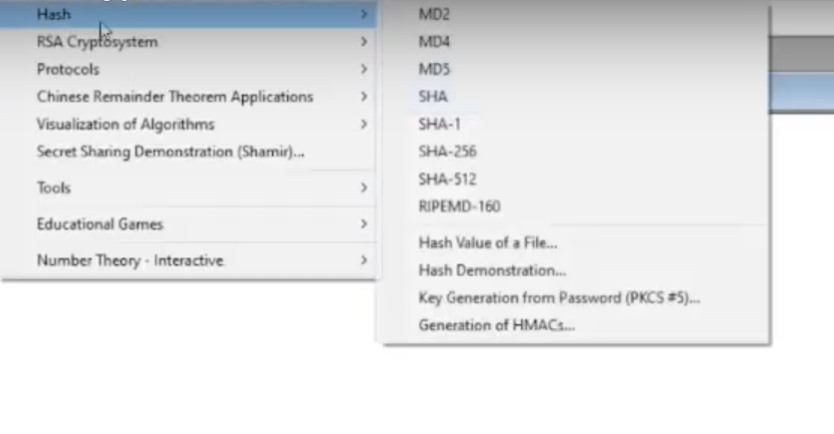
B. Asymmetric Encryption (e.g., RSA):

1. Go to Encrypt/Decrypt → Asymmetric → RSA Demonstration.
2. Enter a sample text and choose key length (e.g., 1024-bit).
3. Click on Generate Keys, then Encrypt.
4. Note the cipher text → then click Decrypt.
5. Original text should appear.

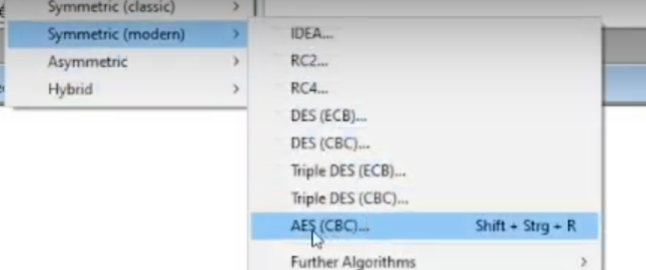
****

C. Hashing (SHA, MD5):

1. Go to Digital Signatures/PKI → Hash → SHA-256 or MD5.
2. Enter any message (e.g., “This is secure”) and generate its hash.
3. Try changing 1 letter and hash again → note how the hash changes (demonstrating sensitivity).

****

**Example:**

****

**A screenshot of a computer

AI-generated content may be incorrect.**

Jayesh

****

**A screenshot of a computer

AI-generated content may be incorrect.**

Jayesh

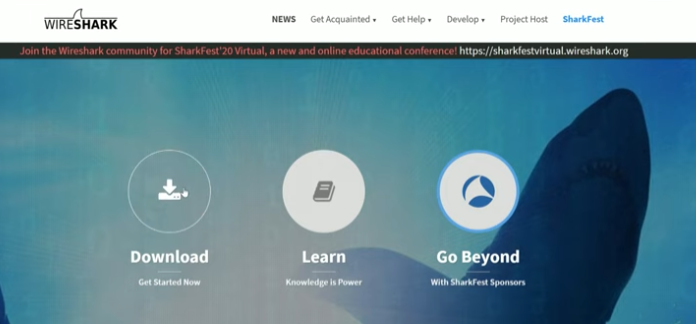


**PRACTICLE- 11**

**AIM:** Study and use open-source packet analyzer-Wireshark to understand security mechanism of various network protocols.

**Tools Required:**

* Install **Wireshark** from <https://www.wireshark.org>

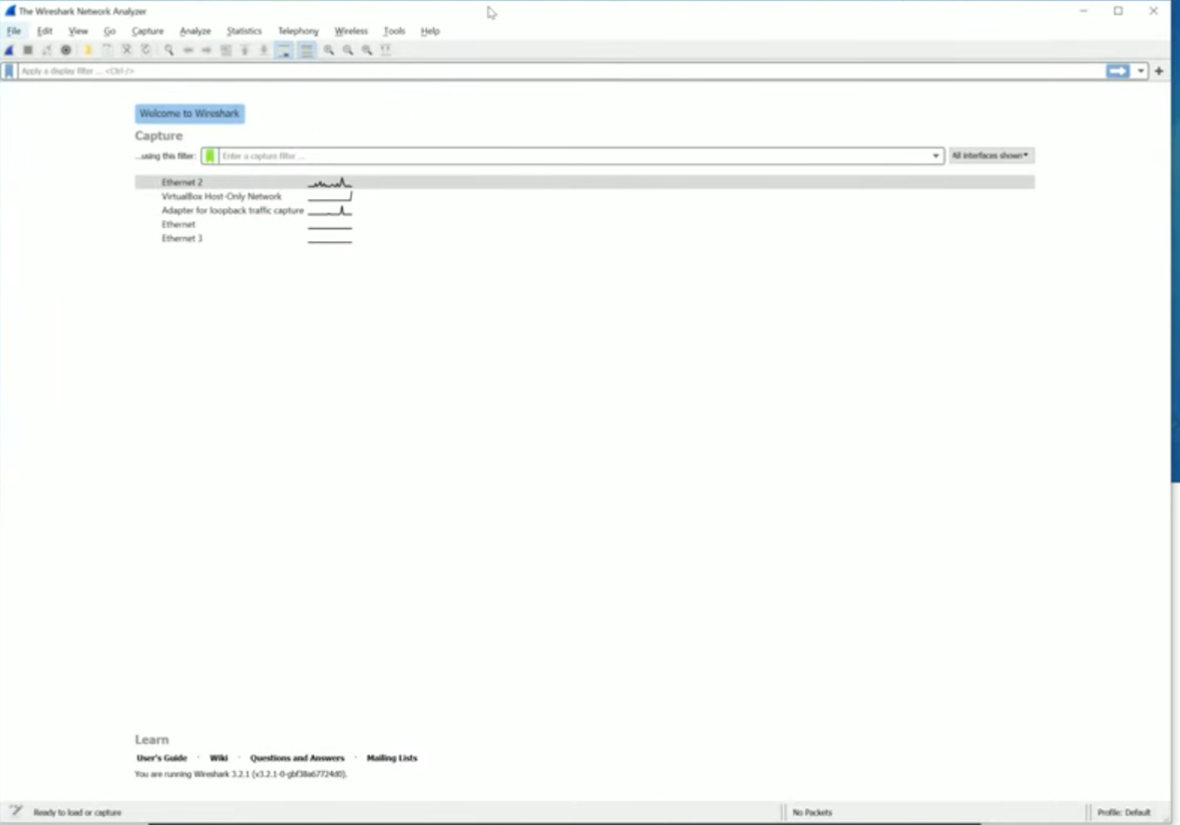


* + Use on a system with internet access (admin privileges needed).

**Steps to Perform in Wireshark:**

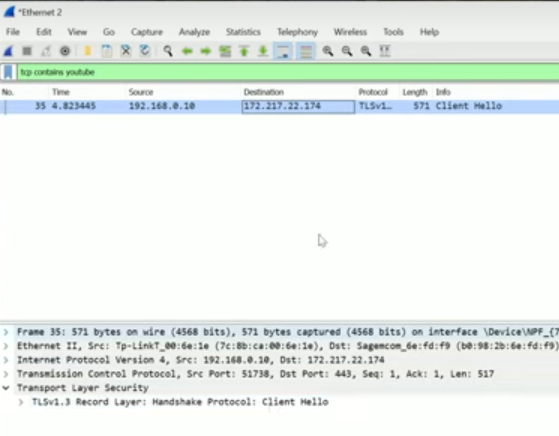
**A. Capture Packets:**

1. Open Wireshark.
2. Choose your active network interface (usually Wi-Fi or Ethernet).





1. Click **Start Capture**.
2. Open a browser and visit some websites like:
   * <https://youtube.com>



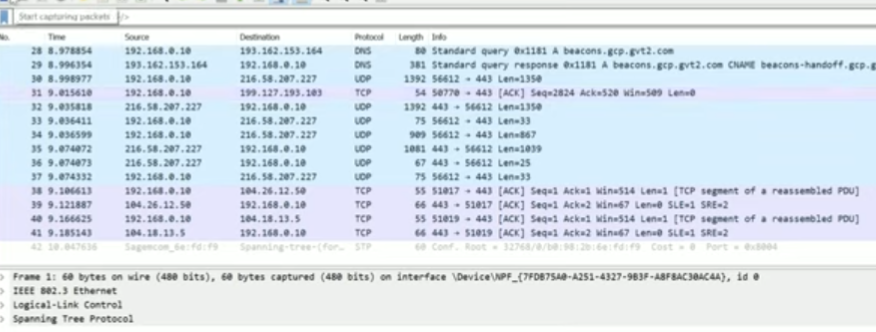
* + 1. <http://test.com> (Note: HTTP site)
  + Return to Wireshark and click **Stop** after a few seconds.

A screenshot of a computer

AI-generated content may be incorrect.

**B. Analyze Protocols:**

1. In the filter bar, type:
   * http to see HTTP packets
   * https for HTTPS packets
   * dns for DNS queries
   * tcp or udp for transport layer
   * icmp for ping packets



1. Click any packet → expand different protocol layers (Ethernet, IP, TCP, etc.)

**C. Understand Security:**

* **HTTP** shows data in plain text → insecure
* **HTTPS** is encrypted (can't read content) → secure
* View certificate in TLS packets by following:
  + Right-click HTTPS packet → “Follow TLS Stream”

**PRACTICLE- 12**

**AIM:** Detail Case study: Real world implementation of Network Security Algorithm/Concept.

**Case:** Secure GUI Chat Application using Python and AES Encryption

**Problem Statement:**

As digital communication grows, the need for private and secure messaging is

more important than ever. However, students and beginner developers often

struggle to grasp secure communication techniques, especially the use of

encryption in real-time applications. Existing tutorials focus on either

GUI or encryption but not both together making it harder to understand

end-to-end secure messaging.

**Objective:**

To build a **GUI-based secure chat application** that enables encrypted

communication between two users using AES encryption. The application

not only demonstrates encryption and decryption in real time but also helps

learners understand how cryptographic principles are applied to actual

communication tools.

**Network Security Concept Involved:**

**AES (Advanced Encryption Standard)**

* **Type:** Symmetric Key Encryption
* **Technique:** Block Cipher using Substitution-Permutation Network
* **Security Principle:** Confidentiality using 128-bit key-based encryption

**Project Features:**

| **Feature** | **Description** |
| --- | --- |
| **User Interface (GUI)** | Simple, user-friendly interface using Python’s Tkinter |
| **Real-time Chat** | Messages sent between users are encrypted and decrypted instantly |
| **AES Encryption** | All messages are encrypted using a shared secret key before transmission |
| **Decryption** | Encrypted messages are decrypted back into readable form  upon receipt |
| **Key Sharing** | Static key entry (for simplicity), with validation to ensure  proper length |
| **Learning Focused** | Built as an educational tool to show how cryptography is  integrated in apps |

**Technologies Used:**

| **Stack** | **Tools/Languages** |
| --- | --- |
| Frontend | Python (Tkinter for GUI) |
| Backend | Python (Socket Programming) |
| Encryption | AES (via pycryptodome library) |

**Encryption-Decryption Flow:**

**Encryption:**

1. User types a message in the chat input field.
2. The message is padded and encrypted using AES with a pre-defined key.
3. The encrypted message is sent via a socket to the receiver.
4. Receiver's GUI displays the incoming ciphertext (optional view).

**Decryption:**

1. Receiver receives the ciphertext through the socket.
2. The ciphertext is decrypted using the same AES key.
3. The decrypted plaintext message is shown in the receiver's chat window.

**Real-World Relevance:**

* Demonstrates **end-to-end encryption** principles similar to modern messaging apps.
* Helps students and enthusiasts understand how **secure communication protocols** are implemented.
* Acts as a **foundational tool** for further projects like secured file transfers or encrypted group chats.
* Can be extended to include advanced security measures like **key exchange (Diffie-Hellman)** or **authentication mechanisms**.

**Testing & Validation:**

| **Test Case** | **Expected Output** |
| --- | --- |
| Valid key, normal message | |  | | --- | |  |  |  | | --- | | Encrypted and decrypted message shows correctly | |
| Incorrect key on one end | Decryption fails or results in garbled output |
| Empty input message | Message is not sent; alert shown |

**Outcomes:**

* Provides a hands-on learning experience of applying AES encryption.
* Simplifies concepts of secure socket communication and GUI development.
* Encourages learners to experiment with cryptographic algorithms in real-world tools.
* Serves as a complete educational implementation of secure messaging.

**Conclusion:**

The **Secure GUI Chat Application** bridges the gap between cryptographic theory

and practical communication. By offering a real-time, visual experience of secure messaging, the project empowers students to grasp the importance of data privacy

and encryption. It stands as a strong example of integrating network security

concepts into accessible, educational software.