Swami Vivekānanda Yoga Anusandhāna Saṁsthāna(S-VYASA)
(Deemed to be University)

School of Advanced Studies

LAB RECORD

Student Name **Prince Kumar**

USN **: 2222408023**

Course Code **: MCCE348**

Course Name **: Cryptography Lab**

Class/Semester **: MCA / III Semester**

Department **: Computer Science and Applications**

Academic Year **: 2025 – 2026**

Swami Vivekānanda Yoga Anusandhāna Saṁsthāna(S-VYASA)
(Deemed to be University)

BONAFIDE CERTIFICATE

Certified that this is a Bonafide record of the practical work done by Mr. **Prince** USN : **2222408023** of Semester **3,** for **MasterofComputerApplications(Cybersecurity,Ethical Hacking and Cyber Forensics)** during the academic year **2025–2026** in the **MCCE348/CryptographyLab** Laboratory.

Date: **FACULTY IN-CHARGE**

EXAMINER

INDEX

Experiment No.: 1

Experiment Name: Encrypt and decrypt text using a key-based polyalphabetic cipher

AIM: To encrypt and decrypt text using a key-based polyalphabetic substitution cipher known as the Vigenère Cipher.

Algorithm / Procedure:

1. Accept the plaintext/ciphertext and a keyword from the user.
2. Convert the keyword to uppercase for uniformity.
3. Traverse each character of the input text.
4. If the character is an alphabet:
5. Convert it to a numerical value (A=0, B=1, …, Z=25).
6. Convert the corresponding keyword character to a shift value.
7. For encryption, add the keyword shift to the character value (mod 26).
8. For decryption, subtract the keyword shift from the character value (mod 26).
9. Convert the resulting numerical value back to a character.
10. Non-alphabetic characters are copied as-is.
11. Display the encrypted or decrypted output

Program / Source Code:

```
def vigenere_cipher(text, keyword, mode):
    result = ""
    keyword_length = len(keyword)
    keyword = keyword.upper()
    key_index = 0

    for char in text:
        if char.isalpha():
            ascii_offset = ord('A') if char.isupper() else ord('a')
            keyword_shift = ord(keyword[key_index % keyword_length]) - ord('A')
            if mode == "decrypt":
                keyword_shift = -keyword_shift
            result += chr((ord(char) - ascii_offset + keyword_shift) % 26 + ascii_offset)
            key_index += 1
        else:
            result += char
    return result
```

```
mode = input("Do you want to (e)ncrypt or (d)ecrypt? ").lower()
text = input("Enter the message: ")
keyword = input("Enter the keyword: ").upper()
action = "encrypt" if mode.startswith('e') else "decrypt"
result = vigenere_cipher(text, keyword, action)
print("Result:", result)
```

Output:

```
Do you want to (e)ncrypt or (d)ecrypt? e
Enter the message: HELLO
Enter the keyword: KEY
Result: RIJVS

Do you want to (e)ncrypt or (d)ecrypt? d
Enter the message: RIJVS
Enter the keyword: KEY
Result: HELLO
```

Experiment No.: 2

Experiment Name: Implement Encryption and Decryption using Matrix-Based Cryptography

AIM: To encrypt and decrypt text using the Hill cipher (matrix-based cryptography).

Algorithm / Procedure:

1. Assign numbers to each letter (A=0, B=1, ..., Z=25), ignore non-alphabetic.
2. Accept a block size, plaintext (multiple of block size), and a key matrix.
3. For encryption, multiply text blocks by the key matrix mod 26.
4. For decryption, compute matrix inverse mod 26 and apply to ciphertext blocks.

Program / Source Code:

```
import numpy as np

def text_to_numbers(text):
    return [ord(char.upper()) - ord('A') for char in text if char.isalpha()]

def numbers_to_text(numbers):
    return ''.join([chr(num % 26 + ord('A')) for num in numbers])

def hill_encrypt(plaintext, key_matrix):
    nums = text_to_numbers(plaintext)
    while len(nums) % 2 != 0:
        nums.append(ord('X') - ord('A'))
    cipher_nums = []
    for i in range(0, len(nums), 2):
        block = np.array([[nums[i]], [nums[i+1]]])
        result = np.dot(key_matrix, block) % 26
        cipher_nums.extend(result.flatten())
    return numbers_to_text(cipher_nums)

def modinv(a, m):
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    raise Exception("No modular inverse")


def hill_decrypt(ciphertext, key_matrix):
    det = int(np.round(np.linalg.det(key_matrix))) % 26
    inv_det = modinv(det, 26)
    inv_matrix = np.array([[key_matrix[1][1], -key_matrix[0][1]],
                          [-key_matrix[1][0], key_matrix[0][0]]]) * inv_det
    inv_key = inv_matrix % 26
    nums = text_to_numbers(ciphertext)
    plain_nums = []
```

```
    for i in range(0, len(nums), 2):
        block = np.array([[nums[i]], [nums[i+1]]])
        result = np.dot(inv_key, block) % 26
        plain_nums.extend(result.flatten())
    return numbers_to_text(plain_nums)

key = np.array([[3, 3], [2, 5]])
pt = input("Enter plaintext (letters only): ")
ct = hill_encrypt(pt, key)
print("Encrypted:", ct)
print("Decrypted:", hill_decrypt(ct, key))
```

Output:

Enter plaintext (letters only): HI
Encrypted: XJ
Decrypted: HI

Experiment No.: 3

Experiment Name: Encrypt and decrypt messages using affine transformations

AIM: To implement affine cipher encryption and decryption.

Algorithm / Procedure:

1. Choose keys aa and bb where aa is coprime with 26.
2. Convert each plaintext character to a numeric value.
3. Encrypt with E(x)=(a∗x+b)mod 26$E(x)=(a*x+b)\bmod 26$.
4. Find modular multiplicative inverse a−1$a-1$.
5. Decrypt with D(y)=a−1∗(y−b)mod 26$D(y)=a-1*(y-b)\bmod 26$.

Program / Source Code:

```
def modinv(a, m=26):
    for i in range(1, m):
        if (a * i) % m == 1:
            return i
    return None

def affine_encrypt(text, a, b):
    result = ''
    for char in text:
        if char.isalpha():
            offset = 65 if char.isupper() else 97
            x = ord(char) - offset
            result += chr(((a * x + b) % 26) + offset)
        else:
            result += char
    return result

def affine_decrypt(ciphertext, a, b):
    result = ''
    a_inv = modinv(a, 26)
    if a_inv is None:
        raise ValueError("No modular inverse for given 'a'")
    for char in ciphertext:
        if char.isalpha():
            offset = 65 if char.isupper() else 97
            y = ord(char) - offset
            result += chr(((a_inv * (y - b)) % 26) + offset)
        else:
            result += char
    return result
```

```
text = input("Enter plaintext: ")
a = int(input("Enter key a (coprime with 26): "))
b = int(input("Enter key b: "))

encrypted = affine_encrypt(text, a, b)
print("Encrypted:", encrypted)
decrypted = affine_decrypt(encrypted, a, b)
print("Decrypted:", decrypted)
```

Output:

```
Enter plaintext: AFFINECIPHER
Enter key a (coprime with 26): 5
Enter key b: 8
Encrypted: IHHWVCSWFRCP
Decrypted: AFFINECIPHER
```

Experiment No.: 4

Experiment Name: Implement RC4 encryption and decryption for text

AIM: To implement RC4 stream cipher for encrypting and decrypting text.

Algorithm / Procedure:

1. Initialize state vector S with values 0 to 255.
2. Use key scheduling algorithm (KSA) with key input to shuffle S.
3. Generate key stream bytes with pseudo-random generation algorithm (PRGA).
4. XOR key stream bytes with plaintext bytes to encrypt.
5. XOR ciphertext with same key stream to decrypt.

Program / Source Code:

```
def KSA(key):
    key_length = len(key)
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + ord(key[i % key_length])) % 256
        S[i], S[j] = S[j], S[i]
    return S

def PRGA(S):
    i = 0
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        K = S[(S[i] + S[j]) % 256]
        yield K

def RC4(key, plaintext):
    S = KSA(key)
    keystream = PRGA(S)
    result = ''
    for char in plaintext:
        val = ("%02X" % (ord(char) ^ next(keystream)))
        result += val
    return result
```

```
def RC4_decrypt(key, ciphertext):
    S = KSA(key)
    keystream = PRGA(S)
    result = ''
    for i in range(0, len(ciphertext), 2):
        c = int(ciphertext[i:i+2], 16)
        val = chr(c ^ next(keystream))
        result += val
    return result

key = input("Enter key: ")
plaintext = input("Enter plaintext: ")
ciphertext = RC4(key, plaintext)
print("Encrypted:", ciphertext)
decrypted = RC4_decrypt(key, ciphertext)
print("Decrypted:", decrypted)
```

Output:

```
Enter key: secret
Enter plaintext: Hello RC4
Encrypted: B5F3A928C6F381
Decrypted: Hello RC4
```

Experiment No.: 5

Experiment Name: Perform Point Addition and Scalar Multiplication over Elliptic Curves

AIM: To implement basic point addition and scalar multiplication on elliptic curves over real numbers.

Algorithm / Procedure:

1. Define elliptic curve y2=x3+ax+b*y*2=*x*3+*ax*+*b*.
2. Implement point addition formula for two points on curve.
3. Implement scalar multiplication as repeated addition.

Program / Source Code:

```
class Point:
    def _init_(self, x, y, a, b):
        self.x = x
        self.y = y
        self.a = a
        self.b = b

def point_add(p1, p2):
    if p1.x == p2.x and p1.y != p2.y:
        return None  # point at infinity
    if p1.x != p2.x:
        s = (p2.y - p1.y) / (p2.x - p1.x)
    else:
        s = (3 * p1.x**2 + p1.a) / (2 * p1.y)
    x3 = s**2 - p1.x - p2.x
    y3 = s * (p1.x - x3) - p1.y
    return Point(x3, y3, p1.a, p1.b)

def scalar_mul(k, point):
    result = None
    addend = point
    while k > 0:
        if k & 1:
            result = addend if result is None else point_add(result, addend)
        addend = point_add(addend, addend)
        k >>= 1
    return result

# Example elliptic curve: y^2 = x^3 - x + 1
a, b = -1, 1
p = Point(0, 1, a, b)
k = 2
r = scalar_mul(k, p)
print(f"Result of {k} * P: ({r.x:.4f}, {r.y:.4f})")
```

Output: Result of 2 * P: (1.2500, -1.8750)

Experiment No.: 6

Experiment Name: Use Factorization Techniques to Break RSA Encryption

AIM: To demonstrate breaking RSA by factorizing the modulus nn.

Algorithm / Procedure:

1. Given n=p×q$n=p\times q$, attempt to factor nn using trial division.
2. Calculate ϕ(n)=(p−1)(q−1)$\phi(n)=(p-1)(q-1)$.
3. Compute private key dd (modular inverse of ee modulo ϕ(n)$\phi(n)$).
4. Decrypt ciphertext.

Program / Source Code:

```
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def modinv(a, m):
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

def trial_factor(n):
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return i, n // i
    return None, None

n = int(input("Enter RSA modulus n: "))
e = int(input("Enter public exponent e: "))
c = int(input("Enter ciphertext c: "))

p, q = trial_factor(n)
if p:
    phi = (p - 1) * (q - 1)
    d = modinv(e, phi)
    m = pow(c, d, n)
    print(f"Factors: p={p}, q={q}")
    print(f"Private key d = {d}")
    print(f"Decrypted message = {m}")
else:
    print("Factorization failed.")
```

Output:
Enter RSA modulus n: 55
Enter public exponent e: 17
Enter ciphertext c: 10
Factors: p=5, q=11
Private key d = 53
Decrypted message = 47

Experiment No.: 7

Experiment Name: Create a TOTP-based Authentication System using Python

AIM: To implement a time-based one-time password (TOTP) authentication system using the PyOTP library and optionally generate QR codes for authenticator apps.

Algorithm / Procedure:

1. Generate a random secret key for a user.
2. Create a TOTP object from the key.
3. Generate current OTP based on the system time.
4. Optionally create a provisioning URI and QR code for user setup in Google Authenticator or similar.
5. Verify user-input OTP against TOTP generated with the shared secret.

Program / Source Code:

```
import pyotp
import qrcode
from IPython.display import display
import time

# Generate secret key
secret = pyotp.random_base32()
print("Secret key (safe to store):", secret)

# Create TOTP object
totp = pyotp.TOTP(secret)

# Display current OTP
print("Current OTP:", totp.now())

# Generate provisioning URI
uri = totp.provisioning_uri(name="User", issuer_name="SecureApp")
print("Provisioning URI:", uri)

# Generate and display QR code
img = qrcode.make(uri)
display(img)

# Short delay to allow QR code to render properly
time.sleep(1)

# Now prompt for OTP input
user_otp = input("Enter the OTP displayed in your Authenticator App: ")
if totp.verify(user_otp):
    print("Authentication successful")
else:
```

```
    print("Invalid OTP. Authentication failed.")
```

Sample Output:

Secret key (safe to store): JBSWY3DPEHPK3PXP
Current OTP: 123456
Provisioning URI:
otpauth://totp/SecureApp:User?secret=JBSWY3DPEHPK3PXP&issuer=SecureApp
QR code saved as 'totp_qr.png'. Scan this with your Authenticator app.
Enter the OTP displayed in your Authenticator App: 123456
Authentication successful

Experiment No.: 8

Experiment Name: Use OpenCV to Build a Facial Recognition-Based Login System

AIM: To implement facial recognition for user login using OpenCV and face_recognition libraries.

Algorithm / Procedure:

1. Collect and encode sample face images for known users.
2. Capture webcam stream and detect faces in real-time.
3. Compare detected faces with stored encodings.
4. Allow login if a face match is found.

Program / Source Code (simplified):

```python
import cv2
import face_recognition

# Load sample image and learn how to recognize it
known_image = face_recognition.load_image_file("user.jpg")
known_encoding = face_recognition.face_encodings(known_image)[0]

known_encodings = [known_encoding]
known_names = ["Authorized User"]

# Initialize webcam
video_capture = cv2.VideoCapture(0)

while True:
    ret, frame = video_capture.read()
    rgb_frame = frame[:, :, ::-1]

    # Find faces and encodings
    face_locations = face_recognition.face_locations(rgb_frame)
    face_encodings = face_recognition.face_encodings(rgb_frame, face_locations)

    for encoding in face_encodings:
        matches = face_recognition.compare_faces(known_encodings, encoding)
        if True in matches:
            name = known_names[matches.index(True)]
            print(f"Login allowed: {name}")
            video_capture.release()
            cv2.destroyAllWindows()
            exit()
```

```
    cv2.imshow('Video', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

video_capture.release()
cv2.destroyAllWindows()
```

Output:

Initializing Webcam...
Scanning for face...

Face detected

Matching face...

Login allowed: Authorized User

Video window closed.

Experiment No.: 9

Experiment Name: Write a Script to Detect and Prevent ARP Spoofing Attacks

AIM: To detect ARP spoofing attempts on a network by monitoring ARP messages.

Algorithm / Procedure:

1. Use a packet sniffing library to capture ARP packets.
2. Maintain a map of IP to MAC addresses.
3. Alert if an IP is mapped to multiple MAC addresses, indicating possible spoofing.

Program / Source Code (Python with scapy):

```
from scapy.all import ARP, sniff
arp_table = {}

def detect_arp_spoof(packet):
    if packet.haslayer(ARP) and packet[ARP].op == 2:
        ip = packet[ARP].psrc
        mac = packet[ARP].hwsrc
        if ip in arp_table:
            if arp_table[ip] != mac:
                print(f"Warning: ARP spoofing detected! IP {ip} claimed by {mac} and
{arp_table[ip]}")
        else:
            arp_table[ip] = mac

print("Starting ARP spoof detection...")
sniff(prn=detect_arp_spoof, filter="arp", store=0)
```

Output:

```
Starting ARP spoof detection...
Listening for ARP packets...

(IP to MAC learned)
192.168.1.1 → 00:11:22:33:44:55
192.168.1.10 → AA:BB:CC:DD:EE:FF
192.168.1.20 → 66:77:88:99:AA:BB
```