1. **Encrypt and decrypt text using a key-based polyalphabetic cipher.**

```
text = input("Enter text: ").upper()
key = input("Enter key: ").upper()

encrypted = ""
for i in range(len(text)):
    if text[i].isalpha():
        encrypted += chr((ord(text[i]) + ord(key[i % len(key)]) - 130) % 26 + 65)
    else:
        encrypted += text[i]

print("Encrypted:", encrypted)



decrypted = ""
for i in range(len(encrypted)):
    if encrypted[i].isalpha():
        decrypted += chr((ord(encrypted[i]) - ord(key[i % len(key)]) + 26) % 26 + 65)
    else:
        decrypted += encrypted[i]

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

2. **Implement Encryption and Decryption using Matrix-Based**

```
text = input("Enter text: ").upper()

if len(text) % 2 != 0:
    text += 'X'

k11, k12, k21, k22 = 1, 2, 3, 5
ik11, ik12, ik21, ik22 = 21, 2, 3, 25

nums = [ord(c) - 65 for c in text]

enc = []
for i in range(0, len(nums), 2):
```

```
    x = (k11*nums[i] + k12*nums[i+1]) % 26
    y = (k21*nums[i] + k22*nums[i+1]) % 26
    enc.append(x)
    enc.append(y)

encrypted = ''.join(chr(n + 65) for n in enc)
print("Encrypted:", encrypted)

dec = []
for i in range(0, len(enc), 2):
    x = (ik11*enc[i] + ik12*enc[i+1]) % 26
    y = (ik21*enc[i] + ik22*enc[i+1]) % 26
    dec.append(x)
    dec.append(y)

decrypted = ''.join(chr(n + 65) for n in dec)
print("Decrypted:", decrypted)
```

3. **Encrypt and decrypt messages using affine transformations**

```
text = input("Enter text: ").upper()

a = 5
b = 8
a_inv = 21



enc = ""
for c in text:
    if c.isalpha():
        x = (a * (ord(c) - 65) + b) % 26
        enc += chr(x + 65)
    else:
        enc += c

print("Encrypted:", enc)



dec = ""
for c in enc:
    if c.isalpha():
```

```
        x = (a_inv * ((ord(c) - 65) - b)) % 26
        dec += chr(x + 65)
    else:
        dec += c

print("Decrypted:", dec)
```

4. **Implement RC4 encryption and decryption for text**

```
def rc4(text, key):
    S = list(range(256))
    j = 0

    for i in range(256):
        j = (j + S[i] + ord(key[i % len(key)])) % 256
        S[i], S[j] = S[j], S[i]

    i = j = 0
    result = ""

    for char in text:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        k = S[(S[i] + S[j]) % 256]
        result += chr(ord(char) ^ k)

    return result

# MAIN
text = input("Enter text: ")
key = input("Enter key: ")

encrypted = rc4(text, key)
print("Encrypted:", encrypted)

decrypted = rc4(encrypted, key)
print("Decrypted:", decrypted)
```

5. **Perform Point Addition and Scalar Multiplication over Elliptic Curves**

```
# Elliptic Curve: y^2 = x^3 + ax + b mod p
a = 2
```

```
b = 3
p = 97

def modinv(x, p):
    return pow(x, p-2, p)

def point_add(P, Q):
    x1, y1 = P
    x2, y2 = Q

    m = ((y2 - y1) * modinv(x2 - x1, p)) % p
    x3 = (m*m - x1 - x2) % p
    y3 = (m*(x1 - x3) - y1) % p
    return (x3, y3)

P = (3, 6)
Q = (80, 10)

R = point_add(P, Q)
print("P + Q =", R)

k = 3
R = P
for i in range(k - 1):
    R = point_add(R, P)

print("3P =", R)
```

6. **Use Factorization Techniques to Break RSA Encryption**

```
# Public key (small values for lab)
n = 55      # n = p * q
e = 3
cipher = 34

for i in range(2, n):
    if n % i == 0:
        p = i
        q = n // i
        break
```

```
phi = (p - 1) * (q - 1)

for d in range(1, phi):
    if (d * e) % phi == 1:
        break

plain = pow(cipher, d, n)

print("p =", p, "q =", q)
print("Private key d =", d)
print("Decrypted message =", plain)
```