In-Lab:

1. **Affine Cipher:**

```
def encrypt_affine(text, a, b):
    def affine_encrypt_char(c, a, b):
        return chr(((a * (ord(c) - ord('A')) + b) % 26) + ord('A'))

    return ''.join(affine_encrypt_char(c, a, b) if c.isalpha() else c for c in text.upper())

def decrypt_affine(ciphertext, a, b):
    def mod_inverse(a, m):
        for x in range(1, m):
            if (a * x) % m == 1:
                return x
        return None

    a_inv = mod_inverse(a, 26)
    if a_inv is None:
        raise ValueError("a and 26 are not coprime")

    def affine_decrypt_char(c, a_inv, b):
        return chr((a_inv * (ord(c) - ord('A') - b) % 26) + ord('A'))

    return ''.join(affine_decrypt_char(c, a_inv, b) if c.isalpha() else c for c in ciphertext.upper())

# Test the API
plaintext = "HELLO"
a, b = 5, 8
ciphertext = encrypt_affine(plaintext, a, b)
decrypted = decrypt_affine(ciphertext, a, b)

print(f"Plaintext: {plaintext}")
print(f"Ciphertext: {ciphertext}")

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

2. **Caesar Cipher:**

```
def encrypt_caesar(text, shift):
  return ''.join(chr((ord(c) - ord('A') + shift) % 26 + ord('A')) if c.isalpha() else c for c in text.upper())

def decrypt_caesar(ciphertext, shift):
  return ''.join(chr((ord(c) - ord('A') - shift) % 26 + ord('A')) if c.isalpha() else c for c in ciphertext.upper())

# Test the API
plaintext = "HELLO"
shift = 3
ciphertext = encrypt_caesar(plaintext, shift)
decrypted = decrypt_caesar(ciphertext, shift)

print(f"Plaintext: {plaintext}")
print(f"Ciphertext: {ciphertext}")
print(f"Decrypted: {decrypted}")
```

3. **Miller–Rabin Primality Test:**

```
import random

def miller_rabin_test(n, k):
  if n <= 1 or n == 4:
    return False
  if n <= 3:
    return True

  d = n - 1
  while d % 2 == 0:
    d //= 2

  for _ in range(k):
    a = random.randint(2, n - 2)
    x = pow(a, d, n)
    if x == 1 or x == n - 1:
      continue
    while d != n - 1:
      x = (x * x) % n
      d *= 2
      if x == n - 1:
        break
    else:
      return False
    return True

# Test the API
number = 31
k = 4  # Number of iterations
result = miller_rabin_test(number, k)

print(f"Is {number} prime? {result}")
```

4. **Euclidean Algorithm for Finding GCD(A, B):**

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# Test the API
a, b = 56, 98
result = gcd(a, b)

print(f"GCD of {a} and {b} is {result}")
```

5. **Rabin Cryptosystem:**

```
import sympy

def rabin_encrypt(m, n):
    return (m * m) % n

def rabin_decrypt(c, p, q):
    n = p * q
    mp = pow(c, (p + 1) // 4, p)
    mq = pow(c, (q + 1) // 4, q)

    yp = sympy.mod_inverse(p, q)
    yq = sympy.mod_inverse(q, p)

    r = (yp * p * mq + yq * q * mp) % n
    nr = n - r
    s = (yp * p * mq - yq * q * mp) % n
    ns = n - s

    return r, nr, s, ns

# Test the API
p, q = 7, 11
n = p * q
message = 5
ciphertext = rabin_encrypt(message, n)
decrypted = rabin_decrypt(ciphertext, p, q)
```

```
print(f"Message: {message}")
print(f"Ciphertext: {ciphertext}")
print(f"Decrypted possibilities: {decrypted}")
```