

# Cryptography and Information Security

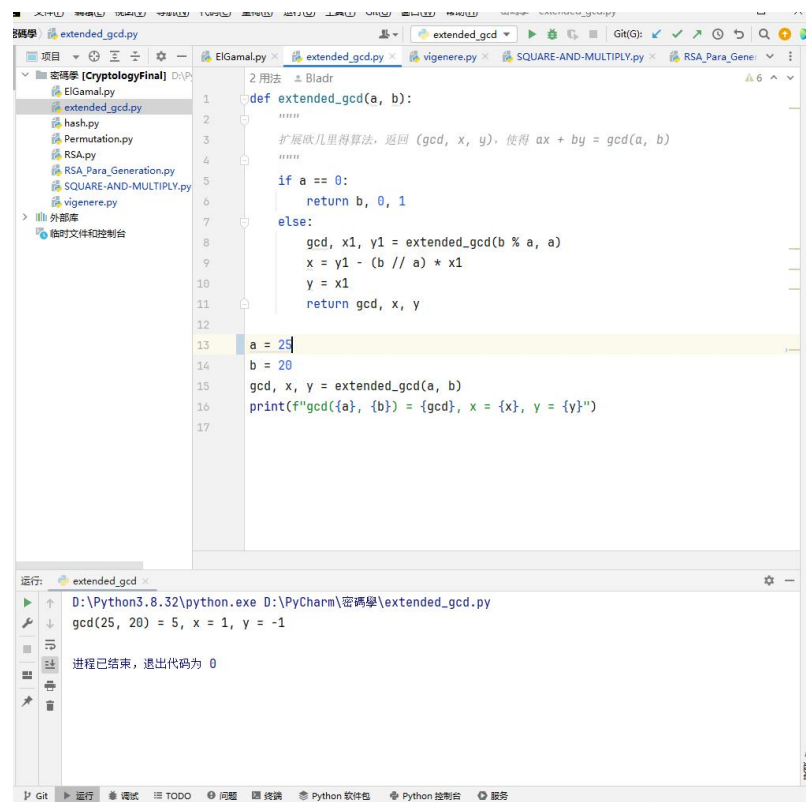
## Final Report

WangYihan 2230017050

### Algorithm 6.2: EXTENDED EUCLIDEAN ALGORITHM(a, b)

```
def extended_gcd(a, b):  
    if a == 0:  
        return b, 0, 1  
    else:  
        gcd, x1, y1 = extended_gcd(b % a, a)  
        x = y1 - (b // a) * x1  
        y = x1  
        return gcd, x, y
```

run result:



```
def extended_gcd(a, b):  
    """  
    扩展欧几里得算法，返回 (gcd, x, y)，使得 ax + by = gcd(a, b)  
    """  
    if a == 0:  
        return b, 0, 1  
    else:  
        gcd, x1, y1 = extended_gcd(b % a, a)  
        x = y1 - (b // a) * x1  
        y = x1  
        return gcd, x, y  
  
a = 25  
b = 20  
gcd, x, y = extended_gcd(a, b)  
print(f"gcd({a}, {b}) = {gcd}, x = {x}, y = {y}")
```

运行: extended\_gcd

D:\Python3.8.32\python.exe D:\PyCharm\密码学\extended\_gcd.py  
gcd(25, 20) = 5, x = 1, y = -1  
进程已结束，退出代码为 0

#### Algorithm 6.4: RSA PARAMETER GENERATION

```
import sympy
import random

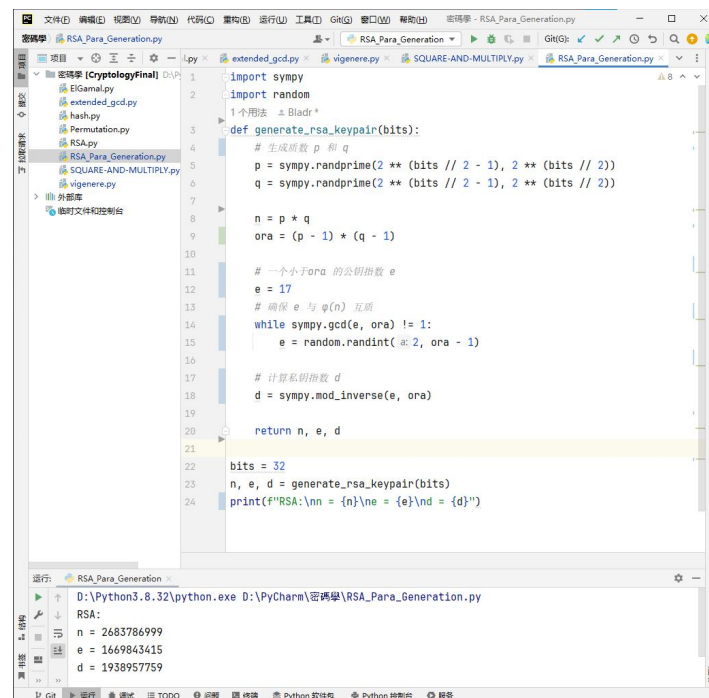
def generate_rsa_keypair(bits):
    # 生成质数 p 和 q
    p = sympy.randprime(2 ** (bits // 2 - 1), 2 ** (bits // 2))
    q = sympy.randprime(2 ** (bits // 2 - 1), 2 ** (bits // 2))

    n = p * q
    ora = (p - 1) * (q - 1)
    e = 17

    while sympy.gcd(e, ora) != 1:
        e = random.randint(2, ora - 1)
    d = sympy.mod_inverse(e, ora)

    return n, e, d
```

run result:



The screenshot shows a Python IDE with a file explorer on the left, a code editor in the center, and a console at the bottom. The code editor displays the `generate_rsa_keypair` function and its execution. The console shows the output of the program, which is the RSA parameters `n`, `e`, and `d`.

```
1 import sympy
2 import random
3
4 def generate_rsa_keypair(bits):
5     # 生成质数 p 和 q
6     p = sympy.randprime(2 ** (bits // 2 - 1), 2 ** (bits // 2))
7     q = sympy.randprime(2 ** (bits // 2 - 1), 2 ** (bits // 2))
8
9     n = p * q
10    ora = (p - 1) * (q - 1)
11
12    # 一个小于 ora 的公钥指数 e
13    e = 17
14    # 确保 e 与 q(n) 互质
15    while sympy.gcd(e, ora) != 1:
16        e = random.randint(2, ora - 1)
17
18    # 计算私钥指数 d
19    d = sympy.mod_inverse(e, ora)
20
21    return n, e, d
22
23 bits = 32
24 n, e, d = generate_rsa_keypair(bits)
25 print(f"RSA:\nn = {n}\ne = {e}\nd = {d}")
```

运行: RSA\_Param\_Generation.py

```
D:\Python3.8.32\python.exe D:\PyCharm\密码学\RSA_Param_Generation.py
RSA:
n = 2683786999
e = 1669843415
d = 1938957759
```

#### Algorithm 6.5: SQUARE-AND-MULTIPLY(x,c,n)

```
def square_and_multiply(a, b, m):
    result = 1
    a = a % m
```

```

while b > 0:
    if (b % 2) == 1:
        result = (result * a) % m
    b = b >> 1 # Equivalent to b // 2
    a = (a * a) % m # Square the a

return result

```

run result:

```

def square_and_multiply(a, b, m):
    result = 1
    a = a % m

    while b > 0:
        if (b % 2) == 1:
            result = (result * a) % m
        b = b >> 1 # Equivalent to b // 2
        a = (a * a) % m # Square the a

    return result

a = 5
b = 17
m = 497
print(square_and_multiply(a, b, m))

```

运行: RSA\_Para\_Generation

```

D:\Python3.8.32\python.exe D:\PyCharm\密碼學\RSA_Para_Generation.py
RSA:
n = 2191469221
e = 17
d = 644522193
进程已结束, 退出代码为 0

```

## Algorithm for Cryptosystem 6.1: RSA Cryptosystem

```

import random
import sympy

def generate_rsa_keypair(bits):
    p = sympy.randprime(2 ** (bits // 2 - 1), 2 ** (bits // 2))
    q = sympy.randprime(2 ** (bits // 2 - 1), 2 ** (bits // 2))

    n = p * q
    ora = (p - 1) * (q - 1)
    e = 17
    while sympy.gcd(e, ora) != 1:
        e = random.randint(2, ora - 1)
    d = sympy.mod_inverse(e, ora)
    return n, e, d

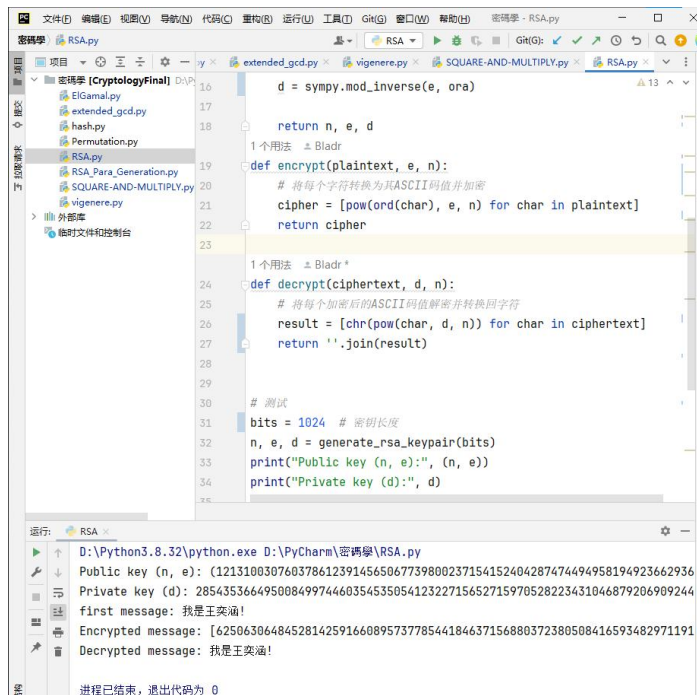
def encrypt(plaintext, e, n):
    cipher = [pow(ord(char), e, n) for char in plaintext]

```

```
return cipher
```

```
def decrypt(ciphertext, d, n):  
    result = [chr(pow(char, d, n)) for char in ciphertext]  
    return ''.join(result)
```

run result



The screenshot shows a Python IDE with a file explorer on the left containing files like ElGamal.py, extended\_gcd.py, hash.py, Permutation.py, RSA.py, RSA\_Param\_Generation.py, SQUARE-AND-MULTIPLY.py, and vigener.py. The main editor displays the RSA.py code, which includes functions for generating RSA keypairs, encrypting, and decrypting. The console at the bottom shows the execution output:

```
D:\Python3.8.32\python.exe D:\PyCharm\密码学\RSA.py  
Public key (n, e): (1213100307603786123914565067739800237154152404287474494958194923662936  
Private key (d): 2854353664950084997446035453505412322715652715970528223431046879206909244  
first message: 我是王奕涵!  
Encrypted message: [6250630648452814259166089573778544184637156880372380508416593482971191  
Decrypted message: 我是王奕涵!  
进程已结束, 退出代码为 0
```

## Algorithm for Cryptosystem 7.1: ElGamal Public-key Cryptosystem

```
import random  
import sympy  
  
def generate_keypair(bits):  
    while True:  
        p = random.getrandbits(bits)  
        if sympy.isprime(p):  
            break  
  
    g = random.randint(2, p - 1)  
    x = random.randint(1, p - 2)  
    y = pow(g, x, p)  
    return (p, g, y), x  
  
def encrypt(result, public_key):  
    p, g, y = public_key
```

```

m = int.from_bytes(result.encode(), 'big')
k = random.randint(1, p - 2)
c1 = pow(g, k, p)
c2 = (m * pow(y, k, p)) % p

return (c1, c2)

def decrypt(ciphertext, private_key, public_key):
    c1, c2 = ciphertext
    p, g, y = public_key
    x = private_key
    s = pow(c1, x, p)
    s_inverse = sympy.mod_inverse(s, p)
    m = (c2 * s_inverse) % p
    result = m.to_bytes((m.bit_length() + 7) // 8, 'big').decode()

    return result

```

run result:

```

def encrypt(result, public_key):
    p, g, y = public_key

    # 将明文转换为整数
    m = int.from_bytes(result.encode(), 'big')

    # 选择一个随机数 k
    k = random.randint(1, p - 2)

    # 计算 c1 = g^k mod p 和 c2 = m * y^k mod p
    c1 = pow(g, k, p)
    c2 = (m * pow(y, k, p)) % p

    return (c1, c2)

def decrypt(ciphertext, private_key, public_key):
    c1, c2 = ciphertext
    p, g, y = public_key
    x = private_key

    # 计算 s = c1^x mod p
    s = pow(c1, x, p)
    s_inverse = sympy.mod_inverse(s, p)
    m = (c2 * s_inverse) % p
    result = m.to_bytes((m.bit_length() + 7) // 8, 'big').decode()

    return result

```

运行: ElGamal

```

D:\Python3.8.32\python.exe D:\PyCharm\密码学\ElGamal.py
Public key (p, g, y): (1008515337756540238635642578820392165116073338323480526328068859962
Private key (x): 7060441711773958711503226573761576768223192547916073503203354178710563743
Original message: 我是王奕涵
Encrypted message: (8432754163489374203976365538648536992265588248246466972803683304502495
Decrypted message: 我是王奕涵

进程已结束, 退出代码为 0

```

## Algorithm for The Permutation Cipher

```

import random

def generate_key(length):
    key = list(range(length))
    random.shuffle(key)

```

```

        return key

def permutation_encrypt(plaintext, key):
    ciphertext = [""] * len(plaintext)
    for i, char in enumerate(plaintext):
        ciphertext[key[i]] = char
    return "".join(ciphertext)

def permutation_decrypt(ciphertext, key):
    plaintext = [""] * len(ciphertext)
    for i, char in enumerate(ciphertext):
        plaintext[key[i]] = char
    return "".join(plaintext)

```

Run result:

### Algorithm for the vigenere cipher

```

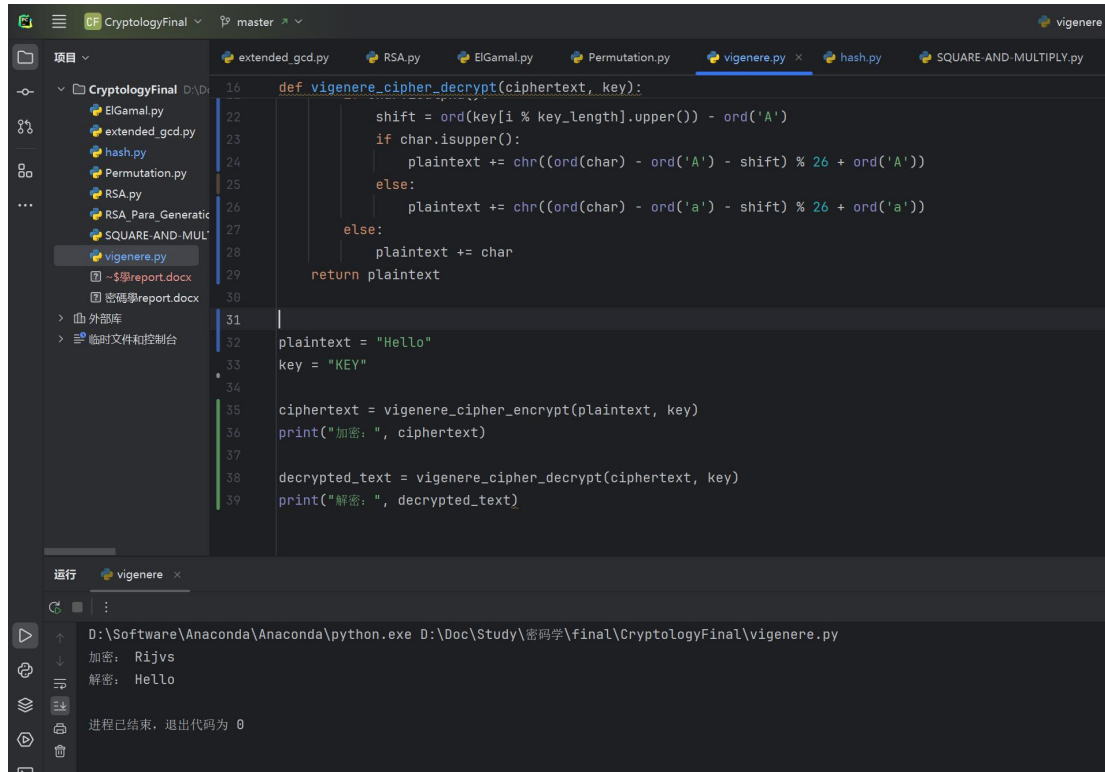
def vigenere_cipher_encrypt(plaintext, key):
    ciphertext = ""
    key_length = len(key)
    for i in range(len(plaintext)):
        char = plaintext[i]
        if char.isalpha():
            shift = ord(key[i % key_length].upper()) - ord('A')
            if char.isupper():
                ciphertext += chr((ord(char) - ord('A') + shift) % 26 + ord('A'))
            else:
                ciphertext += chr((ord(char) - ord('a') + shift) % 26 + ord('a'))
        else:
            ciphertext += char
    return ciphertext

def vigenere_cipher_decrypt(ciphertext, key):
    plaintext = ""
    key_length = len(key)
    for i in range(len(ciphertext)):
        char = ciphertext[i]
        if char.isalpha():
            shift = ord(key[i % key_length].upper()) - ord('A')
            if char.isupper():
                plaintext += chr((ord(char) - ord('A') - shift) % 26 + ord('A'))
            else:
                plaintext += chr((ord(char) - ord('a') - shift) % 26 + ord('a'))
        else:

```

```
    plaintext += char
return plaintext
```

Run result:



```
16 def vigenere_cipher_decrypt(ciphertext, key):
22     shift = ord(key[i % key_length].upper()) - ord('A')
23     if char.isupper():
24         plaintext += chr((ord(char) - ord('A') - shift) % 26 + ord('A'))
25     else:
26         plaintext += chr((ord(char) - ord('a') - shift) % 26 + ord('a'))
27     else:
28         plaintext += char
29     return plaintext
30
31
32 plaintext = "Hello"
33 key = "KEY"
34
35 ciphertext = vigenere_cipher_encrypt(plaintext, key)
36 print("加密: ", ciphertext)
37
38 decrypted_text = vigenere_cipher_decrypt(ciphertext, key)
39 print("解密: ", decrypted_text)
```

运行 vigenere x

D:\Software\Anaconda\Anaconda\python.exe D:\Doc\Study\密码学\final\CryptologyFinal\vigenere.py

加密: Rijvs

解密: Hello

进程已结束, 退出代码为 0

## The application scenarios of Harsh function and MAC

```
import hashlib

def encrypt_mac_address(mac_address):
    mac_address = mac_address.lower().replace(':', '')
    hash_object = hashlib.sha256(mac_address.encode())
    encrypted_mac_address = hash_object.hexdigest()
    return encrypted_mac_address
```

Run result:

```
1 import hashlib
2
3
4 1 个用法 新 +
5 def encrypt_mac_address(mac_address):
6     # 将MAC地址转换为小写并去除冒号
7     mac_address = mac_address.lower().replace(':', '')
8
9     # 使用SHA-256哈希算法进行加密
10    hash_object = hashlib.sha256(mac_address.encode())
11    encrypted_mac_address = hash_object.hexdigest()
12
13    return encrypted_mac_address
14
15 mac_address = "00:11:22:33:44:55"
16 encrypted_mac_address = encrypt_mac_address(mac_address)
17 print("original: ", mac_address)
18 print("decrypt: ", encrypted_mac_address)
19
```

运行 hash x

D:\Software\Anaconda\Anaconda\python.exe D:\Doc\Study\密码学\final\CryptologyFinal\hash.py

original: 00:11:22:33:44:55

decrypt: a9b2ad6f4919c2ddcc2e04825227372ce079c0fe392d636d293d9f048a2c7926

进程已结束，退出代码为 0