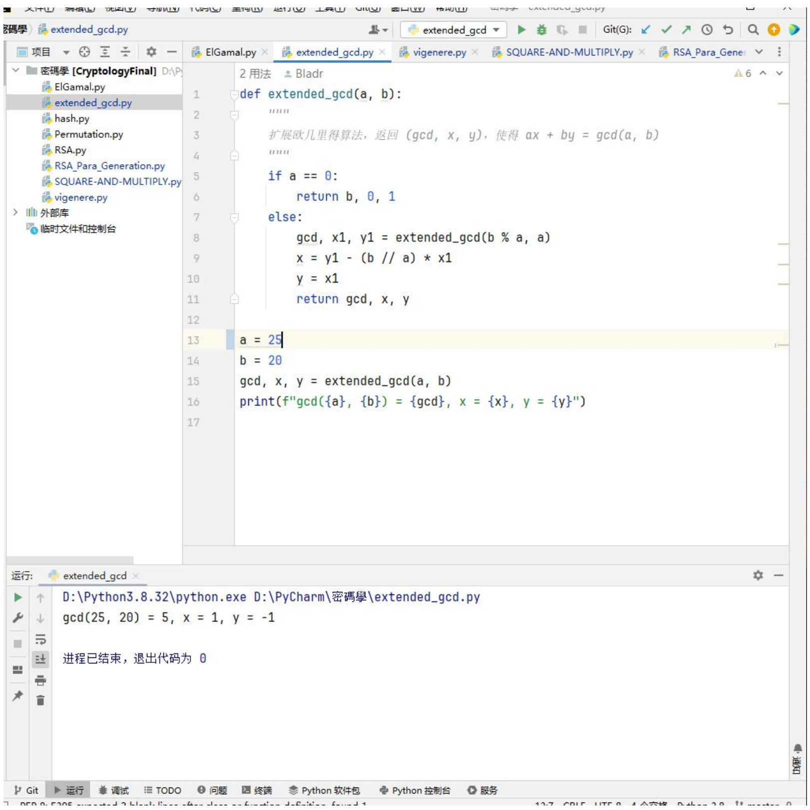
Cryptography and Information Security Final Report

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：



Algorithm 6.4: RSA PARAMETER GENERATION

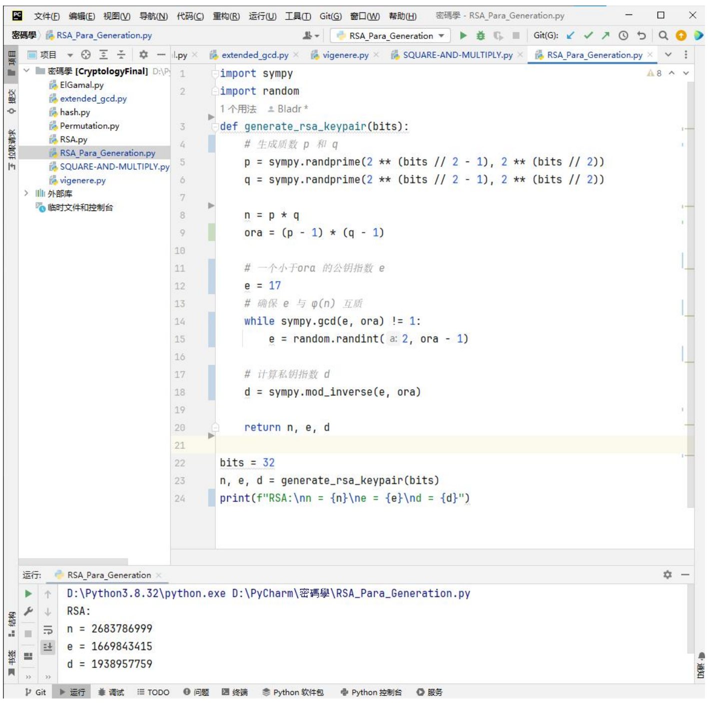
```
import sympy
import random
def generate_rsa_keypair(bits):
    # 生成质数 p 和 q
    p = sympy.randprime(2 ** (bits // 2 - 1), 2 ** (bits // 2))
    q = sympy.randprime(2 ** (bits // 2 - 1), 2 ** (bits // 2))

    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：



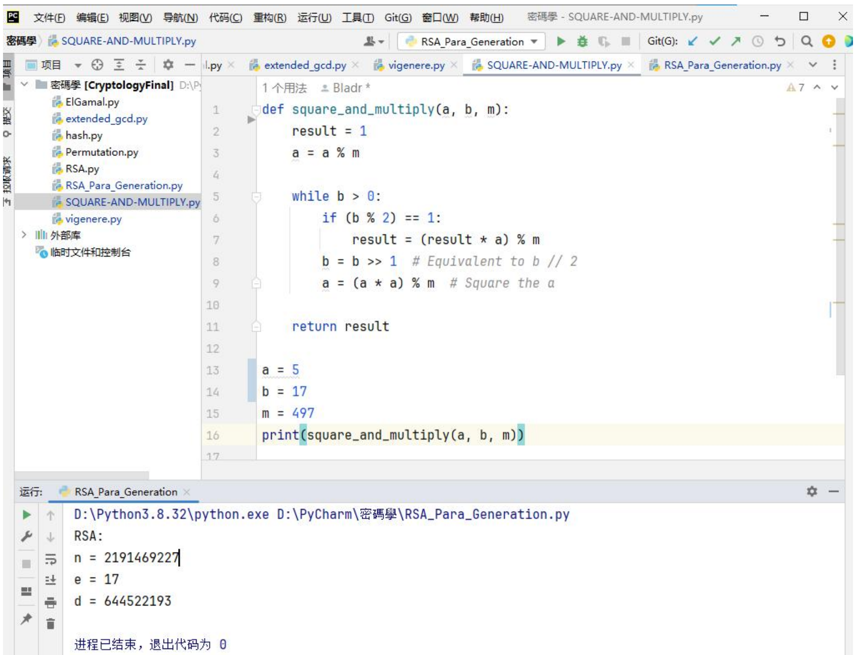
Algorithm6.5: SQUARE-AND-MULTIPLY(x,c,n)

```
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：



Algorithm for Cryptosystem 6.1: RSA Cryptosystem

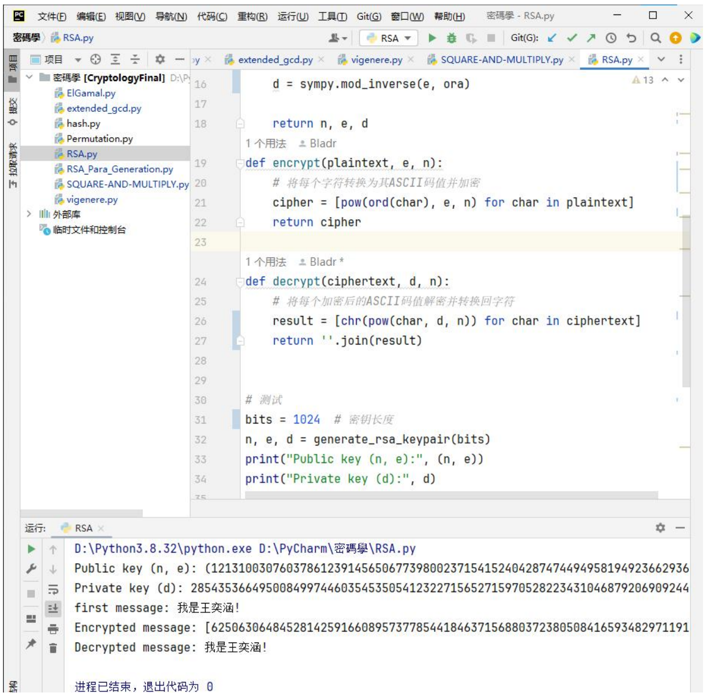
```
import random
import sympy
def generate_rsa_keypair(bits):
    p = sympy.randprime(2 ** (bits // 2 - 1), 2 ** (bits // 2))
    q = sympy.randprime(2 ** (bits // 2 - 1), 2 ** (bits // 2))

    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



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
```

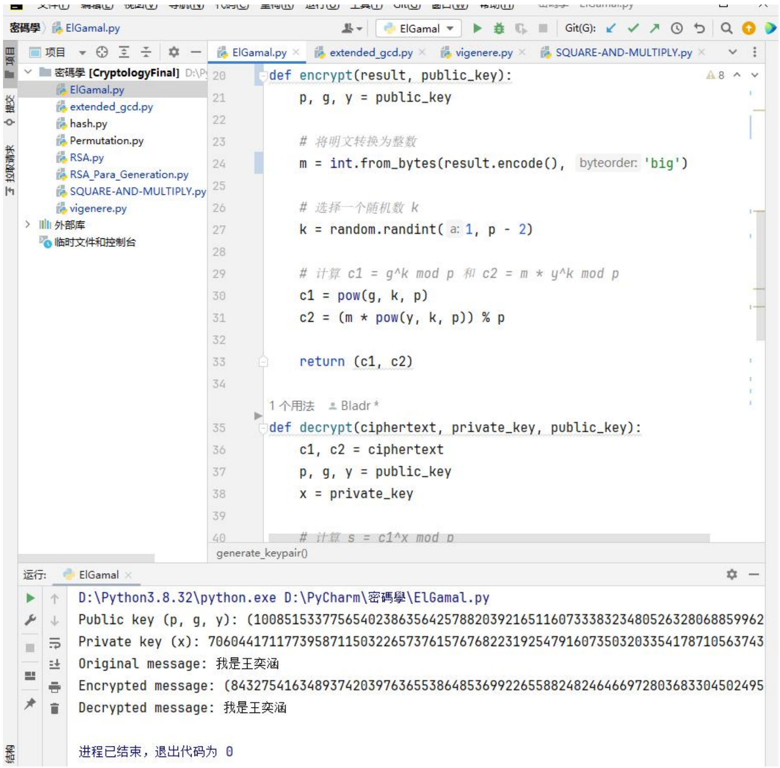
```
    m = int.from_bytes(result.encode(), 'big')
    k = random.randint(1, p - 2)
    c1 = pow(g, k, p)
    c2 = (m * pow(y, k, p)) % p

    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：



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：

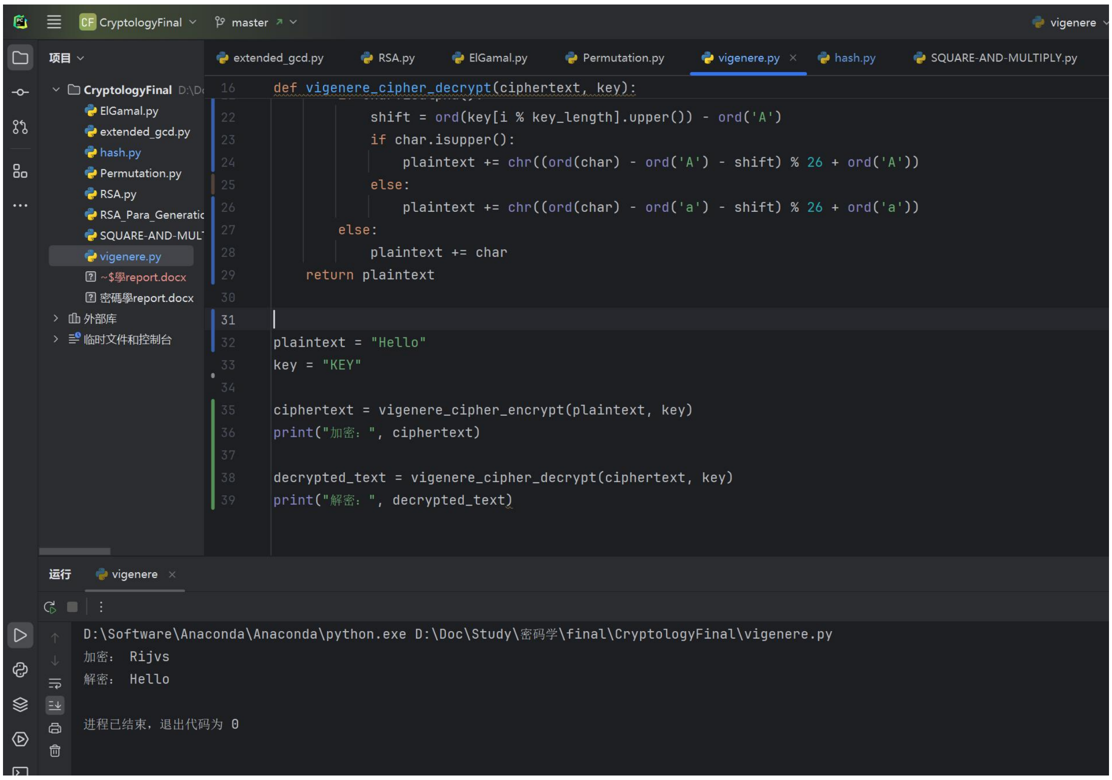
Algoritm for the vigenere cipher

```
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:

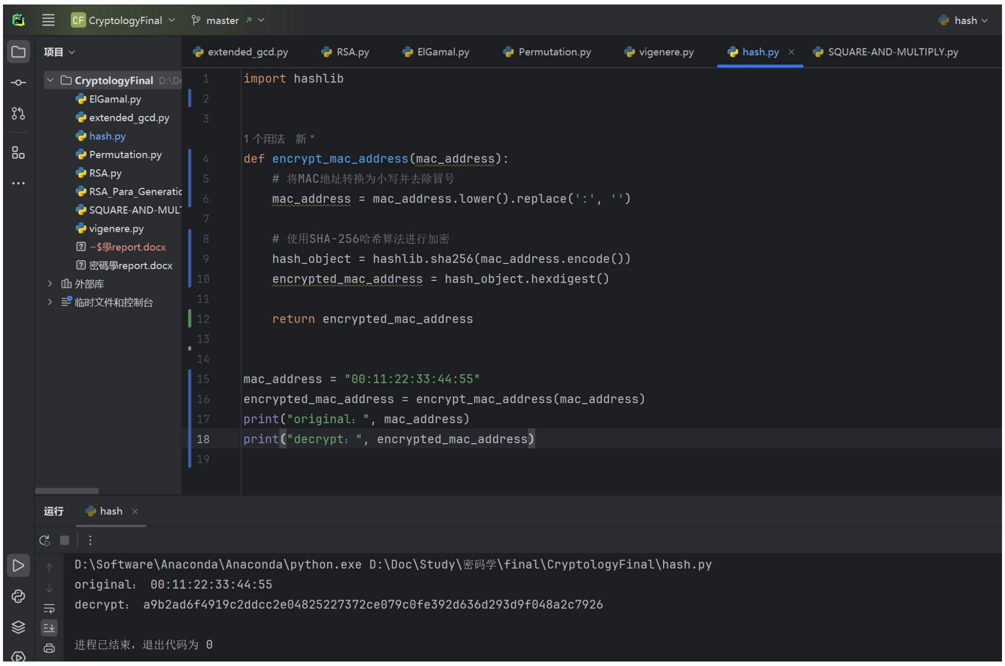


The application scenarios of Harsh function and MAC

```
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:


CryptologyFinal
master
hash
项目
extended_gcd.py
RSA.py
ElGamal.py
Permutation.py
vigenere.py
hash.py
SQUARE-AND-MULTIPLY.py
CryptologyFinal
ElGamal.py
extended_gcd.py
hash.py
Permutation.py
RSA.py
vigenere.py
~$學report.docx
密碼學report.docx
外部库
临时文件和控制台
import hashlib
1 个用法 新 *
def encrypt_mac_address(mac_address):
# 将MAC地址转换为小写并去除冒号
mac_address = mac_address.lower().replace(':', '')
# 使用SHA-256哈希算法进行加密
hash_object = hashlib.sha256(mac_address.encode())
encrypted_mac_address = hash_object.hexdigest()
return encrypted_mac_address
mac_address = "00:11:22:33:44:55"
encrypted_mac_address = encrypt_mac_address(mac_address)
print("original: ", mac_address)
print("decrypt: ", encrypted_mac_address)
运行
hash
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