实 验 报 告

课程名称 信息安全

学　　院 软件学院

专　　业 软件工程

姓　　名 于康

学　　号 20302010040

开 课 时 间 2022 至 2023 学年第 二 学期

实验项目名　　称	**RSA**	成绩	

一、实验目的

1. 实现 RSA 加密
2. 理解分解模数破解 RSA 的过程
3. 理解公共模数攻击

二、实验内容

1.RSA 加密

生成密钥步骤如下：

（1） 选取大素数 p，q；$n = pq$

（2） $\phi(n) = (p-1)(q-1)$

（3） 选取 e，$1 < e < \phi(\mathrm{n})$ 且 $gcd(e, \phi(n)) = 1$

（4） 求出 d，$d \equiv e^{-1}(mod\phi(n))$

（5） $(\mathrm{n}, \mathrm{e}), (\mathrm{n}, \mathrm{d})$

加密解密过程如下：

（1） 加密： $c \equiv \mathrm{m}^{\mathrm{e}}(mod\ \mathrm{n})$

（2） 解密： $\mathrm{m} \equiv \mathrm{c}^{\mathrm{e}}(mod\ \mathrm{n})$

padding：

PKCS#1(v1.5)中规定， 如果使用 PKCS1Padding 进行填充， 当 RSA 的密钥长度是 128B， 则原文数据最多 117B。若原数据不满足长度要求，则需要在加密前进行填充，填充公式为:

EB = 00 || BT || PS || 00 || D:

EB: 填充后的数据

D: 原消息数据

BT: The block type 块类型，取值为 00 or 01（私钥运算时），取值为 02（公钥运算时）

PS: The padding string 填充字符串，长度为 Len(EB) - 3 - Len(D)，最少是 8 字节。

BT=00，PS 为 00

BT=01，PS 为 FF

BT=02，PS 为伪随机生成，非零

在加密过程中，将明文先进行 padding 操作，在进行加密；同样的，在解密过程中，解密得到明文后要进行 unpadding 操作。

2.分解模数破解 RSA

若可以将 n 分解为 p 和 q，当然就可以知道其他所有数值，得到解密所需要的 d。

3.公共模数攻击

利用不同的 e 和相同的 p，q 进行两次加密后，通过两次密文 c 求明文 m。

$$C_1^x * C_2^y = (M^{e_1})^x * (M^{e_2})^y$$

$$= M^{e_1 x} * M^{e_2 y}$$

$$= M^{e_1 x + e_2 y}$$

$$= M^1$$

$$= M$$

三、实验步骤

1.实现 RSA

对于函数 multiplicative_inverse，利用了扩展欧几里得算法来求逆元：

```
def multiplicative_inverse(e, phi):
    '''
    extended Euclid's algorithm for finding the multiplicative inverse
    '''
    # WRITE YOUR CODE HERE!
    # 拓展欧几里得算法
    def extended_euclid(a, b):
        if b == 0:
            return a, 1, 0
        else:
            # ax+by==a*y1+b*(x1-(a/b)*y1)
            # 上一深度的x 等于下一深度的y1，上一深度的y 等于下一深度的x1-(a/b)*y1
            d, x, y = extended_euclid(b, a % b)
            return d, y, x - (a // b) * y

    d, x, _ = extended_euclid(e, phi)
    if d == 1:
        return x % phi
    return None
```

利用能够求得逆元后，可以直接写出函数 key_generation：

```
def key_generation(p, q):
    # WRITE YOUR CODE HERE!
    n = p * q
    phi = (p - 1) * (q - 1)
    e = random.randrange(1, phi)
    d = multiplicative_inverse(e, phi)
    while d is None:
```

```
            e = random.randrange(1, phi)
            d = multiplicative_inverse(e, phi)
        return (n, e), (n, d)
```

根据前面对于 padding 操作的介绍，进行加密前的字节格式应为

```
b"\x00\x02" + padding + b"\x00" + bytes_
```

（其中 bytes_为原数据，padding 为随机产生），而在解密后也应当拆解出原数据，有关 padding 的函数如下所示：

```
    def pkcs1v15_padding(bytes_, size):
        padding_len = size - len(bytes_) - 3
        padding = b""
        while len(padding) < padding_len:
            byte_ = os.urandom(1)
            if byte_ != b'\x00':
                padding += byte_
        return b"\x00\x02" + padding + b"\x00" + bytes_

    def pkcs1v15_unpadding(bytes_):
        if bytes_[0:2] != b"\x00\x02":
            raise ValueError("The format is incorrect")
        padding_index = bytes_.find(b"\x00", 2)
        if padding_index == -1:
            raise ValueError("The format is incorrect")
        return bytes_[padding_index+1:]
```

加密函数输入为需加密的字符串，输出为字节类型加密结果；解密函数输入为字节类型的密文，输出为明文字符串：

```
    def encrypt(pk, plaintext):
        n, e = pk
        plaintext_ = pkcs1v15_padding(plaintext.encode(), (n.bit_length()+7)//8 -
11)
        m = int.from_bytes(plaintext_, 'big')
        c = pow(m, e, n)
        ciphertext = c.to_bytes((n.bit_length()+7)//8, 'big')
        return ciphertext

    def decrypt(sk, ciphertext):
        n, d = sk
        c = int.from_bytes(ciphertext, 'big')
        m = pow(c, d, n)
        plaintext_ = pkcs1v15_unpadding(m.to_bytes((n.bit_length()+7)//8-11,
'big'))
        plaintext = plaintext_.decode()
```

```
        return plaintext
```

2.分解模数破解 RSA

利用 openssl 解析公钥，得到 e，n：

```
root@DESKTOP-M59KV5D:/mnt/c/Users/yuki/Desktop/files/课程/信息安全/lab/lab3# openssl rsa -pubin -text -modulus -in warmup -in pubkey.pem
RSA Public-Key: (256 bit)
Modulus:
    00:c2:63:6a:e5:c3:d8:e4:3f:fb:97:ab:09:02:8f:
    1a:ac:6c:0b:f6:cd:3d:70:eb:ca:28:1b:ff:e9:7f:
    be:30:dd
Exponent: 65537 (0x10001)
Modulus=C2636AE5C3D8E43FFB97AB09028F1AAC6C0BF6CD3D70EBCA281BFFE97FBE30DD
writing RSA key
-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAIhAMJjauXD2OQ/+5erCQKPGqxsC/bNPXDr
yigb/+l/vjDdAgMBAAE=
-----END PUBLIC KEY-----
```

e = 65537

n = 879243482641324068752761405144999371450508936656025929924181716470424916584 61

对大整数进行分解，得到 p 和 q：

p = 27512786035134892817328517438158115229 9

q = 31957631681447894987059016419304804123 9

得到 phi，并进一步求出逆元 d：

phi_n = 879243482641324068752761405144999371444561894884367651143742963084678624649 24

d = 108669487608445991682520826123784959773882712796792315398390496986219949946 73

读取文件 secret.enc 得到密文，利用 bytes2num 函数得到 c 值，解密得到 m 值，再利用 num2str 函数得到明文，注意在 num2str 函数中去除 padding：

```
    for i in range(0, len(tmp), 2):
        if tmp[i:i+2] == '00':
            tmp = tmp[i+2:]
            break
```

3.公共模数攻击

根据上述知识，可知先需要求出 x、y，并对负数情况进行处理；之后就可以直接根据公式求出明文：

```
def attack(c1, c2, e1, e2, n):
    # WRITE YOUR CODE HERE!
    # x*e1 + y*e2 = 1
    _, x, y = gmpy2.gcdext(e1, e2)
    if x < 0:
        x = -x
        c1 = gmpy2.invert(c1, n)
    elif y < 0:
        y = -y
        c2 = gmpy2.invert(c2, n)
    return (pow(c1, x, n) * pow(c2, y, n)) % n
```

四、实验结果及分析

1.实现 RSA

为了验证实现的 RSA 算法的正确性，选取 text 文本，并且选取了两个大素数 p、q，验证经过加密再解密后的结果是否为原文本：

```
text = "helloworld"
print("text: ", text)
p = 3195763168144789498705901641930480412 39
q = 2326847640016985455630670040097558697 17
pk, sk = key_generation(p, q)
plaintext = decrypt(sk, encrypt(pk, text))
print("plaintext: ", plaintext)
if text == plaintext:
    print("Test Pass")
```

运行结果为：

```
PS C:\Users\yuki\Desktop\files\课程\信息安全\lab\lab3> & C:/
Disk_T/python/python.exe c:/Users/yuki/Desktop/files/课程/信
息安全/lab/lab3/20302010040-于康-实验三/20302010040_rsa.py
text:  helloworld
plaintext:  helloworld
Test Pass
```

2.分解模数破解 RSA

最后运行结果为：

```
PS C:\Users\yuki\Desktop\files\课程\信息安全\lab\lab3> & C:/Disk_
T/python/python.exe c:/Users/yuki/Desktop/files/课程/信息安全/lab
/lab3/20302010040-于康-实验三/20302010040_factoring.py
ManyQuestionMarks???
```

得到明文为：ManyQuestionMarks???

3.公共模数攻击

最后运行结果为：

```
PS C:\Users\yuki\Desktop\files\课程\信息安全\lab\lab3> & C:/Disk_
T/python/python.exe c:/Users/yuki/Desktop/files/课程/信息安全/lab
/lab3/20302010040-于康-实验三/20302010040_common_modulus.py
[+] Started attack...
[+] Attack finished!

Plaintext:
 Common Modulus Attack
```

得到明文为：Common Modulus Attack

五、实验总结

该实验进行了 RSA 加密算法的实现，同时也实现了 RSA 加密的两种破解方法。

在实现过程中，可以明显感受到 RSA 加密时的 n 值非常大，虽然分解模数的攻击手段看似操作简单，但施行难度实际上是非常巨大的，RSA 的安全性就依赖于大数的因子分解；而公共模数攻击也给出了特定情况下破解 RSA 加密的实现。