实 验 报 告

课程名称	密码学基础
学　　院	计算机科学技术学院
专　　业	信息安全
姓　　名	冉津豪
学　　号	17307130179

开 课 时 间 **2019** 至 **2020** 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

- RSA 产生公私钥对

  - 1.随机选择两个不相等的质数 p 和 q。
  - 2.计算 p 和 q 的乘积 n。
  - 3.计算 n 的欧拉函数 φ(n)。称作 L，根据公式 φ(n) = (p-1)(q-1)
  - 4.随机选择一个整数 e，也就是公钥当中用来加密的那个数字，条件是 1< e < φ(n)，且 e 与 φ(n) 互质。
  - 5.计算 e 对于 φ(n)的模反元素 d。也就是密钥当中用来解密的那个数字。所谓"模反元素"就是指有一个整数 d，可以使得 ed 被 φ(n)除的余数为 1。ed ≡ 1 (mod φ(n))
  - 6.将 n 和 e 封装成公钥，n 和 d 封装成私钥。

- RSA 加解密

  首先对明文进行比特串分组，使得每个分组对应的十进制数小于 n，然后依次对每个分组 m 做一次加密，所有分组的密文构成的序列就是原始消息的加密结果，即 m 满足 0<=m<n，加密算法为：c≡ m^e mod n; c 为密文，且 0<=c<n。

2. 分解模数破解 RSA

   如果模数 n 可以被分解成 p 和 q，那么就可以通过 e 与密钥生成相同的过程对密文进行解密。

3. 公共模数攻击

   当使用公共的模数 n,不同的私钥 e1, e2 对同一密文进行加密时,如果能截获密文 c1, c2 那么可能可以直接解密。

   若 gcd(e1, e2) = 1, 即 e1 e2 互素时，由扩展欧几里德算法可知：必然存在整数 s1, s2。使得下式①成立：e1s1 + e2s2 = 1, e1, e2 都为正数那么，s1, s2 是一正一负的，这里可以假设 s1 是负数。若记明文为 m, e1,e2 共用模数加密后的密文为 c1, c2。即 c1 = m^e1, c2 = m^e2 那么可以根据①构造下式(运算皆 mod

n)：c1^s1 * c2^s2 = (m^e1)^s1 * (m^e2)^s2 = m^(e1s1 + e2s2) = m^1 = m

三、实验步骤

1. 实现 RSA 算法

- multiplicative_inverse：

  通过 e 和 φ(n)，计算 d 使得 $ed \equiv 1 \pmod{\phi(n)}$。`ext_gcd(a, b)`利用扩展欧几里得算法求最大公约数的过程，返回值 x，y 分别表示 ax + by = 1 的解。

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

def multiplicative_inverse(e, phi):
    '''
    extended Euclid's algorithm for finding the multiplicative
inverse
    '''
    # WRITE YOUR CODE HERE!
    x, y = ext_gcd(phi, e)
    if y < 0:
        return y + phi
    return y
```

- key_generation

  用两个质数 p,q 产生密钥。其中，e 为随机选择一个小于并与 φ(n) 互质的正整数。有了 e，n，再用 multiplicative_inverse 方法求得 d。

```
def selectE(euler_totient):
    while True:
        e = random.randint(0, euler_totient)
        if math.gcd(e, euler_totient) == 1:
            return e

def key_generation(p, q):
    # WRITE YOUR CODE HERE!
    n = p * q
    phi = (p - 1) * (q - 1)
    e = selectE(phi)
    d = multiplicative_inverse(e, phi)
    return n, e, d
```

- Encrypt

  快速幂取模算法完成 plaintext ^ e (mod n)。

```
def encrypt(pk, plaintext):
    # WRITE YOUR CODE HERE!
    n, e = pk
    res = 1
    tmp = plaintext
    while e:
        if e & 0x1:
            res = res * tmp % n
        tmp = tmp * tmp % n
        e >>= 1
    return res
```

- Decrypt

  快速幂取模算法完成 ciphertext ^ d (mod n)。

```
def decrypt(sk, ciphertext):
    # WRITE YOUR CODE HERE!
    n, d = sk
    res = 1
    tmp = ciphertext
    while d:
        if d & 0x1:
            res = res * tmp % n
        tmp = tmp * tmp % n
        d >>= 1
    return res
```

2. 分解模数破解 RSA

由公钥文件 pubkey.pem 获公钥参数 n, e。在 http://www.factordb.com/index.php 分解 n 得到因子 p 和 q，即得到 $\phi(n)$ 再由 e，$\phi(n)$ 得到私钥参数 d。同时，我们在这里验证了我们的 RSA 加解密过程，及 multiplicative_inverse 的正确性，解密密文可得：

```
解密正确!
加密正确!
□(Ev□FQQv□ ManyQuestionMarks???
```

```
if __name__ == '__main__':
    n = "0xC2636AE5C3D8E43FFB97AB09028F1AAC6C0BF6CD3D70EBCA281BFFE97FBE30DD"
    n = int(n, 16)
    e = 65537
    p, q = 275127860351348928173285174381581152299, 319576316814478949870590164193048041239
    phi_n = (p - 1) * (q - 1)
```

```
        d = multiplicative_inverse(e, phi_n)
        fi = open('secret.enc', 'rb')
        cipher = fi.read()
        cipher = bytes2num(cipher)
        fi.close()
        std_plaintext = pow(cipher, d, n)
        plaintext = decrypt((n, d), cipher)
        if std_plaintext == plaintext:
            print("解密正确！")
        cipher2 = encrypt((n, e), plaintext)
        if cipher == cipher2:
            print("加密正确！")
        print(num2str(plaintext))
```

3. 公共模数攻击

根据模数攻击原理，利用扩展欧几里得算法完成攻击。

```
[+] Started attack...
[+] Attack finished!

Plaintext:
 When does school start
```

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

def modinv(a, m):
    x, y = ext_gcd(a, m)
    return x % m

def attack(c1, c2, e1, e2, n):
    # WRITE YOUR CODE HERE!
    s1, s2 = ext_gcd(e1, e2)
    if s1 < 0:
        s1 = - s1
        c1 = modinv(c1, n)
    elif s2 < 0:
        s2 = - s2
        c2 = modinv(c2, n)
    m = (pow(c1, s1, n) * pow(c2, s2, n)) % n
    return m

```

```
25      if __name__ == '__main__':
26          n =
        10310906590233462022610116200879396350425602793911702009187679
        90396908019447356042590186555348601832050310690832542902585772
        91605287053538752280231959857465853228851714786887294961873006
        23415307918721628551682383210242411093406295427234611190757139
        39643636300793435985116020133166046419048520189691789190516 27
27          ct1 =
        89165267757592056998430515273393236824675618209628120636976415
        32454160877763931290632876189619633556391899581306544609711151
        12996225835595212794161183442626595878734974943260495650772083
        91797840846364711580568322182209616487454086620832166912794622
        82635752916518512231065855388817562960448654426593986065256 8
28          e1 = 15
29          ct2 =
        20135243573102312365489445555179471711050237357771531944565197
        20790905685679145266007153398135181361540604268682866688360274
        35656926804662251288789077763396384988888366429201228092846902
        43270675115097201512434337054133554094147631311406424433602757
        21562509284691369379007842445862436922857577601674353294693 4
30          e2 = 13
31
32          print('[+] Started attack...')
33          message = attack(ct1, ct2, e1, e2, n)
34          print('[+] Attack finished!')
35          print('\nPlaintext: \n ' +
        bytearray.fromhex(hex(message)[2:]).decode())
```

四、实验结果

分解模数方法解密结果为：□□v□FQQv□ ManyQuestionMarks???

公共模数攻击结果为：When does school start

根据输出结果，我们可以判断过程无误。

五、实验总结

分解模数方法虽然简单粗暴，但计算时间过长。当我们使用分解模数攻击的方法再对第三部分的 n 进行尝试时，因为计算时间过长，长时间无法得到响应。

公共模数攻击需要得到相同的模数 n，相同的消息 m，不同 e1，e2 加密的密文 c1, c2，条件相比于分解模数方法更加苛刻，且仅仅只能解决这个特定的 m，有较大局限性。但对于分解模数方法束手无策的长密钥，仍然完成了攻击。

对于这两种攻击方法，将 RSA 的密钥模数的长度提高，随机化明文 m 便能极大地完成防御。总体而言，RSA 的安全性是能够得到保障的。