Github（或者 Coding）账号： https://github.com/Wkingxc/cryptography-rsa-homework

个人博客关于密码学大作业的链接:
https://wkingxc.com/2023/01/15/RSA%E5%A4%A7%E7%A4%BC%E5%8C%85/

题目（中文）：RSA 大礼包

摘要（中文）：

RSA 密码算法是使用最为广泛的公钥密码体制。该体制简单且易于实现，只需要选择 5 个参数即可（两个素数p和q、模数$N=pq$ 、加密指数e和解密指数d。设m为待加密消息 RSA 体制破译相当于已知m^e mod N能否还原m的数论问题。目前模数规模为 1024 比特的 RSA 算法一般情况下是安全的，但是如果参数选取不当，同样存在被破译的可能。

有人制作了一个 RSA 加解密软件采用的 RSA 体制的参数特点描述见密码背景部分）。已知该软件发送某个明文的所有参数和加密过程的全部数据（加密案例文件详见附件 3-1。Alice 使用该软件发送了一个通关密语，且所有加密数据已经被截获，请问能否仅从加密数据恢复该通关密语及RSA体制参数？如能请给出原文和参数，如不能请给出已恢复部分并说明剩余部分不能恢复的理由。

题目描述（清楚描述题目中文，写出自己的理解，请勿复制原题目）

共有 **20** 个截获的 **RSA** 加密的块，运用所学知识，查询资料，运用常见的对 **RSA** 加密的攻击方式来对这些 **Frame** 进行攻击，尝试复原出加密的信息。

过程（包括背景，原理：必要的公式，图表；步骤，如有必要画出流程图，给出主要实现步骤代码）

一、 密文结构解析&分析

1.遍历所有的模数 N，判断是否存在模数相同的加密片段，如果猜测可以用共模攻击

2.遍历寻找任意两个模数 N 的公因子，如果得到不为 1 的公因子则可以因数碰撞

3.遍历所有加密指数 e，寻找低加密指数及对应的加密对，可以用低指数广播攻击

```
e[0]=4678646536268633491726599684377984323360699258542497648174505533546867869794877498845030561212796792653392326826041255700012515356962234035324609604060428488350558733782932294963363760918079744775451399203901890478653711508788800552854790064033927005262891544078735727134541681831380844812709888576701574
```

```
8889
e[1]=e[2]=e[5]=e[6]=e[9]=e[10]=e[13]=e[14]=e[17]=e[18]=e[19]=65537
e[2]=65537
e[3]=e[8]=e[12]=e[16]=e[20]=5
e[4]=152206992575706893484835984472544529509325440944131662631741403414037956695665533186650071476146389737020554215956181827422540843366433981607643940546405002217220286072880967331118344806315756304650248634546597784597963886656422706197757265316981889118026978865295597135470735576032282694348773714479076093197
e[7]=e[11]=e[15]=3
```

对上述的公钥 e 进行分析,主要有以下几种，其中 e 比较小的有 3 和 5，其中

Frame7，Frame11，Frame15 采用低加密指数 e=3 进行加密

Frame3，Frame8，Frame12，Frame16 和 Frame20 均采用低加密指数 e=5 进行加密

采用费马分解尝试 p,q 差距比较小的帧和 Pollard p-1 分解进行尝试 p,q 差距比较大的帧

Frame10 可以用费马分解法破解

Frame2，6，19 可以用 p-1 分解法破解

二、 共模攻击

公共模数攻击

n[0]==n[4],还因为可能存在重复发包

设两个用户的公钥分别为 e_1 和 e_2，且两者互质。明文消息为 m，密文分别为:

$$c_1 = m^{e_1} \bmod n$$
$$c_2 = m^{e_2} \bmod n$$

当攻击者截获 c_1 和 c_2 后，就可以恢复出明文。用扩展欧几里得算法求出 $re_1 + se_2 = 1 \bmod n$ 的两个整数 r 和 s，由此可得:

$$\begin{aligned} c_1^r c_2^s &\equiv m^{re_1} m^{se_2} \bmod n \\ &\equiv m^{(re_1+se_2)} \bmod n \\ &\equiv m \bmod n \end{aligned}$$

RESUILT:{'Frame0': b'My secre', 'Frame4': b'My secre'}

三、 公因数碰撞攻击

$$\gcd(n_1, n_2) = p$$

这样很快就能将他们各自的私钥求解出来了。
代码过于简单, 直接 `gcd` 两个因数就行, 结果不为1就说明有相同因数。

```
#公因数碰撞攻击
def same_factor_attack():
    p_of_prime=gmpy2.gcd(n[1],n[18])
    q1=n[1]//p_of_prime
    q18=n[18]//p_of_prime
    phi1=(p_of_prime-1)*(q1-1)#计算欧拉函数
    phi18=(p_of_prime-1)*(q18-1)#计算欧拉函数
    d1=gmpy2.invert(e[1],phi1)#计算私钥
    d18=gmpy2.invert(e[18],phi18)#计算私钥
    m1=pow(c[1],d1,n[1])#解密
    m18=pow(c[18],d18,n[18])#解密
    m1=hex(m1)[2:]#去掉 0x
    m18=hex(m18)[2:]#去掉 0x
    m1=binascii.unhexlify(m1)[-8:]#hex->str
    m18=binascii.unhexlify(m18)[-8:]#hex->str
    sloved['Frame1']=m1
    sloved['Frame18']=m18
```

'Frame1': b'. Imagin', 'Frame18': b'm A to B'

四、 低指数广播攻击

Frame3、Frame8、Frame12、Frame16、Frame20采用该种攻击方法

低加密指数广播攻击适合于n很大,e很小的情况, 其适用的情况是如n下的, 当一条消息m发送给不同的接收者时, 每个接收者的 n 都不同, 但是加密公钥都是相同的, 我们假设公钥为3那么就有

$$\begin{cases} C_1 = m^3 \bmod n_1 \\ C_2 = m^3 \bmod n_2 \\ C_3 = m^3 \bmod n_3 \end{cases}$$

由中国剩余定理知道,我们是可以将 m^3 算出来的, 那么对其开立方就将明文给求出来了。

```
#中国剩余定理
def chinese_remainder_theorem(backup):
    #计算 N 的乘积
    N=1
    for a,n in backup:
        N*=n
    #计算 Ni
    Ni=[]
    for a,n in backup:
        Ni.append(N//n)
    #计算 Ni 的模逆元
```

```
    Ni_inverse=[]
    for i in range(0,len(Ni)):
        Ni_inverse.append(gmpy2.invert(Ni[i],backup[i][1]))
    #计算 x
    x=0
    for i in range(0,len(Ni)):
        x+=backup[i][0]*Ni[i]*Ni_inverse[i]
    x=x%N
    return x,N

#低指数 3
def low_exponent_attack3():
    frame_range=[7,11,15]
    backup=[]
    for i in frame_range:
        backup.append([c[i],n[i]])
    x,N=chinese_remainder_theorem(backup)
    #开三次方根
    m=gmpy2.iroot(x,3)[0]
    m=hex(m)[2:]#去掉 0x
    m=binascii.unhexlify(m)[-8:]#hex->str
    sloved['Frame7']=m
    sloved['Frame11']=m
    sloved['Frame15']=m
#低指数 5
def low_exponent_attack5():
    frame_range=[3,8,12,16,20]
    backup=[]
    for i in frame_range:
        backup.append([c[i],n[i]])
    x,N=chinese_remainder_theorem(backup)
    #开五次方根
    m=gmpy2.iroot(x,5)[0]
    m=hex(m)[2:]#去掉 0x
    m=binascii.unhexlify(m)[-8:]#hex->str
    sloved['Frame3']=m
    sloved['Frame8']=m
    sloved['Frame12']=m
    sloved['Frame16']=m
    sloved['Frame20']=m
```

'Frame7': b'\xb8\xbc\xa2S)s\xcd\xd2', 'Frame11': b'\xb8\xbc\xa2S)s\xcd\xd2', 'Frame15': b'\xb8\xbc\xa2S)s\xcd\xd2', 'Frame3': b't is a f', 'Frame8': b't is a f', 'Frame12': b't is a f', 'Frame16': b't is a f', 'Frame20': b't is a f'

可以看到 Frame7，Frame11，Frame15 采用低加密指数广播攻击出来是乱码，说明该种方法不正确

Frame3、Frame8、Frame12、Frame16、Frame20 可以采用该种攻击方法

五、 Pollard p-1 分解法

Frame2、Frame6、Frame19均采用该种攻击方法

如果 p 与 q 都不超过 10^{20} 次方, 若其中一个 $(p-1)$ 或 $(q-1)$ 的因子都很小的时候（适用于p-1或q-1能够被小素数整除的情况,在这里为了方便说明我们假设为 $(p-1)$）,可以如下操作:
选取一个整数 k, 使其满足 $(p-1) \mid k!$，由费马小定理知道, a 与 p 互素 的时候有

$$a^{p-1} = 1 \bmod p$$

所以 $a^{k!} = 1 \bmod p$ ， 即 p | $(a^{k!}-1)$ 。那么对于 n 与 $(a^{k!}-1)$ 必有公因数为 p, 这样就可以把 n 分解出来了。但是对于 k 的选取还是有要求的，太小了 $(p-1) \mid k!$不会成立，太大了花费时间会很多。

> Pollard p-1 Factoring Algorithm (n, B) :
> `$a \leftarrow 2^{B !}(\bmod n)$`
> $d \leftarrow \gcd(a-1, n)$
> if $1 < d < n$ then return d else return failure

```
#Pollard's p-1 算法
def pollard_p_1(n):
    b=2**20
    a=2
    # while True:
    #       a=gmpy2.powmod(a,b,n)
    #       d=gmpy2.gcd(a-1,n)
    #       if d!=1 and d!=n:
    #             return d
    #       a+=1
    for i in range(2,b+1):
        a=gmpy2.powmod(a,i,n)
        d=gmpy2.gcd(a-1,n)
        if d!=1 and d!=n:
            q=n//d
            n=q*d
    return d
def pollard_data(n):
    frame_range=[2,6,19]
    for i in frame_range:
        temp_n=n[i]
        temp_c=c[i]
        temp_e=e[i]
        p=pollard_p_1(temp_n)
        q=temp_n//p
        phi=(p-1)*(q-1)
        d=gmpy2.invert(temp_e,phi)
        m=pow(temp_c,d,temp_n)
        m=hex(m)[2:]#去掉 0x
        m=binascii.unhexlify(m)[-8:]#hex->str
        sloved['Frame'+str(i)]=m
```

'Frame2': b' That is', 'Frame6': b' "Logic ', 'Frame19': b'instein.'

六、 费马分解法

费马分解法

如果 p 和 q 相差不大的话我们可以通过费马分解把 p 和 q 求出来, 原理如 下

$$\mathrm{n}=\mathrm{p}*\mathrm{q}=\frac{1}{4}(p+q)^2-\frac{1}{4}(p-q)^2$$

由于 p 与 q 相差不大, 所以 $\mathrm{p}-\mathrm{q}$ 相对于 n 和 $(p+q)^2$ 来说可以忽略不计, 所以有～

$$2\sqrt{\mathrm{n}}\approx\mathrm{p}+\mathrm{q}$$

也就是说通过不断尝试就可以把 p 和 q 给计算出来了

p、q比较接近时，可以使用这种攻击方法

```
# 费马分解
def fermat_factorization(n):
    a=gmpy2.iroot(n,2)[0]+1
    max=200000
    for i in range(0,max):
        b2=a*a-n
        b=gmpy2.iroot(b2,2)[0]
        if gmpy2.is_square(b2):
            p=a-b
            q=a+b
            return p,q
        a+=1
def fermat_data():
    frame_range=[10]
    for i in frame_range:
        p,q=fermat_factorization(n[i])
        phi=(p-1)*(q-1)#计算欧拉函数
        d=gmpy2.invert(e[i],phi)#计算私钥
        m=pow(c[i],d,n[i])#解密
        m=hex(m)[2:]#去掉 0x
        m=binascii.unhexlify(m)[-8:]#hex->str
        sloved['Frame'+str(i)]=m
```

经检验,Frame10 可以采用该种攻击方法

'Frame10': b'will get'

七、 实验结果

最后根据解出来的零碎信息，结合搜索引擎得到明文：

My secret is a famous saying of Albert Einstein. That is "Logic will get you from A to B. Imagination will take you everywhere."

总结（完成心得与其它，主要自己碰到的问题和解决问题的方法）

基本了解过 RSA 攻击方面的知识，但对代码编写十分陌生。整个实验过程都是参考互联网上大佬的博客和同学们的指导。

解密的过程是有趣的，了解其背后的原理是困难的，从中能够感受到密码学大牛们的强大。在此实验中，我对 python 语言的使用更加熟练了，同时我对 RSA 密码学的安全性有了基本的认识，并理解如何保护 RSA 加密算法免受攻击。同时，也可以发现加密算法并不是绝对安全的，随着技术的发展和攻击方式的变化，加密算法需要不断地进行更新和改进，以保障数据的安全。

参考文献（包括参考的书籍，论文，URL 等，很重要）

[2016 全国高校密码数学挑战赛-赛题三]
https://www.tr0y.wang/2017/10/31/RSA2016/

[现代密码学大作业]
https://blog.csdn.net/m0_63571390/article/details/122375466?spm=1001.2014.3001.5501

[关于 RSA 的几种攻击手段]
https://blog.csdn.net/pigeon_1/article/details/114371456

[Pollard's rho 算法]
https://blog.csdn.net/qq_39972971/article/details/82346390

[分解因子算法——Pollard rho 算法]
https://blog.csdn.net/weixin_46395886/article/details/115073059