Github 账号： https://github.com/Juli-Eyre

个人博客关于密码学大作业的链接：https://github.com/Juli-Eyre/crypto_experiment

题目（中文）：RSA 大礼包

摘要（中文）：

简要描述要求，目的等

通过对 Frame0 到 Frame20 进行了细致的观察与分析，采用 Fermat 分解法和 Pollard p-1 分解法成功分解了 Frame2、Frame6、Frame10 和 Frame19 的模数并由此 得到了正确的明文消息；使用公共模数攻击法和低加密指数攻击法找到了存在某些安全缺陷的消息片段，成功破译了 Frame0、Frame3、Frame4、Frame8、Frame12、 Frame16 和 Frame20 的明文消息；采用因数碰撞法，用欧几里德算法遍历所有模数，求出 Frame1 和 Frame18 的模数的公因数，进而成功分解了 Frame1 和 Frame18 的模数得到正确的加解密参数；采用 coppersmith 攻击(已知 m 的高位)，破译了 Frame7、Frame11、Frame15，得到明文消息；利用已经得到的若干明文片段，通过语义分析等方法，采用猜测明文攻击，得到了其余分片的所有明文，成功得到了明文。利用已得到的几个素数参数，找到了随机数生成的规律。

题目描述（清楚描述题目中文，写出自己的理解）

已知帧数据的数据格式如下，其中数据都是 16 进制表示，结构如下：1024bit 模数 N | 1024bit 加密指数 e | 1024bit 密文 c。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？

过程（包括背景，原理：必要的公式，图表；步骤，主要代码）

1.攻击原理：

（1）低指数加密攻击

若有几组密文是由同一明文、同一加密密钥加密得到，则 利用中国剩余定理，然后开 e 次方即可还原明文

（2）共模攻击

若有几组密文在加密时使用了相同的模数，则可用扩展欧几里德算法还原明文。

由$gcd(e_1, e_2) = 1$,得

$$e_1 * x + e_2 * y = 1$$

$$c_1^x c_2^y \equiv \left(m_1^{e_1}\right)^x \left(m_2^{e_2}\right)^y \equiv (m^{e_1})^x (m^{e_2})^y \equiv m(modN)$$

（3）因数碰撞法（模不互素攻击）

若 p 或 q 在不同组的加密中出现多次，则生成的不同模数 n 可能有相同的因子。这里，以 p 同，q 不同为例：

$$\begin{cases} n_1 = p * q_1 \\ n_2 = p * q_2 \end{cases}$$

再利用$gcd(n_1, n_2) = p$即可还原明文。

（4）费马分解法

若 p 和 q 相差不大：$n = p * q = \frac{1}{4}(p+q)^2 - \frac{1}{4}(p-q)^2$

$$= x^2 - y^2 = (x-y)(x+y)$$

由 p 和 q 相差不大，忽略 p-q，可以从$x = \sqrt{n}$遍历，计算$x^2 - n$，当其为完全平方数时，即可求出x, y，$p = x + y, q = x - y$

（5）Pollard p-1 分解法

设大整数的一个因子是 p, p-1 的所有素因子都不大于 B，合适的选取 B 使得$(p-1)|B!$，则由欧拉定理知道：

$$g^{B!} \equiv 1(modp),$$

$$p \mid (g^{B!} modn) - 1$$

$gcd(g^{B!} - 1, n)$就是 n 的一个因子

（6）coppersmith 攻击(已知 m 的高位)

设明文 m 的高位为m_0，当 e 比较小时，有：

$$c \equiv (m_0 + x)^e (modn)$$

设多项式$f(x)=(m+x)^e-c$，有$f(x)=k*n,(k=0,1,2\ldots)$，遍历得到x后，依据coppersmith 定理，可以求出剩下的所有明文部分

（7）coppersmith 攻击 (Broadcast Attack)

步骤如下：

- 计算所有 n 的乘积 N
- 用中国剩余定理计算 T_i
- 建立在模 N 下的多项式环
- 计算 $g_i=(am+b)^e-c$，这里就是将 m 线性填充
- 计算 $g=\sum T_i*g_i$
- 将多项式的最高次项系数化简为 1
- 寻找 g 的 small root

（8）对随机数发生器的攻击

由题目中提到“素数 p 由某一随机数发生器生成”和“素数 q 可以随机选择，也可以由 2)中的随机数发生器产生”知道存在素数是由同一个随机数发生器生成的，尝试对随机数发生器的结构进行预测。

由已分解得到的 Frame1、Frame2、Frame6、Frame10、Frame18 和 Frame19 这六个分片中的 12 个素数，对这 12 个素数序列中进行统计，如 0-1 分布特征、游程分布特征、移位相加特征等，观察分析其共性特点，可以得到以下结论：

A. 每对素数中至少有一个由同一个素数生成器生成的素数
B. 原素数在二进制表示下呈现出以 16 或 16 的倍数为周期的特征
C. 将素数序列按每 16bit 为一组进行拆分，纵向观察相同比特位的分布规律：第 0 位比特每 2 组为一循环周期，第 1 位比特每 4 组为一循环周期，第 2 位比特每 8 组为一循环周期，第 3 位比特每 16 组为一循环周期，第 4 位比特每 32 组为一循环周期

参考 lcg 线性同余随机数生成器，和经典的 RC4 算法和 BBS 算法，将递推关系式中每次得到的随机序列串联得到更长的随机序列；在各种数字特征中，发现伪随机序列的结构即相似与 移位寄存器序列的前后递推关系，即序列中可由前序列计算出后续序列（或由后 向前），同时又符合同余发生器的四则运算特性，通过解同余方程，可以得到该随机数发生器的递推函数：

$$X_n = 365X_{n-1} - 1 \bmod 2^{16}$$

可以用已知的素数验证递推函数的准确性

2. 关键代码

（1）低指数加密攻击

```
def small_e_attack(nlist , clist,e=5 ):
    m = chinese_remainder_theorem(nlist , clist)
    tmp = iroot(m , e)
    if tmp[1] == 1:
        return tmp[0]
    return 0
```

（2）共模攻击

```
def same_module_attack(N , e1 , e2 , c1 , c2):
    ##e1*x+e2*y=1
    x = invert(e1 , e2)
    y = (x * e1 - 1) // e2
    true_c2 = invert(c2 , N)
    return (powmod(c1 , x , N) * powmod(true_c2 ,y , N)) % N
```

（3）费马分解法

```
def fermat(n):
    x = iroot(n, 2)[0]
    for _ in range(20000):
        x += 1
        if iroot(x ** 2 - n, 2)[1] == 1:
            y = iroot(x ** 2 - n, 2)[0]
            p = (x + y)
            q = (x - y)
            return p,q
```

（4）pollard p-1 法：

```
def Pollard_p_1(N):
    a = 2
```

```
    g = a
    ##当 n 太大时，就 变化 g，降低复杂度
    while 1:
        for n in range(1,200000):
            g = powmod(g, n, N)
            if is_prime(n):
                d = gcd(g-1, N)
                if 1 < d < N:
                    return d, N//d
                elif d >= N:
                    g = next_prime(a)
                    break
        else:
            break
```

（5）coppersmith 攻击(已知 m 的高位)

为 sage 脚本，可借助本地 sage 环境或 sage 在线运行

```
for m in m_lst:
    nbits = 512
    kbits = 64
    mbar = m&(pow(2,nbits)-pow(2,kbits))
    PR.<x> = PolynomialRing(Zmod(n))
    roots = f.small_roots()
    if roots:
        print(bytes.fromhex(hex(mbar+roots[0])[2:]))
        break
```

（6）已知 p q 求 m

```
def RSA_std(p , q , e , c):
    phi = (p-1)*(q-1)
    d = invert(e , phi)
    m = powmod(c , d , p*q)
    return m
```

（7）按照格式转为字符串

```
def get_plain(mm):
```

```
    tmp = hex(mm)[2:]
    number = int(tmp[16:24],16)
    plain = long_to_bytes(int(tmp[-16:] , 16))
    m[number] = plain
    return 1
```

（8）随机数发生器

```
def crack_PRG(n):
    a = 365
    b = -1
    m = pow(2,16)

    for j in range(m):#种子
        x = j
        p = bin(x)[2:].zfill(16)
        for _ in range(63):
            y = (a*x+b)%m
            x = y
            p += bin(x)[2:].zfill(16)
            if n%int(p,2) == 0:
                return int(p,2)
```

（9）william p+1 分解没有跑出来结果

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）

关于 RSA 密码体系中各参数的选择：

1. 在选择加密指数时，应尽量选择 16bit 以上的数；另外，当明文信息很短时，可以使用独立随机值填充的方法，降低明文信息的相关性，使得攻击者无法用低加密指数攻击法破译算法系统
2. 将同一个明文信息发送给不同的人时，尽量不要选取相同的模数 N
3. 明文空间要足够大，避免猜测明文攻击
4. p q的选择：

p、q 的长度不宜相差过大，防止被试除法轻易破解；p、q 在数值上的差距应当尽可能大，防止被费马分解法轻易破解；p-1 和 q-1 要有大的素因数，防止被 Pollard p-1 分解法轻易破解；选择 p 使得 p 和(p-1)/2 均为素数，防止消息加密后没有被隐藏；不要用同一个素数 p（q）生成不同的模数，防止被因数碰撞法轻易破解

5. 选取随机数时，要使用安全的随机数发生器，不能使用线性同余 glibc 等不是为密码学设计的弱伪随机数发生器。

其他方面：

1. 看懂题目很重要
2. 知道攻击原理，数学知识很重要， coppersmith 原理没有很好的数论知识看不大懂
3. 要加强自己的编程能力
4. 直接使用 Python 的内置库会方便很多
5. 要熟练运用 gmpy2 和 Crypto 的大数库来运算，如 powmod 比 pow 快一倍

参考文献（包括参考的书籍，论文，URL 等）

[1]Whitfield Diffie, Martin Hellman. New Directions in Cryptography[J]. *IEEE Transactions on Information Theory*, 1976, 22(6): 644-654.

[2] Ronald Rivest, Adi Shamir, Leonard Adlemen. A Method for Obtaining Digital Signatures and Public-key Cryptosystems[J]. *Communications of the ACM*, 1978, 21(4): 120-126.

[3] 谢健全, 杨春华. RSA 中几种可能泄密的参数选择[J]. 计算机工程, 2006, 32(16): 118-119.

[4] 王小云, 胡磊, 王明强. 公钥密码发展研究[A]. 2014-2015 密码学学科发展报告[M]. 中国科学技术出版社, 2016

[5] https://blog.csdn.net/pigeon_1/article/details/114371456

[6] https://www.cnblogs.com/21r000/p/15542587.html

[7] https://nonuplebroken.com/2019/11/21/RSA%20Coppersmith%E7%9B%B8%E5%85%B3%E6%94%BB%E5%87%BB/#known-high-bits-message-attack