密码学专题2实验指导

- 非对称加密
  - 非对称加密综述
  - RSA
    - RSA原理讲解
    - 程序实现
    - 攻击方法
    - 实验内容
    - 拓展：格密码、格基规约与Coppersmith攻击
    - 拓展实验内容
- 拓展：文献查找与实现综述
  - 文献查找实战

写在最前面：CryptoHack是个很好的密码学习题练习平台，会有基础练习题帮助了解基础知识，也有较高难度的挑战题

非对称加密

鸣谢：`@4qwerty7`

前置知识：对称加密、密码学安全哈希、离散数学中的群论部分、crypto基础中的RSA部分...

公钥密码体制综述（来点抽象的）

在公钥密码体制中，密码不再像是对称加密那样是一串字节流，而是由公钥和私钥组成的密钥对，通信双方不再持有相同的密码，而是一个持有公钥，一个持有私钥，因此是非对称的。

为什么要这么做?

在公钥密码体制中，我们希望私钥 x（或者说，一些随机产生的值）的持有者代表一个明确的身份，持有私钥意味着你就是你自己。

但是，私钥是随机产生的，放在私钥持有者的电脑上，别人什么都不知道，怎么判断你确实持有私钥而非另一串临时产生的随机数呢？

这就要引入公钥的概念，公钥 $y = f(x)$，大家知道公钥和身份的对应关系，就可以了。同时，公钥代表着对私钥的一个“承诺”。

一个小扩展

在公钥密码体制中，验证者在维护一定的可以证明你的公钥的可信第三方（CA机构，一个默认不会作弊的中心化机构）的公钥列表的同时，需要在私钥持有者处获取它的公钥以及CA机构对它公钥的签名（即它提供的公钥可信的证明）。

但是私钥持有者不一定在你想和它通信时在线（对于网络服务提供者来说没有问题，但是，如果私钥持有者是你的一份邮件的收件人呢？）。

为此，有公钥（或者说标识）是与你身份本身强关联的信息（比如姓名、邮箱地址等）的密码体系——标识密码体系。显然，这里私钥不再是私钥持有者自己能够生成的了（因为任何人都持有标识，你能从标识算出自己的私钥，凭什么其他人不能执行相同的算法来获得私钥？），而是需要一个有公开公钥的、可用其私钥算出任意标识对应的私钥的可行第三方。标识密码体系，例如SM9，通常采用椭圆曲线上的双线性配对来实现。

f 的性质

我们会过来来看看，从私钥算出公钥的 f 不得不有什么性质：

1. f 是**可逆**的，即没有信息丢失。如果有丢失信息，那丢失的那部分信息就是私钥持有者存储的一个和公钥无关的随机数而已，没有任何意义。
2. f 是容易计算的，这点显然。
3. f 是**单向**的，即从公钥推出私钥是计算上十分困难的，不然的话把私钥保密这件事情没有任何用途。

什么是容易的？什么是困难的？

前面提到，“容易计算”和“计算上十分困难”两个概念，他们是什么意思？

“容易计算”意味着在想要算出公钥的私钥持有者来看，它可以接受的时间内，可以计算出公钥来。这意味着计算 f 的算法通常在确定图灵机上是多项式的、乃至是线性的。

“计算上十分困难”在密码学中常用的定义是敌手（这里是一个有公钥想算出私钥的hacker）用确定图灵机、在多项式的时间内，算出私钥的概率是可以忽略不计的（关于私钥的大小 n，概率小于 $\frac{1}{\text{任何关于}n\text{的多项式}}$）。（这是因为这里不可能是0，因为私钥长度必然是有限的，导致可能的私钥数量必然是有限的，而f是可计算的，一个运气爆棚的敌手完全可能随机几次就随机到了私钥x，并通过 $y = f(x)$ 验证这点）

PS：典型的多项式时间内解决问题的能力与确定图灵机不等价的、被认为是在不远的未来可以造出来的量子计算机被纳入密码学的考量，因此，“用确定图灵机、在多项式的时间内”的要求在“后量子”的算法中被改为“用量子计算机、在多项式的时间内”。

公钥密码体制与P/NP

f 是多项式可计算的，意味着，存在属于 NP 的从公钥推出私钥的算法。

一般要求 f 的逆函数至今没有多项式算法，并假设其属于 $NP \setminus P$。

可见，公钥密码体制十分依赖 $P \neq NP$，因此如果 $P = NP$，将可能导致相关体系轰然倒塌...

PS：注意，量子计算机在多项式时间内解决问题的能力与非确定图灵机不等价，因此，造出量子图灵机不意味着任何 NP 问题都变得多项式可解了，公钥密码体制就倒塌了。

应用场景

双方密钥交换

两个私钥(p, p')持有者相互告知对方自己的公钥(P, P')，接下来双方就能用自己的私钥和对方的公钥各自计算出一个秘密值，分别为 $s = KeyExchange(P', p), s' = KeyExchange(P, p')$，且 $s = s'$（对于其他人，最多只拥有双方的公钥，是算不出 s 的），它可以作为双方接下来用对称加密通信时使用的密钥。

PS：使用双线性配对可以实现三方密钥交换。

公钥加密算法

公钥 P 的所有者用公钥对消息 m 进行加密得到密文 c，这个密文有且仅有私钥 p 持有者可以用私钥解密。

可以注意到，双方密钥交换能够实现公钥加密算法，即加密者首先随机生成一个私钥 p' 并算出公钥 P'，算出 $s = KeyExchange(P, p')$ 作为对称密钥来加密 m 得到 c'，令 $c = (c', P')$；私钥持有者算出 $s = KeyExchange(P', p)$ 并以此来解密 c' 得到 m。椭圆曲线上的公钥加密算法就是这样实现的。

公钥签名算法

签名的本质是，向验证签名的人证明：我是谁（我持有公钥 P 对应的私钥 p）；我认可消息 m。

换言之，签名者要向验证者展示一项自己拥有 p （“超能力”）才能做到的事情，同时，这个“展示”和 m 强相关，比如 RSA 中能得出满足 $m \equiv c^e \pmod N$ 的 c。

PS：可以注意到，公钥加密和公钥签名是完全不同的两件事，只是RSA的签名和解密、验证与加密恰好相似。

依赖的假设（回到具体点的）

前面的记叙已经提到了对 f 的逆运算的困难性的要求，一般由一个公认的难题（而这个难题很困难，是一个公认的假设）来充当 f 的逆运算：

1. 大质因数分解难题：随机生成两个大素数是容易的（由概率多项式的素数判定算法 Miller-Rabin 和素数密度公式可知），把他们乘起来更简单，但是，从乘积算回去是困难的。
2. 离散对数难题：在一些群（例如其阶含大质因子的 $GF(p)$ 群上或者某个椭圆曲线上）上，已知 a, k 求 a^k 容易的，但是已知 a, a^k 求 k （可以把 k 看作 $\log_a\left(a^k\right)$，即一个离散（k 必为整数否则无定义，因此对数函数不连续）的对数）很困难。
3. ...

PS：目前没有上述假设属于 $NP \setminus P$ 或者属于 NPC 的证明。

除此之外，这些密码学算法还依赖一个假设的机器（即预言机，oracle），能够在 $[0, 2^n]$ 内随机选取一个数字作为随机数。有些密码算法依赖密码学安全哈希函数的输出作为随机数，因此也要求哈希结果可以被认为是在 $[0, 2^n]$ 内随机选取的一个数字。如果能够推断出随机数产生的规律，那么这些密码学算法就会被攻破。

例如：RSA中你知道对方随机产生 p, q 的规律，即使你不会分解 N，也能猜出 p, q。

To OIer/*CPCer

以下是几道密码学入门级的程序设计竞赛问题，分别从3个方面展示了所依赖的假设被破坏时的情况，可以试着挑战一下：

1. 质因数不够大而容易被分解，则RSA公钥加密的内容可以被其他人得知：CCPC 2019 Final - K. Mr. Panda and Kakin；
2. 群的阶不含大质因子时，离散对数易被求得：2019 HDU Multi-University Training Contest 5 - 9. discrete logarithm problem；
3. 弱的随机数产生算法容易被预测其产生的值：ICPC 2020 Asia EC Final - C. Random Shuffle。

RSA

RSA，是1977年由罗纳德·李维斯特（Ron Rivest）、阿迪·萨莫尔（Adi Shamir）和伦纳德·阿德曼（Leonard Adleman）一起提出的加密算法，取三人的姓氏首字母为该加密方法的名称

RSA原理讲解

RSA的原理讲解，在网络上能找到许许多多的资料，这里简单推荐一些

RSA算法原理（一） - 阮一峰的网络日志

素数（六）基于欧拉函数的RSA算法加密原理是什么？RSA算法详解

费马小定理

若p为质数，a与p互质，则$a^{p-1} \equiv 1(mod\ p)$

有兴趣的可以自己去证明一下，或者在网上寻找证明

欧拉函数

欧拉函数Φ(n)的定义就是，对于正整数n，小于等于n且与其互质的正整数的个数

欧拉函数是一个十分重要的内容，需要同学们（自学）掌握，这里放出两个最重要的定理：

(1) 若p为质数，则$\Phi(p) = p - 1$

(2) 若m和n互素，则$\Phi(mn) = \Phi(m) * \Phi(n)$

欧拉定理

欧拉定理是费马小定理的拓展形式

对于与n互质的数m，$m^{\Phi(n)} \equiv 1(mod\ n)$

RSA加密方法

首先取一个数N=pq，p和q为两个不相等的质数

再取一个值m<N，这个值是你的密文

现在任取一个e与Φ(N)互质

此时明文c可以通过$c \equiv m^e(mod\ N)$获得，同样取c<N

根据欧拉函数的性质，可以推出$\Phi(N) = \Phi(p) * \Phi(q) = (p-1)(q-1)$

则必定会有一个d，满足$ed \equiv 1(mod\ \Phi(N))$，称d为e的逆元

这个结论的普适性证明证明需要用到群论知识，有兴趣或有抽象代数基础的同学可以查找资料，证明一下，不过在模同余类中，$e * e^{\Phi(N)-1} = e^{\Phi(N)} \equiv 1(mod\ N)$，可以很容易证明e存在模逆元

另外，d的求得需要用到扩展欧几里得算法(extgcd)，即给定a,b,不仅可以求出两数的最大公约数gcd(a,b)，还可以求出ax+by=gcd(a,b)式中的x,y的一组整数解（详细的算法思路可以自己搜索，

另外推荐一道经典题目练手1061 -- 青蛙的约会），当a=e,b=Φ(N)时，gcd(a,b)=1，经过化简就能化为$ax \equiv 1(mod\ \Phi(N))$，x就是我们所要的d的值

这时，神奇的事情发生了

$$c^d \equiv m^{ed} \equiv m^{\Phi(n)+1} \equiv m(mod\ N)$$

可以看出c经过一次幂运算又变回了m，这就是RSA加密的数学原理

将(C,N)看作公钥，(D,N)看作私钥，就能进行非对称加密了

程序实现

RSA加密算法总结

在这里简单总结一下RSA的加密和解密步骤

1. 选取大质数p和q
2. 计算N=pq
3. 计算Φ(N)=(p-1)(q-1)
4. 选取一个与Φ(N)互质的e
5. 加密密文m，明文$c \equiv m^e(mod\ n)$
6. 计算e在模Φ(N)同余类下的逆元d
7. 解密明文c，$c^d \equiv m(mod\ n)$

密码学库介绍

由于RSA的安全性保证是基于大数质因数分解的困难性，所以RSA加密需要使用高精度算法。不过，目前已经有许许多多的库便于我们进行密码学运算，比如可以在C语言中使用的OpenSSL加密算法库，C++中的CryptoPP密码学库，还有python中的科学计算库sympy，和密码学库Crypto

gmpy2

在本次Crypto2实验中，我们用到的是python中的扩展库gmpy2，这是一个较为常用的高精度计算库

常用的密码学库，特别是gmpy2库和Crypto库的安装，可以参考**环境配置保姆级教学**，在此感谢`@yyy2015c01`大佬提供的技术支持/拜谢

考虑到gmpy2库的安装较为复杂，在这里提供了一个装有gmpy2库、Crypto库、pwntools/socket库和vscode的Ubuntu22.04虚拟机，以方便同学们完成作业

用gmpy2实现RSA加密

```
import gmpy2
from Crypto.Util.number import getPrime
from Crypto.Util.number import bytes_to_long, long_to_bytes
p = getPrime(512)
q = getPrime(512)
N = p * q
PhiN = (p - 1) * (q - 1)
m = bytes_to_long(b'This_is_RSA_WoW')
# The Encrypt Step
e = 0x10001
assert gmpy2.gcd(e,PhiN) == 1
c = gmpy2.powmod(m, e, N)
print("c = " + hex(c))
# The Decrypt Step
d = gmpy2.invert(e, PhiN)
m = gmpy2.powmod(c, d, N)
print(b'm = ' + long_to_bytes(m))
```

攻击方法

然而，有时候，一些特殊的参数构造会使得RSA存在一定风险

N分解攻击

某些特殊的p,q取值会使得N很容易被质因数分解

比如，当N较小时(小于256bit)可以以较短时间直接暴力分解

另外，p，q差值太大或者太小时，可以使用Fermat或Pollard rho法进行分解

在这里推荐一些工具or网址

yafu:该工具可以用于自适应整数因式分解，基本覆盖全平台，内置有Fermat、Pollard rho、SIQS、GNFS等多种算法

factordb:在线数据库，但大多数都查不到

Integer factorization calculator:可以当作是网页版的yafu，有时处理大数质数分解会比yafu还好，强烈推荐

低加密指数小明文攻击

当e极小(如e=3)，且m也较小时

$$c \equiv m^3 (mod\ n)$$

则可以写成

$$m^3 = c + kn$$

进一步

$$m = \sqrt[3]{c + kn}$$

m很小时只需暴力枚举k的值，也有可能会使密文被爆破

举例：

```
import gmpy2
n=0xb99a120d9b8ec040e1bcea45959d2f0076439c44cc898658144af9f3701532420cfcbceb03bb32d99f9e051703bec8bfa50a10d39942597ab4cff3aa42842f45e981975c0bd0cf0560845a0930a094fd52e4cc1d3fd350babf2864ee677d8f7059872f0add2b0993df764fda8cf468dc2077531e21151b309337b62a512c0ab7
e=0x3
c=0x9d6a3d450e03a161f8849877080571ca3828fbe158e4fc27bd687d6a75150266d90e54dbade7592b467245011797cdc2bcf0b0d1bc245cfa87b4c0c32f9bee969e2283ab0895ffd0a1e27f09499f80b5e3bcf95bd9aa9fe941b8bb64d7cbd9acfb216b1f8e391ab801ec0f4d82aae27bfd93ac9a37b4d098d7c66c3e8c4ef795
i = 0
while 1:
    res = gmpy2.iroot(c+i*n,3)
    if(res[1] == True):
        m=res[0]
        print(hex(m))
        break
    print("i="+str(i))
    i = i+1
```

共模攻击

在RSA的使用中,使用了相同的模n、不同的模e对相同的明文m进行了加密，那么就可以在不分解n的情况下还原出明文m的值

已知

$$m^{e_1} \equiv c_1 (mod\ n)$$
$$m^{e_2} \equiv c_2 (mod\ n)$$

那么，可以想办法构造$m^{ae_1+be_2}$使得$ae_1+be_2\equiv 1$，此时回忆一下extgcd，当两个e互素时，可以求出a和b的值

```
import gmpy2
from Crypto.Util.number import long_to_bytes
n =
0xa44bc0ae5e054a5ead7eba9ff6844807c89f212475f70c93cd70745570eac8f7fc4c97e853c47ebd9e82
7409a029176c83c072e980ba96ba6070045777d09a554b32dcae81d03b2bfe39c87da7460c3da4b7fe2b90
b7d43adf17c00d95e3249e27643654124a0ed659fe8ea072003df4ed1d9f8fa6b6a6e31c984096e2a3228d
e1 = 0xb733349f37e8547ae9
e2 = 0xff80413c40bbcb1ad3
c1 =
0x18bbc1c4ef4ed1e5f9c7f1fa26d82de5b56e632bd6db8e5336ced7504bea71914d2a0a822860ba2c90ae
936ac6c113e0f1f49986d9610187e52ac9a4ba4ffd5e48da9fa89d5239feec1bd5a527fe30ea96497a0504
6a6c722a7d13f68d0a0f28d7030e108d0dbbb4c6b60bc62ac417d43e9a88f84ccbad3cadfcfc83e31612ad
c2 =
0x2888a1d502de6dad3d600fa297c9b35ce8ddc2a8d5d242d9a4eccec29d94981346aa8ff44f91f8ed9afa
02b4c539c47b966e285655e0aa0c17c663a4d2ae3e96dc597a7c77d0cb8e02edd84ca863e181a1123b71f8
1713b075d9f39e6a8d2a332cca259af9abeaae65f5faaf181945a609f989039cebfff7f808e107eb6a9b3d
gcd, s, t = gmpy2.gcdext(e1, e2)
m = gmpy2.powmod(c1, s, n) * gmpy2.powmod(c2, t, n) % n
print(long_to_bytes(m))
```

低指数广播攻击

首先先要引入一个数论知识：中国剩余定理

中国剩余定理（CRT）：

对于两两互素的正整数$m_1,m_2,\cdots,m_k$，同余方程组

$$\begin{aligned}x&\equiv a_1(mod\ m_1)\\x&\equiv a_2(mod\ m_2)\\&\vdots\\x&\equiv a_k(mod\ m_k)\end{aligned}$$

在模$M=m_1m_2\cdots m_k$同余类下有且仅有唯一正整数解

其解为$x\equiv\sum_{i=1}^{k}a_iM_i^{-1}M_i$，其中$M_i=M/m_i$，$M_iM_i^{-1}\equiv 1(mod\ m_i)$

再看回低指数广播攻击，其特点是，使用极小的e（通常为3），对相同的m，多次使用不同的n进行加密

通过CRT，就可以得出m^e在模$n_1n_2n_3$同余类下的解，因为e极小，所以大概率$m^e<n_1n_2n_3$（当e≤3时是一定的）

```
n1 =
0x87e9810feb6ecc7a4e55233a76626965eba9242667db3b3aa1be75c8240262d18543600149c9e90a49bc
d5262679883c37a9b65a8b05b48de1a0696576e79df32e6a2e6685d732972cf9c9da7897f6a1b87cb26176
75b71ec67e48f8cdca7d78c92c99a32681bb7e2eafc61c65637356119498a72dc671e829e5cf8d0a79ddb7
n2 =
0x81e247044f044a0cba17906b829c650b388814e757d4f45657ad8782a2bfe3e740d87c1f0ba9e00662c6
a5ec0bffa9cc2cc2974f374733fc4bd6a743fd2c222b395f7da5009fad4e748dabba0aaaf793b1b31134c6
c85f7274e30bc7ef1f8f1530eb947002e7529395e2af892cc2ddbf829989cb0f9f77a431df46388e2ca25f
n3 =
0x8533bfb4a4e5106c72f9bdf58e9b6459af80e74e1f82c9725efa30f92c584a1f506c977efda553153bbc
778c3eee7dcff2f3252a988700a8fd11be160834b8740387d94d2d197232886da6b787f00030f783011fe1
f662ce56cb66eafd1f115dc9348c43caf9a419dd01c1249c773f1106be76f3c26a02446c156374531aabaf
c1 =
0x4213640633166627acd96336c4abc12de583e852b7afb2ae83a7ef802ff3412e77cd0fd5ed18f94178e5
6157cf6c6b46adf66e545c904c0fad52b09210c598f3c0e554dbe2984a5f00d713d85e57ef73ca5184821a
648d5c41ba2b27046421c88f67d1833cf4110dcd0e3dcb3b91776676a717c352619b9e1c564ecdcf75fe26
c2 =
0x7418e32b2273d227fb36fbd1c794e609f0e72167e9e46021728799c7718b19d4f9215537a28269c3ce67
5df156c4281a9660b41307cd49211342009c5557d78dfbe47d5f59326398f8e1a9e535ff76f537f83ba9ac
fa6667a33de60afe9423643111af0ffeffdb14c81108b51000070ac9c8062c07367bc3127cef9c03ab16f4
c3 =
0x39147c85bb76babf38b27b02708ee4d00d349ecbe173c6ec20937902e012d3afbc4ce6e7b0dfa4b0bd49
8fc5130a2988253f4832d97004fd25b39a8cf38661871fc14503593800e84988fbb33d5eb3dbb1c180a11b
a39673f82ef0cc9a6ff56a1b1c37ef9405d32ccaa76f048d1a202ae95870176b893e08ddf0f3e95e13a7d4
e = 3
N1=n2*n3
N2=n1*n3
N3=n1*n2
N=n1*n2*n3
t1=gmpy2.invert(N1,n1)
t2=gmpy2.invert(N2,n2)
t3=gmpy2.invert(N3,n3)
m3=(c1*t1*N1+c2*t2*N2+c3*t3*N3)%N
mi=gmpy2.iroot(m3,e)
m=mi[0]
print(long_to_bytes(m))
```

维纳攻击

首先，需要先简要介绍一下连分数趋近

$$\frac{318583}{559019}=0+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{3+\cfrac{1}{13+\cfrac{1}{28+\cfrac{1}{3+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{5+\frac{1}{2}}}}}}}}}}}}=[0,1,1,3,13,28,3,1,1,2,5,2]=$$

0.5698965509

现在依次选取连分数的前几项构建连分数趋近

$$[0] = 0 = 0$$
$$[0,1] = 0 + \frac{1}{1} = 1$$
$$[0,1,1] = 0 + \frac{1}{1+\frac{1}{1}} = \frac{1}{2} = 0.5$$
$$[0,1,1,3] = 0 + \frac{1}{1+\frac{1}{1+\frac{1}{3}}} = \frac{4}{7} = 0.5714285714$$
$$[0,1,1,3,13] = 0 + \frac{1}{1+\frac{1}{1+\frac{1}{3+\frac{1}{13}}}} = \frac{53}{93} = 0.5698924731$$
$$[0,1,1,3,13,28] = 0 + \frac{1}{1+\frac{1}{1+\frac{1}{3+\frac{1}{13+\frac{1}{28}}}}} = \frac{1488}{2611} = 0.5698965913$$
$$[0,1,1,3,13,28,3] = 0 + \frac{1}{1+\frac{1}{1+\frac{1}{3+\frac{1}{13+\frac{1}{28+\frac{1}{3}}}}}} = \frac{4517}{7926} = 0.569896543$$
$$[0,1,1,3,13,28,3,1] = 0 + \frac{1}{1+\frac{1}{1+\frac{1}{3+\frac{1}{13+\frac{1}{28+\frac{1}{3+\frac{1}{1}}}}}}} = \frac{6005}{10537} = 0.569896555$$
$$[0,1,1,3,13,28,3,1,1] = 0 + \frac{1}{1+\frac{1}{1+\frac{1}{3+\frac{1}{13+\frac{1}{28+\frac{1}{3+\frac{1}{1+\frac{1}{1}}}}}}}} = \frac{10522}{18463} = 0.569895499$$
$$[0,1,1,3,13,28,3,1,1,2] = 0 + \frac{1}{1+\frac{1}{1+\frac{1}{3+\frac{1}{13+\frac{1}{28+\frac{1}{3+\frac{1}{1+\frac{1}{1+\frac{1}{2}}}}}}}}} = \frac{27049}{47463} = 0.569896551$$
$$[0,1,1,3,13,28,3,1,1,2,5] = 0 + \frac{1}{1+\frac{1}{1+\frac{1}{3+\frac{1}{13+\frac{1}{28+\frac{1}{3+\frac{1}{1+\frac{1}{1+\frac{1}{2+\frac{1}{5}}}}}}}}}} = \frac{145767}{255778} = 0.5698965509$$

由此可看出，连分数趋近是一种很好的趋近方法，比如圆周率π的约率$\frac{22}{7}$和密率$\frac{333}{106}$就与其连分数的第2、4项展开相等

而对于较小的d，$ed - k\Phi(N) = 1$，进一步可化为$|\frac{e}{\Phi(N)} - \frac{k}{d}| = \frac{1}{d\Phi(N)}$，而根据Wiener的论文Cryptanalysis of short RSA secret exponents，当$d < \frac{1}{3}N^{\frac{1}{4}}$时，$\frac{e}{N}$可以用$\frac{k}{d}$近似，因此可以用连分数序列表示$\frac{e}{N}$，然后通过上方所述的近似，查看是否有d满足条件

还是以上面的$\frac{318583}{559019}$为例，代表了n=559019，e=318583的情况，代入趋近分母为d，趋近分子为k，可计算出Φ(N)，进一步可算出p和q（韦达定理），再验证pq=N是否成立。经过测试可以得出上面的情况d=7

攻击脚本在实验指导书中不作展示，有兴趣的同学可以自行编写或者在网上搜索相关攻击脚本

选择明/密文攻击

> 有一个自动RSA加/解密的机器，使用者不知道它的n和e，但是它真的是安全的吗?

现在，对自动RSA加密机，输入2、4、8

$$
\begin{aligned}
c_2 &= 2^e \pmod n \\
c_4 &= 4^e \pmod n \\
c_8 &= 8^e \pmod n
\end{aligned}
$$

可以计算出$c_2^2 \equiv c_4 \pmod n$，$c_2^3 \equiv c_8 \pmod n$，这样的话$c_2^2 - c_4$和$c_2^3 \equiv c_8$就都是n的倍数，可以多用几组求出n的具体值，然后再用求离散对数的方法求出e的值（需要e较小）

> 那么，如果限制了c的长度，你还能hack the machine吗?

对于选择明密文攻击，有兴趣的同学可以去做做校巴上的Republican Signature Agency这道题

实验内容

本次实验的基础部分内容如下：

ZJU School-Bus中的**Endless RSA 1**(ACTF 2019)和**RSA Adventure**(ACTF 2020)

其中Endless RSA 1有7小问，RSA Adventure有5小问

基础部分给分详见下方的分值表

题目	难度	分值
Endless RSA 1		
Level 0	不可能不会	10
Level 1	☆	5
Level 2	☆☆	10
Level 3	☆☆☆	15
*Level 4	☆☆☆	20
Level 5	☆☆☆☆	20
Level 6	☆☆☆☆	20
RSA Adventure		
*Level 1	☆☆☆	15
Level 2	☆☆☆☆	20
Level 3	☆☆☆	15
Level 4	☆☆☆☆	20
Level 5	☆☆☆☆☆	25

*注：Endless RSA 1的Level 4和RSA Adventure的Level 1基本上是完全一样的题，所以两题最多只记录一次分数（即任意完成该两小问中的一个即可得到20pts，但两小问都做出也只能获得20pts）

需要说明的是：

1. 理论上来说，在前一个level的题目没有解决前，后面level的题是不能开始的（当然如果你有能跳题的方法请务必告诉我/doge），所以安排了两道题供大家做，如果某一题卡住了可以切换到另一题做

2. 基础部分实验需要提交实验报告。每道做出来的题均需要写在实验报告中，否则无法给分。**实验报告请务必简洁**，建议**可以写出每道题的思路并贴上攻击脚本（payload）。**其中对于“每道题的思路”，如果在课上讲过这种形式的攻击方法的，只需**写明攻击方法名称**（或者在网上找到的相同攻击方法的不同名称）；对于课上没有涉及到的题，可以**简要叙述攻击的原理**；若你从网络上搜索到了攻击的思路（或脚本），**可以直接将网址贴在实验报告中**

3. ~~关于本次的给分：只要提交了实验报告就能获得60分（如果真的一题都做不出也可以在实验报告里随便写一下自己的心得什么的）。所以，本次实际分数构成=0.6*100+0.4*(基础部分得分)，当然挑战部分也有给分的加分~~作业DDL延到7月底了，PoW脚本大家都有了，而且专题也是溢出制，理论上来说保底分应该要取消掉了，不过考虑到有一些同学有后续的短学期课程，所以保底分改为20分，并且至少要做掉一小问

4. 需要注意的是，这两题在开始关卡前有一个**PoW**（proof of work）验证，请自行查找通过PoW的方法并编写脚本，如果实在解决不了请询问助教（老师？）

5. 本次crypto2实验对python的要求会比较高，如果认为自己对python的了解还是不够的话，请务必善用搜索引擎，并积极向助教们提问（对于密码学库的问题尽量咨询密码学方向助教，不过其它python相关问题可以询问所有助教）

格密码、格基规约与Coppersmith攻击

Lattice学习笔记01：格的简介

格基规约相关 - Coinc1dens' Lotus Land

格基规约算法：数学基础

格概述 - CTF Wiki

Coppersmith 相关攻击 - CTF Wiki

线性代数

Gram-Schmidt正交化是一种让线性无关向量组等价变为标准正交向量组的方法，通过这种方法求出某个向量空间的标准正交基，能大幅减少后续计算的运算量

$$
\begin{aligned}
&\beta_1 = \alpha_1 \\
&\textbf{for } i \ in \ (2, n] \textbf{ do} \\
&\quad \mu_{ij} = \frac{(\alpha_i, \beta_j)}{(\beta_j, \beta_j)}, 0 < j < i \\
&\quad \beta_i = \alpha_i - \sum_{j=1}^{i-1} \mu_{ij} \beta_j \\
&\textbf{end}
\end{aligned}
$$

格

而格则是若干线性无关向量量$\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$整系数线性组合构成的集合，即$\mathcal{L} = \sum_{i=1}^{n} \mathbf{b}_i \cdot \mathbb{Z} = \{\sum_{i=1}^{n} c_i \mathbf{b}_i : c_i \in \mathbb{Z}\}$，而$\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_n)$就称为格基

引理：$\mathbf{B}_1$，$\mathbf{B}_2$等价的充要条件是存在单模矩阵$\mathbf{U}$使得$\mathbf{B} = \mathbf{CU}$

定义：$det(L(B)) = \sqrt{det(B^T B)}$

而对于格中的问题，SVP，SMP，CVP等问题均为计算困难性问题

最短向量问题（SVP）

给定格L及其基B，找到格L中的最短非零向量v

最近向量问题（CVP）

给定格L和目标向量t，找到一个格中的非零向量v，使得对于格中的任意非零向量u满足
$\|v - t\| \leq \|u - t\|$

连续最小长度问题（SMP）

求出给定秩为 n 的格 L的连续最小长度，即找到格 L 中 n 个线性无关向量s_i，对于任意1≤i≤n，满足格中 i 个线性无关的向量$\|v_j\| \leq \|s_i\| = \lambda_i, 1 \leq j \leq i$的最小值

另外还有最短线性无关向量问题（SIVP）等格上问题也是计算困难的

格密码

正因格中许多计算是属于计算困难性问题，因此可以用来作为非对称加密，便称之为格密码，格密码最引人注目的特点就是它是抗量子计算攻击的。在这里对格密码不多作叙述，有兴趣的同学可以自行查找资料

格基规约

我们对把一个Lattice的基进行变换，找到一组非常接近垂直的基的过程成为格基规约，完成格基规约后就能快速求解SVP、CVP等问题

高斯格基规约算法

这是一种较为古老的二维格基规约算法，比较像GCD和施密特正交化的结合

时间复杂度为$O(log \mid v \mid \cdot [1 + log \mid v \mid - log\lambda_1(\mathcal{L})])$

LLL算法和BKZ算法

现代格基规约算法主要使用LLL算法以及其改良算法BKZ算法，BKZ的具体的算法不在这里叙述了，有兴趣的同学可以在参考网站中自学

LLL(Lenstra–Lenstra–Lovász)算法，可以看作是一种高斯算法在高维格中的推广算法

若格基B满足以下两个性质

(1) 对于任意的j < i ≤ n，有$|\mu_{i,j}| = |\frac{(\mathbf{b}_i, \mathbf{b}_j^*)}{(\mathbf{b}_j^*, \mathbf{b}_j^*)}| \leq \frac{1}{2}$ (2) 任取i有$\delta \|\mathbf{b}_i^*\|^2 \leq \|\mathbf{b}_{i+1}^* + \mu_{i+1,i}\mathbf{b}_i^*\|^2$（Lovász condition）

则称其为格L的一组δ-LLL约化基

而LLL算法的大致思路是，按顺序对基做施密特正交化（取整），再判断与前一个基是否满足Lovász condition，若满足则继续处理下一个，不满足则交换两者并回退到已处理过的基中最大的那个，直至Lovász condition全部满足

LLL算法的约化能力是$\|b_1\| \leq (\frac{2}{4\delta-1})^{n-1}\lambda_1(\mathcal{L})$，能够在$O\left(n^6 \ln^3 B\right)$的时间内，输出质量较高的约化基，且满足$\|b_1\| \leq 2^{\frac{n-1}{4}} det(L)^{\frac{1}{n}}$

BKZ算法的约化能力下界是$|b_1| \leq \beta^{\frac{n-1}{2(\beta-1)}+\frac{3}{2}} det(B)^{\frac{1}{n}}$，不过尚未被证明是多项式时间的算法

在开源软件Sage中可以调用一个简单的命令完成格基的LLL算法或BKZ算法规约

SageMath

可以在官网下载，也可以直接在线跑sage脚本Sage Cell Server

```
st1= matrix([
[846835985, 9834798552],
[87502093, 123094980]])
stl=st1.LLL()
#stl=st1.BKZ()
print(stl)
```

另附上LLL算法的具体简易实现

```
def LLL(B, delta):
    Q = gram_schmidt(B)

    def mu(i,j):
        v = B[i]
        u = Q[j]
        return (v*u) / (u*u)

    n, k = B.nrows(), 1
    while k < n:

        # length reduction step
        for j in reversed(range(k)):
            if abs(mu(k, j)) > .5:
                B[k] = B[k] - round(mu(k, j))*B[j]
                Q = gram_schmidt(B)

        # swap step
        if Q[k]*Q[k] >= (delta - mu(k,k-1)**2)*(Q[k-1]*Q[k-1]):
            k = k + 1
        else:
            B[k], B[k-1] = B[k-1], B[k]
            Q = gram_schmidt(B)
            k = max(k-1, 1)

    return B
```

Coppersmith攻击

通过构造包含小整数根的模多项式方程，我们可以通过Coppersmith攻击来在多项式时间内找到所有方程的小整数根，这样对某些特殊情况RSA的破解会有很大的作用

首先，Don Coppersmith提出了一个引理，即对于一个定义域为有限域且具有较小系数的多项式，其在有限域上寻找多项式的根可以直接看作在整数/实数域下找整数根

引理：定义$h(x)=\sum a_i x^i \in \mathbb{Z}[x]$的范数$\|h\|=\sqrt{\sum_i |a_i|^2}$，设h(x)为d次多项式，X为正整数，假设$\|h(xX)\|<N/\sqrt{d}$，对于$|x_0|<X$满足$h(x_0)\equiv 0(mod\ N)$，则可以直接将其视作整数域下的根

$$|h(x_0)|=|\sum a_i x_0^i|=|\sum a_i X^i(\frac{x_0}{X})^i|\leq\sum|a_i X^i(\frac{x_0}{X})^i|\leq\sum|a_i X^i|\leq\sqrt{d}\|h(xX$$

我们需要构造小函数h(x)=g(x)f(x)，然而很难找到符合线性组合的g(x)

接下来，Don Coppersmith使用了一些神奇的构造

$$g_{u,v}(x)=N^{m-v}x^u f^v(x)$$

因为$f(x_0) = 0(mod\ N)$，所以同理$g_{u,v}(x_0) = 0(mod\ N)$

然后就要考虑如何继续构造成小范数多项式的线性组合

现在，将$g_{u,v}(x)$看作格的行向量，u取0,1,...,d-1，v取0,1,...,m，这样就能构造出一个ω行ω列的矩阵(ω=d(m+1))

根据$g_{u,v}(x)$的定义，$f(x_0) \equiv 0(mod\ N)$的小根同样也会满足$g_{u,v}(x_0) \equiv 0(mod\ N^m)$

回想一下格基规约的最终目的，是构造出较小的格基，注意到格基和此处的多项式范数定义均为L2范数（欧氏距离），所以大胆放心地使用格基规约，就能约化出一堆小范数多项式，这就是我们需要的h(x)

由LLL的性质可推出$\|v\| \leq 2^{\frac{\omega}{4}} det(L)^{\frac{1}{\omega}}$，那只要$2^{\frac{\omega}{4}} det(L)^{\frac{1}{\omega}} < N^m/\sqrt{\omega}$就能完成小根攻击的效果

然后我们就可以愉快地解决小根问题啦Hi~ o(￣▽￣)ブ

想要理解更深的原理可以看一下论文Twenty Years of Attacks on the RSA Cryptosystem

当然，sage里自带了small_roots方法，只需一行代码就能求出小根

```
sage.rings.polynomial.polynomial_modn_dense_ntl.small_roots(self, X=None, beta=1.0, epsilon=None, **kwds)
```

其中$X = ceil(\frac{1}{2} N^{\frac{\beta^2}{\delta} - \epsilon})$，β的意义是n的某个因数b使得$b \geq n^{\beta}$，所以0<β≤1

下面是coppersmith攻击的常见例题

已知明文高位

```
n=0x2680048649769800c79ec1f3ceab3909b8dc665c2f44dea93678a3c604c1e87cd7500a59e126bbeed1
0318c83807c7701d8968448200a43b7f64e697129c8fa88b52f0e62ce3166a0d817f195452e0de6fe27e1d
2939a88756e2b38694837513199aa6d92ba88dd895020a53ef007b0f57478c7f363d4d262db4c2da5e8b15
75
c=0xbdf9abbdf60072d7421a33bd7294c76fcea8f5cac435e552b003d9192de4391a049fc8911b15378091
bd4f87a6173671abcd13e92d353d5b1f0798d49538d4da98931b5f021ca7743036b996893b8d81910859ed
ba619c28ec237e5f7abc0d9a4fdb1a85ba44c22b9031d3317998eb41d4d11ab34ab1f731ba50697317029c
d
mbar=0x472796951dbf2909faba23a804ff52df69a44e166d7a7bd2c57eeba22e4ea2dcb99123fdb17db03
8a13af1061c41daab206a88881c802300000000000000000000
kbits = 72
e=3
PR.<x> = PolynomialRing(Zmod(n))
f = (mbar + x)^e - c
x0 = f.small_roots(X=2^kbits, beta=1)[0]  # find root < 2^kbits with factor = n
print(hex(mbar + x0))
#0x472796951dbf2909faba23a804ff52df69a44e166d7a7bd2c57eeba22e4ea2dcb99123fdb17db038a13
af1061c41daab206a88881c8023cd25397dfaff8641ec
```

已知p，q其中一个的高位

```
n=0x63d1b2ecd4fd3da6ce2afe6c7b2e9086920a461f1b6a0d0c5782563c1e6af681cd03ea7ff4b4cb5d6f
980fca6f605db0376c337e0921798f395fb0088b895eebe4b977251c9000c9c80055edbdc89068e0e88f31
1641407c8082aaab9497ee685005063af64b735143d18bbb0c72ee02220bd68c8250e5bcfde8d3d7e66bf2
b5
p=0x8b8a9cc1a5373864eaf7671d0a12547163c83408f1e99dbfd531f86dc7af585bae265a749f63d05016
b4596ecdc530850000000000000000000000000000000000
pbits = 512  #p的位数
kbits = 128     #未知位数
pbar = p & (2^pbits-2^kbits)
print("upper %d bits (of %d bits) is given" % (pbits-kbits, pbits))
PR.<x> = PolynomialRing(Zmod(n))
f = x + pbar
# f = (x + pbar) % p = 0
x0 = f.small_roots(X=2^kbits, beta=0.4)[0]  # find root < 2^kbits with factor >= n^0.4
print(hex(x0 + pbar))
#0x8b8a9cc1a5373864eaf7671d0a12547163c83408f1e99dbfd531f86dc7af585bae265a749f63d05016b
4596ecdc53085ca1dbee39771f2c4258f31116c6d7003
```

已知d低位求p,q

```
def partial_p(p0, kbits, n):
    PR.<x> = PolynomialRing(Zmod(n))
    nbits = n.nbits()

    f = 2^kbits*x + p0
    f = f.monic()
    roots = f.small_roots(X=2^(nbits//2-kbits), beta=0.3)  # find root < 2^(nbits//2-
kbits) with factor >= n^0.3
    if roots:
        x0 = roots[0]
        p = gcd(2^kbits*x0 + p0, n)
        return ZZ(p)

def find_p(d0, kbits, e, n):
    X = var('X')

    for k in range(1, e+1):
        results = solve_mod([e*d0*X - k*X*(n-X+1) + k*n == X], 2^kbits)
        for x in results:
            p0 = ZZ(x[0])
            p = partial_p(p0, kbits, n)
            if p:
                return p


if __name__ == '__main__':

n=0x5c6b8458509910007051be15d867161643d8c62a3032200261b8fef53403d4a7639e4f810fc34ddbce
0c49156bab8686ecea8a2cfeba2349304535e483fd40cfedb8c45163f130f98d23edb8bca3c049f34f11c9
2c9ec7df9a9096221564a59f9e165106144b4b2f890d19e1bc82d24b4e2fe27b34428fd635f49f3e589fb5
a7
    e = 3
    d =
0xc54fb4b61153358ca26946efd6c472b248715edb73d853900400e2488535847b5030dd97a619693fc9bf
8151a1562b291193f87518fae5c1674266cd6f0a91bb

    nbits = n.nbits()
    kbits = floor(nbits*(0.5))
    d0 = d & (2^kbits-1)
    print("lower %d bits (of %d bits) is given" % (kbits, nbits))

    p = find_p(d0, kbits, e, n)
    print(hex(p))
```

对于coppersmith的利用，还有许多其它复杂的攻击方法，如Broadcast Attack with Linear Padding，Related Message Attack，Boneh and Durfee attack等，有兴趣的同学可以自行查阅论文资料学习，通常这些题都是直接套模板的，没有太大意义

拓展实验内容

crypto2课程结束后将会放上拓展实验的挑战题（1-3题），与格基规约和coppersmith攻击相关，不是必须做的，但是如果做出来了crypto2专题会有加分（但是不会溢出）

文献查找与实现综述

在CTF实战中，许多题目的攻击方法自己完全不熟悉，或者在有限的时间内无法自己推导出攻击方法，这时可以搜索以前的文献中是否存在对攻击方法的详细证明和推导，并编写代码脚本实现攻击。从出题人的角度来说，阅读文献而了解到一种情况的攻击方法，也是出题的一种思路

下面，就用一道例题来进行文献查找的实战

文献查找实战——equivalent(from D3CTF)

task.py

```python
from collections import namedtuple
from Crypto.Util import number
from Crypto.Random import random


PublicKey = namedtuple('PublicKey', ['a'])
SecretKey = namedtuple('SecretKey', ['s', 'e', 'p'])



def keygen(sbit, N):
    s_ = [random.getrandbits(sbit) | 1 for _ in range(N)]

    pbit = sum(s_).bit_length() + 1
    p = number.getPrime(pbit)
    while not (sum(s_) < p < 2*sum(s_)):
        p = number.getPrime(pbit)
    e = random.randint(p//2, p-1)

    a_ = [e*s_i % p for s_i in s_]

    assert 2**N > max(a_)

    pk = PublicKey(a_)
    sk = SecretKey(s_, e, p)

    return (pk, sk)


def enc(m, pk):
    assert 0 <= m < 2
    n = len(pk.a)
    r_ = [random.getrandbits(1) for _ in range(n-1)]
    r_n = (m - sum(r_)) % 2
    r_.append(r_n)
    c = sum(a_i*r_i for a_i, r_i in zip(pk.a, r_))
    return c


def dec(c, sk):
    e_inv = number.inverse(sk.e, sk.p)
    m = (e_inv * c % sk.p) % 2
    return m


def encrypt(msg, pk):
    bits = bin(number.bytes_to_long(msg))[2:]
    cip = [enc(m, pk) for m in map(int, bits)]
    return cip
```

```python
def decrypt(cip, sk):
    bits = [dec(c, sk) for c in cip]
    msg = number.long_to_bytes(int(''.join(map(str, bits)), 2))
    return msg


if __name__ == "__main__":
    from secret import FLAG

    pk, sk = keygen(sbit=80, N=100)

    msg = FLAG.removeprefix(b"d3ctf{").removesuffix(b"}")
    cip = encrypt(msg, pk)

    assert msg == decrypt(cip, sk)

    with open("data.txt", 'w') as f:
        f.write(f"pk = {pk}\n")
        f.write(f"cip = {cip}\n")
```

data.txt详见附件

实验内容

课上会给出15-20min来进行文献查找，如果能找到相关文献就能获得当次作业赋分附加分25或50分（论文查找给分分两档，猜猜为什么/doge，本次作业得分加满为止），在同学实践结束后会进行实验讲解