

CTF Crypto: 从入门到进阶

Triode

目录

1	前言	1
2	基础知识	3
2.1	初等数论	3
2.1.1	整除与最大公约数	3
2.1.2	二元一次不定方程与扩展欧几里得算法	4
2.1.3	质数、同余与素性测试	5
2.1.4	同余式与孙子定理	7
2.2	线性代数	12
2.2.1	矩阵与行列式	12
2.2.2	线性变换	12
2.2.3	向量空间	12
2.3	抽象代数	13
2.3.1	集合	13
2.3.2	代数系统	13
2.3.3	群	13
2.3.4	环	13
2.3.5	域	13
2.3.6	格	13
3	对称密码	15
3.1	流密码	15
3.2	分组密码	15
4	非对称密码	17
4.1	RSA	17
4.1.1	直接分解模数 n	18

Chapter 1

前言

Chapter 2

基础知识

2.1 初等数论

由于现代密码学多数以数论为基础，所以在学习密码学之前，我们需要了解一些初等数论的知识，在这里对理论知识只作简要的介绍，若想深入学习初等数论相关知识，推荐阅读《数论概论》([美]Joseph H. Silverman 著) 或者《初等数论》(闵嗣鹤、严士健编)。

2.1.1 整除与最大公约数

整除是数论中的基本概念，所以在学习初等数论之前，我们需要了解整除的定义。

整除的定义 设 a, b 是任意两个整数 (其中 $b \neq 0$)，若存在一个整数 q 使得等式

$$a = bq$$

成立，则称 a 被 b 整除 (或称 b 整除 a)，记作 $b|a$ ，此时称 b 为 a 的因数， a 为 b 的倍数。反之，若整数 q 不存在则称 a 不能被 b 整除 (或称 b 不能整除 a)，记作 $b \nmid a$ 。

下面简要介绍一些整除相关的定理：

- 若 $b|a$ 且 $c|b$ ，则有 $c|a$
- 若 a_1, a_2, \dots, a_n 都是 m 的倍数，而 k_1, k_2, \dots, k_n 是任意 n 个整数，则 $k_1a_1 + k_2a_2 + \dots + k_na_n$ 也是 m 的倍数
- (带余除法) 若 a, b 是任意两个整数 (其中 $b > 0$)，则存在唯一的一对整数 q, r 使得等式 $a = bq + r$ 成立，其中 $0 \leq r < b$ 称为余数， q 称为不完全商 (有时也简称为商)

整除的概念以及相关定理在数论乃至在密码学中都是十分重要的。

在简要了解了整除的概念之后，我们将要了解一个重要的概念——最大公约数。

公约数的定义 设 a_1, a_2, \dots, a_n 是 $n(n \geq 2)$ 个整数，若存在一个整数 d 使得 $d|a_1, d|a_2, \dots, d|a_n$ ，则称 d 是 a_1, a_2, \dots, a_n 的一个公约数。

在 a_1, a_2, \dots, a_n 的所有公约数中，最大的一个称为 a_1, a_2, \dots, a_n 的最大公约数，记作 (a_1, a_2, \dots, a_n) ，而若 $(a_1, a_2, \dots, a_n) = 1$ ，则称 a_1, a_2, \dots, a_n 互素，若 a_1, a_2, \dots, a_n 中每两个数互素，则称 a_1, a_2, \dots, a_n 两两互素。

在了解了最大公约数的概念之后，我们需要首先了解最大公约数的一些性质：

- 若 a_1, a_2, \dots, a_n 为任意 n 个不为 0 的整数, 则 a_1, a_2, \dots, a_n 的公约数与 $|a_1|, |a_2|, \dots, |a_n|$ 的公约数相等, 且 $(a_1, a_2, \dots, a_n) = (|a_1|, |a_2|, \dots, |a_n|)$
- 若 a 是一个非零整数, 则 $(0, a) = |a|$
- 设 a, b, c 是任意三个不全为 0 的整数, 且有 $a = bq + c$ (其中 $q \neq 0$), 则 $(a, b) = (b, c)$

由上面提到的性质我们可以得出一系列等式:

$$\begin{aligned}
 a &= bq_1 + r_1, & 0 \leq r_1 < b \\
 b &= r_1q_2 + r_2, & 0 \leq r_2 < r_1 \\
 r_1 &= r_2q_3 + r_3, & 0 \leq r_3 < r_2 \\
 &\dots & \\
 r_{n-2} &= r_{n-1}q_n + r_n, & 0 \leq r_n < r_{n-1} \\
 r_{n-1} &= r_nq_{n+1} + r_{n+1}, & r_{n+1} = 0
 \end{aligned} \tag{2.1.1.0.1}$$

根据这一系列等式我们可以知道对于任意两整数 a, b , (a, b) 就是这一系列等式中的最后一个非零余数 r_n , 这一系列等式就是著名的辗转相除法¹, 也称为欧几里得算法. 欧几里得算法的流程如下:

Algorithm: 欧几里得算法 $\gcd(a, b)$

Input: a, b

Output: (a, b)

while $b \neq 0$ **do**

$r \leftarrow a \bmod b$

$a \leftarrow b$

$b \leftarrow r$

end

return a

幸运的是, Python 中的 `math`, `gmpy2` 等库都提供了计算最大公约数的函数 (一般为 `gcd`), 我们可以直接调用这些函数来计算最大公约数.

在整除和最大公约数的基础上, 我们可以进一步得到一个重要的定理:

裴祖定理 设 a, b 是任意两个不全为 0 的整数, 则对于任意整数 x, y , 都有

$$(a, b) | ax + by$$

通过裴祖定理定理, 我们可以知道对于任意两个不全为 0 的整数 a, b , 都存在整数 x, y 使得 $ax + by = (a, b)$, 这个结论在密码学中有着重要的应用.

2.1.2 二元一次不定方程与扩展欧几里得算法

所谓二元一次不定方程, 就是指形如 $ax + by = c$ 的方程, 其中 a, b, c 是已知的整数, x, y 是未知的整数, 在密码学中通常要求解这类问题, 由裴祖定理我们很容易可以知道形如 $ax + by = c$ 的方程存在整数解的充要条件是 $(a, b) | c$, 而如若知道方程 $ax + by = c$ 的一组整数解为 $(x, y) = (x_0, y_0)$, 我们很容易可以知道这组方程的所有解为可以表示为 $(x, y) = (x_0 - bt, y_0 + at)$ (其中 $t \in \mathbb{Z}$). 所以我们的问题就转化为了求解 $ax + by = (a, b)$ 的其中一组整数解, 为求解这一方程, 我们可以使用扩展欧几里得算法.

扩展欧几里得算法的流程如下:

¹辗转相除法与更相减损术 (《九章算术·卷一》中提及的“约分术”: 可半者半之; 不可半者, 副置分母、子之数, 以少减多, 更相减损, 求其等也. 以等数约之) 略有不同, 辗转相除法的效率会略优于更相减损术

Algorithm: 扩展欧几里得算法 $\text{extgcd}(a, b, x, y)$

```

if  $b = 0$  then
     $x \leftarrow 1$ 
     $y \leftarrow 0$ 
    return  $a$ 
end
 $d \leftarrow \text{extgcd}(b, a \bmod b, x, y)$ 
 $x_0 \leftarrow x$ 
 $y_0 \leftarrow y$ 
 $x \leftarrow y_0$ 
 $y \leftarrow x_0 - \lfloor \frac{a}{b} \rfloor y_0$ 
return  $d$ 

```

通过使用扩展欧几里得算法，我们可以在求 a, b 的最大公约数的同时求出方程 $ax + by = (a, b)$ 的一组整数解 (x_0, y_0) ，从而可以求解二元一次不定方程 $ax + by = c$ 的所有整数解。

在实际应用中，我们可以使用 gmpy2 库中的 `gcdext` 函数来求解二元一次不定方程 $ax + by = (a, b)$ ，实际用例如下：

```

1  import gmpy2
2  a=123456789
3  b=987654321
4  x,y,d=gmpy2.gcdext(a,b)
5  print(x,y,d)

```

该函数接受两个参数 a, b ，并返回三个值 d, x, y ，其中 d 为 a 与 b 的最大公约数， x, y 则是方程 $ax + by = d$ 的其中一组整数解。

在后面也会对拓展欧几里得算法的应用进行更加详细的介绍。

2.1.3 质数、同余与素性测试

质数（prime number，又称素数）是指除了 1 和它本身之外没有其他因数的大于 1 的自然数，比如 2, 3, 5, 7, 11, \dots 都是质数，而不是质数的数称为合数。在密码学中，质数是一个很重要的概念。

关于质数，有一个很重要的定理——算术基本定理：

算术基本定理 每一个大于 1 的自然数都可以唯一表示为一系列质数的乘积，即对于任意一个大于 1 的自然数 n ，都存在唯一的一组质数 p_1, p_2, \dots, p_k 使得：

$$n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$$

其中 $p_1 < p_2 < \dots < p_k$ 是质数， e_1, e_2, \dots, e_k 是正整数。

在初步了解了质数与算术基本定理之后，我们即将需要了解同余，同余是数论中的一个重要概念，它描述了整数之间的一种关系，这也是 RSA 公钥体系以及很多密码学理论的基础。

同余的定义 给定一正整数 m ，若对整数 a, b ，用 a 除以 m 所得的余数等于 b 除以 m 所得的余数，则称 a 与 b 对模 m 同余，记作 $a \equiv b \pmod{m}$ ，否则称 a 与 b 对模 m 不同余。

通过对定义的延申，我们可以得到同余的一些性质：

- (同余关系的自反性) $a \equiv a \pmod{m}$

- (同余关系的对称性) 若 $a \equiv b \pmod{m}$, 则 $b \equiv a \pmod{m}$
- (同余关系的传递性) 若 $a \equiv b \pmod{m}$, $b \equiv c \pmod{m}$, 则 $a \equiv c \pmod{m}$
- 若 $a_1 \equiv b_1 \pmod{m}$ 且 $a_2 \equiv b_2 \pmod{m}$, 则 $a_1 + a_2 \equiv b_1 + b_2 \pmod{m}$, 而且 $a_1 a_2 \equiv b_1 b_2 \pmod{m}$
- 若 $a + b \equiv c \pmod{m}$, 则 $a \equiv c - b \pmod{m}$
- 若有 $a = a'p, b = b'p$ 而且 $(p, m) = 1$, 则如果 $a \equiv b \pmod{p}$, 那么有 $a' \equiv b' \pmod{m}$
- 若 $a \equiv b \pmod{m}$, 则 $a^k \equiv b^k \pmod{m}$ (其中 k 是整数)
- 若 $a \equiv b \pmod{m}$, 则对于任意整数 k , 有 $ka \equiv kb \pmod{mk}$
- 若 $a \equiv b \pmod{m}$, 则对于 a, b, m 的任意一个公约数 d , 有 $\frac{a}{d} \equiv \frac{b}{d} \pmod{\frac{m}{d}}$
- 若对于整数 m_1, m_2, \dots, m_k 均有 $a \equiv b \pmod{m_i} (i = 1, 2, \dots, k)$, 则对于它们的最小公倍数 m , 有 $a \equiv b \pmod{m}$
- 若 $a \equiv b \pmod{m}$, 则对于 m 的任一正因数 d , 有 $a \equiv b \pmod{d}$
- 若 $a \equiv b \pmod{m}$, 则必然有 $(a, m) = (b, m)$

在同余的基础上, 我们有一个重要的定理——欧拉定理. 在介绍欧拉定理之前, 我们需要先了解欧拉函数:

欧拉函数 对于任意一个正整数 n , 欧拉函数 $\varphi(n)$ 定义为 $1, 2, \dots, n-1$ 中与 n 互素的数的个数. 设 $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, 则我们可以得到欧拉函数的计算公式:

$$\varphi(n) = n \left(1 - \frac{1}{p_1}\right) \left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_k}\right)$$

同时我们可以得到一个重要的结论: 若 a, b 为两个互素的正整数, 则 $\varphi(ab) = \varphi(a)\varphi(b)$.

此时我们就可以引入一个在数论中以及密码学中十分重要的定理——欧拉定理:

欧拉定理 对于任意一个大于 1 的正整数 n , 若 $(a, n) = 1$, 则有 $a^{\varphi(n)} \equiv 1 \pmod{n}$.

当 n 为质数时, 我们可以得到一个推论——费马小定理:

费马小定理¹ 对于任意一个质数 p , 若 $(a, p) = 1$, 则有 $a^{p-1} \equiv 1 \pmod{p}$.

欧拉定理是 RSA 公钥体系的基础, 在 RSA 公钥体系中, 我们往往需要通过欧拉定理来求出私钥, 这将会在之后的章节讲到.

接下来是本节中最后一个重要的概念——素性测试, 素性测试可以用于判定所给自然数是否为素数, 这在密码学中有着重要的应用.

素性测试一般分为两类: 确定性素性测试和概率性素性测试, 其中确定性素性测试可以绝对确定一个数是否为素数, 而概率性素性测试可以较快速地以很高的概率确定一个数是否为素数, 但是存在一定的概率错误地将一个合数判断为素数.

在确定性素性测试中, 最简单的算法是试除法, 即对于一个数 n , 我们可以从 2 到 $n-1$ 遍历所有的整数, 若存在一个整数可以整除 n , 则可以断定 n 为合数, 否则 n 为素数, 显然这个算法的时间复杂度为 $O(n)$, 而我们知道, 如果 n 为合数, 则必然存在一个小于 \sqrt{n} 的因数, 所以我们可以将试除法的时间复杂度降低到 $O(\sqrt{n})$, 那么我们可以将该算法改进为:

¹费马小定理的逆定理并不成立, 所以费马小定理并不能用来准确地判断一个数是否素数, 因为存在合数 $341 = 11 \times 31$, 使得 $2^{341-1} \equiv 1 \pmod{341}$, 这类合数一般被称为伪素数 (pseudoprime number)

Algorithm: 试除法 isPrime(n)

```

for  $i = 2$  to  $\sqrt{n}$  do
    if  $i|n$  then
        return  $n$  不是素数
    end
end
return  $n$  是素数
  
```

然而若 n 是一个很大的整数, $O(\sqrt{n})$ 的时间复杂度仍然是不可接受的, 所以我们需要更高效的确定性素性测试算法, 例如 AKS 算法, 不过这些确定性素性测试算法在效率上仍然不如概率性素性测试中的 Miller-Rabin 算法.

在概率性素性测试中, 最著名的算法是 Miller-Rabin 算法, 该算法基于费马小定理与二次探测定理, 现在先介绍二次探测定理:

二次探测定理 对于质数 p , 若有 $x^2 \equiv 1 \pmod{p}$, 则该方程小于 p 的正整数解只有两个: $x \equiv 1 \pmod{p}$ 或 $x \equiv p-1 \pmod{p}$.

基于费马小定理与二次探测定理, 在一轮测试中, 使用随机底数 a 对数 n 进行测试, 则对于一个用费马小定理判断为可能是质数的数 n , 则必然满足 $a^{n-1} \equiv 1 \pmod{n}$, 此时我们可以将 $n-1$ 写成 $2^s d$ 的形式 (其中 d 为奇数), 所以在一轮测试中, 我们可以通过判断 $a^{2^i d} (i = 0, 1, 2, \dots)$ 在模 n 下是否等于 1 或 $n-1$ 来判断 n 是否为素数, 若对中间某一个状态 $a^{2^i d} \equiv n-1 \pmod{n}$, 则可以初步判断 n 为素数, 为提高准确率, 我们需要多进行几轮测试, 若 n 经过 k 轮测试都通过, 则可以认为 n 为素数, 否则 n 为合数. 由此我们可以得到 Miller-Rabin 算法的流程如下页算法所示.

Miller-Rabin 算法的时间复杂度为 $O(k \log n)$, 是一个很优秀的算法.

在实际应用中, 我们并不需要自己实现素性测试算法, 在很多 Python 库中都有素性测试的函数, 例如在 pycryptodome 库中有 isPrime 函数, sympy 库中有 isprime 函数, gmpy2 库中有 is_prime 函数, 这些函数都可以帮助我们快速、准确地判断一个数是否为质数.

2.1.4 同余式与孙子定理

在了解同余的概念之后, 我们可以进一步了解同余式以及解法.

同余式 对多项式 $f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_0$ (其中 $a_0, a_1, \dots, a_n \in \mathbb{Z}$), 则对于正整数 m

$$f(x) \equiv 0 \pmod{m}$$

称为模 m 的同余式, 而若 a_n 模 m 不为零, 则称该式的次数为 n .

若存在整数 x_0 使得 $f(x_0) \equiv 0 \pmod{m}$, 则称 x_0 为同余式 $f(x) \equiv 0 \pmod{m}$ 的一个解.

在介绍高次 (次数大于 2) 同余式的解法之前, 我们先介绍一次同余式的解法.

对于一条一次同余式 $ax \equiv b \pmod{m}$ (在之后的讨论中, 我们默认 a 不为 m 的倍数), 我们可以通过同余的定义将其转化为一个二元一次不定方程 $ax + my = b$, 那么我们很容易可以知道这个方程有解的条件是 $(a, m) | b$, 所以我们通过二元一次不定方程的知识可以知道这个方程有解的充要条件是 $(a, m) | b$, 从而我们可以通过扩展欧几里得算法求出这个方程的解.

对于一类特殊的同余式 $ax \equiv 1 \pmod{m}$, 若这类同余式有解 (即 a 与 m 互质), 则称 a 在模 m 下有乘法逆元, 该同余式的解记作 $x \equiv a^{-1} \pmod{m}$, 称为 a 模 m 下的乘法逆元 (简称逆元). 对于在模 m 下确定存在逆元的数 a , Python 提供了多种方法来求解 a 在模 m 下的逆元, 例如可以通过 Python 自

Algorithm: Miller-Rabin 素性测试 MillerRabin(n, k)

```

if  $n < 2$  or  $n \% 2 = 0$  then
  | return  $n$  不是素数
end
if  $n = 2$  then
  | return  $n$  是素数
end
将  $n - 1$  写成  $2^s d$  的形式
for  $i = 1$  to  $k$  do
  | 随机选取  $a \in [2, n - 2]$ 
  |  $x \leftarrow a^d \pmod n$ 
  | if  $x = 1$  or  $x = n - 1$  then
  | | 进入下一轮测试
  | end
  | for  $j = 1$  to  $s$  do
  | |  $x \leftarrow x^2 \pmod n$ 
  | | if  $x = 1$  or  $x = n - 1$  then
  | | | 进入下一轮测试
  | | end
  | end
  | return  $n$  不是素数
end
return  $n$  是素数

```

带的 `pow` 函数通过 `pow(a,-1,m)` 来求解 a 在模 m 下的逆元, 也可以通过 `gmpy2` 库中的 `invert` 函数或者 `pycryptodome` 库中的 `inverse` 函数来求解 a 在模 m 下的逆元.

在了解了一次同余式的解法后, 我们将要进一步了解一元一次同余式组的解法, 一元一次同余式组的形式如下:

$$\begin{cases} x \equiv b_1 \pmod{m_1} \\ x \equiv b_2 \pmod{m_2} \\ \dots \\ x \equiv b_n \pmod{m_n} \end{cases} \quad (2.1.4.0.1)$$

《孙子算经》最早给出了这类问题的解法, 故该类问题的解法被称为孙子定理, 孙子定理在国外亦被称为中国剩余定理 (Chinese remainder theorem, 简称 CRT), 该定理详细地介绍了同余式组 (2.1.4.1) 如何求解:

孙子定理 (中国剩余定理, CRT) 设 m_1, m_2, \dots, m_k 是 k 个两两互质的正整数, 令 $m = m_1 m_2 \dots m_k$, 记 $M_i = \frac{m}{m_i} (i = 1, 2, \dots, k)$, 则同余式组 (2.1.4.1) 的解为:

$$x \equiv \sum_{i=1}^k M'_i M_i b_i \equiv M'_1 M_1 b_1 + M'_2 M_2 b_2 + \dots + M'_k M_k b_k \pmod{m}$$

其中 M'_i 是 M_i 在模 $m_i (i = 1, 2, \dots, k)$ 下的逆元, 即 $M'_i M_i \equiv 1 \pmod{m_i}$.

据此我们可以得到通过孙子定理求解同余式组的流程:

在实际应用中, 我们可以通过 Python 的 `sympy` 库中的 `crt` 函数或者 `sage` 中的 `crt` 函数来求解同余式组 (2.1.4.1).

Algorithm: CRT($b_1, b_2, \dots, b_k; m_1, m_2, \dots, m_k$)

```

 $m \leftarrow m_1 m_2 \cdots m_k$  for  $i = 1$  to  $k$  do
     $M_i \leftarrow \frac{m}{m_i}$ 
     $M'_i \leftarrow \text{inv}(M_i, m_i)$  //求解  $M_i$  在模  $m_i$  下的逆元
end
 $x \leftarrow 0$ 
for  $i = 1$  to  $k$  do
     $x \leftarrow x + M'_i M_i b_i \pmod{m}$ 
end
return  $x$ 

```

在了解了孙子定理 (之后均简称为 CRT) 之后, 我们就可以通过之前所学的知识来求解高次同余式了. 通过同余式组的知识, 我们知道如下事实: 若 m_1, m_2, \dots, m_k 是 k 个两两互质的正整数, $m = m_1 m_2 \cdots m_k$, 那么同余式 $f(x) \equiv 0 \pmod{n}$ 与下述同余式组等价:

$$\begin{cases} f(x) \equiv 0 \pmod{m_1} \\ f(x) \equiv 0 \pmod{m_2} \\ \dots \\ f(x) \equiv 0 \pmod{m_k} \end{cases}$$

而且如果设同余式 $f(x) \equiv 0 \pmod{m_i} (i = 1, 2, \dots, k)$ 的解数为 n_i , 则同余式 $f(x) \equiv 0 \pmod{n}$ 的解数为 $n_1 n_2 \cdots n_k$.

所以假设当 $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ 时, 要求解同余式 $f(x) \equiv 0 \pmod{n}$, 我们只需要先求出 $f(x) \equiv 0 \pmod{p_i^{\alpha_i}} (i = 1, 2, \dots, k)$ 的解, 然后通过 CRT 就可以求出 $f(x) \equiv 0 \pmod{n}$ 的所有解了.

假设 p 是一个很大的质数, 如果想要在 sage 中求解 $f(x) \equiv 0 \pmod{p^\alpha} (\alpha \text{ 为大于 } 1 \text{ 的整数})$ 的解是比较困难的, 但是求解 $f(x) \equiv 0 \pmod{p}$ 是比较容易的, 所以我们可以通过求解 $f(x) \equiv 0 \pmod{p}$ 的解, 然后通过一些方式来求出 $f(x) \equiv 0 \pmod{p^\alpha}$ 的解, 下面介绍通过 Hensel 引理来通过同余式 $f(x) \equiv 0 \pmod{p}$ 的解来求 $f(x) \equiv 0 \pmod{p^\alpha}$ 的解.

Hensel 引理 设 p 是一个质数, $x \equiv x_{\alpha-1} \pmod{p}$ 为同余式 $f(x) \equiv 0 \pmod{p^{\alpha-1}}$ 的一个解 ($\alpha \geq 2$), 且 $f'(x_{\alpha-1}) \not\equiv 0 \pmod{p}$, 则在模 p 意义下必然存在一个正整数 $t_{\alpha-1}$ 使得 $f(x_{\alpha-1} + t_{\alpha-1} p^{\alpha-1}) \equiv 0 \pmod{p^\alpha}$.

Hensel 引理向我们表明了一个事实: 存在一种方法可以通过同余式 $f(x) \equiv 0 \pmod{p}$ 的解来求解 $f(x) \equiv 0 \pmod{p^\alpha}$ 的解, 通过 Hensel 引理的证明, 我们可以得到在引理中提及的 $t_{\alpha-1}$ 实际上满足:

$$t_{\alpha-1} \cdot f'(x_{\alpha-1}) \equiv -\frac{f(x_{\alpha-1})}{p^{\alpha-1}} \pmod{p} \quad (2.1.4.0.2)$$

由于 $x_\alpha \equiv x_{\alpha-1} + t_{\alpha-1} p^{\alpha-1} \equiv \cdots \equiv x_1 \pmod{p}$, 所以必然有 $f'(x_\alpha) \equiv f'(x_1) \pmod{p}$, 所以式 (2.1.4.2) 可以得到一解:

$$t_{\alpha-1} \equiv -\frac{f(x_{\alpha-1})}{p^{\alpha-1}} f'(x_{\alpha-1})^{-1} \pmod{p}$$

令该解为 $t_{\alpha-1} \equiv t'_{\alpha-1} \pmod{p}$, 则有 $t_{\alpha-1} = t'_{\alpha-1} + p t_\alpha (t_\alpha \in \mathbb{Z})$, 所以我们可以得到 $f(x_\alpha) \equiv 0 \pmod{p^\alpha}$ 的一解为:

$$x \equiv x_{\alpha-1} + p^{\alpha-1} (t'_{\alpha-1} + p t_\alpha) \equiv x_{\alpha-1} - f(x_{\alpha-1}) f'(x_{\alpha-1})^{-1} \pmod{p^\alpha}$$

通过 Hensel 引理, 假设同余式 $f(x) \equiv 0 \pmod{p}$ 的一个解为 $x \equiv x_1 \pmod{p}$ 我们可以得到同余式 $f(x) \equiv 0 \pmod{p^\alpha} (\alpha = 2, 3, \dots)$ 的解满足以下递推关系:

$$x_\alpha \equiv x_{\alpha-1} - f(x_{\alpha-1}) f'(x_{\alpha-1})^{-1} \pmod{p^\alpha}$$

结合上面的知识, 当遇到同余式 $f(x) \equiv 0 \pmod{n}$ ($n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$) 时, 我们就可以分别求解 $f(x) \equiv 0 \pmod{p_i}$ ($i = 1, 2, \dots, k$) 的解, 然后通过上面的递推关系和 CRT 来求出 $f(x) \equiv 0 \pmod{n}$ 的解了.

连分数

形如:

$$a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{\ddots + \frac{1}{a_n}}}}$$

的分数称为连分数, 为缩减篇幅方便书写, 一般常用符号

$$a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \cdots \frac{1}{a_n}}}$$

或者

$$[a_1, a_2, a_3, \dots, a_n]$$

来表示上面的连分数, 其中第二种表示方法比较常用. 而若 a_1 为整数, $a_2, a_3, \dots, a_n \cdots$ 为正整数, 则称连分数 $[a_1, a_2, a_3, \dots, a_n, \dots]$ 为简单连分数. 若简单连分数中 a 的个数有限, 则称其为有限简单连分数, 若 a 的个数无限, 则称其为无限简单连分数.

连分数中有一个比较重要的概念——渐进分数:

渐进分数 对于一个连分数 $[a_1, a_2, a_3, \dots, a_n]$, 我们定义其第 k ($1 \leq k \leq n$) 个渐进分数为:

$$\frac{p_k}{q_k} = [a_1, a_2, a_3, \dots, a_k]$$

其中 p_k 和 q_k 是整数, 且 $(p_k, q_k) = 1$ (即 $\frac{p_k}{q_k}$ 是一个既约分数).

对于无限简单连分数 $[a_1, a_2, a_3, \dots]$, 其第 k 个渐进分数同样按照如上定义, 而且若 $\lim_{k \rightarrow \infty} \frac{p_k}{q_k} = A$, 则称 A 为无限简单连分数 $[a_1, a_2, a_3, \dots]$ 的值.

实际上, 每一个连分数都表示一个实数, 而且每个实数都可以被表示为简单连分数 (有限或无限), 这在密码学应用中有着重要的意义.

下面介绍连分数的一个应用——求解佩尔 (Pell) 方程:

佩尔 (Pell) 方程 形如 $x^2 - dy^2 = 1$ 的方程称为佩尔方程, 其中 d 是一个正整数.

若 d 是一个完全平方数 (即 $\sqrt{d} \in \mathbb{Z}$), 则佩尔方程 $x^2 - dy^2 = 1$ 无正整数解, 否则该方程有无穷多组正整数解, 在下面的讨论中, 我们只讨论 d 不是完全平方数的情况.

对于佩尔方程 $x^2 - dy^2 = 1$, 若 x_0, y_0 是它的一组正整数解, 且 $x_0 + \sqrt{d}y_0$ 是形如 $x + \sqrt{d}y$ (x, y 均为 $x^2 - dy^2 = 1$ 的正整数解) 的最小数, 则方程的一切正整数解可以由:

$$x + \sqrt{d}y = (x_0 + \sqrt{d}y_0)^n$$

确定. 由此我们可以得到通过佩尔方程的最小正整数解 x_0, y_0 来求解佩尔方程的所有正整数解的迭代方法 (其中 $n \geq 1$):

$$\begin{cases} x_n = x_0 x_{n-1} + d y_0 y_{n-1} \\ y_n = x_0 y_{n-1} + y_0 x_{n-1} \end{cases}$$

表示为矩阵形式则为：

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} x_0 & dy_0 \\ y_0 & x_0 \end{bmatrix} \begin{bmatrix} x_{n-1} \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} x_0 & dy_0 \\ y_0 & x_0 \end{bmatrix}^n \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

若我们得到了佩尔方程任一组正整数解，我们均可以通过上面的递推关系得到所有的正整数解，而如果想要得到佩尔方程的一组正整数解，我们需要使用到连分数，实际上，若将 \sqrt{d} 展开为连分数，得到 $[a_1, a_2, \dots, a_n, \dots]$ ，我们总能找到找到 a_{n+1} 满足 $a_n = 2a_1$ ，则 \sqrt{d} 的第 n 个渐进分数 $[a_1, a_2, \dots, a_n] = \frac{x}{y}$ 所确定的 x, y 就可能为佩尔方程 $x^2 - dy^2 = 1$ 的一组解（注意，并不是所有满足条件的渐进分数都确定该方程的一组正整数解，在实际应用中，对于每一个满足上述条件的渐进分数，我们均需要将其分子 x ，及分母 y 代入方程中进行测试，直到符合方程为止）。

2.2 线性代数

2.2.1 矩阵与行列式

2.2.2 线性变换

2.2.3 向量空间

2.3 抽象代数

抽象代数 (又称近世代数), 是代数学的一个分支, 其最根本的任务是研究各种抽象的代数结构以及其性质, 其为密码学提供数学基础和工具, 例如在 AES 加密算法中, 我们会用到抽象代数中有限域的知识, 而在椭圆曲线密码学中, 则会经常用到抽象代数中群论的知识.

2.3.1 集合

2.3.2 代数系统

2.3.3 群

2.3.4 环

2.3.5 域

2.3.6 格

Chapter 3

对称密码

对称密码指的是加密和解密使用相同密钥的密码算法，其加解密过程大致如下图所示：

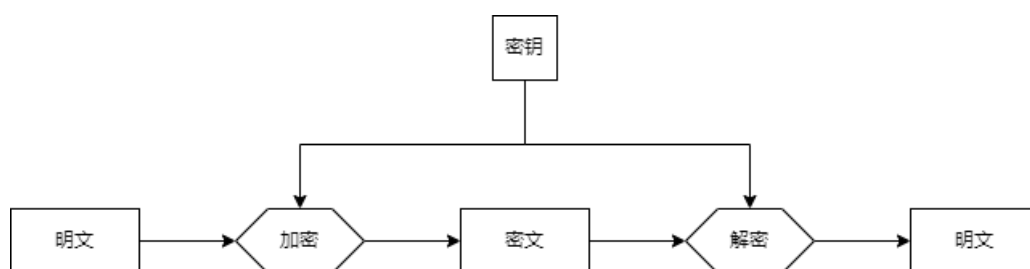


图 3.1: 对称密码加解密过程

在分类上，对称密码可以分为流密码（又称序列密码）和分组密码，

3.1 流密码

3.2 分组密码

Chapter 4

非对称密码

非对称密码指的是加密和解密使用不同密钥的密码算法，其加解密过程大致如下图所示：

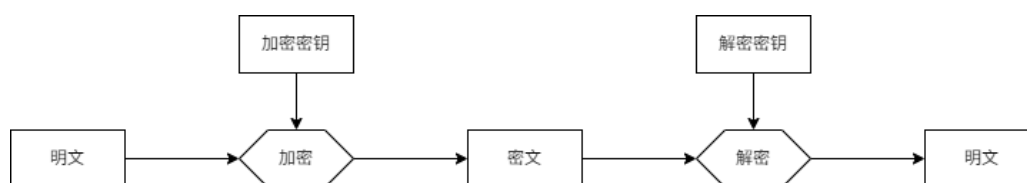


图 4.1: 非对称密码加解密过程

上图中加密密钥与解密密钥不相同，在加密密钥和解密密钥之中必有一个是公开的，另一个是保密的，一般称公开的密钥为公钥，保密的密钥为私钥，故有时候非对称密码也被称为公钥密码。

在 CTF 比赛中常见的非对称密码有 RSA、ECC(Elliptic Curve Cryptography) 等，其中 RSA 是 CTF 中最常见且最基础的非对称密码算法。

4.1 RSA

RSA 算法是由 Ron Rivest、Adi Shamir 以及 Leonard Adleman 三人于 1977 年提出的非对称加密算法，其安全性基于大数分解的困难性，一般情况下，RSA 算法的加解密步骤如下：

加密：

1. 选择两个大质数 p, q ，计算 $n = pq$ ，以及 n 的欧拉函数 $\varphi(n) = (p-1)(q-1)$;
2. 选择一个整数 e (一般选取 $e = 65537$)，使得 $1 < e < \varphi(n)$ 且 $(e, \varphi(n)) = 1$;
3. 计算 d 使得 $ed \equiv 1 \pmod{\varphi(n)}$ ，即 d 是 e 关于模 $\varphi(n)$ 的乘法逆元;
4. 对于明文 m ，计算密文 c ，即 $c \equiv m^e \pmod{n}$;
5. 将公钥对 (n, e) 公开，私钥对 (n, d) 保密.

解密：

1. 对于密文 c ，通过私钥 (n, d) 计算明文 m ，即 $m \equiv c^d \pmod{n}$.

RSA 算法的安全性基于大数分解的困难性，在一般情况下，我们已知公钥 (n, e) 的时候，是很难通过分解 n 得到 p, q ，从而计算出私钥 $d \equiv e^{-1} \pmod{\varphi(n)}$ 的。

实际上在 CTF 比赛中，出题人提供的 RSA 加密算法往往是有一定的缺陷的，在本节中，我们将主要介绍一些基础的对 RSA 算法的攻击手段。

4.1.1 直接分解模数 n

尽管先前提到 RSA 算法的安全性基于大数分解的困难性，但是在 CTF 比赛中，RSA 加密过程中选取的素数 p, q 可能较小，或者生成的质数存在某些特殊性质，从而导致直接分解 n 变得可行，常见的分解方法有如下几种：

采用工具分解

在 CTF 比赛中，RSA 加密过程中生成的质数可能会比较小，抑或者质因数分解结果被人为地加入了某些数据库中公开，这种情况下，我们可以直接使用一些工具来尝试分解模数 n ，常见的工具有 factordb(<http://www.factordb.com/>)、yafu 等。

factordb 是一个在线的质因数分解数据库，对于一个数 n ，我们可以尝试在 factordb 中搜索 n ，若数据库中不存在该数的分解结果，则可以直接使用一些工具来尝试分解模数 n ，常见的工具有 factordb 上搜索这个数：



图 4.2: 使用 factordb 搜索 463901 的质因数

搜索可以得到如下结果：

Search	Sequences	Report results	Factor tables	Status	Downloads	API	Login
463901							
Factorized							
factordb.com - 4 queries to generate this page (0.00 seconds) (limits) (Privacy Policy) (Imprint) (Imprintsum)							
Result:							
status (2)	digits	number					
FF	6 (show)	463901 = 523 * 887					

图 4.3: factordb 搜索结果

此时我们可以得到 463901 的质因数分解结果为 523×887 。但是并非所有的数都能在 factordb 中找到分解结果，若 factordb 中没有该数的分解结果，对于下面这个由两个 512 bits 的质数相乘得到的大整数 n ：

```
8320573729512995489018533734363020219212422510765051110587777028952826122698
8719710929077482516264118525947936654690138203865762947587961213513161052433
7776321996760960256340875966659583104019116897853228345370770890781441937044
8432857596083737050341679106062233488210113949748561973115754159540001230771
9799
```

图 4.4: 两个 512 bits 的质数相乘得到的大整数 n

此时我们尝试使用 factordb 搜索该数，会得到如下结果：

Search	Sequences	Report results	Factor tables	Status	Downloads	API	Login
(832057372951299548901853373436302021921242251076505111058777702895282612269887197109290774825162641) Factorized							
Result:							
status (2)	digits	number					
C*	308 (show)	8320573729...99_308* = 8320573729...99_308*					

图 4.5: factordb 无法搜索出结果

显然, factordb 无法搜索出该数的分解结果, 一般在求解 RSA 问题的时候遇到这种情况就说明我们可能需要采用其它的方法来求解这道 RSA 问题了.

如果想要更方便地使用 factordb 来分解, 我们可以通过 Python 调用其 API 来实现自动化分解, 有开发者利用 factordb 的 API 开发了一个名为 factordb-python 的库, 安装方式可以参考 factordb-python, 举个例子, 我们以 LitCTF 2023 中 factordb 题目为例, 题目给出如下数据:

```
1 e = 65537
2 n = 87924348264132406875276140514499937145050893665602592992418171647042491658461
3 c = 87677652386897749300638591365341016390128692783949277305987828177045932576708
```

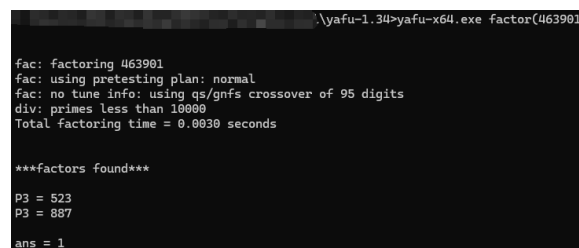
那么我们可以通过 factordb-python 库来分解模数 n , 代码如下:

```
1 from Crypto.Util.number import *
2 from factordb.factordb import FactorDB # 导入factordb-python库
3
4 e = 65537
5 n = 87924348264132406875276140514499937145050893665602592992418171647042491658461
6 c = 87677652386897749300638591365341016390128692783949277305987828177045932576708
7
8 f = FactorDB(n) # 创建factordb对象
9 f.connect() # 连接到factordb
10 factors = f.get_factor_list() # 获取质因数列表
11
12 p = factors[0]
13 q = factors[1]
14
15 phi = (p - 1) * (q - 1)
16 d = inverse(e, phi)
17 m = pow(c, d, n)
18
19 print(long_to_bytes(m))
```

运行代码即可得到 flag.

除 factordb 之外, 我们还可以使用 yafu 等工具来分解模数 n , yafu 是一个开源的质因数分解工具, 支持多种分解算法, 下载地址<https://sourceforge.net/projects/yafu>.

这个工具需要通过命令行来使用, 对于给定的整数 n , 我们可以在 yafu 目录下的命令行中输入 `yafu-x64.exe factor(n)` (其中 n 在实际使用中换成一个具体的数), 例如对于先前的 $n = 463901$, 我们可以在 yafu 目录下的命令行中输入 `yafu-x64.exe factor(463901)`, 得到如下图所示结果:



```

.\yafu-1.34>yafu-x64.exe factor(463901)

fac: factoring 463901
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
Total factoring time = 0.0030 seconds

***factors found***

p3 = 523
p3 = 887

ans = 1
  
```

图 4.6: yafu 分解结果

参考文献

- [1] Joseph H. Silverman. 数论概论（第四版）[M]. 北京：机械工业出版社, 2020.
- [2] 闵嗣鹤, 严士健. 初等数论（第四版）[M]. 北京：高等教育出版社, 2020.
- [3] 张苍. 九章算术 [M]. 邹涌, 译解. 重庆：重庆出版社, 2016.
- [4] Michael O Rabin. Probabilistic algorithm for testing primality[J]. *Journal of Number Theory*, 1980, 12(1):128-138.
- [5] 杨子胥. 近世代数（第四版）[M]. 北京：高等教育出版社, 2020.
- [6] Christof Parr, Jan Pelzl. 深入浅出密码学——常用加密技术原理及应用 [M]. 马小婷, 译. 北京：清华大学出版社, 2012.