南開大學

信息安全数学基础探究报告

学院：	网络空间安全学院
专业：	信息安全
学号：	2311208
姓名：	魏来
班级：	一班

目录

一、 大整数分解问题 1

1、 数学问题 1

2、 算法分析 1

2.1、 试除法（Eratosthenes 筛法） 1

2.2、 费马分解算法 2

2.3、 Pollard ρ 分解算法 3

2.4、 Pollard ρ-1 分解算法 5

2.5、 其他效率更高算法的原理概览 6

3、 在密码学中的应用 6

二、 RSA 问题 8

1、 总述 8

2、 数学问题 8

3、 RSA 公钥加密算法 8

3.1、 预备知识 8

3.2、 算法分析 10

4、 常见 RSA 攻击和防范 15

4.1、 针对模数 N 的攻击 15

4.2、 针对参数 e 的攻击 21

4.3、 针对参数 d 的攻击 27

5、 在密码学中的应用 31

三、 零知识证明问题 33

1、 对密码原语的介绍 33

1.1、 历史缘起 33

1.2、 密码学背景 33

1.3、 零知识的含义 34

1.4、 三个核心性质 34
1.5、 四个证明阶段 35
1.6、 零知识证明的类型 35
2、 具体的应用场景 38
2.1、 隐私保护的身份验证 38
2.2、 区块链与加密货币 38
2.3、 数字版权保护与数字内容认证 39
2.4、 电子投票系统 39
2.5、 金融与支付系统 39
2.6、 医疗数据的隐私保护 39
3、 其中包含的数学问题 40
3.1、 同态加密与承诺方案 40
3.2、 椭圆曲线与离散对数 44

一、 大整数分解问题

1、 数学问题

给定一个大整数 N，它是两个大素数的乘积，但其因子 p 和 q 未知，我们将寻找 p 和 q，使其满足 $N = p \cdot q$ 的问题称作大整数分解问题。尽管整数分解问题容易理解和解决，但寻找高效的大整数分解算法却困难重重。随着近些年计算机计算能力的飞速提升和密码学的广泛应用，大整数分解已然成为世界关注的热点问题。

2、 算法分析

2.1、 试除法（Eratosthenes 筛法）

a. 算法思想

试除法是最早也是最简单的整数分解方法。试除法用素数表里小于 $\sqrt{n}$ 的素数一个一个地去验证其是否整除 n.

b. 算法核心伪代码

```
int factorize(int n, int* prime_list, int* factors)
{
    if (n <= 1)
        return 0;
    else
    {
        int j = 0;
        for (int i = 0; prime_list[i] * prime_list[i] <= n; i++)
        {
            while (n % prime_list[i] == 0)
            {
                factors[j] = prime_list[i];
                j++;
                n = n / prime_list[i];
            }
        }
        if (n > 1)
            factors[j] = n;
        j++;
        return j;
    }
}
```

c. 算法特点

试除法肯定能完全分解大整数，但是效率却非常低。这种方法所需的计算复杂度为 $O(\frac{\sqrt{n}}{\ln n})$.

2.2、 费马分解算法

a. 算法思想

对于奇整数 n，当我们能够获得方程

$$n = x^2 - y^2$$

的整数解时，我们也就获得了 n 的两个因子，因为

$$n = (x - y)(x + y)$$

反过来，当获得如下整数分解

$$n = ab \quad (a \geq b \geq 1)$$

的时候，我们也就获得了上面方程的整数解，因为

$$n = \left(\frac{a+b}{2}\right)^2 - \left(\frac{a-b}{2}\right)^2$$

该方法过程为，首先确定最小的整数 k，使得 $k^2 \geq n$。然后，对下面的数列

$$k^2 - n, (k+1)^2 - n, (k+2)^2 - n, (k+3)^2 - n, \cdots, ((n+1)/2)^2 - n$$

按顺序进行测试，直到找到一个整数 $m \geq \sqrt{n}$ 使得 $m^2 - n$ 是一个平方整数，从而也就找到了一对因子，如果一直运行到上面数列最后一个数才找到平方整数，那么 n 就是素数没有非平凡因子。

b. 算法核心伪代码

```
#include <cmath>

bool is_square(int x)
{
    int s = (int)sqrt(x);
    return s * s == x;
}

int factorize(int n, int& factor1, int& factor2)
{
```

```
    if (n <= 1)
        return 0;
    else
    {
        int k = (int)ceil(sqrt(n));
        for (int i = 0;; i++)
        {
            int m = (k + i) * (k + i) - n;
            if (is_square(m))
            {
                factor1 = k + i - (int)sqrt(m);
                factor2 = k + i + (int)sqrt(m);
                if (factor1 == 1 || factor2 == n)
                    return 1;
                else
                    return 2;
            }
        }
    }
}
```

c. 算法特点

该算法使分解大整数的问题转化成了寻找满足要求的平方数，大大提高了分解的效率。虽然费马分解效率比试除法高得多，但随着要分解的大整数不断变大，费马算法的效率依然显得很低。这种方法所需的计算复杂度为 $O(n)$.

2.3、 Pollard ρ 分解算法

a. 算法思想

首先，确定一个简单的二次以上整系数多项式，例如 $f(x) = x^2 + a\ a \neq -2\ 0$。然后，从一个初始值 x_0 开始，利用迭代公式

$$x_{k+1} \equiv f(x_k) \pmod{n}$$

计算一个序列 $x_1, x_2, x_3 \cdots$ 令 d 为 n 的一个非平凡因子，因为模 d 的剩余类个数比模 n 的剩余类个数少很多，所以很可能存在 x_k 和 x_j 属于同一个模 d 的剩余类，同时又属于不同的模 n 的剩余类. 因为 $d|(x_k - x_j)$ 而 $n \nmid (x_k - x_j)$，所以 $((x_k - x_j), n)$ 是 n 的非平凡因子。如果在该算法中，我们对序列中的任意两个量的差都进行计算的话，那么计算的开销就太大了，改进方法是只对 $(x_{2k} - x_k)$ 进行计算.

b. 算法核心伪代码

```cpp
#include <cmath>

int gcd(int a, int b)
{
    if (b == 0)
        return a;
    return gcd(b, a % b);
}

int F(int x, int a, int n)
{
    return (x * x + a) % n;
}

int pollard_rho(int n, int x0, int a, int steps)
{
    if (n % 2 == 0)
        return 2;
    int x = x0;
    int y = x;
    int d = 1;
    for (int i = 0; i < steps && d == 1; i++)
    {
        x = F(x, a, n);
        y = F(F(y, a, n), a, n);
        d = gcd(y - x, n);
    }
    if (d > 1 && d < n)
        return d;
    else
        return -1;
}
```

c. 算法特点

该算法基于生日悖论，具有简单且易于实现，内存需求低等优点，适合中小规模数的分解。然而，当它在遇到非常大的质因子时效率较低，其时间复杂度为 $O(n^{1/4})$。对于某些具有特殊结构的数（如 Carmichael 数、RSA 模数等），Pollard ρ 算法的表现可能不佳。此外，由于它是一种概率算法，不能保证每次都能成功分解复合数。因此，如

果算法在运行预先规定的步数后不成功，可以重新选择 x_0 或者 $f(x)$ 再开始处理。

2.4、 Pollard ρ-1 分解算法

a. 算法思想

这个方法对如下的情况起作用：奇合数 n 有一个素因子 ρ，而且 $\rho-1$ 是小素数之积。

该算法需要预先选择一个整数 k，只要 k 充分大，就可以保证 $(\rho-1)|k!$，接着选择一个整数 a，使 $1<a<\rho-1$，计算 $a^{k!}\equiv m \pmod{n}$。因为存在整数 j，使 $k!=j(\rho-1)$，所以

$$m\equiv a^{k!}\equiv a^{j(\rho-1)}\equiv\left(a^{\rho-1}\right)^{j}\equiv 1^{j}\equiv 1 \pmod{\rho}$$

也就是 $\rho|(m-1)$，因此 $(m-1,n)>1$，只要 $m\not\equiv 1 \pmod{n}$，$(m-1,n)$ 必为 n 的非平凡因子。

b. 算法核心伪代码

```cpp
#include <cmath>

int gcd(int a, int b)
{
    if (b == 0)
        return a;
    return gcd(b, a % b);
}

int mod_exp(int a, int k, int n) {
    int result = 1;
    while (k > 0) {
        if (k % 2 == 1)
            result = (result * a) % n;
        a = (a * a) % n;
        k /= 2;
    }
    return result;
}

int pollard_rho_minus_one(int n, int a, int k) {
    int m = mod_exp(a, tgamma(k + 1), n);
    int d = gcd(m - 1, n);
    if (d > 1 && d < n) {
```

```
        return d;
    }
    else {
        return -1;
    }
}
```

c. 算法特点

该算法基于费马小定理，是一种概率算法，不能保证每次都能成功分解复合数。其时间复杂度为 $O(n^{1/4})$。在实际计算时，通常取 a 为 2 或 3 等小素数，k 取 20 以内的一些小数，防止阶乘过大。

2.5、 其他效率更高算法的原理概览

虽然以上几种基础算法能够解决一些大整数分解问题，但它们都有一定的局限性。在它们被提出的几年后，科学家又先后提出椭圆曲线法、连分式分解法、二次筛法（当前用于分解十进制位数小于 110 位的整数的最快方法）和数域筛法（目前运行效率最高的整数分解算法之一）。随后 Shor 算法实现了量子计算环境下在多项式时间内分解大整数。

以下是二次筛法和数域筛法等算法的基本思路：

如果我们能够找到两个整数 x 和 y，满足

$$x^2 \equiv y^2 \pmod{n}$$

且

$$x \not\equiv \pm y \pmod{n}$$

那么 $(x-y,n)$ 和 $(x+y,n)$ 是 n 的非平凡因子。

连分数分解算法、二次筛法和数域筛法之间的不同主要在于如何构造满足上面同余条件的两个整数 x 和 y。

3、 在密码学中的应用

大整数分解在密码学中的应用，尤其是在公钥加密系统中，具有极其重要的作用。其核心概念是基于大整数分解的难度，这成为了许多加密算法安全性的基础。

最典型的例子是 RSA 加密算法，它的安全性依赖于将一个由两个大质数相乘形成的数分解为质因数的困难性。虽然乘法很容易进行，但要从结果反推出原始质数却是极其困难的，尤其当数值非常大时，分解的难度成指数级增长。因此，只要质数足够大，未经授权的人几乎不可能通过分解获得加密信息。这就是为什么 RSA 等加密系统在现

代通信中广泛应用，如保护电子邮件、数字签名等。反之，如果能够有效地将大整数分解为其素数因子，那么就可以破译使用该密钥的加密信息。

随着量子计算的发展，Shor 算法的出现使得这一分解过程在量子计算环境下可以变得可行，因此对现有加密算法提出了潜在的威胁。

总的来说，大整数分解问题在密码学中的应用主要是用于确保信息传输的安全，但随着计算能力的不断进步，尤其是量子计算的潜力，未来的加密技术将需要进一步发展来应对新的挑战。

二、 RSA 问题

1、 总述

在探究报告一中我说到，大整数分解问题的一个重要应用是 RSA 加密算法，在本次报告中，我将具体探究 RSA 问题。RSA 问题是 Rivest、Shamir 和 Adleman 三位密码学家于 1978 年在著名的 RSA 公钥密码体制中提出的，其反映了 RSA 加密算法（或字签名算法）的安全等级。尽管目前仍没有人能够证明解决 RSA 问题的困难度与解决整数分解问题的困难度相同，我们仍然认为 RSA 问题是一个困难问题。这样的话，我们就可以基于 RSA 问题的困难度来设计 RSA 加密算法，换句话说，如果破解 RSA 加密算法的困难度等同于解决 RSA 问题的难度，那么我们可以将 RSA 加密算法看作是安全的。

2、 数学问题

定义 2.2.1 令 p 和 q 是两个比特长度相近的大素数。若 $n = pq$ 是长度至少为 1024 比特的整数，并且 $p-1$ 和 $q-1$ 有大素数因子，则称 n 为成熟合数 (Ripe Composite Number)。

定义 2.2.2(RSA 问题) 令 $n = pq$ 是一个成熟合数，e 是一个正奇数且满足 $(e, \varphi(n)) = 1$。给定一个随机整数 $c \in \mathbb{Z}_n^*$，我们将寻找一个整数 m 使其满足 $m^e \equiv c \pmod{n}$ 的问题称作 RSA 问题。

3、 RSA 公钥加密算法

3.1、 预备知识

RSA 算法基于一个十分简单的数论事实：将两个大质数相乘十分容易，但是想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。RSA 算法是现今使用最广泛的公钥密码算法，也是号称地球上最安全的加密算法。接下来我将对一些基本的概念和理论进行解释。

a. 密码分类

根据密钥的使用方法，可以将密码分为对称密码和公钥密码：

对称密码：加密和解密使用同一种密钥的方式；

公钥密码：加密和解密使用不同的密码的方式，因此公钥密码通常也称为非对称密码。

在公钥密码系统中，这两个不同的密钥之间存在着相互依存关系：即用其中任一个密钥加密的信息只能用另一个密钥进行解密。这使得通信双方无需事先交换密钥就可进行保密通信。其中加密密钥和算法是对外公开的，人人都可以通过这个密钥加密文件然

后发给收信者，这个加密密钥又称为公钥；而收信者收到加密文件后，它可以使用他的解密密钥解密，这个密钥是由他自己私人掌管的，并不需要分发，因此又成称为私钥。

b. 公钥体制算法的思想

有上述公钥体制的任务中我们知道，区别于其他密钥体制算法，公钥算法需要做到的是将加密密钥（公钥）和加密算法公开，破坏者也不能由公钥和密文破译出明文。只有使用解密密钥（私钥）和解密算法才能解出明文。而且保证通过公钥不能推出私钥。

上述内容可以通过所谓的“单向函数”来实现（传统的密钥体制中加密算法和解密算法是互逆的）。所谓“单向函数”是指加密函数 E 和解密函数 D。但是已知加密函数 E 推导其逆运算却非常困难（也就是推导私钥的过程）。所以若不知道解密函数（或私钥）不可能解出明文。

单向函数的实现依赖的是大整数因子分解的难度，根据算数基本定理：任何大于 1 的整数都可以分解成素数乘积的形式，并且，如果不计分解式中素数的次序，该分解式是唯一的。这个定理在理论上十分漂亮，但是操作起来却非常困难。下表列出了现代最快的分解算法在大型计算机上分解一个大数所用的时间。

整数的位数	操作次数	所需时间
50	1.4×10^{10}	3.9 小时
75	9.0×10^{12}	104 天
100	2.3×10^{15}	74 年
200	1.2×10^{23}	3.8×10^{9} 年
300	1.3×10^{29}	4.9×10^{12} 年
500	1.3×10^{35}	4.2×10^{17} 亿年

图 1: 现代最快的分解算法在大型计算机上分解一个大数所用的时间

c. 相关数学名词和数学理论

数学名词（具体内容详见课本）：

素数、互素、同余、模运算及其性质、欧拉函数、欧拉定理、费马小定理

数学推论：

推论 2.1 若 n 可以拆成两个互质的正整数之积，如 $n = p \times q$，则有：

$$\phi(n) = \phi(pq) = \phi(p)\phi(q)$$

推论 2.2 对质数 m，有：

$$\phi(m) = m - 1$$

推论 2.3 若 n 可以拆解成两个质数 p 和 q 的积，则：

$$\phi(n) = (p-1)(q-1)$$

推论 2.4 如果 p 是一个素数，则：

$$\phi(p^e) = p^e - p^{e-1}$$

定义 2.3.1 模反元素：如果两个正整数 a 和 n 互质，则一定有整数 b，使得：ab - 1 能被 n 整除，或者说 ab 被 n 整除的余数是 1，记作：

$$ab \equiv 1 \pmod{n}$$

3.2、 算法分析

a. 算法思想

RSA 公钥加密算法是 Rivest, Shamir 和 Aldeman 于 1977 年提出的一种公钥加密体制, 是当前最常用的公钥加密算法之一。RSA 算法包括密钥生成, 加密和解密三个过程。

密钥生成过程包括如下步骤: 随机生成两个大素数 p 和 q (通常来说, 选取的 p 和 q 的比特长度相同); 计算 $N = p \cdot q$, 称作 RSA 的模数; 计算 N 的欧拉函数 $\varphi(N) = (p-1)(q-1)$; 随机选取一个整数 e, 其满足 $1 < e < \varphi(N)$ 且 $\gcd(e, \varphi(N)) = 1$; 利用扩展欧几里得算法计算得到一个整数 d, 使其满足 $d \cdot e \equiv 1 \pmod{\varphi(N)}$。令 (N, e) 为公钥, (p, q, d) 为私钥。

加解密过程如下: 如果用户 B 想要将消息 m 的密文发送给用户 A，那么 B 将使用 A 的公钥来计算得到密文 $C = m^e \pmod{N}$，然后将密文 C 发送给 A。A 接收到密文 C 后，使用私钥 d 来计算出明文 $m = C^d \pmod{N}$。

b. 算法原理图解

RSA 密钥的生成过程如下图：

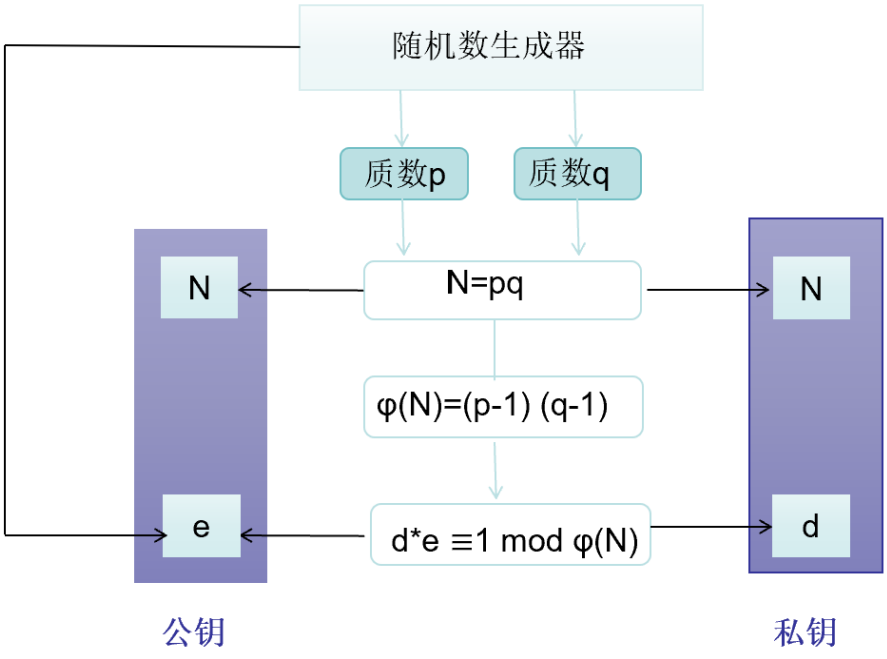

图 2: RSA 密钥的生成过程

RSA 加密算法的具体步骤如下图：

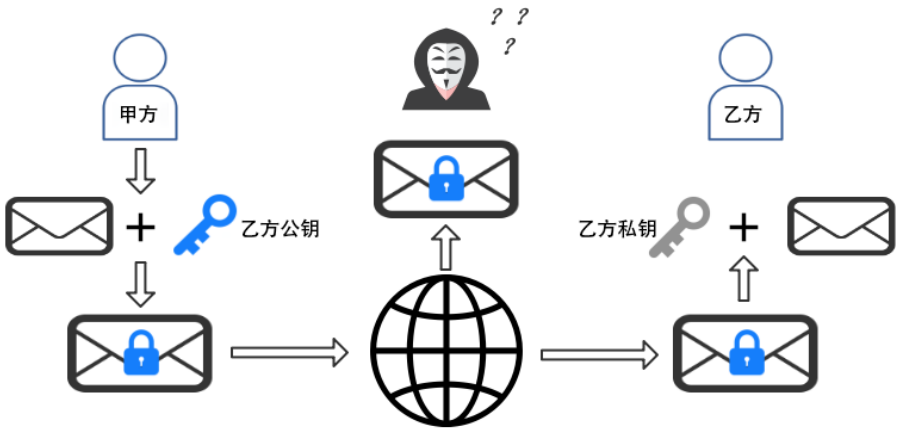

图 3: RSA 加密算法

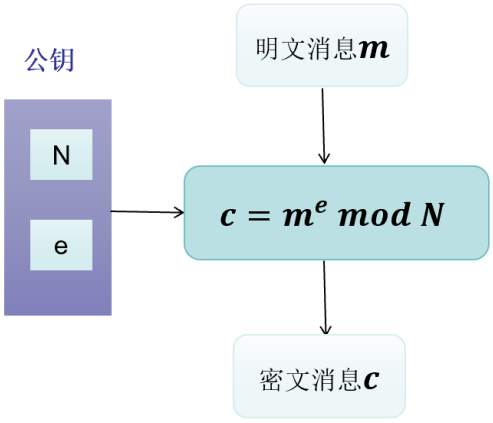

图 4: RSA 加密过程

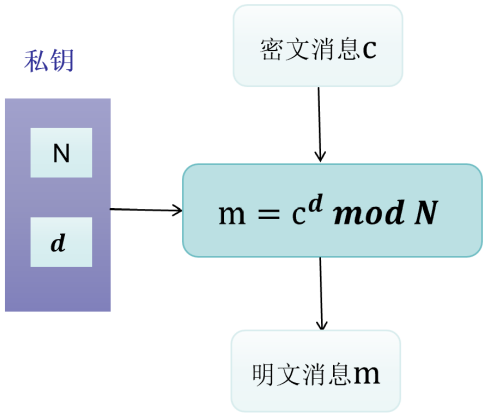

图 5: RSA 解密过程

1. **选择两个大素数**：首先，接收方选择两个大素数 p 和 q，这些素数用于生成公钥和私钥。

2. **计算模数** n：然后，计算 $n = p \times q$。这个 n 会成为公钥的一部分，并且也是私钥计算的一部分。

3. **计算欧拉函数** $\varphi(n)$：接下来，计算欧拉函数 $\varphi(n) = (p-1) \times (q-1)$，它表示小于 n 且与 n 互素的数的个数。

4. **选择公钥指数** e：选取一个整数 e，使其满足 $1 < e < \varphi(n)$ 且 e 与 $\varphi(n)$ 互素。这个 e 是加密过程中使用的公钥指数，通常选择较小的常数值（例如 65537），以提高加密效率。

5. **计算私钥指数** d：通过扩展欧几里得算法，计算私钥指数 d，使其满足

$$d \times e \equiv 1 \pmod{\varphi(n)}$$

这意味着 d 是 e 在模 $\varphi(n)$ 下的乘法逆元。

6. **发布公钥，保留私钥**：接收方将 (n, e) 作为公钥发布给外界，而 d 则作为私钥保留，只有接收方知道 d 的值。

c. 算法证明

我们要证明的是 $x^{ed} \equiv x \pmod{n}$，由于 $n = p \times q$，且 p 和 q 是素数，因此可以分别证明：

$$x^{ed} \equiv x \pmod{p} \quad (1)$$

和

$$x^{ed} \equiv x \pmod{q} \quad (2)$$

接下来利用中国剩余定理将二者合并即可得到 $x^{ed} \equiv x \pmod{n}$。

首先证明式 (1)。根据欧拉定理，如果 x 与 p 互素，则有

$$x^{\varphi(p)} \equiv 1 \pmod{p}$$

由于 $\varphi(p) = p - 1$，我们有：

$$x^{p-1} \equiv 1 \pmod{p}$$

接下来，考虑 $ed - 1 \equiv 0 \pmod{\varphi(n)}$，即 $ed = k(p-1)(q-1) + 1$ 对某个整数 k 成立。因此：

$$x^{ed} = x^{k(p-1)(q-1)+1} = x \cdot \left(x^{(p-1)(q-1)}\right)^k$$

根据费马小定理，$x^{p-1} \equiv 1 \pmod{p}$，所以 $x^{ed} \equiv x \pmod{p}$。

同样的道理可以应用到模 q 的情况，因此：

$$x^{ed} \equiv x \pmod{q}$$

根据中国剩余定理，得出：

$$x^{ed} \equiv x \pmod{n}$$

这证明了 RSA 解密过程的正确性。

d. 算法核心伪代码

```
// 生成公钥和私钥
int prime1 = generateLargePrime(); // 生成第一个大素数 prime1
int prime2 = generateLargePrime(); // 生成第二个大素数 prime2
```

```
int modulus = prime1 * prime2; // 计算 n = prime1 * prime2, 作为公钥的一部分
int totient = (prime1 - 1) * (prime2 - 1); // 计算欧拉函数 (n)
int publicKey; // 加密所用的公钥指数 e
do {
    publicKey = randomValue(2, totient - 1); // 生成与 (n) 互质的随机整数 e
} while (calculateGCD(publicKey, totient) != 1); // 确保 e 与 (n) 互素
int privateKey = calculateModInverse(publicKey, totient); // 通过扩展欧几里得算法计算私钥 d

// 加密过程
int plaintext = getPlaintextMessage(); // 获取待加密的明文消息
int ciphertext = fastModularExponentiation(plaintext, publicKey, modulus); // 加密, 计算密文 ciphertext = (plaintext^e) mod n

// 解密过程
int decryptedMessage = fastModularExponentiation(ciphertext, privateKey, modulus); // 解密, 计算明文 decryptedMessage = (ciphertext^d) mod n

// 辅助函数

// 计算 a 和 b 的最大公约数 (GCD)
int calculateGCD(int a, int b) {
    if (b == 0) {
        return a;
    } else {
        return calculateGCD(b, a % b);
    }
}

// 扩展欧几里得算法, 计算 a 在模 m 下的乘法逆元
int calculateModInverse(int a, int m) {
    int m0 = m, x0 = 0, x1 = 1; // 初始变量
    if (m == 1) {
        return 0; // 如果模数为1, 没有逆元
    }
```

```
    while (a > 1) { // 当 a 大于 1 时，继续计算
        int quotient = a / m; // 计算商
        int temp = m;
        m = a % m; // 更新 m
        a = temp; // 更新 a
        temp = x0;
        x0 = x1 - quotient * x0; // 更新 x0 和 x1
        x1 = temp;
    }
    if (x1 < 0) {
        x1 += m0; // 确保结果为正数
    }
    return x1; // 返回模逆元
}

// 快速幂算法，计算 (base^exponent) mod modulus
int fastModularExponentiation(int base, int exponent, int modulus) {
    int result = 1; // 初始值为 1
    base = base % modulus; // 处理 base 可能大于 modulus 的情况
    while (exponent > 0) {
        if (exponent % 2 == 1) { // 如果 exponent 是奇数
            result = (result * base) % modulus; // 更新 result
        }
        base = (base * base) % modulus; // base 自乘
        exponent /= 2; // 指数减半
    }
    return result; // 返回结果
}

// 生成大素数（伪函数，用于示意）
int generateLargePrime() {
    // 使用合适的算法生成大素数，例如米勒-拉宾素性测试
    return pseudoRandomPrime();
}

// 获取待加密的明文消息（伪函数，用于示意）
int getPlaintextMessage() {
    // 从用户输入或文件中获取待加密的明文消息
    return inputMessage();
```

```
}

// 生成随机数（伪函数，用于示意）
int randomValue(int min, int max) {
    // 返回一个在 min 和 max 之间的随机整数
    return pseudoRandom(min, max);
}
```

e. 算法特点

RSA 算法通过生成大素数、计算模数和公私钥对实现安全的加密解密过程，其主要特点是基于大整数分解的困难性。尽管 RSA 算法本身是安全的，但如果在实现细节上出现漏洞，可能会导致攻击成功。因此，构建安全的 RSA 系统时，除了需要生成足够大的素数，还应严格选择和配置参数（如模数 N、加密密钥 e 和解密密钥 d）。接下来，我们将讨论常见的 RSA 攻击方式及其防范措施，确保算法的安全性。

4、 常见 RSA 攻击和防范

4.1、 针对模数 N 的攻击

RSA 系统的安全性通常被认为与大整数因式分解问题等同，即如果能够分解 N 的因子，就可以破解 RSA 密码系统。因此，N 参数的选择对 RSA 系统的安全性至关重要。在 RSA 中，模数 N 是由两个大素数 p 和 q 的乘积得到的，即 $n = p \times q$。如果攻击者能够分解 N 得到 p 和 q，就可以计算出 $\varphi(N) = (p-1)*(q-1)$，进而通过公开的 e 和 $ed \equiv 1(mod\varphi(N))$ 解出解密密钥 d。因此，RSA 系统中模数 N 的选择与其安全性密切相关。一般来说，确定模数 N 时可遵循以下原则：

（1）p 和 q 的差值应较大：

当 p 和 q 相差不大时，在已知 N 的情况下，可以假设它们的平均值计算公式为：

$$\frac{p+q}{2} = \sqrt{a}$$

然后利用以下等式：

$$(\frac{p+q}{2})^2 - N = (\frac{p-q}{2})^2$$

如果 $(\frac{p+q}{2})^2$ 可以开方，就能解出 $\frac{p+q}{2}$ 和 $\frac{p-q}{2}$，从而达到分解 N 的目的。

（2）p-1 和 q-1 的最大公因数应尽可能小：

若 $gcd((p-1),(q-1))$ 较大，可采用迭代方法攻击。对密文 $c = m^e mod n$ 重复进行 e 次幂运算：c^e, c^{ee}.....，直到 c 的 e^t 次幂模 n 等于 c 为止。此时，c 的 e^{t-1} 次幂模 n 即为 m。当 t 不大时，这种攻击方法较为有效。根据欧拉定理：

$$e^t = 1 mod \varphi(n)$$

可推导出 t 的最小值：

$$t = \varphi(\varphi(n)) = \varphi((p-1)(q-1))$$

因此，如果 $gcd((p-1),(q-1))$ 较小，$\varphi(\varphi(n))$ 就会很大，导致 t 值较大，使得迭代攻击方法效果不佳，从而增强加密的安全性。

（3）p 和 q 应为强素数：

对于素数而言，如果满足以下条件，则称之为强素数：

强素数的定义：一个素数如果满足以下条件，则被称为强素数：

条件一：存在两个大素数 p_1 和 p_2，使得 $p_1|p-1$ 且 $p_2|p+1$。

条件二：存在四个大素数 r_1、r_2、s_1、s_2，使得 $r_1|p_1-1$、$s_1|p_1+1$、$r_2|p_2-1$、$s_2|p_2+1$。

在这里，r_1、r_2、s_1、s_2 被称为三级素数，p_1、p_2 被称为二级素数，p 则被称为一级素数。只有两个强素数的积所构成的 N，其因子分解才是一个困难的数学问题。

如果不是强素数，则分解 n 相对容易。证明如下：

设 a 为 $a_1, a_2, ..., a_n$ 中的最大值，构造

$$B = p_1^a p_2^a ... p_n^a$$

由于

$$p_1^{a_1} p_2^{a_2} ... p_n^{a_n} | p_1^a p_2^a ... p_n^a$$

所以

$$B = k(p-1)$$

因为

$$x^B \equiv x^{p-1} \mod p$$

若 $x \neq kp$，则根据费马定理，即

$$x^B \equiv 1 \mod p$$

即

$$x^B = k_1 p + 1$$

令

$$x^B \equiv y \mod n$$

有

$$k_1 p + 1 \equiv y \mod pq$$

因此

$$k_1 p \equiv y - 1 \mod qp$$

若 $k_1 p > qp$，则

$$k_1 p \equiv k_2 p \mod qp$$

若 $k_1 p > qp$，则

$$k_1 p \equiv k_1 p \mod qp$$

有

$$k_1 p \equiv kp \mod qp$$

因此

$$y - 1 = kp$$

所以

$$y = kp + 1$$

最终

$$p = gcd(y - 1, n)$$

尝试设 $x = 2\ 3\ 4...$，直到 $y \neq 1$，即可得到 p.

（4）p 和 q 应足够大，使 N 的因式分解在计算上不可行：

RSA 的安全性依赖于大数的因子分解。如果能够因子分解模数 N，RSA 就会被攻破。因此，模数 N 必须足够大，以至于在计算上无法进行因子分解。因子分解问题是密码学中最基本的难题之一，但随着计算机计算能力的迅速提升，因子分解的速度已有很大进步。为确保安全性，实际应用中选择的素数 P 和 q 应该足够大，使得计算机无法对其进行大数因式分解。

以下是一种针对多组密文使用同一个模数 n 加密的情况的经典攻击方法——共模攻击。

1、共模攻击

a. 算法思想

在 RSA 加密系统中，共模攻击的基本思想是利用多组加密指数不同但模数相同的密文，结合扩展欧几里得算法来恢复明文。在特定条件下（即不同加密指数互质），可以通过如下步骤实现这一攻击：

背景： RSA 加密公式为：

$$c_1 \equiv m^{e_1} \mod n$$

$$c_2 \equiv m^{e_2} \mod n$$

其中，m 为明文，e_1 和 e_2 分别为两个不同的公钥加密指数，n 为相同的模数。c_1 和 c_2 是对应的密文。

攻击条件：当 e_1 和 e_2 互质时，满足：

$$\gcd(e_1, e_2) = 1$$

根据扩展欧几里得算法，存在整数 s_1 和 s_2，使得：

$$e_1 \cdot s_1 + e_2 \cdot s_2 = 1$$

这就是贝祖等式（Bézout's identity）。

推导过程：首先，根据加密公式 $c_1 \equiv m^{e_1} \mod n$ 和 $c_2 \equiv m^{e_2} \mod n$，可以得到：

$$c_1^{s_1} \equiv (m^{e_1})^{s_1} \mod n = m^{e_1 \cdot s_1} \mod n$$

$$c_2^{s_2} \equiv (m^{e_2})^{s_2} \mod n = m^{e_2 \cdot s_2} \mod n$$

将这两个结果相乘，我们可以得到：

$$c_1^{s_1} \times c_2^{s_2} \equiv m^{e_1 \cdot s_1} \times m^{e_2 \cdot s_2} \mod n$$

利用指数相加的性质：

$$m^{e_1 \cdot s_1} \times m^{e_2 \cdot s_2} = m^{e_1 \cdot s_1 + e_2 \cdot s_2}$$

因为 $e_1 \cdot s_1 + e_2 \cdot s_2 = 1$，因此：

$$m^{e_1 \cdot s_1 + e_2 \cdot s_2} = m^1 = m$$

最终结论：我们可以得出如下等式：

$$c_1^{s_1} \times c_2^{s_2} \mod n = m$$

通过这一等式，即可在不分解模数 n 的情况下，直接解出明文 m。

b. 算法核心伪代码

```
// 使用扩展欧几里得算法的共模攻击伪代码

// 扩展欧几里得算法，计算 a 和 b 的最大公约数 gcd，并返回 x 和 y 使得 ax + by = gcd(a, b)
int extendedGCD(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1; y = 0;
        return a; // gcd(a, 0) = a
    }
    int x1, y1;
    int gcd = extendedGCD(b, a % b, x1, y1); // 递归求解
```

```
    x = y1;
    y = x1 - (a / b) * y1;
    return gcd;
}

// 共模攻击主函数
int commonModulusAttack(int c1, int c2, int e1, int e2, int n) {
    int s1, s2;

    // 使用扩展欧几里得算法求解 s1 和 s2，使得 e1 * s1 + e2 * s2 = 1
    int gcd = extendedGCD(e1, e2, s1, s2);
    if (gcd != 1) {
        throw std::invalid_argument("e1␣和␣e2␣必须互质");
    }

    // 处理 s1 和 s2 的符号问题，如果 s1 < 0，计算模逆元
    if (s1 < 0) {
        s1 = -s1;
        c1 = calculateModInverse(c1, n); // c1 的模逆元
    }
    // 如果 s2 < 0，同理计算 c2 的模逆元
    if (s2 < 0) {
        s2 = -s2;
        c2 = calculateModInverse(c2, n); // c2 的模逆元
    }

    // 计算 c1^s1 和 c2^s2 模 n
    int part1 = fastModularExponentiation(c1, s1, n);
    int part2 = fastModularExponentiation(c2, s2, n);

    // 计算最终结果 m = (c1^s1 * c2^s2) mod n
    int m = (part1 * part2) % n;
    return m;
}

// 辅助函数

// 计算 a 在模 m 下的乘法逆元，使用扩展欧几里得算法
int calculateModInverse(int a, int m) {
```

```
    int x, y;
    int gcd = extendedGCD(a, m, x, y);
    if (gcd != 1) {
        throw std::invalid_argument("无模逆元");
    }
    return (x % m + m) % m; // 确保结果为正
}

// 快速幂算法，计算 (base^exponent) mod modulus
int fastModularExponentiation(int base, int exponent, int modulus) {
    int result = 1;
    base = base % modulus;
    while (exponent > 0) {
        if (exponent % 2 == 1) {
            result = (result * base) % modulus;
        }
        base = (base * base) % modulus;
        exponent /= 2;
    }
    return result;
}

// 测试伪代码
int main() {
    // 输入密文 c1, c2 和相应的加密指数 e1, e2 及模数 n
    int c1 = 12345;  // 示例密文 1
    int c2 = 67890;  // 示例密文 2
    int e1 = 7;      // 加密指数 1
    int e2 = 11;     // 加密指数 2
    int n = 98765;   // 模数

    // 使用共模攻击解密明文
    int decryptedMessage = commonModulusAttack(c1, c2, e1, e2, n);

    // 输出解密后的明文
    std::cout << "解密得到的明文:␣" << decryptedMessage << std::endl;
    return 0;
}
```

c. 算法特点

- **适用性**：适用于多组密文使用同一个模数 n 加密的情况，且不同的公钥指数 e 之间两两互素。
- **效率**：算法的核心是扩展欧几里得算法和模幂运算，计算效率较高。
- **安全性影响**：这种攻击方法揭示了在 RSA 系统中重复使用相同模数的潜在风险。
- **预防措施**：为防止共模攻击，应为每个用户或每次通信使用不同的模数 n。
- **局限性**：攻击成功的前提是获得两个使用相同模数、不同公钥指数加密的密文，且两个公钥指数必须互质。

4.2、 针对参数 e 的攻击

1、低加密指数攻击

a. 算法思想

低加密指数攻击（Small Exponent Attack）是针对 RSA 加密系统的一种攻击方式，适用于公钥加密指数 e 选取过小的情况。在这种情况下，如果明文 m 的值较小，则容易导致密文 c 的特定结构，从而使攻击者可以通过简单的数学运算恢复出明文。

背景： RSA 加密的基本公式为：

$$c \equiv m^e \mod n$$

其中，m 为明文，e 为加密指数，n 为模数。加密后的密文为 c。当 e 选取过小时，尤其是 $e = 3$ 时，可能引发以下安全问题。

攻击条件：

- e 的值较小，例如 $e = 3$
- $m^e < n$，即明文的 e 次方仍然小于模数 n
- n 和 c 的值较大

在这种情况下，加密公式简化为：

$$c = m^e$$

因此，可以通过对密文直接开 e 次方根来恢复明文 m。

推导过程：

当 $m^e < n$ 时：根据 RSA 加密公式 $c = m^e \mod n$，如果明文 m 过小，导致 $m^e < n$，则密文实际上等于 m^e，没有取模的效果，即：

$$c = m^e$$

在这种情况下，攻击者只需对密文 c 进行 e 次方根运算，即可直接获得明文：

$$m = \sqrt[e]{c}$$

当 $m^e > n$ 时：

如果 m^e 大于 n，但不是特别大，仍然可以通过爆破的方法恢复明文。根据加密公式的变形，我们有：

$$m^e = k \cdot n + c$$

其中 k 为某个正整数。通过枚举 k，我们可以反复尝试 k 的值，直到找到满足以下条件的 m^e：

$$c + k \cdot n = m^e$$

一旦满足该条件，就可以对结果进行 e 次方根运算，得到明文 m。

最终结论：在低加密指数攻击中，如果加密指数 e 选取得过小，攻击者可以通过简单的数学运算恢复出明文，具体为：

1. 当 $m^e < n$ 时，直接对密文进行 e 次方根运算。
2. 当 $m^e > n$ 时，通过爆破 k 的值来尝试恢复明文。

b. 算法核心伪代码

```
// 低加密指数攻击伪代码

// 计算整数的 e 次方根
int nthRoot(int value, int e) {
    return pow(value, 1.0 / e); // 返回 e 次方根
}

// 爆破 k 的主函数，尝试找到 m
int lowExponentAttack(int c, int e, int n) {
    // 尝试直接计算 e 次方根
    int m = nthRoot(c, e);
    if (pow(m, e) == c) {
        return m; // 成功找到 m
    }

    // 如果直接开根失败，尝试爆破 k
    for (int k = 1; k < MAX_K; ++k) {
        int candidate = c + k * n;
        m = nthRoot(candidate, e);
```

```
            if (pow(m, e) == candidate) {
                return m; // 成功找到 m
            }
        }

        throw std::runtime_error("攻击失败，未找到明文");
}

// 测试伪代码
int main() {
    // 输入密文 c，加密指数 e 及模数 n
    int c = 123456; // 示例密文
    int e = 3; // 加密指数，常见值 e = 3
    int n = 987654; // 模数

    // 使用低加密指数攻击解密明文
    int decryptedMessage = lowExponentAttack(c, e, n);

    // 输出解密后的明文
    std::cout << "解密得到的明文:␣" << decryptedMessage << std::endl;
    return 0;
}
```

c. 算法特点

- **适用性**：当 e 取值过小（例如 e = 3）且明文较小时，该攻击方法较为有效。
- **效率**：算法的核心步骤是计算 e 次方根和尝试爆破 k，这在现代计算机上效率较高。
- **安全性影响**：这种攻击暴露了在 RSA 系统中使用小加密指数的潜在风险，特别是在不对明文进行填充或其他防护措施的情况下。
- **预防措施**：为防止低加密指数攻击，建议选择较大的加密指数 e，并使用合适的填充方案（如 PKCS#1）来避免明文 m 过小。
- **局限性**：攻击成功的前提是加密指数 e 足够小且明文较小，且没有额外的加密填充措施。

2、低加密指数广播攻击

a. 算法思想

低加密指数广播攻击（Small Exponent Broadcast Attack）是针对 RSA 加密系统的一种攻击方式，适用于公钥加密指数 e 选取较小时的情况，且攻击者掌握了不同模数 n 下的多组加密密文 c。在这种情况下，通过中国剩余定理可以联合多组密文，直接恢复明文。

背景： RSA 加密的基本公式为：

$$c \equiv m^e \mod n$$

其中，m 为明文，e 为加密指数，n 为模数，密文为 c。当使用相同的 e 和不同的 n 对相同的明文 m 进行加密时，可以利用广播攻击将这些密文联合起来，恢复出明文。

攻击条件：

- 公钥加密指数 e 较小，例如 $e = 3$
- 明文 m 被加密了多次，并使用了不同的模数 n_1, n_2, n_3 进行加密
- 攻击者可以获得多组密文 c_1, c_2, c_3，但对应的明文相同

在这种情况下，攻击者可以列出以下同余方程组：

$$\begin{aligned} c_1 &= m^e \mod n_1 \\ c_2 &= m^e \mod n_2 \\ c_3 &= m^e \mod n_3 \end{aligned}$$

通过中国剩余定理，可以将这些方程合并，恢复出 m^e，最终通过开 e 次方根得到明文 m。

推导过程：

中国剩余定理的应用：

对于已知的方程组：

$$c_i = m^e \mod n_i$$

设 M 为所有模数 n_1, n_2, n_3 的乘积：

$$M = n_1 \times n_2 \times n_3$$

对于每一个 n_i，我们计算出对应的 M_i：

$$M_i = M / n_i$$

接着，计算每个 M_i 对应模数 n_i 的模逆 t_i，满足：

$$t_i \cdot M_i \equiv 1 \mod n_i$$

通过中国剩余定理，将每一个 $M_i \cdot t_i \cdot c_i$ 累加起来：

$$x = (c_1 \cdot M_1 \cdot t_1 + c_2 \cdot M_2 \cdot t_2 + c_3 \cdot M_3 \cdot t_3) \mod M$$

这个 x 就是 m^e 的值。接下来，计算 x 的 e 次方根，即：

$$m = \sqrt[e]{x}$$

从而恢复出明文 m。

b. 算法核心伪代码

```
//低加密指数广播攻击伪代码

// 扩展欧几里得算法，计算模逆
int modInverse(int a, int n) {
    int t = 0, new_t = 1;
    int r = n, new_r = a;
    while (new_r != 0) {
        int quotient = r / new_r;
        int temp = t; t = new_t; new_t = temp - quotient * new_t;
        temp = r; r = new_r; new_r = temp - quotient * new_r;
    }
    if (r > 1) throw std::runtime_error("模逆不存在");
    if (t < 0) t = t + n;
    return t;
}

// 计算整数的 e 次方根
int nthRoot(int value, int e) {
    return pow(value, 1.0 / e); // 返回 e 次方根
}

// 低加密指数广播攻击主函数
int lowExponentBroadcastAttack(int c[], int n[], int e, int size) {
    // 计算所有 n 的乘积 M
    int M = 1;
    for (int i = 0; i < size; ++i) {
        M *= n[i];
    }

    // 累加结果
```

```
    int x = 0;
    for (int i = 0; i < size; ++i) {
        int M_i = M / n[i];
        int t_i = modInverse(M_i, n[i]);
        x += c[i] * M_i * t_i;
    }

    x = x % M;

    // 计算 e 次方根，得到明文 m
    return nthRoot(x, e);
}

// 测试伪代码
int main() {
    // 输入密文数组 c，模数数组 n，加密指数 e
    int c[] = {123456, 234567, 345678}; // 示例密文
    int n[] = {987654, 876543, 765432}; // 模数数组
    int e = 3; // 加密指数，常见值 e = 3

    // 使用低加密指数广播攻击解密明文
    int decryptedMessage = lowExponentBroadcastAttack(c, n, e, 3);

    // 输出解密后的明文
    std::cout << "解密得到的明文:␣" << decryptedMessage << std::endl;
    return 0;
}
```

c. 算法特点

- **适用性**：该算法适用于加密指数 e 取值较小且加密了相同的明文多次的情况，尤其是在没有使用填充的情况下。
- **效率**：利用中国剩余定理将多个密文合并为一个方程，随后通过开 e 次方根快速恢复明文。
- **安全性影响**：低加密指数广播攻击展示了使用小加密指数 e 的潜在安全隐患，尤其是在没有任何填充的纯 RSA 加密中。这种情况下，明文恢复非常容易。
- **预防措施**：为防止这种攻击，建议使用较大的加密指数，如 $e = 65537$，并结合使用填充方案（如 PKCS#1），避免直接攻击密文。

- **局限性**：该攻击要求加密指数较小，同时需要获取足够多的不同模数下的密文。如果只有少量模数或使用了填充，则攻击难以成功。

4.3、 针对参数 d 的攻击

1、低解密指数攻击（维纳攻击）

a. 算法思想

低解密指数攻击（Small Decryption Exponent Attack）与低加密指数攻击相似，主要利用了 RSA 中解密指数 d 较小的特性。由于 d 是 e 的模 $\phi(n)$ 的逆元，若 e 过大，则会导致解密指数 d 过小。Wiener 攻击（Wiener Attack）是一种常用的低解密指数攻击，当 $d < \frac{1}{3}n^{1/4}$ 时，攻击者可以通过连分数逼近恢复私钥 d。

背景： RSA 加密系统的公钥由模数 $n = p \cdot q$ 和加密指数 e 构成。私钥由模数 n 和解密指数 d 构成，且 d 满足以下方程：

$$d \cdot e \equiv 1 \pmod{\phi(n)}$$

其中 $\phi(n) = (p-1)(q-1)$ 是欧拉函数。

RSA 的加密过程如下：

$$c = m^e \mod n$$

解密过程通过解密指数 d 还原明文 m：

$$m = c^d \mod n$$

d 是 e 对 $\phi(n)$ 的模反元素。当 e 较大时，d 会变得较小，Wiener 攻击的核心就是利用低解密指数 d 的弱点。当 d 足够小时，$\frac{d}{e}$ 可以用连分数逼近。

攻击条件：

- 加密指数 e 较大
- 解密指数 d 较小，满足 $d < \frac{1}{3}n^{1/4}$
- 模数 n 较大，但不能快速分解

在这种情况下，攻击者可以使用 Wiener 攻击，通过连分数逼近来恢复私钥。

推导过程： Wiener 攻击的解密过程依赖于连分数和渐进分数的数学性质，为了详细解释，我们需要理解连分数的展开、渐进分数的推导过程以及如何在攻击中利用这些性质来恢复 d。

连分数展开

连分数是一种表示有理数的方式，它可以用以下形式表示：

$$\frac{d}{e} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \dots}}}$$

其中，$a_0, a_1, a_2, \dots$ 是连分数的系数。

Wiener 攻击的关键在于对 $\frac{e}{n}$ 进行连分数展开，因为 $\frac{d}{e}$ 是一个很小的数，连分数逼近能够很好地逼近这个比值。

渐进分数（近似值）

对于连分数展开，渐进分数（convergents）是每一步的逼近值，通常表示为：

$$\frac{p_k}{q_k}$$

其中，p_k 和 q_k 分别是第 k 步的连分数分子和分母。这些渐进分数的性质表明，它们逐渐逼近实际的 $\frac{d}{e}$，即 $p_k/q_k \approx d/e$。

攻击步骤

Wiener 的连分数攻击过程如下：假设 RSA 模数 $N = p \cdot q$ 且 $q < p < 2q$; 此外，假设攻击者知道解密指数 $d < \frac{1}{3}N^{1/4}$，且加密指数 e 提供给了攻击者。此时 d 和 e 满足

$$e \cdot d \equiv 1 (\mathrm{mod} \varphi(N))$$

其中 $\varphi(N) = (p-1)(q-1)$。我们注意到，上式意味着存在一个整数 k 使得

$$e \cdot d - k \cdot \varphi(N) = 1.$$

因此，我们有

$$\left| \frac{e}{\varphi(N)} - \frac{k}{d} \right| = \frac{1}{d \cdot \varphi(N)}$$

由于 $N = p \cdot q$ 且 $q < p < 2q$，有

$$N - \varphi(N) = p + q - 1 < p + q < 3q < 3\sqrt{pq} = 3\sqrt{N}.$$

由此，我们可以认为分数 $\frac{e}{N}$ 是分数 $\frac{k}{d}$ 的近似值，而且

$$\left| \frac{e}{N} - \frac{k}{d} \right| = \left| \frac{ed - kN}{dN} \right| = \left| \frac{ed - k\varphi(N) - kN + k\varphi(N)}{dN} \right|$$

$$= \left| \frac{1 - k(N - \varphi(N))}{dN} \right|$$

$$\leq \frac{3k\sqrt{N}}{dN} = \frac{3k}{d\sqrt{N}}.$$

由于 $e < \varphi(N)$，显然我们有 $k < d$。又由假设条件 $d < \frac{1}{3} \cdot N^{1/4}$，可得

$$\left| \frac{e}{N} - \frac{k}{d} \right| < \frac{1}{3d^2} < \frac{1}{2d^2}.$$

因此根据有关定理可知分数 $\frac{k}{d}$ 一定是有理数 $\frac{e}{N}$ 的一个渐近分数。因此，当 $d < \frac{1}{3} \cdot N^{1/4}$ 时，我们可以得到一个高效的分解 RSA 模数 N 的算法：首先计算 $\frac{e}{N}$ 的每个渐近分数 $\frac{k_i}{d_i}$；然后计算 $T_i = N - \frac{ed_i - 1}{k_i} + 1$；对二次方程 $y^2 - T_i y + N = 0$ 进行求解，若解为 p，则 N 被分解；若不是，则继续计算 $\frac{k_{i+1}}{d_{i+1}}$，直到 N 被分解。此时，我们不但对 N 进行了分解，也得到了解密指数 d。

b. 算法核心伪代码

```
//维纳攻击伪代码

// 计算整数平方根
int isqrt(int n) {
    return static_cast<int>(sqrt(n));
}

// 连分数展开函数
std::vector<int> continuedFractionExpansion(int e, int n) {
    std::vector<int> fractions;
    while (n != 0) {
        fractions.push_back(e / n);
        int temp = e % n;
        e = n;
        n = temp;
    }
    return fractions;
}

// 计算渐进分数函数
std::pair<int, int> computeConvergent(const std::vector<int>&
   fractions, int limit) {
    int numerator = 0, denominator = 1;
    for (int i = limit; i >= 0; --i) {
        int temp = numerator;
        numerator = denominator;
        denominator = fractions[i] * denominator + temp;
    }
    return {denominator, numerator}; // 返回 (k, d)
```

```
}

// Wiener攻击核心函数
int wienerAttack(int e, int n) {
    // 计算连分数展开
    std::vector<int> cf = continuedFractionExpansion(e, n);

    for (size_t i = 0; i < cf.size(); ++i) {
        // 计算渐进分数
        auto [k, d] = computeConvergent(cf, i);
        if (k == 0) {
            continue;
        }

        // 计算欧拉函数phi
        int phi = (e * d - 1) / k;
        int b = n - phi + 1;
        int discriminant = b * b - 4 * n;

        // 检查判别式是否为完全平方数
        if (discriminant >= 0) {
            int sqrt_disc = isqrt(discriminant);
            if (sqrt_disc * sqrt_disc == discriminant) {
                // 找到解密指数 d
                return d;
            }
        }
    }
    throw std::runtime_error("攻击失败，未找到私钥␣d");
}

// 测试伪代码
int main() {
    int e = 17993; // 示例加密指数
    int n = 90581; // 示例模数

    try {
        // 使用Wiener攻击恢复私钥 d
        int d = wienerAttack(e, n);
```

```
68          std::cout << "找到的解密指数 d: " << d << std::endl;
69      } catch (const std::exception& ex) {
70          std::cerr << ex.what() << std::endl;
71      }
72      return 0;
73  }
```

c. 算法特点

- **适用性**：该算法适用于解密指数 d 较小（即 $d < \frac{1}{3}n^{1/4}$）的场景，尤其当加密指数 e 很大时，Wiener 攻击效果显著。
- **效率**：Wiener 攻击通过连分数展开逼近解密指数 d，在满足条件的情况下可以在多项式时间内完成攻击，计算复杂度较低。
- **安全性影响**：RSA 加密系统中，若解密指数 d 过小，攻击者可以利用 Wiener 攻击等技术恢复私钥，破坏系统的安全性。
- **预防措施**：为了避免低解密指数攻击，应选择合适的解密指数 d，确保其不满足 $d < \frac{1}{3}n^{1/4}$ 的条件，从而增强系统的安全性。
- **局限性**：攻击成功的前提是 d 较小，如果解密指数 d 不满足攻击条件，攻击将无法奏效。

5、 在密码学中的应用

RSA 算法在密码学中有着广泛的应用，其安全性主要基于大整数分解的困难性。这使得它在以下几个领域得到了广泛的使用：

1. **公钥加密**
   RSA 是最典型的公钥加密算法之一。在公钥加密系统中，信息的发送方使用接收方的公钥对消息进行加密，而接收方使用私钥解密。RSA 公钥加密通过生成一对密钥——公钥用于加密，私钥用于解密，确保了通信的机密性。加密过程涉及将明文转化为大整数，并使用指数运算与模运算进行加密，保证即使第三方截获加密的消息，由于大整数分解的困难性，他们无法轻易破解密文。

2. **数字签名**
   RSA 的另一重要应用是数字签名。签名过程与加密相反，消息发送方使用自己的私钥对消息签名，而接收方则使用发送方的公钥验证签名的有效性。数字签名提供了消息的真实性验证，确保消息未被篡改，同时也提供了消息发送方的不可否认性，防止发送方否认曾发送过消息。

3. **密钥交换**
   RSA 还用于安全密钥交换。在某些场景下，双方需要交换对称加密所需的密钥，RSA 可以用来安全地加密传输密钥。双方通过各自的公钥和私钥，可以确保在交换密钥的过程中，只有合法的通信方能够解密获取密钥，从而确保后续的对称加密通信安全。

4. **证书颁发机构 (CA)**
   证书颁发机构在现代互联网安全中扮演了重要角色，尤其是在 HTTPS 协议的实现中。RSA 常用于生成和验证 SSL/TLS 证书。通过 RSA 加密，证书颁发机构可以验证服务器的身份，确保客户端与服务器之间的通信安全。证书包含 RSA 公钥，用于加密传输的对称密钥，从而确保加密连接的安全性。

5. **文件加密**
   RSA 也可以用于加密重要的文件，尤其是那些需要跨不安全网络传输的文件。在这种应用中，文件的加密过程通常结合 RSA 和对称加密算法使用。文件先通过对称密钥加密，而对称密钥则通过 RSA 加密传输。这样既能利用 RSA 的安全性，又能保持对称加密的效率。

6. **物联网安全**
   随着物联网（IoT）设备的普及，如何确保设备之间的安全通信成为一个重要课题。RSA 可以用于 IoT 设备的身份认证和加密通信，防止恶意设备接入网络或窃取敏感数据。通过 RSA 公私钥对，IoT 设备可以进行安全认证，确保只有合法设备能够参与通信。

7. **区块链技术**
   在区块链系统中，RSA 同样扮演了重要的角色，尤其是在数字货币的交易和验证中。RSA 通过数字签名确保交易的合法性，防止交易记录被篡改。在某些区块链平台上，RSA 也可以用于节点间的安全通信，保护区块链网络的安全。

三、 零知识证明问题

零知识证明（Zero-Knowledge Proof, ZKP）是一种重要的密码原语，用于在不泄露实际信息的情况下证明某个陈述的真实性。它广泛应用于密码学协议，如身份验证、区块链隐私保护等领域。零知识证明的核心思想是通过特定的算法，使得验证者确信证明者掌握了正确的秘密信息，而不暴露该秘密。

1、 对密码原语的介绍

零知识证明（Zero-Knowledge Proof, ZKP）作为一种复杂的密码学技术，其发展源于密码学的核心目标：在公开环境下保证信息的隐私和安全。它最早于 1985 年由三位计算机科学家——Shafi Goldwasser、Silvio Micali 和 Charles Rackoff 提出，他们在论文中正式定义了“零知识”的概念。这一概念的提出，填补了密码学在数据隐私和交互安全中的重要空白，特别是如何在不暴露信息的前提下进行有效的验证。这种新颖的方法为密码学带来了深远的影响，也逐渐推动了现代信息技术的发展。

1.1、 历史缘起

零知识证明的提出和发展离不开密码学的历史背景。在 20 世纪 70 年代，现代密码学的基础逐步建立，主要关注如何在开放的通信环境下实现数据的机密性和完整性。这一时期出现了非对称加密、公钥密码学等概念，例如 RSA 加密算法和 Diffie-Hellman 密钥交换协议，它们使得双方能够安全地共享加密密钥和敏感数据。

在此之后，研究者逐渐意识到：如何验证某人知道某个秘密而不需要公开该秘密，这种需求尤其在身份验证和隐私保护场景中显得尤为重要。例如，在不透露密码的情况下证明你知道密码，或在不透露交易细节的情况下确认交易的有效性。零知识证明应运而生，为这种需求提供了一种新颖的解决方案。其提出不仅延展了密码学理论，也开启了对隐私计算的新探索。

1.2、 密码学背景

a. 公钥密码学与数据隐私

公钥密码学（如 RSA、椭圆曲线加密等）使得两方可以在没有预先共享密钥的情况下进行加密通信。其核心是使用一对密钥：公开的加密密钥和私密的解密密钥。这种方式让数据在传输过程中保持私密性，然而，如果验证一个人知道私钥或能正确解密某个密文，却不会泄露该密钥内容，传统的密码学方法并不直接支持这一需求。

b. 身份验证问题

在传统的身份验证方法中，证明者往往需要提供密码或某种密钥，但这些验证方法可能会引发隐私泄露的问题。为了解决这一问题，零知识证明设计了一个交互过程，通过数学算法确保验证者能够相信证明者的声明是真实的，但又不会得到多余的细节信

息。这种设计从根本上改变了验证方式，使得证明的过程在保持信息隐私的同时，依然能够被信任。

c. 复杂性理论与概率论

零知识证明的实现依赖于复杂性理论中的 NP（非确定性多项式）问题和概率论。其安全性依赖于计算难题的不可解性。一个典型的例子是离散对数问题，证明者可以向验证者展示其“知道”某个问题的解，而验证者无法直接获得这个解。通过引入随机性的交互式设计，零知识证明能有效避免验证者通过多轮交互获得不该获得的隐私信息。

1.3、 零知识的含义

零知识证明的“零知识”并不是指没有任何信息，而是指不泄露证明内容以外的任何信息。证明者使用一些算法和技巧向验证者传递足够的信息，以保证验证者确信这个声明为真，但又不会让验证者获取任何关于实际秘密的细节。

举个简单的例子：假设山洞内有两条路径 A 和 B，中间有一扇上锁的门需密码打开。Alice 想证明她知道密码但不想透露，便从路径 A 或 B 进入，Bob 在外随机要求她从另一条路径返回。Alice 若每次都能满足要求，则 Bob 可确信她确实知道密码，而无需得知密码内容。

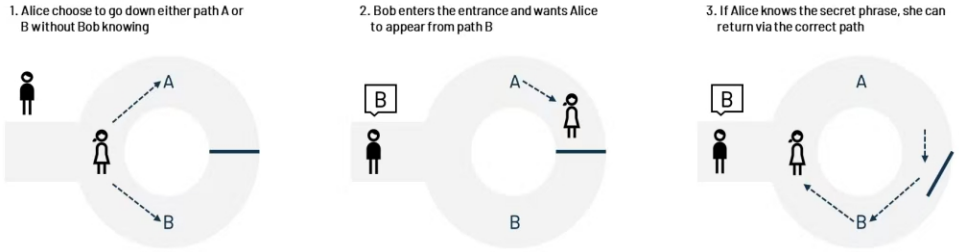

图 6: 零知识证明的示例

在实际的密码学应用中，零知识证明的原理类似于这种山洞场景。想象有一种计算电路，输入数据后可以得出一个特定的输出，比如抛物线上的某个值。如果证明者能多次计算出正确的值，那么验证者就可以确信证明者知道生成这条曲线的函数。每一轮的验证都增加了证明的可信度，而不需透露函数的具体内容。这种方式在保护隐私的同时实现了验证，特别适用于区块链等注重数据隐私和验证的领域。

1.4、 三个核心性质

- **完备性**：如果证明者提供的信息是真实的，那么验证者在遵循协议的过程中，必然会接受这个证明。
- **可靠性**：如果证明者提供的信息是假的，那么无论证明者采取何种策略，验证者都不应接受这个证明。

- **零知识性**：如果陈述是真实的，验证者无法从交互中获得任何多余的信息，只能确认陈述的真实性。这个性质通过设计复杂的加密算法和协议来实现。

1.5、 四个证明阶段

- **承诺（Commit）**：证明者针对要证明的内容生成一个承诺，表明自己掌握了某些信息，但不直接展示。这一承诺保持不可更改，等待验证者后续提出的挑战。
- **挑战（Challenge）**：验证者根据随机选择的数值对证明者的承诺发起验证挑战，要求证明者展示其承诺的真实性。
- **回应挑战（Response）**：证明者将根据验证者随机选择的数值进行回应，将挑战和最初的承诺结合后做出回复。这一步要求承诺内容不可随意修改。
- **验证（Verify）**：验证者检查证明者的回应是否符合要求。如果回应有误，验证者拒绝证明；若回应无误，则验证者接受证明。

1.6、 零知识证明的类型

1、交互式零知识证明

a. 概念解释

交互式零知识证明是一种密码学协议，允许证明者（Prover）向验证者（Verifier）证明其拥有某种知识（例如某个秘密或某种数学上的解）而不泄露该知识本身。在这种证明方式中，证明者（拥有秘密的人，比如知道密码或私钥）和验证者（想确认对方知道秘密的人）需要多次交流。验证者会不断随机提出“挑战”问题，证明者则需要提供正确的回答，从而让验证者相信她确实知道秘密。这种方式的安全性很高，但需要双方实时在线，并多次交互。因此，交互式零知识证明通常适用于身份验证或需要持续确认的场景。

b. 典型的交互式零知识证明协议——Schnorr 协议

(1) 初始化（Initialization）

- 证明者（我们称为 Alice）拥有一个私钥 $sk = a$。
- Alice 的公钥 $pk = g^a \pmod{P}$，其中：
  - g 是一个生成元（Generator），通常是公开已知的。
  - P 是一个大素数，作为模数。
  - pk 就是 Alice 公布的身份标识，用来证明她拥有 sk 而不暴露具体的 sk 值。

此时，验证者（我们称为 Bob）知道 Alice 的公钥 pk，但不知道她的私钥 sk。

(2) 承诺（Commitment）

- Alice 生成一个随机数 r。
- 她计算 $x = g^r \pmod{P}$，得到一个承诺值 x。
- 这个 x 是一个" 临时" 的计算值，用于让 Bob 作为一个验证基准。
- 然后，Alice 将这个值 x 发送给 Bob。此时，Bob 只知道 x，并不知道生成它的具体随机数 r。

(3) 挑战（Challenge）

- Bob 随机生成一个挑战值 c 并发送给 Alice。这个 c 是一个随机数，用于测试 Alice 是否真的知道私钥 a。
- 这个随机挑战确保了 Alice 不能事先预测 Bob 会问什么问题，确保每次交互都是独立的。

(4) 回应（Response）

- Alice 根据收到的挑战值 c，计算响应值 $y = r + c \cdot a \pmod{q}$，其中 q 是与 P 相关的素数（通常是 $P-1$ 的一个因数）。
- 这里的 y 包含了 Alice 对于挑战 c 的响应，同时隐含了 Alice 的私钥 a。
- Alice 将 y 发送给 Bob。

(5) 验证（Verification）

- Bob 收到 y 后，验证以下等式是否成立：$g^y \stackrel{?}{=} x \cdot pk^c \pmod{P}$
- 若等式成立，Bob 可以相信 Alice 知道私钥 a，因为仅知道 pk 而不知道 a 是无法通过该等式验证的。

数学证明：

为了理解为什么这个验证步骤有效，我们可以简单推导一下验证等式：

1. 回到 Alice 计算的 y 值：$y = r + c \cdot a \pmod{q}$
2. 将 y 带入验证等式的左边，我们得到：$g^y = g^{r+c\cdot a} \pmod{P}$
3. 根据幂的加法规则 $g^{r+c\cdot a} = g^r \cdot g^{c\cdot a}$，所以：$g^y = g^r \cdot (g^a)^c \pmod{P}$
4. 因为 Alice 的公钥 $pk = g^a$，我们可以替换得到：$g^y = x \cdot pk^c \pmod{P}$
5. 因此，如果 Alice 的 y 是基于实际的 a 计算得出的，那么验证等式成立。

这证明了如果 Alice 知道 a，则等式成立，从而 Bob 能够确认 Alice 知道 $sk = a$ 而无需直接暴露 a。

2、非交互式零知识证明

a. 概念解释

非交互式零知识证明（Non-Interactive Zero-Knowledge, NIZK）是交互式零知识证明的简化版本，它允许证明者在没有验证者参与的情况下完成证明。在非交互式零知识证明中，去除了反复的挑战与响应过程，只需要一次性生成一个“证明文件”，证明者便可将其公开发布或传递给验证者。任何人都可以通过这个证明文件验证证明者确实知道秘密，而无需与证明者进行交互。因此，非交互式零知识证明特别适合在区块链等场景中使用，因为它们需要支持快速、公开验证，不可能一对一交互。

c. Fiat-Shamir 变换——将交互式转化为非交互式

Fiat-Shamir 变换是一种方法，用于将交互式零知识证明协议（如 Schnorr 协议）转化为非交互式的形式。它通过将验证者的” 挑战” 随机数替换为一个由哈希函数生成的伪随机数，使整个证明过程只需一次性生成并公开发布，从而省去验证者和证明者之间的多次交互。以下是基于 Schnorr 协议的 Fiat-Shamir 变换流程的详细说明：

(1) 初始化

- 与交互式 Schnorr 协议相同，假设证明者拥有一个私钥 $sk = a$ 和相应的公钥 $pk = g^a \pmod{P}$，其中 g 是一个公开的生成元，P 是素数模数。

(2) 承诺（Commitment）

- 证明者生成一个随机数 r，计算出承诺值：
- $x = g^r \pmod{P}$
- 此时，x 表示证明者的承诺，类似于” 锁住” 的信息，将在验证过程中使用。

(3) 挑战（Challenge）

- 在交互式协议中，挑战 c 是由验证者随机生成的数值，但在 Fiat-Shamir 变换中，挑战是由证明者自己通过哈希函数生成。具体地，证明者计算：
- $c = H(x \parallel M)$
- 其中 H 是一个加密安全的哈希函数，M 是想要证明的消息或特定内容。这里，哈希函数 H 可以被认为是一种” 随机预言机”，模拟了验证者的随机挑战，使得验证者无需在线也能确定唯一的挑战。

(4) 回应（Response）

- 证明者计算响应 y：
- $y = r + c \cdot a \pmod{q}$

- 其中 q 是与 P 相关的另一个素数。然后证明者将 (c, y) 和消息 M 作为证明值发布。

(5) 验证（Verification）

1. 计算 g^y 并检查其是否满足：$g^y \stackrel{?}{=} x \cdot pk^c \pmod{P}$

2. 验证 c 是否等于 $H(x \parallel M)$，确保挑战值的一致性。

如果以上两步均成立，则证明通过，验证者可以确定证明者确实知道秘密私钥 $sk = a$ 而无需再和其交互。这种方式特别适合区块链等场景，因为其无需多次交互，可离线验证，提高了效率。

2、 具体的应用场景

零知识证明（Zero-Knowledge Proof, ZKP）在多个实际场景中展现了其独特的优势，特别是在安全和隐私保护方面。以下是几个典型的应用场景：

2.1、 隐私保护的身份验证

零知识证明最经典的应用是进行无密码身份验证。在传统的身份验证系统中，用户需要提供密码或其他敏感信息来验证身份，而零知识证明可以实现用户在不透露任何敏感信息的前提下完成身份验证。例如：

- **无需密码的登录**：用户可以使用零知识证明来证明自己知道某个秘密（如私钥），而无需向服务器提交该秘密。这可以减少密码泄露的风险。

- **门禁系统**：在门禁系统中，通过零知识证明，持有合法密钥的用户无需暴露密钥本身，即可进入门禁区域。

2.2、 区块链与加密货币

区块链技术本质上是公开透明的，这意味着每笔交易的细节（如交易金额、发送方和接收方）都可以被查看。零知识证明在区块链领域的应用，主要用于保护交易隐私和提高交易效率。

- **zk-SNARKs**：如 Zcash（ZEC）使用 zk-SNARKs（简洁非交互式零知识证明）来隐藏交易的详细信息，只公开交易的发生，而不透露交易的金额或双方身份。

- **以太坊 2.0**：零知识证明也被用于以太坊 2.0 的扩展性解决方案，如 zk-Rollups，通过将大量的交易打包在一起并使用零知识证明来验证这些交易的合法性，从而显著减少链上数据存储和计算的需求。

2.3、 数字版权保护与数字内容认证

在数字版权保护领域，零知识证明可以用来验证数字作品的版权归属，而不需要公开该作品的详细信息。例如：

- **数字水印验证**：零知识证明可以验证某个作品的所有者拥有合法的版权，而无需公开数字水印的具体内容。
- **防伪验证**：在数字内容认证中，可以使用零知识证明来确认内容的真实性和完整性，而不需要公开原始的内容数据。

2.4、 电子投票系统

零知识证明在电子投票系统中有广泛的应用前景。通过使用零知识证明，可以确保投票者的隐私，同时保证投票的公正性和准确性。

- **投票隐私保护**：使用零知识证明，选民可以在不透露投票选择的情况下证明他们已经合法地投票。选票的隐私得以保护，同时投票计数的完整性也能得到保证。
- **防止重复投票**：零知识证明可以用于验证每个投票者只投了一次票，而无需透露他们具体的投票信息。

2.5、 金融与支付系统

在金融和支付系统中，零知识证明可以用于保证交易的隐私和合规性。例如：

- **银行业务中的隐私保护**：通过零知识证明，银行可以验证客户的资产是否满足某些要求，而不需要公开客户的所有资产信息。
- **防止洗钱和欺诈行为**：金融机构可以使用零知识证明来验证交易的合法性和资金的合规性，而不需要公开交易的详细信息，从而减少洗钱和欺诈的风险。

2.6、 医疗数据的隐私保护

零知识证明还可以应用于医疗数据共享场景中。患者可以使用零知识证明在不泄露详细病历的前提下，证明他们满足某些条件。

- **医疗保险审核**：保险公司可以通过零知识证明来验证投保人是否符合理赔要求，而不需要获取所有的医疗记录。
- **医学研究中的数据共享**：研究人员可以通过零知识证明来共享数据分析的结果，而不需要公开患者的隐私数据。

3、 其中包含的数学问题

零知识证明涉及诸多数学问题，比如图同构问题、哈密顿回路问题、多项式求根问题、离散对数问题、二次剩余问题等。这些问题大多具有特殊的数学性质：易于验证但难以求解，这种不对称性正是构造零知识证明的基础。下面我将就同态加密与承诺方案和椭圆曲线与离散对数问题进行详细分析：

3.1、 同态加密与承诺方案

a. 数学问题的定义

（1）同态加密 (Homomorphic Encryption)：

同态加密允许在密文状态下进行特定运算，解密后结果与直接在明文上运算一致。其形式化定义为：

定义 1 (同态加密)**.** *给定加密函数 E、解密函数 D 和消息 m_1, m_2：*

- *对于加法同态：* $D(E(m_1) \oplus E(m_2)) = m_1 + m_2$
- *对于乘法同态：* $D(E(m_1) \otimes E(m_2)) = m_1 \times m_2$

其中 $\oplus, \otimes$ 为密文域上的运算。

根据支持的运算类型，同态加密可分为：

1. **部分同态加密 (PHE)**：
   - 加法同态：如 Paillier 密码系统
   - 乘法同态：如 RSA 加密
   - 优点：效率较高，适用于特定场景
2. **全同态加密 (FHE)**：
   - 支持任意次数的加法和乘法运算组合
   - 代表方案：Gentry 构造的基于格的 FHE
   - 主要挑战：计算开销大，需要优化

（2）承诺方案 (Commitment Scheme)：

承诺方案允许一方对消息进行绑定，并在之后揭示该消息。形式化定义如下：

定义 2 (承诺方案)**.** *一个承诺方案包含以下算法：*

- $Setup(1^\lambda) \to pp$*：生成公共参数*
- $Commit(pp, m; r) \to c$*：生成承诺值*

- $Open(pp, c, m, r) \to 0, 1$: 验证承诺

其中 λ 为安全参数,m 为消息,r 为随机数。

安全性要求:

1. **绑定性 (Binding)**:
   - 完美绑定: 计算上不可能找到不同的 (m, r) 和 (m', r') 使得 $\text{Commit}(m, r) = \text{Commit}(m', r')$
   - 计算绑定: 在多项式时间内找到碰撞的概率可忽略
2. **隐藏性 (Hiding)**:
   - 完美隐藏: 承诺值在信息论上不泄露消息信息
   - 计算隐藏: 在多项式时间内无法从承诺值推测消息

b. 数学问题的分析与应用
同态加密的性质与零知识证明结合:
(1) 基于模运算的加法同态加密性质分析:

1. 基本定义与构造

定义 3 (模运算加法同态加密)**.** 加密函数 $E(m)$ 定义如下:

- 密钥生成: 选择大素数 p, q, 计算 $n = pq$, 选择生成元 g
- 加密: $E(m) = g^m \cdot r^p \bmod n$, 其中 r 为随机数
- 解密: 利用私钥 p, q 恢复明文 m

2. 同态性质证明

证明. 对于两个密文 $E(m_1) = g^{m_1} \cdot r^p \bmod n$ 和 $E(m_2) = g^{m_2} \cdot s^p \bmod n$:

$$\begin{aligned} E(m_1) \cdot E(m_2) &= (g^{m_1} \cdot r^p) \cdot (g^{m_2} \cdot s^p) \bmod n \\ &= g^{m_1} \cdot g^{m_2} \cdot (r \cdot s)^p \bmod n \\ &= g^{m_1+m_2} \cdot (r \cdot s)^p \bmod n \\ &= E(m_1 + m_2) \end{aligned}$$

这证明了加法同态性质。 □

(2) Paillier 加密系统与零知识证明的结合:

1. 基本算法框架

- **密钥生成**:
  1. 选择两个大素数 p, q 满足 $\gcd(pq, (p-1)(q-1)) = 1$
  2. 计算 $n = pq$ 和 $\lambda = \text{lcm}(p-1, q-1)$
  3. 选择 $g \in \mathbb{Z}_{n^2}^*$ 使得 g 的阶为 n 的倍数
  4. 公钥为 (n, g), 私钥为 λ
- **加密过程**:
  1. 输入明文 $m \in \mathbb{Z}_n$
  2. 选择随机数 $r \in \mathbb{Z}_n^*$
  3. 计算密文: $c = g^m \cdot r^n \bmod n^2$
- **解密过程**:
  1. 计算 $\mu = (L(g^\lambda \bmod n^2))^{-1} \bmod n$
  2. 明文 $m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n$
  3. 其中 $L(x)$ 函数定义为 $L(x) = \frac{x-1}{n}$

2. 零知识证明构造

- **证明系统设置**:
  - 公共参数: Paillier 加密的公钥 (n, g)
  - 证明者私有输入: 明文 m 和随机数 r
  - 公开输入: 密文 $c = E(m) = g^m \cdot r^n \bmod n^2$
- **证明协议流程**:
  1. 证明者选择随机值:

$$v \xleftarrow{R} \mathbb{Z}_n \qquad\qquad s \xleftarrow{R} \mathbb{Z}_n^*$$

  2. 计算承诺:

$$t = g^v \cdot s^n \bmod n^2$$

  3. 收到挑战 $e \xleftarrow{R} \mathbb{Z}_n$ 后计算响应:

$$z_1 = v + em \bmod n \qquad\qquad z_2 = s \cdot r^e \bmod n$$

4. 验证等式:

$$g^{z_1} \cdot z_2^n \stackrel{?}{\equiv} t \cdot c^e \pmod{n^2}$$

(3) 具体应用实例:

范围证明 (Range Proof):

证明加密值落在区间 $[a, b]$ 中:

- **协议构造**:

1. 证明者计算:

$$w_1 = m - a \geq 0$$
$$w_2 = b - m \geq 0$$
$$c_1 = E(w_1) = g^{w_1} \cdot r_1^n \bmod n^2$$
$$c_2 = E(w_2) = g^{w_2} \cdot r_2^n \bmod n^2$$

2. 利用同态性质证明:

$$c \cdot g^{-a} \equiv c_1 \pmod{n^2} \text{ 且} g^b \cdot c^{-1} \equiv c_2 \pmod{n^2}$$

3. 使用平方分解证明非负性:

$$w_1 = \sum_{i=1}^{k} x_i^2 \qquad\qquad w_2 = \sum_{i=1}^{k} y_i^2$$

- **验证过程**:

1. 验证同态关系:

$$E(m-a) \stackrel{?}{=} c_1 \text{ 且} E(b-m) \stackrel{?}{=} c_2$$

2. 验证平方分解证明
3. 检查所有零知识证明的有效性

相等性证明:

证明两个密文包含相同的明文:

- **问题设置**:

  - 两个密文: $c_1 = E(m, r_1)$, $c_2 = E(m, r_2)$
  - 证明它们加密了相同的值

- **证明构造**:

1. 利用同态性质计算:

$$\phi = c_1 \cdot c_2^{-1} = E(0, r_1/r_2)$$

2. 证明者计算承诺:

$$v \xleftarrow{R} \mathbb{Z}_n, s \xleftarrow{R} \mathbb{Z}_n^* \qquad t = g^v \cdot s^n \bmod n^2$$

3. 响应挑战:

$$z_1 = v + e \cdot 0 = v \qquad z_2 = s \cdot (r_1/r_2)^e$$

- **验证等式**:

$$g^{z_1} \cdot z_2^n \stackrel{?}{\equiv} t \cdot \phi^e \pmod{n^2}$$

乘法关系证明:
证明三个密文满足乘法关系:

- **问题设置**:
  - 密文: $c_1 = E(x)$, $c_2 = E(y)$, $c_3 = E(z)$
  - 证明 $z = x \cdot y$
- **证明步骤**:

1. 利用同态性质:

$$c_1^y = E(xy) = E(z) = c_3$$

2. 构造零知识证明:

选择随机值: $r \xleftarrow{R} \mathbb{Z}_n^*$　　计算承诺: $t = c_1^r$ 响应挑战e: $z = r + ey \bmod n$

3. 验证等式:

$$c_1^z \stackrel{?}{\equiv} t \cdot c_3^e \pmod{n^2}$$

这些零知识证明协议充分利用了 Paillier 加密系统的同态特性, 实现了在保护隐私的同时验证各种关系的目的。

3.2、 椭圆曲线与离散对数

a. 数学问题的定义
(1) 椭圆曲线 (Elliptic Curve):
椭圆曲线是满足特定方程的点集，通常用于密码学的椭圆曲线方程为:

$$E : y^2 = x^3 + ax + b \pmod{p}, \tag{1}$$

其中 $a, b \in \mathbb{F}_p$ 且满足 $4a^3 + 27b^2 \neq 0$，保证曲线无奇点。点集 $E(\mathbb{F}_p)$ 包括曲线上的所有点 (x, y) 和一个无穷远点 $\mathcal{O}$。

椭圆曲线上的加法定义如下：

1. 对于任意点 $P, Q \in E(\mathbb{F}_p)$，定义加法 $P + Q$。
2. 存在加法单位元 $\mathcal{O}$ 使得 $P + \mathcal{O} = P$。
3. 每个点 P 都有逆元 $-P$，使得 $P + (-P) = \mathcal{O}$。

（2）离散对数 (Discrete Logarithm)：

离散对数问题在椭圆曲线上的形式化定义为：

定义 4 (椭圆曲线离散对数问题 (ECDLP))**.** 给定椭圆曲线 $E(\mathbb{F}_p)$、基点 $G \in E(\mathbb{F}_p)$ 和点 $P \in E(\mathbb{F}_p)$，找到整数 k 满足：

$$P = kG, \tag{2}$$

其中 $k \in \mathbb{Z}$ 且运算基于椭圆曲线的加法定义。

ECDLP 是一个难解问题，其复杂性随着椭圆曲线阶的增长以指数增加，广泛用于构建密码系统。

b. 数学问题的分析与应用

（1）ECDLP 与零知识证明的结合：

椭圆曲线上的离散对数问题在零知识证明中被广泛用于验证某个计算声明的正确性，而不泄露声明的细节。以下通过两种具体问题展开说明：

（1.1）离散对数相等性证明：

证明两个点 Q_1, Q_2 对应的标量相等，即：已知 $Q_1 = kP_1, Q_2 = kP_2$，证明 k 是相同的标量，但不暴露 k。

问题设置 椭圆曲线 E 上的基点 P_1, P_2；点 $Q_1 = kP_1, Q_2 = kP_2$；证明者私有输入：标量 k；验证者公开输入：P_1, P_2, Q_1, Q_2。

协议流程 证明者选择随机标量 $r \in \mathbb{Z}_n$；计算承诺值 $T_1 = rP_1$ 和 $T_2 = rP_2$，将 T_1, T_2 发送给验证者；验证者生成随机挑战 $e \in \mathbb{Z}_n$，并发送给证明者；证明者计算响应值 $z = r + ek \pmod{n}$，并发送给验证者；验证者验证以下等式是否成立：

$$zP_1 \stackrel{?}{=} T_1 + eQ_1 \quad \text{且} \quad zP_2 \stackrel{?}{=} T_2 + eQ_2. \tag{3}$$

协议分析 零知识性：证明者的随机选择 r 和挑战响应的结构，确保 k 不会被泄露；完备性：若 $Q_1 = kP_1, Q_2 = kP_2$，则验证等式必然成立；可靠性：无法生成伪造证明，因为挑战 e 是由验证者随机生成。

（1.2）椭圆曲线范围证明（Range Proof）：

证明一个点 $Q = kP$ 的标量 k 位于区间 $[a, b]$，但不暴露 k 的具体值。

问题设置 椭圆曲线 E 上的基点 P；点 $Q = kP$；区间 $[a, b]$；证明者私有输入：标量 k；验证者公开输入：P, Q, a, b。

协议流程 证明者将 k 分解为 $w_1 = k - a$ 和 $w_2 = b - k$，确保 $w_1 \geq 0$ 且 $w_2 \geq 0$；分别计算点 $C_1 = w_1 P$ 和 $C_2 = w_2 P$，并将 C_1, C_2 发送给验证者；利用零知识证明，证明 w_1, w_2 为非负数：证明 C_1, C_2 可表示为若干点平方和；验证 $Q = aP + C_1$ 且 $bP - Q = C_2$。

协议分析 零知识性：k 的具体值对验证者完全隐藏；完备性：若 $k \in [a, b]$，则等式和非负性证明均成立；可靠性：无法伪造区间外的点，因为椭圆曲线离散对数问题确保标量的唯一性。

（2）ECDLP 的具体应用实例：

（2.1）椭圆曲线签名验证：

ECDLP 在数字签名中被用来验证签名的合法性。例如，ECDSA 使用以下方法：

利用公开密钥 $Q = dP$ 和签名 (r, s)，验证者计算点 $R = u_1 P + u_2 Q$；验证 R 的横坐标是否等于 r，若相等则签名合法。

（2.2）隐私交易中的零知识验证：

在隐私交易中，使用椭圆曲线证明发送者确实拥有交易输出的所有权，同时隐藏金额和交易细节：

利用 ECDLP 确保发送者的私钥正确；使用范围证明保证交易金额有效。