信息安全数学基础期末探究报告

蒋枘言 2313546

第一部分 大整数分解问题

一、问题概述

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.

大整数分解是数论中的一个基本问题，从其诞生到现在已有数百年历史，真正引起数学家、计算机科学家以及密码学家的极大关注是近几十年的事情，它不仅是公钥加密算法 RSA 的最直接的攻击手段，也是 RSA 安全性分析最关键的切入点，因而整数分解问题的任何一点进展都将引起密码学界的关注. 大数分解问题属于 NP 类问题，但既未被证明是多项式时间可解的 P 问题，也未被证明是 NP 完备问题. 另外，大整数分解问题的研究直接影响到数论及通信领域中其他一些问题的解决，因而对其研究具有极其重要的理论意义和应用价值.

二、解决思路

解决大整数分解问题有一些常见思路. 试除法，也称 Eratosthenes 筛法，是最早也是最简单的整数分解方法. 17 世纪法国著名数学家费马用平方差来重写要分解的大整数，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，直到 20 世纪七八十年代，一些比较高效的算法才逐渐出现. 1974 年，J.M.Pollard 提出了 Pollard $\rho-1$ 分解法，它是基于费马小定理的分解方法. 1975 年，J. M.Pollard 又提出了 Pollard ρ 分解法，它是基于“生日悖论”的分解方法，J.M.Pollard 提出的这两种算法都是概率算法，有一定的局限性. 1986 年，H. Lenstra 提出了更高效的椭圆曲线法. 1975 年，Morrison 和 Brillhart 提出了连分式分解法，它是最早的基于分解基的分解算法. 1981 年，C.Pomerance 在线性筛法的基础上提出了二次筛法，二次筛法整体来说仍属于分解基算法. 二次筛法的成名在于它成功分解了 RSA-129，当时轰动世界. 二次筛法是当前用于分解十进制位数小于 110 位的整数的最快方法，渐近计算时间为 $e^{(1+O(1))(\ln n)^{\frac{1}{2}}(\ln\ln n)^{\frac{1}{2}}}$. 随后又出现了数域筛法，数域筛法是目前运行效率最高的整数分解算法之一，数城筛法则是当前分解大整数（十进制位数大于 110 位的整数）的最快方法，渐近计算时间为 $e^{(1.923+O(1))(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}}$. 1999 年，Peter Shor 提出了在量

子计算机上运行的整数分解算法，被称作 Shor 算法. 该算法能够在多项式时间内分解大整数，这就意味着在量子计算环境下大整数分解问题不再是一个困难问题.

二次筛法和数域筛法是当前效率最高的分解算法，但是涉及的数学知识较为复杂，因此下面只详细介绍基础的费马分解算法、Pollard ρ 分解算法和 Pollard ρ －1 分解算法.

（一）试除法（Eratosthenes 筛法）

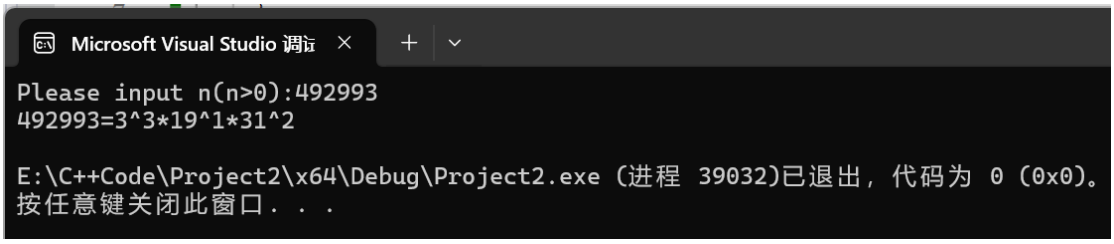
试除法最简单的整数分解算法，通过逐一尝试小于 N 的每个素数来试除待分解的整数. 如果找到一个素数能够整除 N，则这个数就是 N 的一个因子. 事实上，由于因数成对出现，我们只需要尝试小于 $\sqrt{N}$ 的每个素数即可. 试除法肯定能够完全分解大整数，但时间复杂度较高，达到了约 $O\left(\frac{\sqrt{N}}{\ln N}\right)$，不适用于大型素数的判断.

用 C++实现试除法的代码如下：

```
#include <iostream>
using namespace std;
void primeFactorization(int n) {
    if (n == 1) {
        cout << "1=1^1" << endl;
        return;
    }
    int factor[100]{}, exp[100]{};
    int count1 = 0, count2 = 0;
    int initial = n;
    for (int i = 2; i <= initial; i++) {
        if (n % i == 0) {
            if (count1 == 0) {
                factor[count2] = i;
                exp[count2]++;
            }
            else {
                if (i == factor[count2]) {
                    exp[count2]++;
                }
                else {
                    count2++;
                    factor[count2] = i;
                    exp[count2]++;
                }
            }
            count1++;
            n /= i;
            i = 1;//因为进入下一个循环要执行一遍 i++，故要令 i = 1
        }
    }
    cout << initial << "=";
```

```
        for (int i = 0; i < count2; i++) {
            cout << factor[i] << "^" << exp[i] << "*";
        }
        cout << factor[count2] << "^" << exp[count2] << endl;
}
int main() {
        cout << "Please input n(n>0):";
        int n;
        cin >> n;
        primeFactorization(n);
}
```

我们对整数 492993 进行测试，得到结果 $492993=3^3\times19\times31^2$，如下图所示.



（二）费马分解算法

我们很容易得到

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

那么，我们想对整数 n 进行分解，由 $n=ab$ 可得

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

移项即得

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

设 $m=\frac{a+b}{2}$，则 $\left(\frac{a-b}{2}\right)^2=m^2-n$，当寻找到正确的 a 和 b 时，m^2-n 应该是一个完全平方数. 于是，我们首先确定最小的整数 k，使得 $k^2\geq n$. 然后，对下面的数列

$$k^2-n,(k+1)^2-n,(k+2)^2-n,\cdots,\left(\frac{n+1}{2}\right)^2-n$$

按顺序进行测试，直到找到一个整数 $m\geq\sqrt{n}$ 使得 m^2-n 是一个完全平方数，从而

也就找到了一对因子. 当然，对于上面数列的最后一个数$\left(\frac{n+1}{2}\right)^2-n=\left(\frac{n-1}{2}\right)^2$一定是一个完全平方数，此时如果一直运行到上面数列的最后一个数才找到完全平方数，那么 n 就是素数.

例如，我们对 $n=119143$ 进行整数分解. 因为 $345^2<119143<346^2$，所以 $k=346$. 有

$$\begin{aligned}&346^2-119143=573\\&347^2-119143=1266\\&348^2-119143=1961\\&349^2-119143=2658\\&350^2-119143=3357\\&351^2-119143=4058\\&352^2-119143=4761=69^2\end{aligned}$$

所以 $n=(352-69)(352+69)=283\times421$.

费马分解算法虽然在分解两个因子接近的合数时表现出一定的效率，但是在合数的两个因子相差较大时，算法效率会显著降低。此外，算法的计算复杂度同样较高，可能会增加加密或解密操作的计算负担，影响性能。对于含大质数因子的整数，费马分解算法的计算时间过长，使得其在实际应用中受到限制。

（三）Pollard ρ 分解算法

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

$$x_{k+1}\equiv f(x_k)(\mathrm{mod}\,n)$$

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，但是又属于不同的模 n 的剩余类. 因为 $d\mid(x_k-x_j)$ 而 $n\nmid(x_k-x_j)$，所以 (x_k-x_j,n) 是 n 的非平凡因子.

例如，想要求 $n=2189$ 的一个非平凡因子，选择 $x_0=1$ 和 $f(x)=x^2+1$，得到序列 $x_1=2,x_2=5,x_3=26,x_4=677,x_5=829,\cdots$，由于 $(x_5-x_3,n)=11$，故 11 是 2189 的一个因子.

在该算法中，如果我们对序列的任意两个量的差都进行计算的话，那么计算

的开销就会太大了. 改进方法就是只对 $(x_{2k}-x_k)$ 进行计算.

例如，想要求 $n=30623$ 的一个非平凡因子，选择 $x_0=3$ 和 $f(x)=x^2-1$，得到如下序列

$$x_1=8, x_2=63, x_3=3968, x_4=4801,$$
$$x_5=21104, x_6=28526, x_7=18319, x_8=18926, \cdots.$$

由于

$$(x_2-x_1, n)=1, (x_4-x_2, n)=1,$$
$$(x_6-x_3, n)=1, (x_8-x_4, n)=113,$$

故 113 是 30623 的一个因子.

Pollard ρ 算法的性能取决于起始值 x_0 的选择和函数 $f(x)$的设计. 在实践中，通常需要多次尝试不同的起始值和函数来增加成功分解整数 N 的机会. 若算法在运行预先规定的步数后不成功，可以重新选取 x_0 或者 $f(x)$再开始处理. 但 Pollard ρ 算法并不是一个确定性算法，它可能在某些情况下失败，即无法将整数 N 分解为素因数. 对于特定的整数 N，Pollard ρ 算法的效率可能会有所不同. 在实际应用中，通常与其他分解算法结合使用，以提高分解的成功率.

Pollard ρ 算法的原理是概率论中的“生日悖论”. “生日悖论”研究的问题是：一个有 k 名同学的班级中，至少有两名同学生日在同一天的概率是多少. 不妨忽略闰年，设一年有 365 天，至少有两名同学生日在同一天的概率为 $P(k)$，根据概率论的知识可以得出

$$P(k)=1-\prod_{i=1}^{k-1}\left(1-\frac{i}{365}\right).$$

当 k 按顺序取一些的值的时候，$P(k)$的值如下表所示：

k	$P(k)$	k	$P(k)$	k	$P(k)$
1	0	9	0.094624	17	0.315008
2	0.002740	10	0.116948	18	0.346911
3	0.008204	11	0.141141	19	0.379119
4	0.016356	12	0.167025	20	0.411438
5	0.027136	13	0.194410	21	0.443688
6	0.040462	14	0.223103	22	0.475695
7	0.056236	15	0.252901	23	0.507297
8	0.074335	16	0.283604	100	0.999999693

可见，一个班级中有 23 名同学时，至少有两名同学生日在同一天的概率就已经超过了 50%. 当有 100 名同学时，至少有两名同学生日在同一天的概率已经非常接近 1，也就是说几乎可以肯定至少存在两名同学生日在同一天. 这个结论与人们的直觉相反，所以称为“生日悖论”.

将此问题进行推广：设 G 是一个有限群，$|G|=n$，对 G 中的元素进行随机抽样，每次抽样随机从 G 中选择一个元素，连续进行 k 次抽样，则 G 中至少有一个元素被选中两次的概率为

$$P(k,n)=1-\prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right).$$

易知当 x 足够小的时候，有 $1-x\approx \mathrm{e}^{-x}$，所以当 n 足够大的时候有

$$1-\frac{i}{n}\approx \mathrm{e}^{-\frac{i}{n}}.$$

故有

$$P(k,n)=1-\prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right)\approx 1-\prod_{i=1}^{k-1}\mathrm{e}^{-\frac{i}{n}}=1-\mathrm{e}^{-\sum_{i=1}^{k-1}\frac{i}{n}}=1-\mathrm{e}^{-\frac{k(k-1)}{2n}},$$

当 k 足够大的时候有 $k(k-1)\approx k^2$. 所以，当 k 和 n 都足够大的时候有

$$P(k,n)\approx 1-\mathrm{e}^{-\frac{k(k-1)}{2n}}\approx 1-\mathrm{e}^{-\frac{k^2}{2n}},$$

得

$$k\approx\sqrt{n\left(2\ln\left(\frac{1}{1-P(k,n)}\right)\right)}.$$

将 $P(k,n)=0.5$ 代入得 $k\approx\sqrt{n\cdot 2\ln 2}\approx\sqrt{n}$. 所以当 n 足够大且 k 的数量级大于 $\sqrt{n}$ 的数量级时，基本上可以使 $P(k,n)>0.5$. 在这里的 Pollard ρ 算法中，模 d 的剩余类但又不是模 n 的剩余类构成群 G，从 G 的元素中不断抽取 x_k 和 x_j，有一定概率抽到 x_k 和 x_j 属于同一个剩余类. 于是就能够求出 n 的非平凡因子.

（四）Pollard $\rho-1$ 分解算法

这个方法对如下情况起作用：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 $(\rho-1)\mid k!$，接

着选择一个整数 a，使$1<a<\rho-1$，计算$a^{k!}\equiv m(\mathrm{mod}\, n)$. 因为存在整数 j 使得 $k!=j(\rho-1)$，所以

$$m\equiv a^{k!}\equiv a^{j(\rho-1)}\equiv (a^{\rho-1})^{j}\equiv 1^{j}\equiv 1(\mathrm{mod}\,\rho),$$

也就是$\rho\,|\,(m-1)$. 因此$(m-1,n)>1$，只要$m\not\equiv 1(\mathrm{mod}\, n)$，$(m-1,n)$一定为 n 的非平凡因子.

例如，想要求$n=2987$的一个非平凡因子，选择$a=2,k=7$，

$$2^{7!}\equiv 755(\mathrm{mod}\, 2987),$$
$$(755-1,2987)=29,$$

所以，29 是 2987 的一个非平凡因子.

三、在密码学中的应用

大整数分解问题在密码学中的应用主要体现在公钥密码体制中. 其中，RSA 算法是一种基于大整数分解问题的公钥密码体制. 大整数分解问题是国际数学界几百年来尚未解决的难题，也是现代密码学中公开密钥 RSA 算法密码体制建立的基础.

在 RSA 算法中，加密和数字签名的安全性依赖于将大整数 N 分解为两个较大的素数 p 和 q 的乘积. 这个分解过程是非常困难的。因此，对于攻击者来说，找到大整数 N 的素因数分解是一个极其困难的任务. 当一个 RSA 密钥对生成时，需要选择两个大素数 p 和 q，并计算它们的乘积 N. 如果一个攻击者能够成功地分解 N，那么它可以利用这些质因数来推导出私钥，并将加密的消息解密，或者伪造数字签名. 因此，大整数分解问题的困难性是保护 RSA 算法安全性的基础.

简单来说，RSA 算法的安全性依赖于大整数分解的难度. 只要密钥长度足够长，目前来说用 RSA 加密的信息实际上是不能被破解的.

第二部分 二次剩余问题

一、问题概述

我们先来定义 Blum 数. 若整数 n 满足 $n=pq$，其中 p 和 q 是素数且 $p\equiv 3(\mathrm{mod}\,4), q\equiv 3(\mathrm{mod}\,4)$，则 n 称为 Blum 数. 除了特殊说明，以下提到的二次剩余均在 $\mathbb{Z}_n^*$ 中，其中 n 是 Blum 数.

令 $n=pq$ 是两个大素数之积. 对于一个任意选取的模 n 的二次剩余 a，求出一个 x 使得 $x^2\equiv a(\mathrm{mod}\,n)$ 的问题称为二次剩余问题. 对于一个随机选取的整数 $a\in\mathbb{Z}_n^*$，判定 a 是否为模 n 的二次剩余是一个困难问题.

要判定 a 是否为模 n 的二次剩余时，我们可以使用雅可比符号 $\left(\frac{a}{n}\right)$（若 n 为素数，则 $\left(\frac{a}{n}\right)$ 为勒让德符号）. 我们先定义勒让德符号：设 p 是奇素数，$(a,p)=1$，勒让德符号

$$\left(\frac{a}{p}\right)=\begin{cases}1, & \text{若}a\text{是模}p\text{的二次剩余},\\ -1, & \text{若}a\text{是模}p\text{的二次非剩余}.\end{cases}$$

若正奇数 $m=p_1p_2\cdots p_r$ 是奇素数 $p_i(i=1,2,\cdots,r)$ 的乘积，定义雅可比符号

$$\left(\frac{a}{m}\right)=\left(\frac{a}{p_1}\right)\left(\frac{a}{p_2}\right)\cdots\left(\frac{a}{p_r}\right).$$

当 p 是奇素数，$(a,p)=1$ 时，我们有结论

$$\left(\frac{a}{p}\right)\equiv a^{\frac{p-1}{2}}(\mathrm{mod}\,p).$$

设 m,n 是互素的正奇数，对于勒让德符号和雅可比符号均有二次互反律

$$\left(\frac{n}{m}\right)\left(\frac{m}{n}\right)=(-1)^{\frac{m-1}{2}\cdot\frac{n-1}{2}},$$

也可写成

$$\left(\frac{n}{m}\right)=(-1)^{\frac{m-1}{2}\cdot\frac{n-1}{2}}\left(\frac{m}{n}\right).$$

若能成功分解大整数 n，则雅可比符号可以帮助判断 a 是否为模 n 的二次剩余.

用 C++实现勒让德符号的代码如下：

```
#include<iostream>
using namespace std;

int gcd(int a, int b) {
	if (a % b == 0) {
		return b;
	}
	else {
		return gcd(b, a % b);
	}
}
bool isOddPrime(int a) {
	if (a <= 2) {
		return false;
	}
	for (int i = 2; i < a; i++) {
		if (a % i == 0) {
			return false;
		}
	}
	return true;
}

//模幂运算 base^exp mod
long long squareAndMultiply(long long base, long long exp, long long mod) {
	long long result = 1;
	base = base % mod;
	while (exp > 0) {
		if (exp % 2 == 1) {
			result = (result * base) % mod;
		}
		base = (base * base) % mod;
		exp >>= 1;
	}
	return result;
}

//计算 Legendre 符号(a/p)
int Legendre(int a, int p) {
	if (!isOddPrime(p)) {
		cout << "p 不是奇素数" << endl;
		return -1000;
	}
	if (gcd(a, p) != 1) {
		cout << "a 和 p 不互素" << endl;
		return -2000;
	}
	if (squareAndMultiply(a, (p - 1) / 2, p) % p == 1) {
		return 1;
	}
	if (squareAndMultiply(a, (p - 1) / 2, p) % p == p - 1) {
		return -1;
	}
	return -3000;
```

```
}

int main() {
	int a = 0;
	int p = 0;
	cout << "a=";
	cin >> a;
	cout << "p=";
	cin >> p;

	if (!isOddPrime(p)) {
		cout << "p 不是奇素数。" << endl;
		return 0;
	}
	if (gcd(a, p) != 1) {
		cout << "a 和 p 不互素！" << endl;
		return 0;
	}

	if (Legendre(a, p) == 1) {
		cout << "Legendre(" << a << "/" << p << ")=" << 1 << endl;
		cout << a << "是模" << p << "的二次剩余。" << endl;
	}
	else {
		if (Legendre(a, p) == -1) {
			cout << "Legendre(" << a << "/" << p << ")=" << -1 << endl;
			cout << a << "不是模" << p << "的二次剩余。" << endl;
		}
		else {
			cout << "(" << a << "/" << p << ")=" << Legendre(a, p) << "ERROR" << endl;
		}
	}
}
```

计算勒让德符号$\left(\frac{137}{227}\right)$（即判断 137 是否为模 227 的二次剩余）得$\left(\frac{137}{227}\right)=-1$，即 137 不是模 227 的二次剩余.

二、解决思路

（一）穷举法

对于方程$x^2 \equiv a \pmod{n}$，将 0 至$n-1$依次带入验证，通过穷举的方式一定可以判断 a 是否为模 n 的二次剩余，若是，还可求出一个 x. 但是对于大整数 n，穷举法的效率显然太低，计算开销太大.

用 C++实现穷举法的代码如下：

```
#include<iostream>
using namespace std;

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

//判断 a≡b(mod m)是否成立
bool isMod(long long a, long long b, long long m) {
	return (a - b) % m == 0 ? true : false;
}

int main() {
	int a = 0;
	int m = 0;
	cout << "a=";
	cin >> a;
	cout << "m=";
	cin >> m;

	if (m <= 1) {
		cout << "不满足 m>1。" << endl;
		return 0;
	}
	if (gcd(a, m) != 1) {
		cout << "a 和 m 不互素！" << endl;
		return 0;
	}

	bool res = false;
	int count = 0;
	int* pSol = new int[m];
	for (int i = 0; i < m; i++) {
		if (isMod(i * i, a, m)) {
			res = true;
			pSol[count] = i;
			count++;
		}
	}

	if (res == true) {
		cout << a << "是模" << m << "的二次剩余。" << endl;
		cout << "同余方程 x^2≡" << a << "(mod " << m << ")的解为 x≡";
		for (int i = 0; i < count - 1; i++) {
			cout << pSol[i] << ",";
		}
		cout << pSol[count - 1];
		cout << "(mod " << m << ")。" << endl;
	}
	else {
		cout << a << "不是模" << m << "的二次剩余。" << endl;
```

```
    }
    delete[]pSol;
}
```

我们对方程 $x^2 \equiv 35 (\mathrm{mod}\, 299)$ 进行测试，发现 35 是 299 的二次剩余，方程的解为 $x \equiv 55, 101, 198, 244 (\mathrm{mod}\, 299)$.

（二）Cipolla 算法

Cipolla 算法基于数论中的一些基本概念，如二次剩余、勒让德符号和欧拉判别准则. 算法的核心思想是在一个扩充的数域中寻找解，这个数域是通过引入一个虚数单位 i 来定义的，使得 $\mathrm{i}^2 \equiv a^2 - n (\mathrm{mod}\, p)$，其中 a 是一个随机选择的数，使得 $a^2 - n$ 是模 p 的二次非剩余. 易知模 p 的二次剩余和二次非剩余的个数均为 $\frac{p-1}{2}$，因此随机选择的 a 有一半的概率使得 $a^2 - n$ 是非剩余.

我们定义了 i 之后，所有的数都可以表示为 $A + B\mathrm{i}$ 的形式，其中 A, B 是 $[0, p-1]$ 范围内的整数. 再计算 $y \equiv (a + \mathrm{i})^{\frac{p+1}{2}} (\mathrm{mod}\, p)$，那么 $\pm y$ 就是方程 $x^2 \equiv a (\mathrm{mod}\, n)$ 的解.

Cipolla 算法的期望复杂度是 $O(\log^2 p)$，这比传统的暴力搜索方法要高效得多. 算法的随机性主要在于选择 a 的过程，但因为找到合适的 a 的概率较高，所以算法在实践中表现良好.

三、在密码学中的应用

二次剩余问题在密码学中的应用非常广泛，Rabin 加密算法、GM 加密算法等等都是二次剩余问题的经典应用.

GM 加密算法是一个概率加密算法. 在 GM 加密算法中，私钥为两个素数 (p, q)，公钥为 (n, t)，其中 $n = pq$，t 是一个随机的模 p 和 q 的二次非剩余. 对于一个明文比特 m，加密过程为

$$c \equiv \begin{cases} tx^2 (\mathrm{mod}\, n), m = 1, \\ x^2 (\mathrm{mod}\, n), m = 0. \end{cases}$$

其中 x 为随机选取的整数，且 $1 \le x \le n-1$. 解密过程为

$$m \equiv \begin{cases} 1, \left(\frac{c}{p}\right) = \left(\frac{c}{q}\right) = -1, \\ 0, \left(\frac{c}{p}\right) = \left(\frac{c}{q}\right) = 1. \end{cases}$$

综上所述，二次剩余问题不仅是数论中的一个重要概念，也是现代密码学中不可或缺的一部分，它在公钥密码体制、数字签名、素性测试以及多种密码学协议中都有着关键的应用.

第三部分 同态加密算法

一、密码原语简介

（一）基本概念

同态加密（Homomorphic Encryption, HE）是满足同态运算性质的加密技术，它允许对加密数据进行特定的操作，而不需要先将数据解密，从而实现数据的“可算不可见”。这种技术的核心特点是，对密文进行操作后的结果解密，与先对明文操作再加密的结果相同，保持了同态性。

（二）分类

1. 部分同态加密（Partially Homomorphic Encryption，PHE）

仅支持对密文进行有限次的加法或乘法操作. 例如，RSA 算法和 ElGamal 加密算法分别提供了乘法同态和加法同态的特性.

2. 近似同态加密/类同态加密（Somewhat Homomorphic Encryption，SWHE）

类同态加密支持有限次数的加法和乘法操作，但计算深度有限.

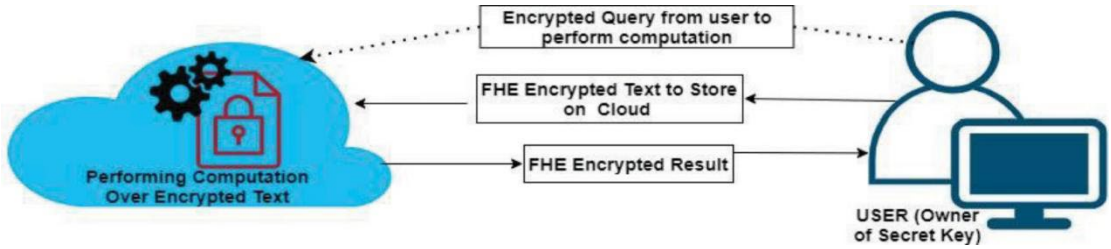
3. 全同态加密（Fully Homomorphic Encryption，FHE）

支持对密文进行任意次数和类型的操作，包括加法、乘法以及更复杂的计算。全同态加密是目前同态加密技术中最强的类型，是同态加密研究的理想目标.

二、应用场景

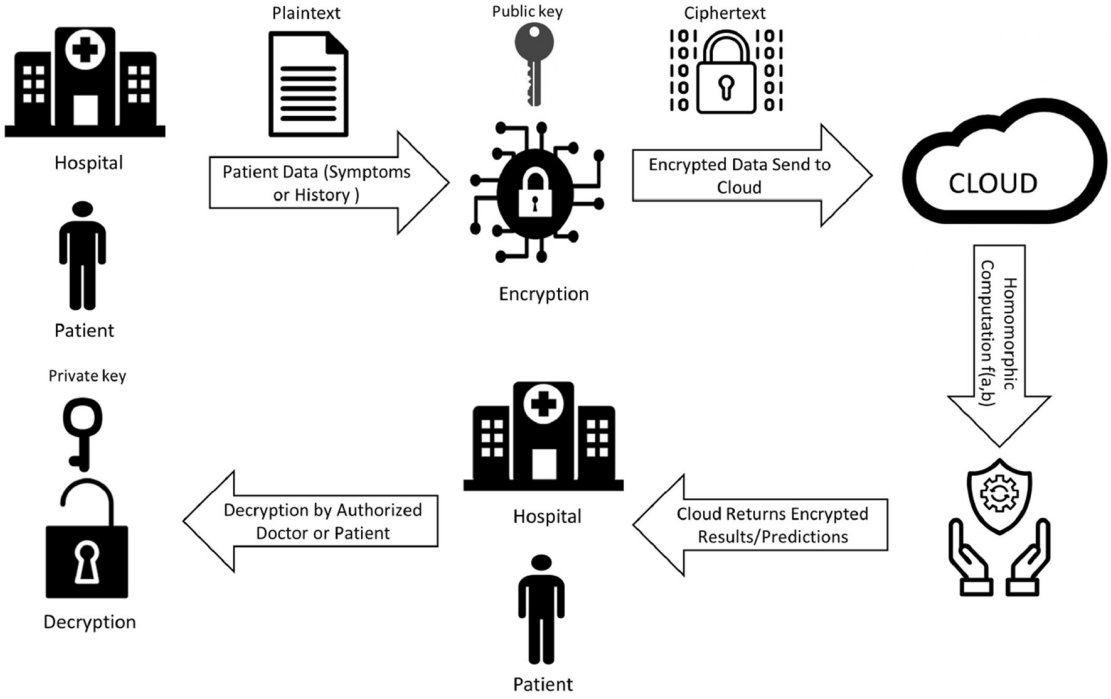
（一）云计算

用户的数据在加密状态下进行处理，云服务提供商无法直接访问或解读数据的原始内容，由云服务器直接对密文进行计算，计算结果解密后与直接在明文上操作的结果一致，从而确保了用户数据的隐私安全，无需担心数据泄露。



（二）医疗健康

保护患者的病历、医疗影像等敏感信息，安全地分析医疗数据，而无需将数据解密，有效防止患者隐私泄露。



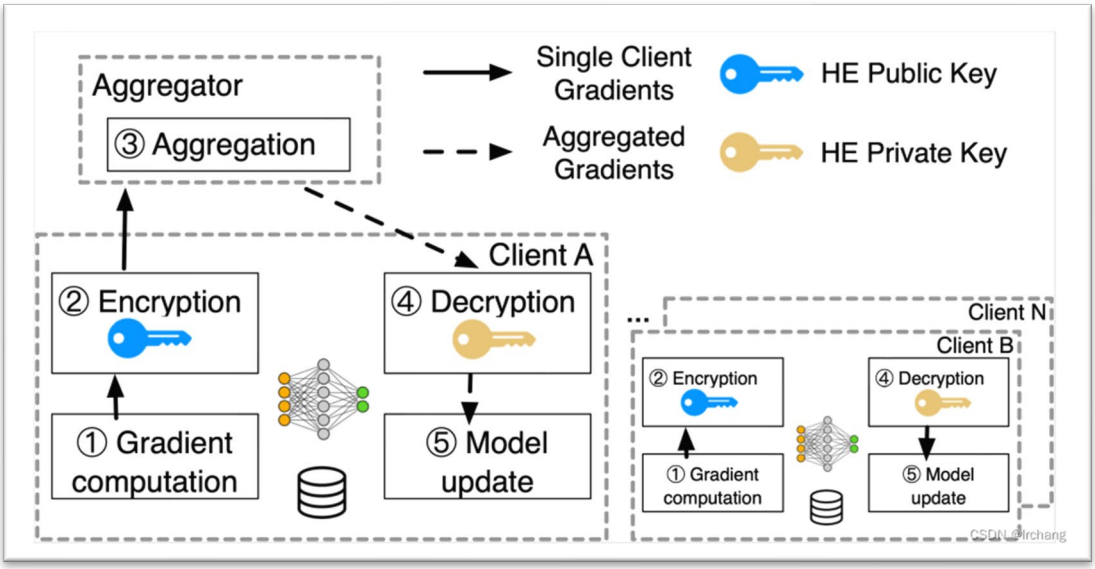
（三）金融服务

处理大量包括个人身份信息、信用卡信息、贷款信息等的客户数据时保护客户数据的隐私和安全，避免因数据泄露而导致的法律和经济风险。

（四）联邦学习

联邦学习（Federated Learning）是一种分布式机器学习方法，它允许多个客户端（如移动设备、浏览器或分布式服务器）协作训练一个共享模型，同时保持数据的隐私和安全。

同态加密用于联邦学习中的参数交互计算过程，实现预测模型的联合确立。在联邦学习中，多个参与方可以在保证各自数据隐私的同时实现联合机器学习建模，即在不获取对方原始数据的情况下利用对方数据提升自身模型的效果。



三、具体算法及数学问题

（一）Paillier 算法

1. 密钥生成

生成两个大素数 p 和 q，计算 $n=pq, g=n+1, \lambda=\mathrm{lcm}(p-1,q-1)$，定义函数 $L(x)=\dfrac{x-1}{n}$，计算 $\mu=(L(g^{\lambda} \bmod n^2))^{-1}(\bmod n)$，得到公钥 (n,g)，私钥 (λ,μ).

2. 加密过程

选取明文 m，选择随机数 $r(0<r<n)$，计算得到密文 $c=g^m r^n \bmod n^2$.

3. 解密过程

计算得到明文 $m=\mu L(c^{\lambda} \bmod n^2)(\bmod n)=\dfrac{L(c^{\lambda} \bmod n^2)}{L(g^{\lambda} \bmod n^2)}$.

4. 正确性分析

由最小公倍数可得

$$(p-1)\mid\lambda,(q-1)\mid\lambda,$$

可令

$$\lambda=k_1(p-1)=k_2(q-1).$$

接着由欧拉定理得

$$g^{\varphi(p)}=g^{p-1}\equiv 1(\bmod p),$$

故

$$g^{\lambda}=g^{k_1(p-1)}\equiv 1(\mathrm{mod}\ p)\ ,$$

同理

$$g^{\lambda}\equiv 1(\mathrm{mod}\ q)\ .$$

所以

$$(g^{\lambda}-1)\mid p,(g^{\lambda}-1)\mid q\ ,$$

所以

$$(g^{\lambda}-1)\mid pq\ ,$$

即

$$(g^{\lambda}-1)\mid n\ .$$

可得

$$(g^{\lambda}-1)\mid n^2,g^{\lambda}\equiv 1(\mathrm{mod}\ n^2),$$
$$(g^{\lambda}\ \mathrm{mod}\ n^2)\equiv 1(\mathrm{mod}\ n).$$

可令

$$g^{\lambda}\ \mathrm{mod}\ n^2=nk_g+1\ ,$$

则

$$L(g^{\lambda}\ \mathrm{mod}\ n^2)=k_g\ .$$

由二项式定理得

$$(1+kn)^m=\mathrm{C}_m^0(kn)^0+\mathrm{C}_m^1(kn)^1+\mathrm{C}_m^2(kn)^2+\cdots+\mathrm{C}_m^m(kn)^m\equiv 1+kmn(\mathrm{mod}\ n^2)\ ,$$

故

$$g^{m\lambda}=(nk_g+1)^m\equiv k_gmn+1(\mathrm{mod}\ n^2)\ ,$$

同理有

$$r^{n\lambda}=(nk_r+1)^n\equiv k_rn^2+1\equiv 1(\mathrm{mod}\ n^2)\ ,$$

故

$$L(c^{\lambda}\ \mathrm{mod}\ n^2)=L(g^{m\lambda}r^{n\lambda}\ \mathrm{mod}\ n^2)=L(k_gmn+1)=k_gm\ .$$

所以

$$\frac{L(c^{\lambda}\ \mathrm{mod}\ n^2)}{L(g^{\lambda}\ \mathrm{mod}\ n^2)}=\frac{k_gm}{k_g}=m\ ,$$

得证.

5. 同态性分析

Paillier 算法是加法同态算法，对应到密文是乘法操作. 现有明文 m_1,m_2，加密后得到密文 c_1,c_2，那么有

$$c_1c_2 = g^{m_1+m_2}(r_1r_2)^n \bmod n^2, \mathrm{Dec}(c_1c_2) = m_1 + m_2.$$

我们可以发现 $\mathrm{Enc}(m_1+m_2) = g^{m_1+m_2}(r)^n \bmod n^2$，实际上此密文和 c_1c_2 没有区别，因为 r 为随机选取.

6. 安全性分析

Paillier 算法的安全性主要有关两个数学问题：大整数分解问题和复合剩余类问题. 大整数分解问题在前文已经详细地说明，这里再简略地说一下复合剩余类问题.

复合剩余类问题（Decisional Composite Residuolity Assumption，DCRA）指的是：给定一个合数 n 和整数 z，很难确定是否存在一个整数 y，使得 $z \equiv y^n (\mathrm{mod}\, n^2)$. 即判断 z 是不是模 n^2 的 n 阶剩余是很困难的.

复合剩余类问题的困难性是 Paillier 算法的安全性的基础. 由于复合剩余类问题的困难性，Paillier 算法能够抵抗多种攻击，包括选择明文攻击（CPA）和选择密文攻击（CCA）. 这意味着即使攻击者拥有大量的密文和对应的明文，或者能够选择密文并获取对应的明文，也无法有效地破解 Paillier 算法的加密.

（二）ElGamal 算法

1. 密钥生成

生成一个大素数 p，g 是 p 的原根，随机选取一个私钥 $x(1 < x < p-1)$，计算得到公钥 $y \equiv g^x (\mathrm{mod}\, p)$.

2. 加密过程

选取明文 m，选择一个随机数 $k(1 < k < p-1)$，计算得到密文 (c_1, c_2)，其中 $c_1 \equiv g^k (\mathrm{mod}\, p), c_2 \equiv my^k (\mathrm{mod}\, p)$.

3. 解密过程

计算 $k' = k^{-1} (\mathrm{mod}\, p-1)$，得到明文 $m \equiv c_2 c_1^{-x} y^{k'c_2} (\mathrm{mod}\, p)$.

4. 同态性分析

ElGamal 算法是乘法同态算法，对应到密文是乘法操作. 现有明文 m_1,m_2，加

密后得到密文 c_1,c_2，那么有 $\mathrm{Dec}(c_1c_2)=m_1m_2$.

5. 安全性分析

ElGamal 算法的安全性主要基于离散对数问题，在这里是有限域上的离散对数问题. 它指的是：设 p,q 为两个素数，$G=\{g^i \mid 0\le i\le q-1, g\in\mathbb{Z}_p^*\}$ 为阶为 q 的有限域 $\mathbb{Z}_p^*$ 上的乘法群. 给定一个元素 $y\in G$，找到一个整数 $x\in\mathbb{Z}_q$，使得 $y=g^x$ 的问题.

ElGamal 加密算法的安全性依赖于离散对数问题的难解性. ElGamal 算法在理论上被认为比 RSA 算法更安全，因为理论上存在更快的方法能够分解大质数，这可能会降低 RSA 的安全性；而 ElGamal 算法依赖的离散对数问题在数学上更难解决.

离散对数问题有一些求解思路，比如穷举法、商客法（小步大步法）、Pollard ρ 算法、Pohlig-Hellman 算法、指数积分法等等，这里不详细展开说明.

（三）BFV 加密方案

1. 密钥生成

从特定的分布（通常是离散高斯分布）中随机采样一个多项式 **s** 作为私钥，从多项式环 $R_q=\mathbb{Z}_q[x]/(x^n+1)$ 中随机采样一个多项式 **a**，从噪声分布（通常是离散高斯分布）中随机采样一个多项式 **e**，计算 $\mathbf{b}=-(\mathbf{a}\cdot\mathbf{s}+\mathbf{e})(\mathrm{mod}\,q)$，公钥为$(\mathbf{a},\mathbf{b})$.

2. 加密过程

选取明文 m，将其编码到多项式环上（一般使用 SIMD 编码）得到 $\mathbf{m}\in R_t$，从 R_q 中随机采样一个多项式 **u**，从噪声分布中随机采样两个多项式 $\mathbf{e_1},\mathbf{e_2}$，计算 $\mathbf{c_1}=\mathbf{a}\cdot\mathbf{u}+\mathbf{e_1}+\Delta\cdot\mathbf{m}(\mathrm{mod}\,q)$，$\mathbf{c_2}=\mathbf{b}\cdot\mathbf{u}+\mathbf{e_2}(\mathrm{mod}\,q)$，其中 $\Delta=\left\lfloor\frac{q}{t}\right\rfloor$，密文为$(\mathbf{c_1},\mathbf{c_2})$.

3. 解密过程

计算 $\mathbf{m}'=\mathbf{c_0}+\mathbf{c_1}\cdot\mathbf{s}(\mathrm{mod}\,q)$，计算得到密文 $\mathbf{m}=\left\lfloor\frac{t\mathbf{m}'}{q}\right\rceil\ (\mathrm{mod}\ t)$.

4. 同态性分析

BFV 加密方案是全同态加密算法，支持加法和乘法操作. 现有编码后的明文 $\mathbf{m_1},\mathbf{m_2}$，加密后得到密文$(\mathbf{c_{11}},\mathbf{c_{12}}),(\mathbf{c_{21}},\mathbf{c_{22}})$，那么有

$$(\mathbf{c_{11}},\mathbf{c_{12}})+(\mathbf{c_{21}},\mathbf{c_{22}})=(\mathbf{c_{11}}+\mathbf{c_{21}},\mathbf{c_{12}}+\mathbf{c_{22}})$$
$$=\big(\mathbf{a}(\mathbf{u_1}+\mathbf{u_2})+(\mathbf{e_{11}}+\mathbf{e_{21}})+\Delta\cdot(\mathbf{m_1}+\mathbf{m_2}),\mathbf{b}(\mathbf{u_1}+\mathbf{u_2})+(\mathbf{e_{12}}+\mathbf{e_{22}})\big).$$

多出来的项$(\mathbf{e_{11}}+\mathbf{e_{21}})$和$(\mathbf{e_{12}}+\mathbf{e_{22}})$称为噪声. 噪声会在同态运算中不断增大，但我们有一种减小噪声积累的方法，即 Bootstrapping. Bootstrapping 是一种通过重复抽样来估计一个统计量的分布情况的统计方法，它就可以使噪声保持在可控的范围内.

5. 安全性分析

BFV 加密方案的安全性依赖于带误差学习问题的扩展——环上带误差学习问题（Ring Learning With Errors，RLWE）的难解性. RLWE 问题指的是给定一个多项式环 $R_q=\mathbb{Z}_q[x]/(x^n+1)$（其中 q 是一个大素数）和一个概率分布 χ（通常是高斯分布），再给定多项式 $\mathbf{a},\mathbf{b}\in R_q$，找到多项式 $\mathbf{s}\in R_q$ 使得 $\mathbf{b}\equiv\mathbf{a}\cdot\mathbf{s}+\mathbf{e}(\mathrm{mod}\,q)$ 是困难的. 其中 $\mathbf{e}$ 是从分布 χ 中采样得到的噪声多项式.

BFV 加密方案的安全性依赖于 RLWE 问题的复杂性。RLWE 问题的复杂性主要来源于其数学结构的复杂性和噪声的引入。由于 RLWE 问题是在环上进行的，因此其数学结构比 LWE 问题更为复杂。同时，噪声的引入也增加了问题的难度，使得从给定的样本中恢复出 $\mathbf{s}$ 变得非常困难。

四、不足与改进

（一）传统同态加密的不足

1. 效率问题

密文膨胀：在进行同态操作时，尤其是乘法操作后，密文的大小会迅速增加，这被称为密文膨胀问题.

计算开销：同态加密算法通常涉及复杂的数学运算，导致较高的计算开销.

2. 安全性限制

噪声积累：在 BFV 等方案中，每次同态操作都可能引入噪声，多次操作后可能导致噪声过大，影响解密结果.

侧信道攻击：某些实现可能容易受到侧信道攻击，泄露加密密钥或明文信息.

3. 密钥管理

密钥大小：全同态加密方案的密钥可能非常大，导致密钥管理困难.

4. 功能限制

部分同态加密：只能进行有限的计算操作，如加法或乘法，不支持复杂的算术运算.

全同态加密：虽然支持任意计算，但效率较低，不适用于大规模数据处理.

（二）一种可能的改进方式——混合同态加密

混合同态加密（Hybrid Homomorphic Encryption）结合了两种或以上的同态

加密技术，通常是为了利用各自的优势并克服单一方案的局限性. 在实际应用中，混合同态加密方案具有以下优势：

1. 功能性增强：混合方案可以结合加性同态和乘性同态的特性，从而支持更广泛的算术运算. 例如，结合 Paillier（加法同态）和 RSA（乘法同态）可以创建一个支持加法和乘法的混合方案.

2. 效率提高：某些操作在一种同态加密方案中可能效率较低，而在另一种方案中效率较高. 混合方案可以针对特定操作选择最优的加密技术，从而提高整体的计算效率.

3. 安全性增强：混合方案可以通过两层或多层加密提供更强的安全保障. 即使攻击者破解了一层加密，仍然需要破解另一层，这大大增加了破解的难度.

4. 减少密文膨胀：全同态加密方案通常伴随着较大的密文膨胀问题，而部分同态加密方案的密文膨胀较小. 混合方案可以结合两者，以减少密文的大小.

5. 适应性：混合方案可以根据不同的应用需求和安全要求灵活配置，例如，在需要高强度加解密操作的场景中使用全同态加密技术，在只需要简单加解密操作的场景中使用部分同态加密技术.