信息安全数学基础探究报告

其一：素性检测探究报告

2212000 宋奕纬

June 2024

1 问题定义

素性检测（Primality Testing）是指判定一个给定的整数是否为素数的过程。素数是仅能被 1 和自身整除的自然数，在数论和密码学中具有重要地位。有效的素性检测算法是现代密码学（如 RSA 加密算法）的基础。素数的稀有性和难以预测性使其在加密领域尤为重要。

2 应用领域

素性检测（Primality Testing）在数论和密码学中具有重要地位。它主要用于确定一个给定的整数是否为素数。素数的稀有性和难以预测性使其在加密领域尤为重要。以下将详细介绍素性检测在五个主要领域的应用。

2.1 RSA 加密算法

RSA 加密算法是最广泛使用的公钥加密算法之一，其安全性依赖于大素数的生成。在 RSA 密钥生成过程中，需要选取两个大素数 p 和 q，并通过素性检测算法（如 Miller-Rabin 测试）来验证这些数是否为素数。只有确保 p 和 q 为素数，才能生成安全的公钥和私钥对，用于加密和解密数据。通过有效的素性检测算法，可以防止选取的数为合数，从而提高 RSA 加密的安全性。

2.2 Diffie-Hellman 密钥交换协议

Diffie-Hellman 密钥交换协议是用于安全地交换加密密钥的公钥协议。该协议要求选择一个大素数 p 和一个基数 g，通过素性检测确保 p 为素数，

以防止中间人攻击和其他安全威胁。素性检测在这个过程中起到关键作用，确保所选的大数 p 确实是素数，从而保证密钥交换的安全性和可靠性。通过这种方式，通信双方可以在不直接传输密钥的情况下安全地共享密钥。

2.3 数字签名

数字签名用于验证数据的完整性和真实性，广泛应用于电子商务、在线通信等领域。许多数字签名算法（如 DSA）依赖于大素数来生成签名密钥。通过素性检测，可以确保选取的大数为素数，从而生成安全的签名密钥。这些密钥用于生成和验证数字签名，防止数据被篡改或伪造，保障通信的安全性和可靠性。

2.4 加密安全的随机数生成器

在加密领域，高质量的随机数是确保算法安全性的关键。加密安全的随机数生成器通过生成大素数来提升随机数的质量和安全性。素性检测用于验证生成的大数是否为素数，确保随机数的质量。高质量的随机数用于生成加密密钥、盐值等，在加密算法中发挥重要作用，提升系统的安全性和抗攻击能力。

2.5 公钥基础设施（PKI）

公钥基础设施（PKI）系统用于管理和分发公钥和私钥，确保数据传输的安全性。在 PKI 系统中，生成的密钥需要通过素性检测算法验证，以确保其安全性和有效性。素性检测确保所选的大数为素数，从而生成安全的密钥对，用于加密、解密和数字签名。通过素性检测，PKI 系统能够有效防止因选取不合适的密钥而导致的安全漏洞，保障数据的安全传输。

3 常用算法

接下来将介绍常用的素性检测算法，包括试除法、费马素性测试、Miller-Rabin 测试、AKS 素性测试和 Solovay-Strassen 测试。

3.1 试除法

试除法是最基本的素性检测方法。其基本思路是依次除以小于其平方根的所有整数来判断一个数是否为素数。如果一个数 n 不能被 2 到 $\sqrt{n}$ 之间的任何整数整除，则 n 为素数。该方法的时间复杂度为 $O(\sqrt{n})$，不适用于大数的素性检测。

C++ 实现

```cpp
#include <iostream>
#include <cmath>

bool isPrime(int n) {
    if (n <= 1) return false;
    if (n <= 3) return true;
    if (n % 2 == 0 || n % 3 == 0) return false;
    for (int i = 5; i * i <= n; i += 6) {
        if (n % i == 0 || n % (i + 2) == 0) return false;
    }
    return true;
}

int main() {
    int n = 29; // 待检测的数
    if (isPrime(n)) {
        std::cout << n << " is prime." << std::endl;
    } else {
        std::cout << n << " is not prime." << std::endl;
    }
    return 0;
}
```

3.2 费马素性测试

费马素性测试基于费马小定理。费马小定理指出，若 p 是素数，则对于任意整数 a $(1 < a < p)$，有 $a^{(p-1)} \equiv 1 \pmod{p}$。费马测试通过选择多个随机数 a，验证上述等式是否成立。若等式不成立，则 n 为合数；若等式成立，则 n 可能为素数。该方法简单高效，但存在费马伪素数。

C++ 实现

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

```

```cpp
using namespace std;

// 快速幂模运算
long long power(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;
        }
        exp = exp >> 1;
        base = (base * base) % mod;
    }
    return result;
}

// 费马素性测试
bool fermatTest(int n, int k) {
    if (n <= 1) return false;
    if (n <= 3) return true;

    srand(time(0));
    for (int i = 0; i < k; i++) {
        int a = 2 + rand() % (n - 3);
        if (power(a, n - 1, n) != 1) {
            return false;
        }
    }
    return true;
}

int main() {
    int n = 29; // 待检测的数
    int k = 5; // 测试次数

    if (fermatTest(n, k)) {
        cout << n << " is probably prime." << endl;
    } else {
        cout << n << " is composite." << endl;
    }
    return 0;
}
```

3.3 Miller-Rabin 测试

Miller-Rabin 测试是一种基于费马小定理的改进随机化算法，通过多次随机选择测试值来提高准确性。

3.3.1 算法原理

Miller-Rabin 测试基于以下数学理论：

1. 将 $n-1$ 表示为 $2^r \times d$，其中 d 是奇数。

2. 对于随机选择的整数 a（$2 \leq a \leq n-2$），计算 $a^d \pmod{n}$。

3. 如果 $a^d \equiv 1 \pmod{n}$ 或者 $a^{(2^j \cdot d)} \equiv -1 \pmod{n}$（其中 $0 \leq j \leq r-1$），则 n 可能为素数，否则 n 为合数。

3.3.2 算法步骤

1. 选择待检测的奇数 n。

2. 将 $n-1$ 分解为 $2^r \times d$，其中 d 为奇数。

3. 选择多个随机数 a，每个 a 满足 $2 \leq a \leq n-2$。

4. 对每个 a：

   (a) 计算 $x = a^d \pmod{n}$。

   (b) 如果 $x = 1$ 或 $x = n-1$，则继续测试下一个 a。

   (c) 否则，重复以下步骤 $r-1$ 次：

      i. 计算 $x = x^2 \pmod{n}$。

      ii. 如果 $x = n-1$，则继续测试下一个 a。

      iii. 如果 $x = 1$，则 n 为合数。

   (d) 如果没有发现 $x = n-1$，则 n 为合数。

5. 如果所有 a 都通过测试，则 n 可能为素数。

3.3.3 C++ 实现

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
```

```cpp

using namespace std;

// 快速幂模运算
long long power(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;
        }
        exp = exp >> 1;
        base = (base * base) % mod;
    }
    return result;
}

// Miller-Rabin测试
bool millerRabinTest(long long d, long long n) {
    long long a = 2 + rand() % (n - 4);
    long long x = power(a, d, n);

    if (x == 1 || x == n - 1) {
        return true;
    }

    while (d != n - 1) {
        x = (x * x) % n;
        d *= 2;

        if (x == 1) return false;
        if (x == n - 1) return true;
    }

    return false;
}

// 检测一个数是否为素数
bool isPrime(long long n, int k) {
    if (n <= 1 || n == 4) return false;
    if (n <= 3) return true;

    long long d = n - 1;
    while (d % 2 == 0) {
        d /= 2;
    }
```

```
50
51      for (int i = 0; i < k; i++) {
52          if (!millerRabinTest(d, n)) {
53              return false;
54          }
55      }
56
57      return true;
58  }
59
60  int main() {
61      srand(time(0));
62      long long n = 31; // 待检测的数
63      int k = 4; // 测试次数
64
65      if (isPrime(n, k)) {
66          cout << n << " is prime." << endl;
67      } else {
68          cout << n << " is not prime." << endl;
69      }
70
71      return 0;
72  }
```

3.4 AKS 素性测试

AKS 素性测试是一种确定性算法，可以在多项式时间内判断一个数是否为素数。该算法基于多项式同余理论。

3.4.1 算法原理

AKS 算法基于以下数学定理：

整数 $n(\geq 2)$ 是素数，当且仅当

$$(x+a)^n \equiv (x^n + a) \pmod{n} \tag{1}$$

这个同余多项式对所有与 n 互素的整数 a 均成立。这个定理是费马小定理的一般化，并且可以通过这个特征：

$$\binom{n}{k} \equiv 0 \pmod{n}, \quad \text{对任何} 0 < k < n \text{ 当且仅当} n \text{ 是素数}$$

来证明此定理。

虽然说关系式 (1) 基本上构成了整个素数测试，但是验证花费的时间却是指数时间。因此，为了减少计算复杂度，AKS 改为使用以下的同余多项式：

$$(x+a)^n \equiv (x^n+a) \pmod{x^r-1, n} \tag{2}$$

这个多项式与存在多项式 g, f 令：

$$(x+a)^n - (x^n+a) = (x^r-1)g + nf \tag{3}$$

意义是等同的。

这个同余式可以在多项式时间之内检查完毕。这里我们要注意所有的素数必定满足此条件 (令 $g=0$ 则 (3) 等于 (1)，因此符合 n 必定是素数)。然而，有一些合数也会满足这个条件式。有关系 AKS 正确性的证明包括了推导出存在一个足够小的 r 以及一个足够小的整数集合 A，令如果此同余式对所有 A 里面的整数都满足，则 n 必定为素数。

具体步骤如下：

1. 检查 n 是否是一个完全幂。
2. 找到最小的 r，使得 $o_r(n) > (\log_2 n)^2$。
3. 检查 $1 < \gcd(a, n) < n$ 的 a。
4. 对于 $a = 1$ 到 $\lfloor \sqrt{\phi(r)} \log_2 n \rfloor$，验证多项式同余关系。
5. 如果 n 通过上述所有测试，则为素数，否则为合数。

3.4.2 C++ 实现

由于 AKS 算法比较复杂，此处为简化后的：

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>

using namespace std;

// 检查n是否是一个完全幂
bool isPerfectPower(int n) {
    for (int b = 2; b <= log2(n); ++b) {
        int a = pow(n, 1.0 / b);
        if (pow(a, b) == n || pow(a + 1, b) == n) {
```

```
            return true;
        }
    }
    return false;
}

// 快速幂模运算
long long power(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;
        }
        exp = exp >> 1;
        base = (base * base) % mod;
    }
    return result;
}

// 计算最大公约数
int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}

// AKS素性测试
bool aksTest(int n) {
    if (n <= 1) return false;
    if (isPerfectPower(n)) return false;

    // Step 2: 找到最小的r, 使得 o_r(n) > (log2(n))^2
    int r = 1;
    int maxK = log2(n) * log2(n);
    while (true) {
        ++r;
        bool nextR = false;
        for (int k = 1; k <= maxK; ++k) {
            if (power(n, k, r) == 0 || power(n, k, r) == 1) {
                nextR = true;
                break;
            }
        }
        if (!nextR) break;
    }

```

```
    // Step 3: 检查1 < gcd(a, n) < n的a
    for (int a = 2; a < min(r, n); ++a) {
        if (gcd(a, n) > 1 && gcd(a, n) < n) return false;
    }

    // Step 4: 验证多项式同余关系
    for (int a = 1; a <= sqrt(phi(r)) * log2(n); ++a) {
        if (power(a, n, n) != power(a, 1, n)) return false;
    }

    return true;
}

int main() {
    int n = 29; // 待检测的数

    if (aksTest(n)) {
        cout << n << " is prime." << endl;
    } else {
        cout << n << " is composite." << endl;
    }
    return 0;
}
```

3.5 Solovay-Strassen 测试

Solovay-Strassen 测试是一种基于雅可比符号的随机化算法。

3.5.1 算法原理

Solovay-Strassen 测试基于以下数学定理：如果 p 是素数，对于任何整数 a，满足 $a^{(p-1)/2} \equiv (\frac{a}{p}) \pmod{p}$。该算法通过验证多个随机数 a 的雅可比符号和指数幂之间的关系来判断素性。

先输入奇整数 $n \geq 3$ 和安全参数 t （计算次数）

1. 随机选取整数 b，b 的范围是 $[2, n-2]$。
2. 计算 $r = b^{(n-1)/2} \pmod{n}$。
3. 如果 $r \neq 1$ 且 $r \neq n-1$，则 n 是合数。
4. 计算 Jacobi 符号 $s = \left(\frac{a}{n}\right)$。

5. 如果 $r \neq s$，则 n 是合数。

6. 上述过程重复 t 次。

3.5.2 C++ 实现

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

// 计算雅可比符号
int jacobi(int a, int n) {
    if (a == 0) return 0;
    if (a == 1) return 1;
    int ans;
    if (a % 2 == 0) {
        ans = jacobi(a / 2, n);
        if (n % 8 == 3 || n % 8 == 5) ans = -ans;
    } else {
        ans = jacobi(n % a, a);
        if (a % 4 == 3 && n % 4 == 3) ans = -ans;
    }
    return ans;
}

// 快速幂模运算
long long power(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;
        }
        exp = exp >> 1;
        base = (base * base) % mod;
    }
    return result;
}

// Solovay-Strassen测试
bool solovayStrassenTest(int n, int k) {
    if (n <= 1) return false;
    if (n % 2 == 0) return false;

```

```
    srand(time(0));
    for (int i = 0; i < k; i++) {
        int a = 2 + rand() % (n - 2);
        int jacobian = (n + jacobi(a, n)) % n;
        int mod = power(a, (n - 1) / 2, n);
        if (jacobian == 0 || mod != jacobian) return false;
    }
    return true;
}

int main() {
    int n = 29; // 待检测的数
    int k = 5; // 测试次数

    if (solovayStrassenTest(n, k)) {
        cout << n << " is probably prime." << endl;
    } else {
        cout << n << " is composite." << endl;
    }
    return 0;
}
```

4 总结

素性检测是数论和密码学中的一个重要问题。有效的素性检测算法能够快速判定一个大数是否为素数，这在加密算法如 RSA 和 Diffie-Hellman 中尤为关键。常用的素性检测算法包括试除法、费马素性测试、Miller-Rabin 测试、AKS 素性测试和 Solovay-Strassen 测试。素性检测在密码学中的应用尤为重要，确保了加密密钥的安全性和数据传输的保密性。未来的研究可以继续优化现有算法，提高大数素性检测的效率，开发新的算法以应对更大规模数的素性检测需求。

其二：二次剩余探究报告

2212000 宋奕纬

June 2024

1 引言

二次剩余（Quadratic Residue）是数论中的一个重要概念，对许多数学问题和密码学应用都有重要影响。本文将详细介绍二次剩余的定义、性质、Blum 整数，以及二次剩余在密码学中的应用。

2 相关定义与性质

2.1 二次剩余定义

在模 n 的算术中，整数 a 称为模 n 的二次剩余，如果存在整数 x 使得 $x^2 \equiv a \pmod{n}$。换句话说，a 是某个整数平方的同余类。

- **二次剩余**：存在整数 x 使得 $x^2 \equiv a \pmod{n}$，则称 a 为模 n 的二次剩余。
- **非二次剩余**：如果不存在这样的整数 x，则称 a 为模 n 的非二次剩余。

2.2 勒让德符号

勒让德符号 $\left(\frac{a}{p}\right)$ 用于判断一个数是否为模素数 p 的二次剩余，定义如下：

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{如果 } a \equiv 0 \pmod{p} \\ 1 & \text{如果 } a \text{ 是模 } p \text{ 的二次剩余} \\ -1 & \text{如果 } a \text{ 是模 } p \text{ 的非二次剩余} \end{cases}$$

勒让德符号的性质：

- 乘法性质：$\left(\frac{a}{p}\right)\left(\frac{b}{p}\right)=\left(\frac{ab}{p}\right)$。
- 欧拉准则：$\left(\frac{a}{p}\right)\equiv a^{(p-1)/2}\pmod{p}$。
- 互反律：$\left(\frac{a}{p}\right)\left(\frac{p}{a}\right)=(-1)^{\frac{(p-1)(a-1)}{4}}$。

2.3 雅可比符号

雅可比符号 $(\frac{a}{n})$ 是勒让德符号的推广，用于模合数 n 的情形，定义如下：

$$\left(\frac{a}{n}\right)=\left(\frac{a}{p_1}\right)^{e_1}\left(\frac{a}{p_2}\right)^{e_2}\cdots\left(\frac{a}{p_k}\right)^{e_k}$$

其中 $n=p_1^{e_1}p_2^{e_2}\cdots p_k^{e_k}$ 是 n 的素因子分解。

雅可比符号的性质：

- 若 n 为奇数，$(\frac{a}{n})\equiv a^{(n-1)/2}\pmod{n}$。
- 乘法性质：$\left(\frac{a}{n}\right)\left(\frac{b}{n}\right)=\left(\frac{ab}{n}\right)$。
- 如果 $a\equiv b\pmod{n}$，则 $\left(\frac{a}{n}\right)=\left(\frac{b}{n}\right)$。

2.4 Blum 整数

Blum 整数是指两个形如 $p\equiv 3\pmod{4}$ 和 $q\equiv 3\pmod{4}$ 的素数的乘积，即 $n=p\cdot q$。Blum 整数在密码学中有重要应用。

Blum 整数的性质

设 $n=p\cdot q$ 为一个 Blum 整数，则 $x^2\equiv a\pmod{n}$ 的解可以由模 p 和模 q 的解通过中国剩余定理得到。具体地，若 x_0 是 $x^2\equiv a\pmod{p}$ 的解，y_0 是 $x^2\equiv a\pmod{q}$ 的解，则 $x^2\equiv a\pmod{n}$ 有以下四个解：

$$x\equiv\pm x_0\pmod{p}$$

$$x\equiv\pm y_0\pmod{q}$$

这些解的具体形式为：

$$x \equiv \pm x_0 \pmod{n}$$

$$x \equiv \pm y_0 \pmod{n}$$

3 二次剩余的判断

3.1 模素数的二次剩余

利用勒让德符号可以判断一个数是否为模素数 p 的二次剩余。若 $\left(\frac{a}{p}\right) = 1$，则 a 是模 p 的二次剩余；若 $\left(\frac{a}{p}\right) = -1$，则 a 是模 p 的非二次剩余。

```cpp
#include <iostream>
#include <cmath>

using namespace std;

// 计算勒让德符号
int legendre(int a, int p) {
    if (a % p == 0) return 0;
    int result = 1;
    if (a < 0) {
        a = -a;
        if (p % 4 == 3) result = -result;
    }
    while (a != 0) {
        while (a % 2 == 0) {
            a /= 2;
            if (p % 8 == 3 || p % 8 == 5) result = -result;
        }
        swap(a, p);
        if (a % 4 == 3 && p % 4 == 3) result = -result;
        a %= p;
    }
    return (p == 1) ? result : 0;
}

int main() {
    int a = 10;
    int p = 13;
    int result = legendre(a, p);
    if (result == 1) {
        cout << a << " is a quadratic residue modulo " << p << endl;
    } else if (result == -1) {
```

```
        cout << a << " is not a quadratic residue modulo " << p << endl;
    } else {
        cout << a << " is divisible by " << p << endl;
    }
    return 0;
}
```

3.2 模合数的二次剩余

对于合数，可以利用雅可比符号进行判断。如果 $\left(\frac{a}{n}\right) \neq a^{(n-1)/2} \pmod{n}$，则 a 不是模 n 的二次剩余。

```
#include <iostream>
#include <cmath>

using namespace std;

// 计算雅可比符号
int jacobi(int a, int n) {
    if (a == 0) return 0;
    if (a == 1) return 1;
    int ans;
    if (a % 2 == 0) {
        ans = jacobi(a / 2, n);
        if (n % 8 == 3 || n % 8 == 5) ans = -ans;
    } else {
        ans = jacobi(n % a, a);
        if (a % 4 == 3 && n % 4 == 3) ans = -ans;
    }
    return ans;
}

int main() {
    int a = 10;
    int n = 21;
    int result = jacobi(a, n);
    if (result == 1) {
        cout << a << " is a quadratic residue modulo " << n << endl;
    } else if (result == -1) {
        cout << a << " is not a quadratic residue modulo " << n << endl;
    } else {
        cout << a << " is divisible by " << n << endl;
    }
    return 0;
}
```

3.3 中国剩余定理

对于合数 $n = p \cdot q$，可以通过分别判断 a 是否是模 p 和模 q 的二次剩余，然后利用中国剩余定理来综合判断。

```cpp
#include <iostream>
#include <cmath>

using namespace std;

// 计算勒让德符号
int legendre(int a, int p) {
    if (a % p == 0) return 0;
    int result = 1;
    if (a < 0) {
        a = -a;
        if (p % 4 == 3) result = -result;
    }
    while (a != 0) {
        while (a % 2 == 0) {
            a /= 2;
            if (p % 8 == 3 || p % 8 == 5) result = -result;
        }
        swap(a, p);
        if (a % 4 == 3 && p % 4 == 3) result = -result;
        a %= p;
    }
    return (p == 1) ? result : 0;
}

// 判断a是否为模p和模q的二次剩余
bool isQuadraticResidue(int a, int p, int q) {
    return legendre(a, p) == 1 && legendre(a, q) == 1;
}

int main() {
    int a = 10;
    int p = 7;
    int q = 11;
    if (isQuadraticResidue(a, p, q)) {
        cout << a << " is a quadratic residue modulo " << p << " and " << q
    << endl;
    } else {
        cout << a << " is not a quadratic residue modulo " << p << " and "
    << q << endl;
    }
    return 0;
```

```
}
```

4 二次剩余的应用

4.1 密码学

二次剩余在密码学中有着广泛的应用，特别是在构造公钥加密算法和零知识证明时。例如，Rabin 加密算法和 Goldwasser-Micali 加密算法都利用了二次剩余的性质进行消息加密和解密。

4.2 Rabin 加密算法

Rabin 加密算法基于 Blum 整数。其安全性依赖于整数分解的难题。加密过程如下：

- 选择两个大素数 p 和 q，满足 $p \equiv 3 \pmod 4$ 和 $q \equiv 3 \pmod 4$。
- 计算 $n = p \cdot q$。
- 公钥为 n，私钥为 (p, q)。
- 对于明文 m，加密为 $c \equiv m^2 \pmod n$。

解密时，利用中国剩余定理，从 c 的四个平方根中选出正确的明文 m。以下是 Rabin 加密算法的 C++ 实现：

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <vector>

using namespace std;

// 计算最大公约数
int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}

// 快速幂模运算
long long power(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;
        }
        exp = exp >> 1;
        base = (base * base) % mod;
    }
    return result;
}

// Rabin加密算法的加密函数
long long rabinEncrypt(long long m, long long n) {
    return (m * m) % n;
}

// Rabin加密算法的解密函数
vector<long long> rabinDecrypt(long long c, long long p, long long q) {
    long long n = p * q;
    long long mp = power(c, (p + 1) / 4, p);
    long long mq = power(c, (q + 1) / 4, q);

    long long y1 = (mp * q * power(q, p - 2, p) + mq * p * power(p, q - 2, q
    )) % n;
    long long y2 = (mp * q * power(q, p - 2, p) - mq * p * power(p, q - 2, q
    )) % n;
    long long y3 = n - y1;
    long long y4 = n - y2;

    vector<long long> roots = {y1, y2, y3, y4};
    return roots;
}

int main() {
    long long p = 7;
    long long q = 11;
    long long n = p * q;
    long long m = 20;

    long long c = rabinEncrypt(m, n);
    cout << "Encrypted message: " << c << endl;

    vector<long long> decrypted = rabinDecrypt(c, p, q);
    cout << "Decrypted messages: ";
    for (long long root : decrypted) {
```

```
        cout << root << " ";
    }
    cout << endl;

    return 0;
}
```

4.3 Goldwasser-Micali 加密算法

Goldwasser-Micali 加密算法利用二次剩余和非二次剩余的性质实现加密。其安全性依赖于二次剩余判定的难题。具体过程如下：

- 选择一个 Blum 整数 $n = p \cdot q$。
- 选择一个非二次剩余 y。
- 公钥为 (n, y)，私钥为 (p, q)。
- 对于每个比特 b，选择一个随机数 r，加密为 $c \equiv r^2 \cdot y^b \pmod{n}$。

解密时，利用私钥 (p, q) 判断 c 是否为二次剩余即可恢复比特 b。以下是 Goldwasser-Micali 加密算法的 C++ 实现：

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <cmath>

using namespace std;

// 计算勒让德符号
int legendre(int a, int p) {
    if (a % p == 0) return 0;
    int result = 1;
    if (a < 0) {
        a = -a;
        if (p % 4 == 3) result = -result;
    }
    while (a != 0) {
        while (a % 2 == 0) {
            a /= 2;
            if (p % 8 == 3 || p % 8 == 5) result = -result;
        }
        swap(a, p);
```

```
        if (a % 4 == 3 && p % 4 == 3) result = -result;
        a %= p;
    }
    return (p == 1) ? result : 0;
}

// 判断是否为二次剩余
bool isQuadraticResidue(int a, int p, int q) {
    return legendre(a, p) == 1 && legendre(a, q) == 1;
}

// Goldwasser-Micali加密
int gmEncrypt(int b, int n, int y) {
    srand(time(0));
    int r = rand() % n;
    int c = (r * r) % n;
    if (b == 1) {
        c = (c * y) % n;
    }
    return c;
}

// Goldwasser-Micali解密
int gmDecrypt(int c, int p, int q) {
    if (isQuadraticResidue(c, p, q)) {
        return 0;
    } else {
        return 1;
    }
}

int main() {
    int p = 7;
    int q = 11;
    int n = p * q;
    int y = 5; // 一个非二次剩余

    int b = 1; // 待加密的比特

    int c = gmEncrypt(b, n, y);
    cout << "Encrypted bit: " << c << endl;

    int decrypted = gmDecrypt(c, p, q);
    cout << "Decrypted bit: " << decrypted << endl;

    return 0;
```

```
}
```

5 总结

二次剩余是数论中的重要概念，在密码学中具有广泛的应用。通过利用二次剩余和 Blum 整数的性质，可以构造高效且安全的加密算法，如 Rabin 加密和 Goldwasser-Micali 加密。理解和研究二次剩余的性质，将有助于在数论和密码学领域取得更大的进展。

其三：基于属性的加密（ABE）探究报告

2212000 宋奕纬

May 2024

1 初探 ABE

1.1 概述

ABE（Attribute-Based Encryption）访问控制机制是一种基于属性的加密技术，它允许数据所有者定义访问策略，将数据加密并授权给用户。用户可以通过满足特定属性要求来解密和访问数据。

传统的加密技术在加密消息后，通常都会有一个或多个明确的接收解密者。但在实际中往往存在以下情况：人们并不关心具体的接收者有哪些，而是希望某些满足条件或者符合策略的接收者能够解密加密后的消息。这里包含了两个关键的问题——一对多通信和灵活的访问控制，传统的加密体制不能很好地同时解决这两个问题：比如公钥基础设施技术既无法实现有效的一对多通信，也不能灵活地进行访问控制；广播加密技术能够实现有效的一对多通信，但是访问控制缺乏灵活性，如果接收者集合过于庞大，会产生不菲的开销，同时身份信息也可能存在被泄露的风险。而基于属性的加密能够很好解决上述两个问题。

ABE 可以看作是基于身份的加密 (Identity-Based Encryption,IBE) 的扩展与延伸。IBE 用一个唯一描述性字符串表示用户的身份，比如可以使用字符串 2212000@nankai@cyberscience 来表示一个南开大学网安学院的学生，而 ABE 则是用一组描述性的属性字符串表示用户的身份，我们可以使用 {南开大学, 网络空间安全学院, 本科学生} 来进行身份的描述。

ABE 是一种一对多的加密模式，能够对加密后的信息进行细粒度的访问控制。它用一个属性集合或访问结构对信息进行加密，当且仅当属性集合满足访问结构时可以被多个访问结构或属性集合解密。例如，用户的身份用一组属性集合 {南开大学, 网络空间安全学院, 本科学生} 描述，信息用访问

结构 {老师 OR(硕士 AND 博士)} 进行加密，当且仅当属性集合满足访问结构，能够正确解密，而不需要关心具体有哪些老师或博士。

1.2 发展历史

1. Sahai 和 Waters 在 2005 年提出了 ABE 的概念并构造了具体方案。方案采用门限访问结构，接收者能够解密发布者加密的消息，当且仅当两者相同的属性个数大于或等于系统设置的门限值。

2. Goyal 等在 2006 年提出了密钥策略基于属性的加密 (KP-ABE) 并构造了具体方案。方案采用访问树表示访问结构，能够表示属性之间的“与”、“或” 和 “门限” 三种关系。密钥策略是指私钥与访问结构关联，密文与属性集合关联，即用属性集合加密，用访问结构解密。KP-ABE 可以应用于审计日志，付费电视等。

3. Bethencourt 等在 2007 年提出了密文策略基于属性的加密 (CP- ABE) 并构造了具体方案。密文策略是指私钥与属性集合关联，密文与访问结构关联，即用访问结构加密，用属性集合解密。CP-ABE 可以应用于云共享，安全邮件列表等。

4. Ostrovsky 等在 2007 年提出了非单调访问结构 ABE 方案。除了能够表示属性之间的 “与”、“或” 和 “门限” 三种关系外，还可以表示 “非” 关系。

5. Chase 在 2007 年提出了多权威中心 ABE 方案。系统中存在一个中心权威和多个相互独立的属性权威，中心权威为属性权威颁发密钥，属性权威为各自属性域内的属性颁发密钥。多个属性权威可以视作多个独立的 ABE 方案。

6. Attrapadung 等在 2009 年提出了双策略 ABE 方案。双策略可以看作是密钥策略和密文策略的结合，用户私钥与一个访问结构和一个属性集合关联，密文与一个访问结构和一个属性集合关联。当且仅当私钥的属性集合满足密文的访问结构，密文的属性集合满足私钥的访问结构时，用户能够解密。

7. 2010 年, Lewko 等提出将对偶系统加密技术应用到 CP-ABE, 实现了标准模型下的适应性安全. 不过局限在于方案基于合数阶双线性群, 依赖大数分解困难问题作为安全依据, 所以效率低下。

8. 之后, Okamoto 和 Takashima 引入对偶配对向量空间 (Dual Pairing Vector Space, DPVS) 技术, 实现基于素数阶双线性群的 CP-ABE 方案. 首次做到了效率, 安全兼得的效果。

9. Green 等在 2011 年提出了外包密文解密 ABE 的概念并构造了具体方案。外包密文解密可以看作是在原有 ABE 方案的基础上添加外包解密功能。

10. 2012 年, Lewko 和 Waters 提出了将标准模型下选择安全的方案转换为适应性安全的通用方法。

1.3 分类

ABE 根据访问策略嵌入实体的不同可以分为两种:

1. CP-ABE: 将访问策略嵌入密文

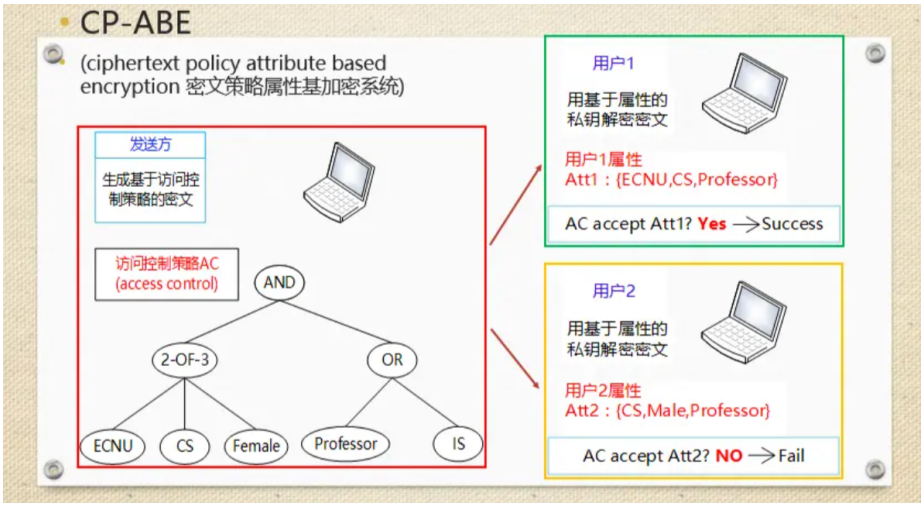

图 1: CP-ABE

所谓密文策略加密系统是指，密文对应于一个访问结构而密钥对应于属性集合，解密当且仅当属性集合中的属性能够满足此访问结构。这种设计比较接近于现实中的应用场景，可以假象每个用户根据自身条件或者属性从属性机构得到密钥，然后加密者来制定对消息的访问控制。

2. KP-ABE：将访问策略嵌入密钥

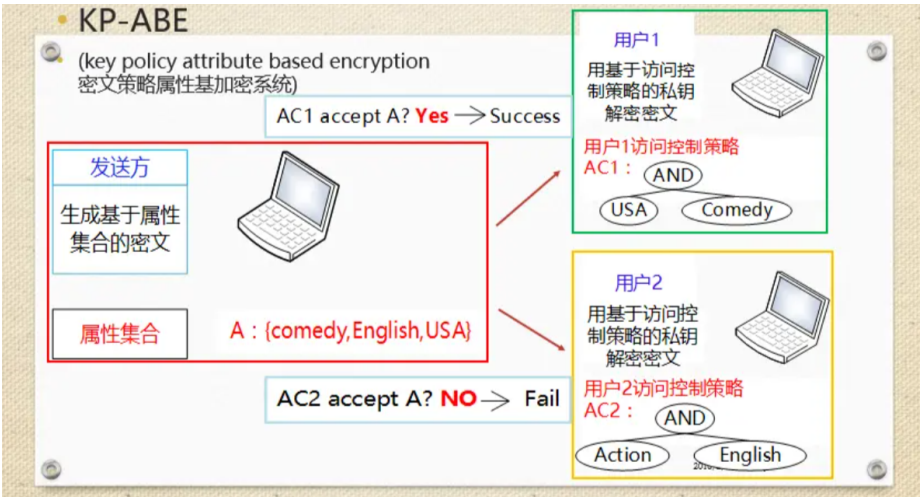

图 2: KP-ABE

所谓密钥策略加密系统是指，密钥对应于一个访问控制而密文对应于一个属性集合，解密当且仅当属性集合中的属性能够满足此访问结构。这种设计比较接近静态场景，此时密文用与其相关的属性加密存放在服务器上，当允许用户得到某些消息时，就分配一个特定的访问结构给用户。

KP-ABE 机制更适用于用户数量较少、但需要高度灵活性的场景；而 CP-ABE 机制则更适用于用户数量较多、但需要高度可扩展性的场景。

更细化的分类如下：

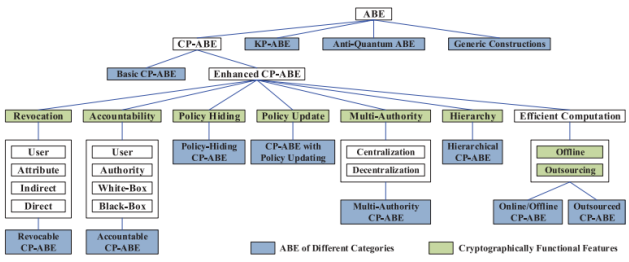

图 3: 分类

2 算法详解

2.1 数学基础

2.1.1 双线性映射

设 $\mathbb{G}_1$ 和 $\mathbb{G}_2$ 是两个乘法循环群，它们的阶均为素数 p。设 g 为 $\mathbb{G}_1$ 的生成元，

$$e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$$

表示双线性映射，如果 e 满足下面三个条件：

- **双线性性：** 任取 $u, v \in \mathbb{G}_1$ 和 $a, b \in \mathbb{Z}_p$，等式 $e(u^a, v^b) = e(u, v)^{ab}$ 成立。
- **非退化性：** $e(g, g) \neq 1$。
- **可计算性：** 任取 $u, v \in \mathbb{G}_1$，双线性映射 $e(u, v)$ 能够进行有效计算。

则称 $\mathbb{G}_1$ 是一个双线性群，又因为

$$e(g^a, g^b) = e(g, g)^{ab} = e(g^b, g^a)$$

所以双线性映射 e 是对称的。

1. 判定 BDH 性假设

   设 $\mathbb{G}_1$ 是一个阶为素数 p 的双线性群，g 是 $\mathbb{G}_1$ 生成元，$e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ 表示双线性映射。随机选取 $a, b, c, z \in \mathbb{Z}_p$，判定性 BDH 假设为：不存在概率多项式时间算法 $\mathcal{B}$，以不可忽略的优势区分 $(A = g^a, B = g^b, C = g^c, e(g, g)^{abc})$ 和 $(A = g^a, B = g^b, C = g^c, e(g, g)^z)$。算法 $\mathcal{B}$ 的优势定义为：

   $$\left|\Pr[\mathcal{B}(A, B, C, e(g, g)^{abc}) = 0] - \Pr[\mathcal{B}(A, B, C, e(g, g)^z) = 0]\right|$$

2. 判定性 q-BiblE 假设

   设 $\mathbb{G}_1$ 是一个阶为素数 p 的双线性群，g 是 $\mathbb{G}_1$ 生成元，$e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ 表示双线性映射。随机选取 $a, s, b_1, \ldots, b_q \in \mathbb{Z}_p$，判定性 q-BDHE 假设

(Decisional q-parallel Bilinear Diffie-Hellman Exponent Assumption) 是指给定敌手

$$y = \begin{cases} g, g^s, g^a, \ldots, g^{a^q}, g^{a^{q+2}}, \ldots, g^{a^{2q}} \\ g^{sb_j}, g^{a/b_j}, \ldots, g^{(a^k/b_j)}, g^{(a^{q+z}/b_j)}, \ldots, g^{(a^{2q}/b_j)} \\ \forall 1 \le j \le q \\ g^{a^{sb_j/b_k}}, g^{a^{tb_j/b_k}}, \ldots, g^{(a^{2q}/b_j)} \end{cases}$$

不存在概率多项式时间算法 $\mathcal{B}$，以不可忽略的优势区分 $e(g,g)^{a^{q+1}s} \in \mathbb{G}_2$ 和随机元素 $R \in \mathbb{G}_2$。算法 $\mathcal{B}$ 的优势定义为：

$$\left|\Pr[\mathcal{B}(y, T = e(g,g)^{a^{q+1}s}) = 0] - \Pr[\mathcal{B}(y, T = R) = 0]\right|$$

2.1.2 访问结构

1. 定义

   设 $P = \{P_1, P_2, \ldots, P_n\}$ 是由 n 个参与者组成的集合，$A \subseteq 2^P$ 是 2^P 的一个非空子集。其中 2^P 表示 P 的所有子集组成的集合，即 A 是由 P 的若干子集组成的非空集合。如果集合 A 是单调的，即对任意的由若干个参与者组成的集合 B 和 C，如果 $B \in A, B \subseteq C$ 则有 $C \in A$，那么我们称 A 是参与者集合 P 上的一个访问结构。设 A 是参与者集合 P 上的一个访问结构，如果 $D \in A$，则称 D 是一个授权集合，否则 D 是一个非授权集合。

   举例：

   比如有 3 个参与者 P_1, P_2, P_3，

   则 2^P 可以表示为 $\{\{\}, \{P_1\}, \{P_2\}, \{P_3\}, \{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}, \{P_1, P_2, P_3\}\}$

   我们可以取出一个 A:$\{\{P_1, P_2\}, \{P_1, P_2, P_3\}, \{P_1\}\}$

   此时 A 中的元素有 $\{P_1, P_2\}$,$\{P_1, P_2, P_3\}$,$\{P_1\}$

   可以看出此时 A 是满足单调的，A 就是参与者集合 P 上的一个访问结构。

   其中的元素 $\{P_1, P_2\}$,$\{P_1, P_2, P_3\}$,$\{P_1\}$ 具有访问权限，是授权集合。

   而参与者集合的其他子集则不具有访问权限，不是授权集合。

2. 表示方式

（1）访问树

访问树是用来表示访问结构的常用方法之一。设 T 是一棵访问树，T 上的任一非叶节点为一个阈门（Threshold Gate），由该非叶节点的所有子节点和某个阈值（Threshold Value）进行描述。设 α 是 T 上的任一节点，num_α 表示 α 的子节点数量，k_α 表示 α 的门限值，则有 $0 < k_\alpha \leq num_\alpha$。当 $k_\alpha = 1$ 时，α 表示一个“或”门；当 $k_\alpha = num_\alpha$ 时，α 表示一个“与”门。如果 α 为叶子节点，则有 $k_\alpha = 1$，并且 α 由某个属性进行描述。

为了方便描述访问树上的操作，我们定义几个函数。$parent(x)$ 表示访问树中节点 x 的父节点。当且仅当 x 为叶子节点时，$att(x)$ 表示与 x 相关联的属性。访问树对每个节点的所有子节点从 1 开始分别进行顺序编号，$index(x)$ 表示 x 在兄弟节点之间的序号。

设 γ 是一个属性集合，T_α 是以 α 为根节点的访问树，如果属性集合 γ 满足一棵访问树 T_α，则用 $T_\alpha(\gamma) = 1$ 表示。$T_\alpha(\gamma)$ 的值可以通过递归计算获得：

- 如果 α 是非叶节点，对其所有子节点 x' 计算 $T_{x'}(\gamma)$，若有至少 k_α 个子节点的计算结果为 1，则 $T_\alpha(\gamma)$ 返回 1。
- 如果 α 是叶子节点，当且仅当 $att(\alpha) \in \gamma$ 时，$T_\alpha(\gamma)$ 返回 1。

简单来说访问树的节点分为叶子结点与非叶节点，非叶节点有两个值，第一个是门限值，第二个是其子节点的个数；叶子节点主要包含属性值。当非叶节点门限是 m 则其子节点至少有 m 个为 1，该节点才为 1（可解密），否则该节点为 0（无法解密）.

由此可知当非叶子节点为 [1 n] 时，该节点相当于一个或门，为 [n n] 时，该节点相当于一个与门。

举例：

首先，从根节点 R 开始，按照自上而下的方式访问树 T 中的每个 x 选择多项式 q_x。选择方式如下：

1. 指定多项式的阶

对于树中的每个节点 x，将多项式 q_x 的阶 d_x 设为比该节点的阈值 k_x 小 1，即 $d_x = k_x - 1$。

2. 指定常数项

从根节点 R 开始，算法选择一个随机的 $s \in \mathbb{Z}_p$，并设置 $q_R(0) = s$。然后对于任意其他节点 x，设置 $q_x(0) = q_{\text{parent}(x)}(\text{index}(x))$。

3. 定义多项式

对所有节点随机选择其他参数，完整地定义 q_x。

上述过程需要根节点的子节点作为下一步的根节点递归执行。

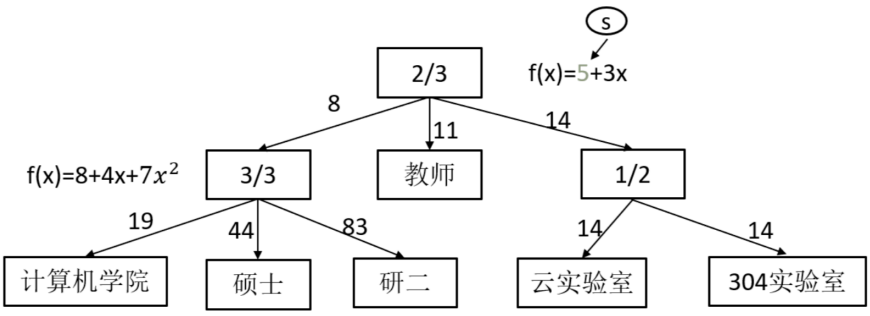

图 4: 示例

如上图所示

从根节点开始，其门限值为 2，子节点有 3 个，随机生成一个多项式，其最高次数为门限值少 1，故根节点的最高次数为 1，然后将常数项设置为秘密数（秘密数为需要秘密保存的数），如此根节点随机的多项式为 f(x)=5+3x，秘密数为 5。此外，将根节点的孩子节点从左至右依次标记为 1,2,3,.....，将节点标记值代入 f(x) 函数中，所得值（即生成新的秘密值）传给该标记的孩子节点秘密保存；故 “3/3” 节点（左边第一个节点）标记为 1，传给 “3/3” 节点的秘密值 f(1)=5+3*1=8，中间 “教师” 节点（中间节点）标记为 2，传给 “教师” 节点的秘密值 f(2)=5+3*2=11，“1/2” 节点（右边节点）标记为 3，传给 “1/2” 节点的秘密值为 f(3)=5+3*3=14。

“3/3” 节点和 “1/2” 节点在接收到父节点传来的值后，按照上述方式生成随机多项式，将常数项设置为父节点传来的值，此外也按照上述方式生成新的秘密值并将它传给子节点，数据如图所示（对于非叶子节点，都按照此方式进行）。对于叶子节点，在接受到父节点的秘密值后，用此叶子节点的属性对秘密值进行加密处理。

这样就构成了访问树，并进行了秘密的分发。

(2) LSSS

LSSS 访问结构也是一种表示访问结构的常用方法，它用分享生成矩阵和映射函数进行描述。分享生成矩阵的构造方法有间接转化和直接构造两种：

一种是将访问树转化为分享生成矩阵。设访问二叉树的叶子节点表示相应的属性，非叶节点表示“与”门或者“或”门，约定 $(1, 0, \ldots, 0)$ 为目标向量。首先，标记根节点为 (1)，设置全局计数器变量 $c = 1$。然后自上而下地标记二叉树中的其它节点：如果节点 α 是一个被向量 $\mathbf{v}$ 标记的“或”门，则将 α 的子节点也标记为 $\mathbf{v}$。如果节点 α 是一个被向量 $\mathbf{v}$ 标记的“与”门，则将 α 的一个子节点标记为 $\mathbf{v}\|1$（“$\|$”为连接符），另外一个子节点标记为 $(0, \ldots, 0)\| - 1$，其中 $(0, \ldots, 0)$ 的长度为 c，计数器变量 c 的值加 1。最后，树中所有节点都被标记，如果标记叶子节点的各向量长度不相等，则用 0 填充至相同长度。标记叶子节点的各向量即为 LSSS 分享生成矩阵的各行向量。

例如，设访问二叉树描述为 (A AND (D OR (B AND C)))，首先标记根节点 A AND 标记为 (1)，根节点的左孩子节点 A 标记为 $(1, 1)$，右孩子节点 OR 及其子节点 D AND 标记为 $(0, 1, -1)$，节点 B 和节点 C 分别标记为 $(0, 1, -1, 1)$ 和 $(0, 0, -1)$，最后，得到分享生成矩阵为：

$$M = \begin{pmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

函数 ρ：$\rho(1) = A, \rho(2) = B, \rho(3) = C, \rho(4) = D$，即各行分别对应属性 $ABCD$。满足访问二叉树的属性集合能线性重构得到目标向量 $(1, 0, \ldots, 0)$，从而还原出秘密。

另一种是直接构造分享生成矩阵。直接构造以门限作基本单元，“与”和“或”分别用 (n, n) 和 $(1, n)$ 门限表示，n 为属性个数。例如，$(A, B, C, 2)$ 表示 $(2, 3)$ 门限，$(A, B, 2$ 表示 A AND B，$(A, B, 1)$ 表示 A OR B。

一个 (t, n) 门限的生成矩阵可以表示为：

$$M_{(t,n)} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 2^2 & \dots & 2^{t-1} \\ 1 & 3 & 3^2 & \dots & 3^{t-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n & n^2 & \dots & n^{t-1} \end{pmatrix}$$

则访问结构 (A AND (D OR (B AND C))) 可以表示如下：

$$(M, \rho) = \left(\begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 0 \\ 1 & 2 & 1 \\ 1 & 2 & 2 \end{pmatrix}, \begin{pmatrix} A \\ D \\ B \\ C \end{pmatrix} \right)$$

2.2 算法介绍

2.2.1 ABE

形式化定义 ABE: 固定消息（明文）集合 M 以及属性集合、谓词 $P : X \times Y \to \{0, 1\}$，一个适用于 P 的 ABE 有 4 个高效随机算法：

- **Setup** 是初始化算法，输出公钥/主密钥对 $(\mathrm{mpk}, \mathrm{msk})$。
- **KeyGen**(msk, y) 是受限密钥生成算法，用来生成和 y 相关联的密钥 sk_y。
- **Enc**(mpk, x, g) 是加密算法，用来产生把消息 g 且和 x 相关联的密文 $\mathrm{ct}_x(g)$。
- **Dec**$(\mathrm{sk}, \mathrm{ct})$ 是解密算法，它输出一条消息（解密结果）或者一个特殊的符号 $\perp$ 表示“不能解密”或者“解密出错”。

正确性要求对任意满足 $P(x, y) = 1$ 的 x, y 以及任意 $g \in M$，都有

$$\Pr \left[\begin{array}{l} (\mathrm{mpk}, \mathrm{msk}) \leftarrow \mathrm{Setup}() \\ \mathrm{sk} \leftarrow \mathrm{KeyGen}(\mathrm{msk}, y) \\ \mathrm{ct} \leftarrow \mathrm{Enc}(\mathrm{mpk}, x, g) \end{array} \middle| \; \mathrm{Dec}(\mathrm{sk}, \mathrm{ct}) = g \right] = 1.$$

2.2.2 KP-ABE

一个 KP-ABE 方案通常由下面四个算法组成：

- **Setup**: 初始化算法。输入安全参数和属性域参数，输出公钥参数 PK 和主密钥 MK。
- **Encryption**: 加密算法。输入明文消息 M、属性集合 γ 和公钥参数 PK，输出密文 E。
- **KeyGen**: 密钥生成算法。输入访问结构 A、主密钥 MK 和公钥参数 PK，输出解密密钥 D。
- **Decryption**: 解密算法。输入由属性集合 γ 加密的密文 E、访问结构 A 对应的解密密钥 D 和公钥参数 PK。如果 $\gamma \in A$，算法完成对密文的解密，并返回明文消息 M。

设 $\mathbb{G}_1$ 是一个阶为素数 p 的双线性群，g 是 $\mathbb{G}_1$ 的生成元，$e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ 表示双线性映射。设 $i \in \mathbb{Z}_p$，S 表示 $\mathbb{Z}_p$ 上的元素集合，拉格朗日系数计算方法如下：

$$\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$$

AP-ABE 算法具体构造（此处以访问树访问结构为例）如下：

Setup: 设系统中共有 n 个属性，属性域记为 $N = \{1, 2, \ldots, n\}$。分别为每个属性 $i \in N$ 随机选取唯一的 $t_i \in \mathbb{Z}_p$，随机选取 $y \in \mathbb{Z}_p$。算法输出公钥参数 $PK : T_1 = g^{t_1}, \ldots, T_n = g^{t_n}, Y = e(g,g)^y$，主密钥 $MK : t_1, \ldots, t_n, y$。

Encryption (M, γ, PK): 随机选取 $s \in \mathbb{Z}_p$，算法输入明文消息 $M \in \mathbb{G}_2$ 和属性集合 γ，输出密文 $E : E = (\gamma, E' = MY^s, \{E_i = T_i^s\}_{i \in \gamma})$。

KeyGen (T, MK): 算法输入访问树 T 和主密钥 MK，输出解密密钥 D。该密钥能够正确解密与属性集合 γ 相关联的密文，当且仅当 $T(\gamma) = 1$。算法操作步骤如下：

首先，从根节点 r 开始，自上而下地访问树 T 中任一节点 x（包括叶子节点）随机选取一个多项式 q_x，其中多项式的度 $d_x = k_x - 1$。对于根节点多项式的选取，设置 $q_r(0) = y$，其他子节点随机选取。对于非根节点多项式的选取，设置 $q_x(0) = q_{\mathrm{parent}(x)}(\mathrm{index}(x))$，其他 d_x 个点随机选取。然后，计算并输出解密密钥：$D = \{D_x = g^{q_x(0)/t_i}\}$，其中 x 为叶子节点，$i = \mathrm{att}(x)$。

Decryption (E, D): 算法输入密文 $E = (\gamma, E' = MY^s, \{E_i = T_i^s\}_{i\in\gamma})$ 和解密密钥 D，若解密成功输出 $\mathbb{G}_2$ 上的一个元素，否则输出 $\perp$。设 x 为访问树中的任一节点，定义一个递归函数 $F_x = \text{DecryptNode}(E, D, x)$。

如果 x 是叶子节点，设 $i = \text{att}(x)$，计算：

$$F_x = \begin{cases} e(D_x, E_i) = e(g^{q_x(0)/t_i}, g^{st_i}) = e(g,g)^{sq_x(0)} & i \in \gamma \\ \perp & \text{otherwise} \end{cases}$$

如果 x 是非叶节点，则对 x 的所有子节点 z 计算 $F_z = \text{DecryptNode}(E, D, z)$，若满足 $F_z \neq \perp$ 的数量不少于 x 的门限值 k_x，则返回 $F_x = \prod_{z\in S'} F_z^{\Delta_{i,S'}(0)}$，其中 S' 为任意 k_x 个满足 $F_z \neq \perp$ 的子节点集合。

当且仅当密文满足解密密钥相关联的访问结构时，解密算法从根节点 r 开始，计算 $\text{DecryptNode}(E, D, r) = e(g,g)^{ys}$。又因为已知 $E' = MY^s = MY^s$，因此算法从密文中还原并返回明文消息 M。

2.2.3 CP-ABE

一个 CP-ABE 方案通常由下面四个算法组成：

- **Setup**: 初始化算法。输入安全参数和属性域参数，输出公钥参数 PK 和主密钥 MK。
- **Encryption**: 加密算法。输入公钥参数 PK、明文消息 M、访问结构 A，输出密文 CT。
- **Key Generation**: 密钥生成算法。输入主密钥 MK 和属性集合 S，输出私钥 SK。
- **Decryption**: 解密算法。输入公钥参数 PK、含有访问结构 A 的密文 CT、私钥 SK。如果属性集合 S 满足访问结构 A，算法完成对密文的解密，并返回明文消息 M。

CP-ABE 算法具体构造（此处以 LSSS 作为访问结构为例）如下：

Setup: 设 $\mathbb{G}_1$ 是一个阶为素数 p 的双线性群，g 是 $\mathbb{G}_1$ 的生成元，双线性映射 $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$，单向函数 $H : \{0,1\}^* \to \mathbb{G}_1$ 是一个预言机，群元素的大小由系统安全参数决定。随机选取 $\alpha, a \in \mathbb{Z}_p$，算法输出公钥参数 $PK = (g, e(g,g)^\alpha, g^a)$ 和主密钥 $MSK = g^\alpha$。

Encrypt $(PK, (M, \rho), m)$: 算法输入公钥参数、LSSS 访问结构 (M, ρ) 以及明文消息 $m \in \mathbb{G}_2$。其中，ρ 是一个单射函数，能够将访问结构中的每个属性与分享生成矩阵 M 的某一行（至少 1 行）关联起来。

设矩阵 M 有 ℓ 行 n 列,算法首先选取一组随机向量 $\vec{v} = (s, y_2, \ldots, y_n) \in \mathbb{Z}_p^n$，随机选取 $r_1, \ldots, r_\ell \in \mathbb{Z}_p$，计算 $\lambda = \vec{v} \cdot M_i$，其中 $i = 1, \ldots, \ell$，M_i 表示矩阵 M 的第 i 行。算法输出密文如下：

$$CT = (C = me(g,g)^{\lambda}, C' = g^s, \{C_i = g^{a^i} H(\rho(i))^{-r_i}, D_i = g^{r_i}\}_{i=1}^{\ell})$$

KeyGen (MSK, S): 算法输入主密钥 MSK 和属性集合 S，随机选取 $t \in \mathbb{Z}_p$，输出私钥如下：

$$SK = (K = g^{\alpha} g^{at}, L = g^t, \{K_x = H(x)^t\}_{x \in S})$$

Decrypt (CT, SK): 算法输入与访问结构 (M, ρ) 相关联的密文 CT，与属性集合 S 相关联的私钥 SK。假设 S 满足 (M, ρ)，定义 $I = \{i : \rho(i) \in S\} \subseteq \{1, \ldots, \ell\}$。根据 LSSS 的线性重构特性，能够找出常数集合 $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$，满足 $\sum_{i \in I} \omega_i \lambda_i = s$，其中 $\{\lambda_i\}$ 是秘密的一组有效分享。算法首先计算：

$$\frac{e(C', K)}{\left(\prod_{i \in I} e(C_i, L) e(D_i, K_{\rho(i)})\right)^{\omega_i}} = \frac{e(g^s, g^{\alpha} g^{at})}{\left(\prod_{i \in I} e(g^{a^i} H(\rho(i))^{-r_i}, g^t) e(g^{r_i}, H(\rho(i))^t)\right)^{\omega_i}} = e(g,g)^{\lambda}$$

然后从 $C = me(g,g)^{\lambda}$ 中还原出明文消息 m。

3 应用场景

ABE 算法的应用场景

1. 云存储安全

场景描述：在云存储中，用户将数据上传至云服务器，云服务提供商负责数据的存储和管理。由于数据在云端，传统的基于用户身份的访问控制模型难以应对多用户共享和复杂的权限管理需求。

ABE 的应用：ABE 通过加密数据并设定访问策略，确保只有符合特定属性的用户才能解密数据。例如，一个公司可以设定策略，只有拥有特定部门和职位属性的员工才能访问特定文件。

具体实现：

- **Setup**：云服务提供商生成公共参数和主密钥。
- **Encryption**：数据拥有者根据访问策略加密数据。
- **KeyGen**：云服务提供商根据用户的属性生成解密密钥。
- **Decryption**：用户使用解密密钥访问数据。

2. 物联网（IoT）

场景描述：物联网设备广泛应用于智能家居、智能城市等领域，这些设备之间需要安全通信和协同工作。

ABE 的应用：ABE 可以在物联网设备间实现安全通信，确保只有具备特定属性的设备或用户才能访问或操作数据。例如，在智能家居中，只有家庭成员才能控制家中的智能锁和摄像头。

具体实现：

- **Setup**：物联网平台生成公共参数和主密钥。
- **Encryption**：设备在传输数据前，根据访问策略加密数据。
- **KeyGen**：平台根据设备属性生成解密密钥。
- **Decryption**：目标设备或用户使用解密密钥访问数据。

3. 电子医疗

场景描述：电子医疗系统中存储和传输大量敏感的患者信息，需要严格保护患者隐私。

ABE 的应用：通过 ABE 技术，只有符合特定条件的医疗专业人员（如特定科室的医生）才能访问患者的敏感信息。

具体实现：

- **Setup**：医疗系统生成公共参数和主密钥。
- **Encryption**：医疗记录加密时附加属性（如科室、医生级别）。
- **KeyGen**：系统根据医生的属性生成解密密钥。
- **Decryption**：医生使用解密密钥访问患者信息。

4. 智能电网

场景描述：智能电网需要实时监控和管理电力供应，涉及大量数据的采集和传输。

ABE 的应用：ABE 可以保护智能电网中传输的数据，确保只有具备特定权限的设备和用户才能访问这些数据。

具体实现：

- **Setup**：电网管理系统生成公共参数和主密钥。
- **Encryption**：传感器在传输数据前，根据访问策略加密数据。
- **KeyGen**：系统根据用户或设备属性生成解密密钥。
- **Decryption**：用户或设备使用解密密钥访问数据。

5. 社交网络

场景描述：社交网络平台上用户分享大量个人信息，需要保护用户隐私。

ABE 的应用：用户可以使用 ABE 加密个人数据，并设定访问策略，确保只有特定的朋友或群组成员才能访问这些数据。

具体实现：

- **Setup**：社交网络平台生成公共参数和主密钥。
- **Encryption**：用户在上传数据时，根据访问策略加密数据。
- **KeyGen**：平台根据朋友或群组成员的属性生成解密密钥。
- **Decryption**：朋友或群组成员使用解密密钥访问数据。

6. 政府和军事

场景描述：政府和军事部门需要处理大量机密信息，需要确保信息只被授权人员访问。

ABE 的应用：ABE 可以用于加密机密信息，确保只有具备特定权限和属性的人员才能访问这些信息，保护国家安全和机密信息。

具体实现：

- **Setup**：政府或军事部门生成公共参数和主密钥。
- **Encryption**：机密信息加密时附加属性（如部门、职位）。
- **KeyGen**：系统根据人员属性生成解密密钥。
- **Decryption**：授权人员使用解密密钥访问机密信息。

总结

ABE 技术通过灵活的属性和策略设置，实现了细粒度的访问控制和高安全性的加密机制，广泛应用于需要严格权限管理和隐私保护的各个领域。这些应用场景展示了 ABE 在现实世界中的巨大潜力和广泛应用前景。