基于Paillier的半同态加密库 副本

一、Paillier算法实现原理

Paillier是一个支持加法同态的公钥密码系统，由Paillier在1999年的欧密会（EUROCRYPT）上首次提出。在众多PHE方案中，Paillier方案由于效率较高、安全性证明完备的特点，在各大顶会和实际应用中被广泛使用，是隐私计算场景中最常用的PHE实例化方案之一。

算法详情

密钥生成

- 随机选择两个大质数 p ， q ；
- 计算 $n = pq$ ，和 $\lambda = lcm(p-1, q-1)$ ；
- 随机选择整数 $g \in Z_{n^2}^*$ ；
- 计算 $\mu = (L(g^\lambda \; mod \; n^2))^{-1} \; mod \; n$ ，其中， L 函数定义为： $L(x) = \frac{x-1}{n}$ ；
- 获得公钥 $pk = (n, g)$ ，私钥 $sk = (\lambda, \mu)$ 。

代码实现如下：

```cpp
void Paillier::KeyGen(unsigned long bitLen)
{
    gmp_randinit_default(gmp_rand);
    mpz_t r;
    mpz_init(r);
    mpz_rrandomb(r, gmp_rand, bitLen); // r <--- rand
    mpz_nextprime(p, r);               // p是大素数
    mpz_set(r, p);
    mpz_nextprime(q, r); // q是大素数

    mpz_mul(n, p, q);       // n = p*q
    mpz_add_ui(g, n, 1);    // g = n+1
    mpz_mul(nsquare, n, n); // nsqaure = n * n;

    mpz_sub_ui(p, p, 1); // p = p-1
    mpz_sub_ui(q, q, 1); // q = q-1
    mpz_lcm(lambda, p, q);         // lambda = lcm(p-1, q-1)
    // mpz_mul(lambda, p, q);
    mpz_invert(lmdInv, lambda, n); // lmdInv = lambda^{-1} mod n

    mpz_clear(r);
}
```

> 关于符号的说明

- $gcm(a,b)$ ：表示两个数的最大公因数。
- $lcm(a,b)$ ：表示两个数的最小公倍数。
- Z_n^* ：表示模 n 意义下， $[0, n-1]$ 中所有与 n 互质的元素的集合。
- $Z_{n^2}^*$ ：表示模 n^2 意义下的可逆元素集合。即，对于正整数 n ， $Z_{n^2}^*$ 包含了模 n^2 意义下与 n^2 互质的所有元素。即，如果有 $x \in Z_{n^2}^*$ ，则 x 满足以下条件：
  - x 和 n^2 互质，即 $gcd(x, n^2) = 1$
  - x 在模 n^2 意义下有逆元，即，存在 y ，使得 $xy \equiv 1 \ (mod\ n^2)$

关于取值和参数优化：

一般取 $g = n + 1$,则通过数学推导，有 $\mu = \lambda^{-1} mod\ n$

> 因此，一般在编程时，只保存私钥为 $sk = \lambda$

下面，对此进行证明：

在不影响算法正确性的前提下，为了简化运算，算法在密钥生成阶段，一般取 $g=n+1$ 。如果 $g=n+1$ ，则有如下推导：

$$\mu=(L(g^{\lambda}\ mod\ n^2))^{-1} \mod n$$

$$=(L((n+1))^{\lambda}\ mod\ n^2))^{-1} \mod n$$

$$=(\frac{(n+1)^{\lambda}\ mod\ n^2-1}{n})^{-1} \mod n$$

根据二项定理，知道：

$$(n+1)^{\lambda}=\sum_{k=0}^{\lambda}\binom{\lambda}{k}n^k$$

根据上述公式，我们知道前面的 $\lambda-1$ 项都是 n^2 的倍数，在模 n^2 的时候都会是0，则有如下简约：

$$\begin{aligned}(n+1)^{\lambda}\ mod\ n^2 &= \sum_{k=0}^{\lambda}\binom{\lambda}{k}n^k\ mod\ n^2\\ &=\lambda n+1\end{aligned}$$

因此：

$$\begin{aligned}\mu &= (\frac{(n+1)^{\lambda}\ mod\ n^2-1}{n})^{-1}\ mod\ n\\ &=(\frac{\lambda n+1-1}{n})^{-1}\ mod\ n\\ &=\lambda^{-1}\ mod\ n\end{aligned}$$

加密过程中的计算加密内容 $g^m r^n\ mod\ n^2$ 的时候，也可简化计算量，如：

> 从上面的公式推导可以知道，如果 $g=n+1$ ，则有：

$$\begin{aligned}g^m &= (n+1)^m\ mod\ n^2\\ &=\binom{m}{0}n^m+\binom{m}{1}n^{m-1}+...+nm+1\ mod\ n^2\\ &=nm+1\ mod\ n^2\end{aligned}$$

这样，就加速了计算过程。

数据加密

- 输入明文消息 m ，满足条件 $0 \leq m < n$
- 选择随机数 r ，满足条件 $0 \leq r < n$ ，且 $r \in Z_{n^2}^*$
- 计算密文 $c = g^m r^n \, mod \, n^2$

代码实现如下：

```C++
void Paillier::Encrypt(mpz_t c, mpz_t m)
{
    if (mpz_cmp(m, n) >= 0)
    {

        throw("m must be less than n");
        return;
    }

    mpz_t r;
    mpz_init(r);
    gmp_randinit_default(gmp_rand);
    mpz_urandomm(r, gmp_rand, n); // r <--- rand

    mpz_powm(c, g, m, nsquare); // c = g^m mod n^2
    mpz_powm(r, r, n, nsquare); // r = r^n mod n^2
    mpz_mul(c, c, r);           // c = c*r
    mpz_mod(c, c, nsquare);     // c = c mod n^2

    mpz_clear(r);
}
```

数据解密

- 输入密文 c
- 计算明文消息 $m = L(c^\lambda \, mod \, n^2) \cdot \mu \, mod \, n$

解密的正确性演算：

首先，选取 $g = n + 1$ ，则根据上面的计算，有 $g^m = nm + 1 \, mod \, n^2$ ，则：

$$
\begin{aligned}
L(c^{\lambda}\ mod\ n^2)\cdot\mu &= \frac{(g^m r^n)^{\lambda}\ mod\ n^2-1}{n}\cdot\mu \\
&= \frac{g^{m\cdot\lambda} r^{n\cdot\lambda}\ mod\ n^2-1}{n}\cdot\mu \\
&= \frac{g^{m\cdot\lambda}\ mod\ n^2-1}{n}\cdot\lambda^{-1} \\
&= \frac{m\cdot\lambda\cdot n-1}{n}\cdot\lambda^{-1} \\
&= m\cdot\lambda\cdot\lambda^{-1}\ mod\ n \\
&= m
\end{aligned}
$$

$r^{n\cdot\lambda}=1\ mod\ n^2$ 的证明

关于 $r^{n\cdot\lambda}=1\ mod\ n^2$ 的正确性的证明，如下所示：

根据 λ 的定义，有： $\lambda=k_1\cdot(p-1)=k_2\cdot(q-1)$ ，根据费马小定理，有：

$$
r^{\lambda}=r^{k_1(p-1)}=(r^{(p-1)})^{k_1}=1\ (mod\ p)
$$

同理，有 $r^{\lambda}=1\ (mod\ q)$ ，即： $r^{\lambda}=1\ (mod\ p\cdot q)=\ (mod\ n)$ ，即：

$$
r^{\lambda}=1+k\cdot n, k\in Z^*
$$

则，根据二项式定理，有：

$$
\begin{aligned}
r^{n\lambda} &= (1+kn)^n\ (mod\ n^2) \\
&= \binom{n}{0}(kn)^n+\binom{n}{1}(kn)^{m-1}+...+(kn)n+1 \\
&= 1\ mod\ n^2
\end{aligned}
$$

代码实现如下：

```C++
void Paillier::Decrypt(mpz_t m, mpz_t c)
{
    if (mpz_cmp(c, nsquare) >= 0)
    {
        throw("ciphertext must be less than n^2");
        return;
    }
    mpz_powm(m, c, lambda, nsquare); // c = c^lambda mod n^2
    // m = (c - 1) / n * lambda^(-1) mod n

    mpz_sub_ui(m, m, 1);    // c=c-1
    mpz_fdiv_q(m, m, n);    // c=(c-1)/n
    mpz_mul(m, m, lmdInv); // c=c*lambda^(-1)
    mpz_mod(m, m, n);       // m=c mod n
}
```

> 注意，本代码并没有对私钥、公钥进行区分！
>
> 相反，为了更简便的进行代码实现，我们将私钥和公钥都放在了一个文件中实现，解密时直接使用 `lambda` ！
>
> 这个在工程实现上是不安全的！

二、同态运算的代码实现

加法

对于密文 c_1 和 c_2 ，通过计算 $c = c_1 * c_2\ mod\ n^2$ 来实现加法运算。其背后的数学原理为：

$$
\begin{aligned}
c &= g^{m_1} r_1^n \cdot g^{m_2} r_2^n \\
&= g^{m_1+m_2} (r_1 \cdot r_2)^n \ \ mod\ \ n^2
\end{aligned}
$$

代码实现：

C++

```cpp
void Paillier::Add(mpz_t res, mpz_t c1, mpz_t c2)
{

    if (mpz_cmp(c1, nsquare) >= 0)
    {
        throw("ciphertext must be less than n^2");
        return;
    }
    if (mpz_cmp(c2, nsquare) >= 0)
    {
        throw("ciphertext must be less than n^2");
        return;
    }
    mpz_mul(res, c1, c2);
    mpz_mod(res, res, nsquare);
}
```

乘法

对于密文 c_1 ，通过计算 $c = c_1^e \ mod \ n^2$ 来实现乘法运算。其背后的数学原理为：

$$\begin{aligned} c = c_1^e &= (g^{m_1} r_1^n)^e \\ &= g^{m_1 \cdot e} (r_1^e)^n \ \ mod \ \ n^2 \end{aligned}$$

即， $Enc(m)^k = Enc(k * m)$

代码实现如下：

```cpp
// 只能是同态标量乘
void Paillier::Mul(mpz_t res, mpz_t c, mpz_t e)
{
    if (mpz_cmp(c, nsquare) >= 0)
    {
        throw("ciphertext must be less than n^2");
        return;
    }
    if (mpz_cmp(e, n) >= 0)
    {
        throw("exponent must be less than n");
    }
    mpz_powm(res, c, e, nsquare);
}
```

减法

Paillier 本身不直接支持减法操作，但由于它支持加法，你可以通过**加上负数**来实现“减法”。

我们想要计算的是：

$$Enc(m_1 - m_2) = Enc(m_1 + (-m_2))$$

关键在于，如何得到 $\mathrm{Enc}(-m_2)$ 。我们从乘法操作中获得启发。即， $Enc(m)^k = Enc(k * m)$ 。当 $k = -1$ 时，有： $Enc(m)^{-1} = Enc(-m) \ mod \ n^2$ ，即有如下推导：

$$\begin{aligned} Enc(m)^{-1} &= (g^m r^n)^{-1} \\ &= g^{-m} r^{-n} \ mod \ n^2 \end{aligned}$$

而 r^{-n} 对应的随机性保证了结果仍然是一个合法的密文。

> 注意： $Enc(m)^{-1} = (g^m r^n)^{-1}$ 是 $Enc(m)$ 的逆元。

也就是说，只需要求减数的乘法逆元，就可以实现减法操作了！

同态减法的完整推导过程

对于 $Enc(m_1) = g^{m_1} r_1^n$ ， $Enc(m_2) = g^{m_2} r_2^n$ ，其减法计算为：

$$\begin{aligned} Enc(m_1 - m_2) &= g^{m_1} r_1^n \cdot (g^{m_2} r_2^n)^{-1} \\ &= g^{m_1 - m_2} \cdot (r_1/r_2)^n \ mod \ n^2 \\ &= Enc(m_1 - m_2) \end{aligned}$$

> 也就是说，通过对减数取逆，再与被减数相加，可以完成减法操作！

代码实现如下：

```cpp
void Paillier::Sub(mpz_t res, mpz_t c1, mpz_t c2)
{
    mpz_t c2_inv;
    mpz_init(c2_inv);

    // 计算 c2 的模逆
    if (mpz_invert(c2_inv, c2, nsquare) == 0)
    {
        throw("c2 模 nsquare 没有逆元");
        mpz_clear(c2_inv);
        return;
    }

    // c1 * c2^{-1} mod nsquare
    mpz_mul(res, c1, c2_inv);
    mpz_mod(res, res, nsquare);

    mpz_clear(c2_inv);
}

```

运行测试用例后，发现：

```cpp
模数 n  的大小是 : 60491
模数 n/2 的大小是 : 30246

========= 开始测试减法 =========
情况1: 数据范围均 < n/2
m1 is 36 ; m2 is 24 ; m1 - m2 = 12
m1 is 36 ; m2 is 24 ; m2 - m1 = 60479
情况2: 数据范围出现 > n/2 的情况
m1 is 30248 ; m2 is 1 ; m1 - m2 = 30247
m1 is 1 ; m2 is 30245 ; m1 - m2 = 30247
========= 结束测试减法 =========
```

发现，

```C++
n = 60491
res = 60479
24 - 36 = -12
n - 12 = 60491 - 12 = 60479 = res // 这个结果表示，是对负数结果取模运算了
```

因此，这里要注意！

减法 `Paillier::Sub(mpz_t res, mpz_t c1, mpz_t c2)` 会出现两种情况：

- 当 $m_1 \geq m_2$ 时，解密后获得正常的计算结果，即： $res = m_1 - m_2$ ；
- 当 $m_1 < m_2$ 时，解密后的结果并不是负数！而是模 n 后的结果！即，$res = (m_1 - m_2)\ mod\ n$

这是因为我们的明文空间是 $Z_n = \{0, 1, 2, 3, ..., n-1\}$ ，解密结果也只能在这个空间里面。

> 注意，为了保证同态减法操作 `Paillier::Sub` 的结果可以被安全地解释为一个有符号整数（正或负），需要对明文的取值范围进行安全限定。
>
> 限制如下：
>
> - 限定明文范围 $m \in [0, B)$ ，其中， $B < n/2$ ！

这样，则同态加/减后的明文结果会落在 $(-B, B)$ 的范围，可以做出如下判断：

- 如果差值是正数 ⇒ 则解密结果 $< n/2$ ；
- 如果差值是负数 ⇒ 则解密结果 $> n/2$ 。

这样，就可以安全的判断数的正负，同时，也可以为后面的比较提供可行的思路。

比较

利用上述的原理解释，通过正负号的判断，来实现比较操作。即，对于 m_1 和 m_2 ，有

- 如果 $Paillier::Decrypt[Enc(m_1 - m_2)] < n/2$ ，则结果为正数，即， $m1 > m_2$
- 如果 $Paillier::Decrypt[Enc(m_1 - m_2)] > n/2$ ，则结果为负数，即， $m1 < m_2$

代码实现如下：

C++

```cpp
int Paillier::Compare(mpz_t c1, mpz_t c2)
{
    mpz_t c_diff, m_diff, n_half;
    mpz_inits(c_diff, m_diff, n_half, NULL);

    // 差值密文
    Sub(c_diff, c1, c2);
    // 解密得到明文差值
    Decrypt(m_diff, c_diff);

    // 计算 n/2
    mpz_fdiv_q_ui(n_half, n, 2);

    int cmp;
    if (mpz_cmp(m_diff, n_half) > 0)
    {
        cmp = -1; // m1 - m2 < 0, 说明 m1 < m2
    }
    else if (mpz_cmp_ui(m_diff, 0) == 0)
    {
        cmp = 0; // 相等
    }
    else
    {
        cmp = 1; // m1 > m2
    }

    mpz_clears(c_diff, m_diff, n_half, NULL);
    return cmp;
}

```

除法

对于密文 c_1 ，通过计算 $c = c_1^{-e} \; mod \; n^2$ 来实现乘法运算

$$\begin{aligned} c = c_1^{-e} &= (g^{m_1} r_1^n)^{-e} \\ &= g^{m_1/e}(r_1^{-e})^n \quad mod \;\; n^2 \end{aligned}$$

即，计算除数 e 对 n^2 的乘法逆元即可。

代码实现如下：

```cpp
void Paillier::Div(mpz_t res, mpz_t c, mpz_t k)
{
    if (mpz_cmp(c, nsquare) >= 0)
    {
        throw("ciphertext must be less than n^2");
        return;
    }

    mpz_t k_inv;
    mpz_init(k_inv);

    // 计算 k 在 mod n 下的逆元
    if (mpz_invert(k_inv, k, n) == 0)
    {
        throw("k has no inverse modulo n");
        mpz_clear(k_inv);
        return;
    }

    // c^{k^{-1}} mod n^2
    mpz_powm(res, c, k_inv, nsquare);

    mpz_clear(k_inv);
}

```

注意：

- 这个是标量除法，除数不能是加密后的密文数据；
- 除数 e 需要和 n 互素，保证除数 e 有模逆元；
- 因为Paillier整数空间，该方法只有当被除数 m 是除数 e 的整数倍时才有效，否则解密失败。

三、测试用例及运行结果

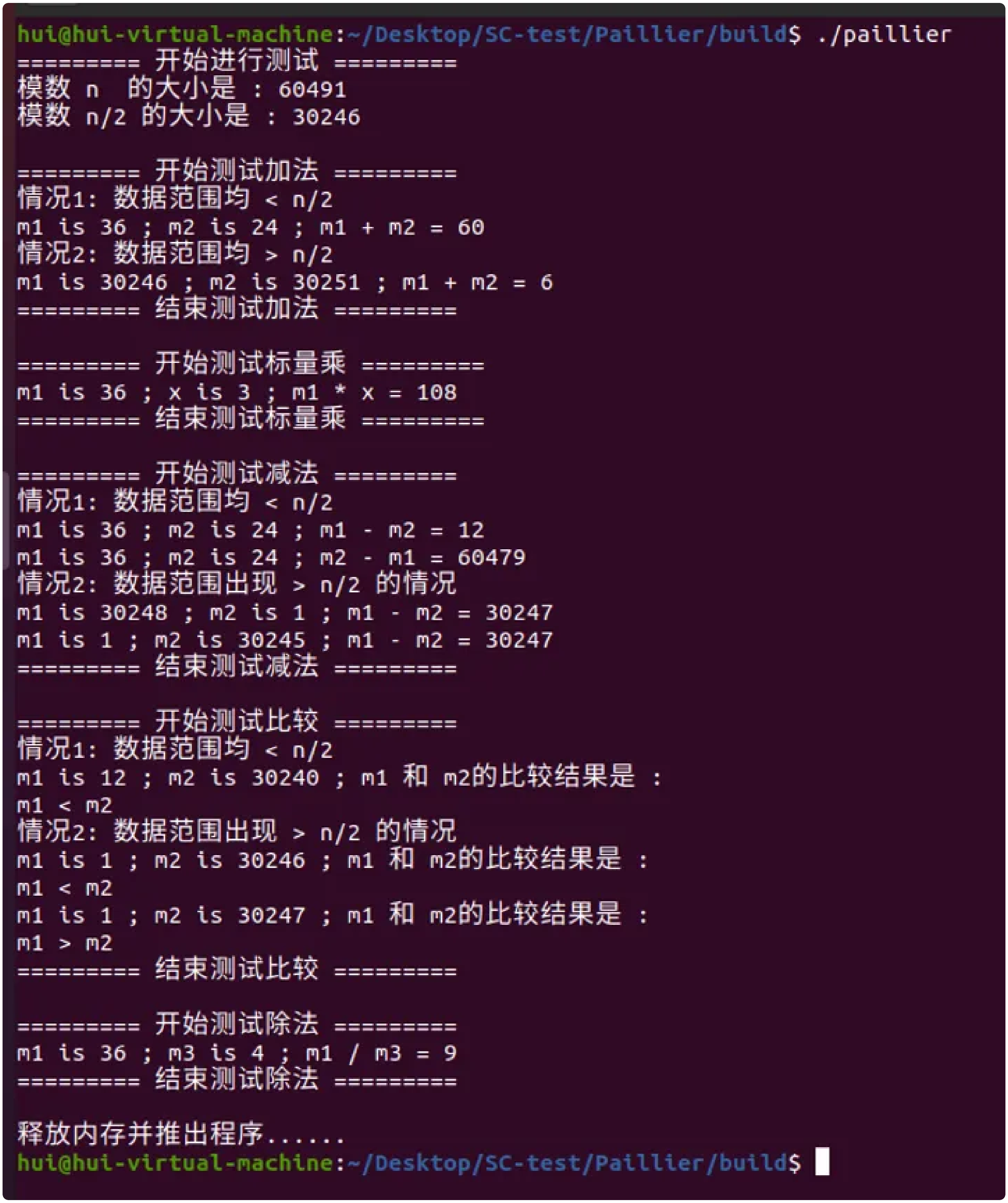
代码整体结构：

```
hui@hui-virtual-machine:~/Desktop/SC-test/Paillier$ tree
.
├── build
├── CMakeLists.txt
└── src
    ├── main.cpp
    ├── paillier.cpp
    └── paillier.h

2 directories, 4 files

```

测试结果：

```C++
hui@hui-virtual-machine:~/Desktop/SC-test/Paillier/build$ ./paillier
========= 开始进行测试 =========
模数 n  的大小是 : 60491
模数 n/2 的大小是 : 30246

========= 开始测试加法 =========
情况1: 数据范围均 < n/2
m1 is 36 ; m2 is 24 ; m1 + m2 = 60
情况2: 数据范围均 > n/2
m1 is 30246 ; m2 is 30251 ; m1 + m2 = 6
========= 结束测试加法 =========

========= 开始测试标量乘 =========
m1 is 36 ; x is 3 ; m1 * x = 108
========= 结束测试标量乘 =========

========= 开始测试减法 =========
情况1: 数据范围均 < n/2
m1 is 36 ; m2 is 24 ; m1 - m2 = 12
m1 is 36 ; m2 is 24 ; m2 - m1 = 60479
情况2: 数据范围出现 > n/2 的情况
m1 is 30248 ; m2 is 1 ; m1 - m2 = 30247
m1 is 1 ; m2 is 30245 ; m1 - m2 = 30247
========= 结束测试减法 =========

========= 开始测试比较 =========
情况1: 数据范围均 < n/2
m1 is 12 ; m2 is 30240 ; m1 和 m2的比较结果是 :
m1 < m2
情况2: 数据范围出现 > n/2 的情况
m1 is 1 ; m2 is 30246 ; m1 和 m2的比较结果是 :
m1 < m2
m1 is 1 ; m2 is 30247 ; m1 和 m2的比较结果是 :
m1 > m2
========= 结束测试比较 =========

========= 开始测试除法 =========
m1 is 36 ; m3 is 4 ; m1 / m3 = 9
========= 结束测试除法 =========

释放内存并推出程序......
```

运行截图:



```
hui@hui-virtual-machine:~/Desktop/SC-test/Paillier/build$ ./paillier
========= 开始进行测试 =========
模数 n  的大小是 : 60491
模数 n/2 的大小是 : 30246

========= 开始测试加法 =========
情况1: 数据范围均 < n/2
m1 is 36 ; m2 is 24 ; m1 + m2 = 60
情况2: 数据范围均 > n/2
m1 is 30246 ; m2 is 30251 ; m1 + m2 = 6
========= 结束测试加法 =========

========= 开始测试标量乘 =========
m1 is 36 ; x is 3 ; m1 * x = 108
========= 结束测试标量乘 =========

========= 开始测试减法 =========
情况1: 数据范围均 < n/2
m1 is 36 ; m2 is 24 ; m1 - m2 = 12
m1 is 36 ; m2 is 24 ; m2 - m1 = 60479
情况2: 数据范围出现 > n/2 的情况
m1 is 30248 ; m2 is 1 ; m1 - m2 = 30247
m1 is 1 ; m2 is 30245 ; m1 - m2 = 30247
========= 结束测试减法 =========

========= 开始测试比较 =========
情况1: 数据范围均 < n/2
m1 is 12 ; m2 is 30240 ; m1 和 m2的比较结果是 :
m1 < m2
情况2: 数据范围出现 > n/2 的情况
m1 is 1 ; m2 is 30246 ; m1 和 m2的比较结果是 :
m1 < m2
m1 is 1 ; m2 is 30247 ; m1 和 m2的比较结果是 :
m1 > m2
========= 结束测试比较 =========

========= 开始测试除法 =========
m1 is 36 ; m3 is 4 ; m1 / m3 = 9
========= 结束测试除法 =========

释放内存并推出程序......
hui@hui-virtual-machine:~/Desktop/SC-test/Paillier/build$
```