南開大學
Nankai University

网络空间安全学院

密码学实验报告

实验 4：RSA 实验报告

姓名：魏伯繁

学号：2011395

专业：信息安全

2022 年 12 月 29 日

目录

1 实验要求与实验目的

1.1 实验目的

通过实际编程了解公钥密码算法 RSA 的加密和解密过程，加深对公钥密码算法的了解和使用。

1.2 实验要求

在本次实验中需要实现以下内容:
1、为了加深对 RSA 算法的了解，根据已知参数：p = 3，q = 11，m = 2，手工计算公钥和私钥，并对明文 m 进行加密，然后对密文进行解密。
2、编写一个程序，用于生成 512 比特的素数。
3. 利用 2 中程序生成的素数，构建一个 n 的长度为 1024 比特的 RSA 算法，利用该算法实现对明文的加密和解密。
4、在附件中还给出了一个可以进行 RSA 加密和解密的对话框程序 RSATool，运行这个程序加密一段文字，了解 RSA 算法原理。

2 实验原理

2.1 公钥密码体制

序列密码和分组密码算法都要求通信双方通过交换密钥实现使用同一个密钥，这在密钥的管理、发布和安全性方面存在很多问题，而公钥密码算法解决了这个问题。

公钥密码算法是指一个加密系统的加密密钥和解密密钥是不同的，或者说不能用其中一个推导出另一个。在公钥密码算法的两个密钥中，一个是用于加密的密钥，它是可以公开的，称为公钥；另一个是用于解密的密钥，是保密的，称为私钥。公钥密码算法解决了对称密码体制中密钥管理的难题，并提供了对信息发送人的身份进行验证的手段，是现代密码学最重要的发明。

2.2 RSA 密码体制

RSA 密码体制是目前为止最成功的公钥密码算法，它是在 1977 年由 Rivest、Shamir 和 Adleman 提出的第一个比较完善的公钥密码算法。它的安全性是建立在“大数分解和素性检测”这个数论难题的基础上，即将两个大素数相乘在计算上容易实现，而将该乘积分解为两个大素数因子的计算量相当大。虽然它的安全性还未能得到理论证明，但经过 40 多年的密码分析和攻击，迄今仍然被实践证明是安全的。

RSA 密码的计算步骤如下：选择两个不同的大素数 p 和 q，n 是二者的乘积，即 n = pq，使

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

对于公钥的选取，需要随机选取正整数 e，使其满足:

$$gcd(\varphi(n), e) = 1$$

即 e 和互素，则将（n，e）作为公钥。
对于私钥的选取，需要求出正数 d，使其满足：

$$e \times d \equiv 1 \ mod(\varphi(n))$$

则将（n，d）作为私钥。
在加密时做如下计算：其中 m 为明文，c 为加密得到的密文

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

在解密时做如下计算：其中 c 是加密得到的密文，m 为恢复得到的明文

$$m \equiv c^d \ mod(n)$$

2.3 米勒-拉宾算法

在编写 RSA 的过程中，一个重要的流程就是寻找大素数，一般而言需要 512 比特大小的素数才是足够安全的，那么如何判断一个 512 比特的数是否是素数呢，很显然，一般的筛法、穷举肯定是不合适，于是我们需要引进一个高校的素数判断方式：米勒-拉宾算法

米勒拉宾的理论基础是两个定理，第一个定理是我们在信息安全数学基础这么课程中学习的费马定理：

引理 1(费马定理) 设 p 是素数，a 为整数，且 $(a,p)=1$，则 $a^{p-1} \equiv 1(\text{mod } p)$。

证明: 考虑 $1,2,3.....(p-1)$ 这 $p-1$ 个数字，给它们同时乘上 a，得到 $a,2a,3a.....(p-1)a$。

$\because \ a \neq b(\text{mod } p),(c,p)=1$

$\therefore \ ac \neq bc(\text{mod } p)$

$\therefore \ 1\times 2\times 3.....(p-1) \equiv a.2a.3a.....(p\text{-}1)a(\text{ mod } p)$

$\therefore \ (p-1)! \equiv (p-1)!a^{p-1}(\text{mod } p)$

$\because \ ((p-1)!,p)=1$

$\therefore \ a^{p-1} \equiv 1(\text{mod } p)$

图 2.1: 费马定理

第二个是由此引申的二次探测定理

引理 2(二次探测定理) 如果 p 是一个素数，且 $0 < x < p$，则方程 $x^2 \equiv 1(\mathrm{mod}\ p)$ 的解为 $x = 1, p-1$。

证明： 易知 $x^2 - 1 \equiv 0(\mathrm{mod}\ p)$

$\therefore\ (x+1)(x-1) \equiv 0(\mathrm{mod}\ p)$

$\therefore\ p \mid (x-1)(x+1)$

$\because\ p$ 为质数

$\therefore\ x = 1$ 或者 $x = p-1$

图 2.2: 二次探测定理

有了上面两个定理作为理论保障，我们便可以给出米勒——拉宾算法的具体步骤

假设 n 是奇素数，则 $n-1$ 必为偶数。令 $n-1 = 2^q \cdot m$。

随机选取整数 $a(0 < a < n)$，由费马定理，$(a^{2^q \cdot m} = a^{n-1}) \equiv 1(\mathrm{mod} n)$。由二次探测定理可知：$a^{2^{q-1} \cdot m} \equiv 1(\mathrm{mod} n)$ 或 $a^{2^{q-1} \cdot m} \equiv n-1(\mathrm{mod} n)$。若 $a^{2^{q-1} \cdot m} \equiv 1(\mathrm{mod} n)$ 成立，则再次由二次探测定理可知：$a^{2^{q-2} \cdot m} \equiv 1(\mathrm{mod} n)$ 或 $a^{2^{q-2} \cdot m} \equiv n-1(\mathrm{mod} n)$，……。如此反复应用二次探测定理，直到某一步 $a^m \equiv 1(\mathrm{mod} n)$ 或 $a^m \equiv n-1(\mathrm{mod} n)$。总之，若 n 是素数，则 $a^m \equiv 1(\mathrm{mod} n)$，或存在 $0 \le r \le q-1$,使 $a^{2^r \cdot m} \equiv n-1(\mathrm{mod} n)$。

所以该算法过程为

- 给定奇数 n，为了判断是否为素数，首先测试 $a^{2^q \cdot m} \equiv 1(\mathrm{mod} n)$ 是否成立。若不成立，则 n 一定为合数；若成立，则继续运行算法做进一步的测试。
- 考察下面的 **Miller 序列**:

  $a^m(\mathrm{mod} n), a^{2m}(\mathrm{mod} n), a^{4m}(\mathrm{mod} n), \ldots\ldots, a^{2^{q-1} \cdot m}(\mathrm{mod} n)$

  若 $a^m \equiv 1(\mathrm{mod} n)$，或者存在某个整数 $0 \le r \le q-1$,使 $a^{2^r \cdot m} \equiv n-1(\mathrm{mod} n)$ 成立，则称 n 通过 **Miller 测试**。

图 2.3: 米勒拉宾算法

当然我们也需要注意:

由上面的分析可知，素数一定通过 Miller 测试。所以，如果 n 不能通过 Miller 测试，则 n 一定是合数；如果 n 能通过 Miller 测试，则 n 很可能是素数。这就是 **Miller-Rabin 算法**。

可以证明 Miller-Rabin 算法给出的错误结果的概率小于等于 $\frac{1}{4}$。若反复测试 k 次，则错误概率可降低至 $(\frac{1}{4})^k$。这是一个很保守的估计，实际使用的效果要好得多。

图 2.4: 米勒拉宾算法的局限性

在本次实验中，我们使用 5 轮米勒-拉宾检测，可以将错误率降低至 0.0005，基本可以认定，如果可以通过五轮米勒-拉宾算法，则可以认为其就是素数。

3 实验流程图

3.1 RSA 流程图

首先展示 RSA 的具体流程图，因为 RSA 属于公钥密码体制，所以他的安全性是由数学定理保证的，并不是单纯依赖于加密解密步骤的复杂性，所以 RSA 的总体流程图也相对简单。其具体的步骤在第二部分也进行了详细的说明，在这里绘制出详细的步骤图帮助理解。

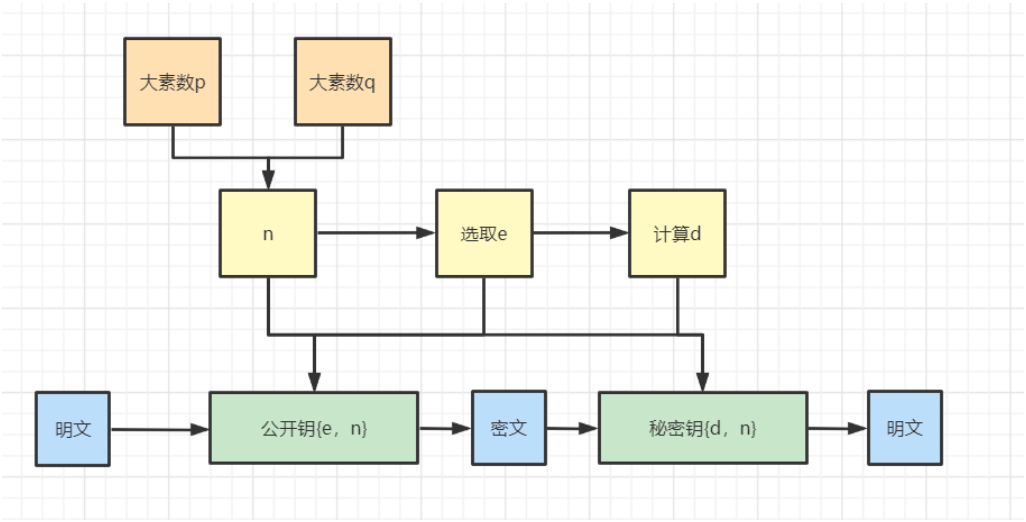

图 3.5: RSA 流程图

3.2 米勒-拉宾算法

虽然 RSA 的步骤并不复杂，但是其需要用到不少的算法，米勒拉宾算法就是其中最重要的算法，它用来帮助我们在短时间内判断一个大数是否可能是素数

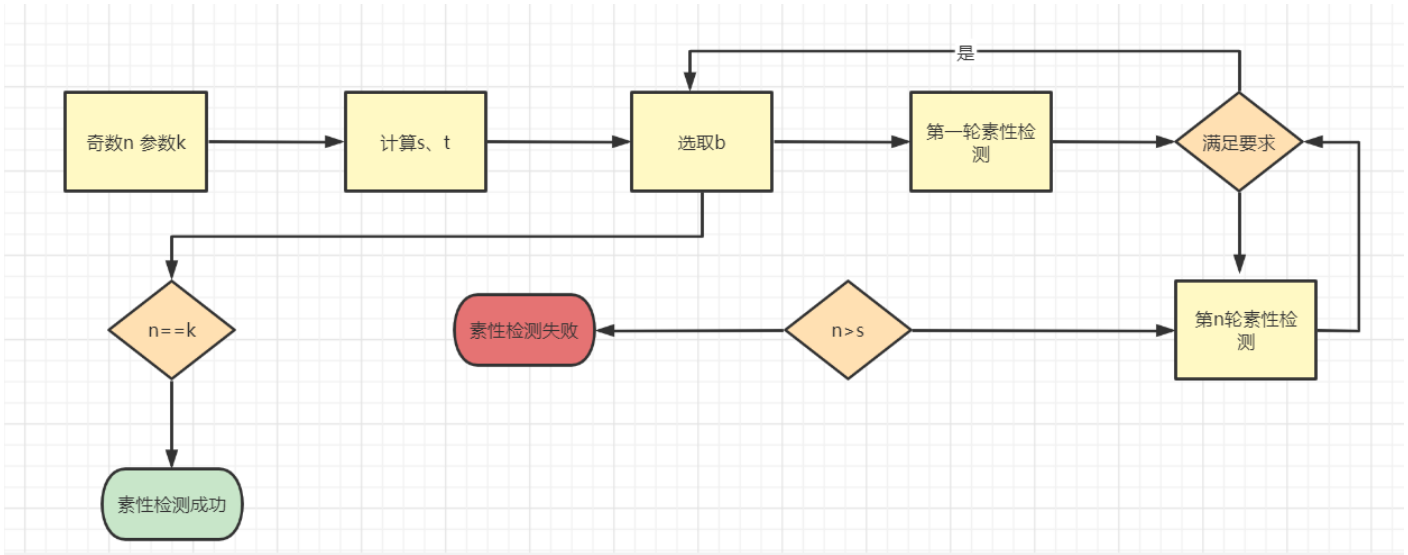

图 3.6: 米勒-拉宾算法流程图

4 实验内容

4.1 手算 RSA

首先，按照实验指导手册的要求，为了更好的理解 RSA 算法，我们动手计算一个 RSA 加密过程，具体的参数为

$$p = 3 \quad q = 11 \quad m = 2$$

则根据 RSA 的加密步骤，我们可以得到如下参数：

$$n = p \times q \ = 33$$

然后求出其欧拉函数

$$\varphi(n) = (p-1) \times (q-1) \ = 20$$

接下来选取 e 并计算 d

$$e = 7 \qquad d = 3$$

于是，我们可以给出公开钥和私密钥

$$\{7, 33\} \qquad \{3, 33\}$$

假设我们需要加密的明文是 4，那么加密过程为：

$$c \equiv 4^7 mod 33 \ \equiv \ 16$$

得到的密文结果为 16，解密过程为：

$$m \equiv 16^3 mod 33 \ \equiv \ 4$$

4.2 代码实现

根据实验手册要求，需要实现 512 位的加密解密流程，512bit 的大素数远远超过了 C++ 可以利用的 64 位 long，所以我们需要自己设计一个大整数类型来完成加减乘除取模运算，并且还需要设计欧几里得算法、扩展的欧几里得算法来使大整数也能被代入运算。

对于大整数，我们需要重载一些算数运算符以及比较运算符，摒弃编写一些功能函数来简化我们的开发工作，比如说拷贝构造函数、判断是否相等、随机生成一个数、随机生成一个素数等等。

大整数类定义

```
1     class BigInt {
2 public:
3   //类内属性
4   vector<int>num;//用来存储真正的数,在这里我们默认0是最高位
5   bool minus = false;//增加负数表示
6
```

```
  //重载运算符
  friend BigInt operator+(const BigInt& a, const BigInt& b);
  friend BigInt operator-(const BigInt& a, const BigInt& b);
  friend BigInt operator*(const BigInt& a, const BigInt& b);
  friend BigInt operator/(const BigInt& a, const BigInt& b);
  friend BigInt operator%(const BigInt& a, const BigInt& b);
  //这些比较目前只比较num, 不比较正负位
  friend bool operator==(const BigInt& a, const BigInt& b);
  friend bool operator!=(const BigInt& a, const BigInt& b) { return !(a == b); }
  friend bool operator>=(const BigInt& a, const BigInt& b);
  friend bool operator<(const BigInt& a, const BigInt& b) {
    bool ret = a >= b;
    return !ret;
  }
  //其他功能性函数
  BigInt();
  BigInt(int size, bool minus = false);
  BigInt(int size, vector<int>& v,bool minus = false);
  BigInt(const BigInt& bi);
  bool isLegal();//判断vector中的数是否合法
  bool strictLegal();//判断长度、数字是否合法
  void generateNum(int size);//随机生成一个数
  void generateOdd(int size);
  void randGenerate(BigInt& n);
  void equal(BigInt& b);
};
```

由于其内容较多，不能逐一展示，针对大整数类，在本报告中仅展示较难实现的除法，下面的代码就是我对自定义的大整数除法的实现

具体的思路为在计算机组成原理课程中学习的除法的实现，我们只需要对除数补零然后查看被除数是否大于补零后的除数，如果大于则商 1 相减，反之则商零然后去掉一个补的零。最后在结尾需要去除前导零。

大整数类除法实现

```
BigInt operator/(const BigInt& a, const BigInt& b) {
  if (a < b) {
    //cout << "非法的除法, 被除数小于除数" << endl;
    BigInt bi;
    return bi;
  }
  BigInt Mya(a);
  vector<int>vi;//存储结果的地方
  for (int i = 0; i < a.num.size(); i++) {
    BigInt myb = b;
    for (int j = 1; j <= ((a.num.size() - 1) - i); j++) {
      myb.num.push_back(0);
    }
    //cout << i << " ";
```

```
    //cout << Mya<<" "<< myb << endl;
    if (Mya >= myb) {
      vi.push_back(1);
      Mya = Mya - myb;
      continue;
    }
    if (Mya < myb) {
      vi.push_back(0);
      continue;
    }
    //cout << Mya << " " << myb << " " << vi[i]<<endl;
  }
  //cout << endl;
  //去除前导零
  reverse(vi.begin(), vi.end());
  for (int i = vi.size() - 1; i >= 0; i--) {
    if (vi[vi.size() - 1] == 0) { vi.pop_back(); }
    else { break; }
  }
  reverse(vi.begin(), vi.end());
  BigInt ret(vi.size(), vi);
  if ((a.minus && b.minus) || (!a.minus && !b.minus)) { ret.minus = false; }
  else { ret.minus = true; }
  return ret;
}
```

接下来展示米勒-拉宾算法的具体内容，我们首先对输入的奇数计算具体的参数 s 和 t，然后就开始 k 轮循环，每一次循环选择一个随机数 b，如果 b 能够完成判断则再次选择一个 b 进行判断，直到达到 k 轮，则返回 true，反之则返回 false

米勒-拉宾实现

```
bool Miller_Rabin(BigInt b,int k) {
  if (b.num[b.num.size() - 1] == 0) { return false; }
  int s=0;//这是2的整数次幂的表示
  BigInt myb(b);
  BigInt t;//另外一部分
  BigInt r;//每一轮迭代都要用到的
  //构造一个代表2的BigInt数
  vector<int>vi; vi.push_back(1); vi.push_back(0);
  BigInt TWO(2, vi);
  vector<int>vi2; vi2.push_back(1);
  BigInt ONE(1, vi2);
  myb = myb - ONE;
  //首先需要确定s和t
  while (true) {
    BigInt bi;
    bi = myb % TWO;
    if (bi.num.size() == 0) {
```

```
        s++;
        myb = myb / TWO;
        continue;
      }
      else {
        t = myb;
        break;
      }
    }
    //cout << "s= " << s << "  " << "t= " << t << endl;
    for (int j = 1; j <= k; j++) {
      BigInt bb;
      Sleep(1000);
      bb.randGenerate(b);
      //cout << "bb=" << bb << endl;
      for (int i = 0; i < s; i++) {
        if (i == 0) {
          r = MyquickPow(bb, t, b);
          if (r == ONE) { break; }
          BigInt check = b - ONE;
          if (r == check) { break; }
          if(s==1){ cout << "fail to pass Miller_Rabin" << endl; return false; }
          continue;
        }
        else {
          r = MyquickPow(r, TWO, b);
          BigInt check = b - ONE;
          if (r == check) { break; }
          if (i == s - 1) { cout << "fail to pass Miller_Rabin"<<endl; return false; }
          continue;
        }
      }
    }
    cout << "pass Miller_Rabin successfully" << endl;
    return true;
}
```

除此之外还有一个非常重要的函数就是 pow 也就是做幂运算的函数，因为在米勒-拉宾算法中需要计算幂指数，而一般来说 t 都是 200 比特以上的数，所以一个一个运算明显是不现实的，这时我们需要使用快速幂运算来实现，也就是说如果计算 a 的 b 次方则可以转换为计算 a2 的 b/2 次方，这样相当于只需要 logn 次就可以计算一次幂运算，极大提升了运算速度。

快速幂算法实现

```
BigInt quickPow(BigInt& a, BigInt& b, BigInt& m) {
  cout << "b=" << b << endl;
  //他其实需要做的就只是调度
  vector<int>vi2; vi2.push_back(1); vi2.push_back(0);
  BigInt TWO(2, vi2);
```

```
    vector<int>vi1; vi1.push_back(1);
    BigInt ONE(1, vi1);
    //如果是2次方直接计算
    if (b == TWO) {
        BigInt ret;
        ret = pow(a, b, m);
        return ret;
    }
    //一次方直接返回
    if (b == ONE) {
        return a;
    }
    //偶数次方
    if (b.num[b.num.size() - 1] == 0) {
        BigInt ret;
        BigInt t = b / TWO;
        BigInt Mya = (a * a)%m;
        ret = quickPow(Mya, t, m);//转而计算a^2的b/2次方
        return ret;

    }
    //奇数次方
    else {
        BigInt ret;
        BigInt t = b / TWO;
        BigInt Mya = (a * a)%m;
        ret = quickPow(Mya, t, m);//转而计算a^2的b/2次方
        ret = (ret * a) % m;//因为是奇数所以还要再乘一个a
        return ret;
    }
}
```

5 实验结果

5.1 512 位素数加密

我们首先选择使用 512 位的密钥，并输入需要加密的文本字符串，随后程序会根据其 ASCII 码将其转换为 16 进制数并进行明文分组。

随后，程序会生成一段文字来展示选取的大素数、公钥组成部分 e、n 以及私钥组成部分 d、n

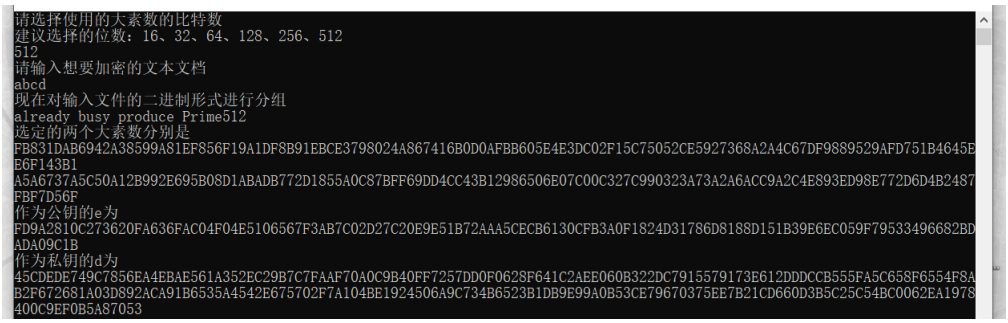

图 5.7: 大素数选取公钥私钥展示

在经过一系列计算后会计算出加密前的文本以及加密后的文本，为了验证程序的正确性程序会做解密，并展示解密结果。

图 5.8: 加解密效果

5.2 熟悉使用 RSAtool2

根据实验手册的要求，我们还需要熟悉使用程序 RSAtools，首先选择需要的密钥的比特数，然后点击 start 开始生成密钥，当生成成功后点击 generate 就可以看到选取到的两个大素数以及其乘积和密钥。

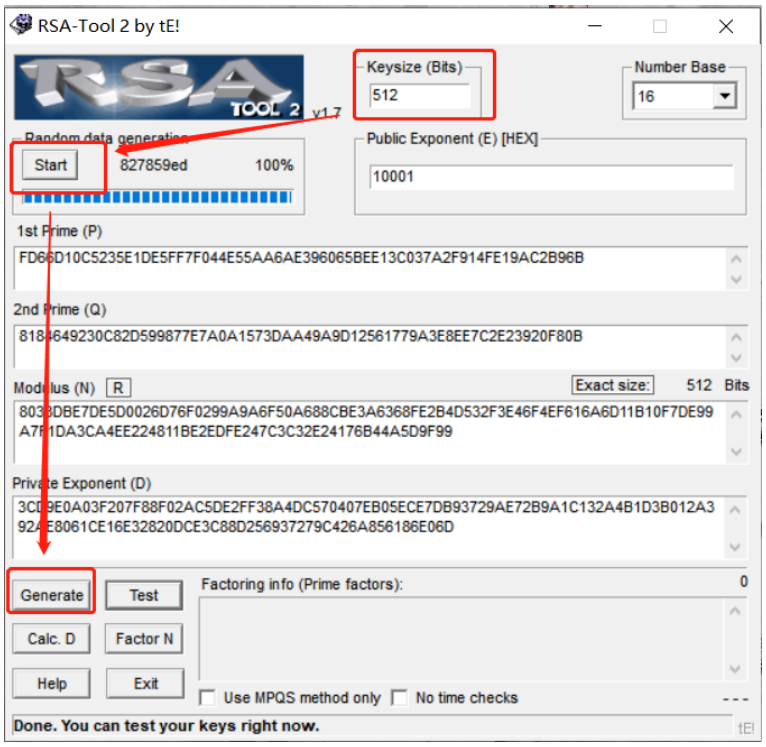

图 5.9: RSAtool2 使用

在生成了对应的密钥对后，我们就可以尝试测试了，点击 Test 按钮进行测试，输入想要加密的明

文，随后就会呈现加密后的密文：

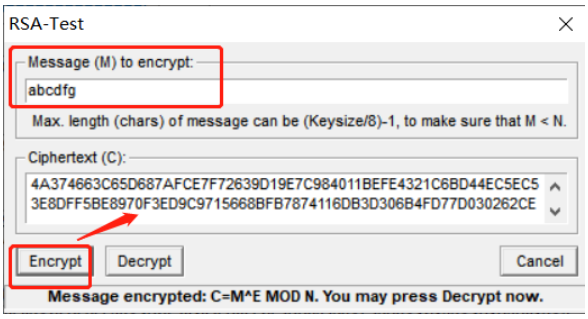

图 5.10: RSAtool2 使用

加下来点击 decrypt 可以进行解密，我们发现解密之后的信息和明文相同，说明加解密成功!

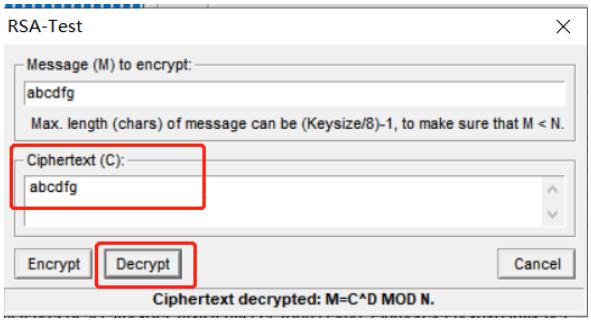

图 5.11: RSAtool2 使用

6 总结

通过本次实验，我熟悉了公钥密码的内涵，充分理解了公钥密码的安全性保证不依赖于过程的复杂性而是依赖于数学定理。同时，通过动手编写程序，我也进一步加深了对大整数运算的理解，动手实现大整数运算以及米勒-拉宾算法素性检测极大的增强了我的代码能力，让我收益匪浅。