《密码学基础原理》实验报告

课程：密码学基础原理　　实验名称：　RSA 算法

姓名：　陈钦　　实验日期：　2022/10/28

学号：　2021131094　　实验报告日期：　2022/10/30

班级：　区块链工程 213 _

教师评语： 签名： 日期：	成绩：

一、实验名称

RSA 算法

二、实验环境（详细说明运行的系统、平台及代码等）

1. 系统：go version go1.19 windows/amd64
2. IDE ：GoLand 2022.2.2

三、实验目的

（1）加深对 *RSA* 算法的理解；

（2）阅读标准和文献，提高自学能力；

（3）加深对模块化设计的理解，提高编程实践能力。

四、实验内容、步骤及结果

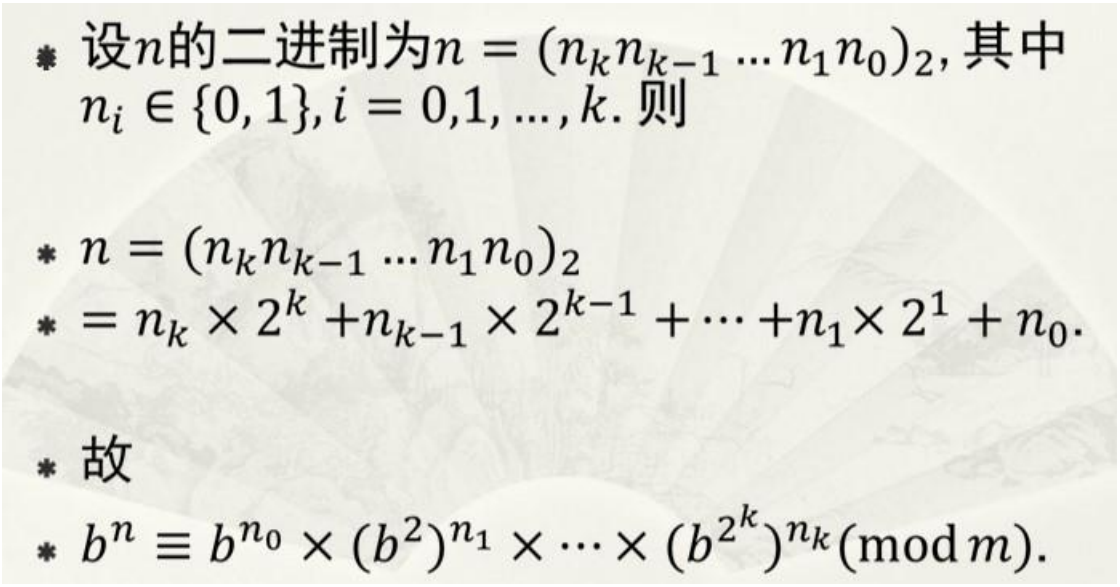
1．实验内容

根据 RSA 算法的原理，编写程序进行计算

2．实验步骤

（1）我们先根据模重复平方法写一个函数，用以解决编程过程中间值过大导致溢出问题。

其原理如图：



根据原理编写的代码如图：

```go
// 模重复平方法
func modRepeateQuadratic_94(b int, n int, m int) int {
    s := 1                                  //用于累加累乘得到结果
    x := strconv.FormatInt(int64(n), base: 2) //将数n转换为二进制
    //从低位开始遍历（右边）
    fmt.Printf( format: "  【模重复平方法过程】")
    for i := len(x) - 1; i >= 0; i-- {
        if int(x[i])-48 == 1 { //减48：因为ASCII码
            s = (s * b) % m //累乘取模
        }
        b = (b * b) % m //将b指数+1
        fmt.Printf( format: "[i=%v,中间值=%v]", i, s)
    }
    fmt.Println()
    return s
}
```

因为实验要求打印计算过程的中间值，因此我们再 for 循环中打印输出每一个值

（2）编写代码求私钥

计算私钥 d 的核心思想是扩展欧几里得算法。欧几里得算法（gcd），是一个求数 A 和 B 的最大公因数的高效算法，我们有：

$$\gcd(a,b)=\gcd(b,a \mod b)$$

我们关注怎么使用它来解决同余方程，计算出我们想要的私钥 d。下面是扩展欧几里得算法的思路：

$$
\begin{aligned}
\gcd(a,n) &= ax_1 + ny_1 \\
\gcd(n, a\%n) &= nx_2 + (a\%n)y_2 \\
&= nx_2 + (a - a/n * n) * y_2 \\
&= nx_2 + ay_2 - a/n * n * y_2 \\
&= ay_2 + n(x_2 - a/n * y_2)
\end{aligned}
$$

因为$\gcd(a,n) = \gcd(n, a\%n)$，所以$x_1 = y_2, y_1 = x_2 - a/n * y_2$，我们得到了计算a与n的最大公因数的过程中x和y的递推公式，据此可以写出算法的简单实现代码。

```
8     // 用于求私钥d
9     var x int = 0 //调用exGcd()之后，x的值就是私钥的d的值
10    var y int = 0 //线性同余方程式y的值，没用，用来占位
```

```
28  // 求私钥d
29  // a：公钥e。n：φ(n)
30  func privateKey_94(a int, n int) int {
31      if n == 0 {
32          x = 1
33          y = 0
34          return a
35      } else if a == 0 {
36          x = 0
37          y = 1
38          return n
39      } else {
40          c := privateKey_94(n, a%n)
41          tmp := x
42          x = y
43          y = tmp - a/n*y
44          return c
45      }
46  }
```

为了后续的打印输出，我们将这个方法封装起来

```
48  // 求私钥d
49  // a：公钥e。n：φ(n)
50  func exGcd_94(a int, n int) {
51      privateKey_94(a, n)
52      fmt.Printf( format: "(2)私钥d= %v\n", x+n)
53  }
```

（3）对整个 AES 算法进行封装，为了输出美观整理，进行打印输出整理

```go
// RSA算法封装
func RSA_94(e int, p int, q int, m int) {
    n := p * q
    Fn := (p - 1) * (q - 1)
    fmt.Printf( format: "用户输入的值为: e=%v, p=%v, q=%v, m=%v\n", e, p, q, m)
    fmt.Printf( format: "(1)n=%v, n的欧拉函数值=%v\n", n, Fn)
    exGcd_94(e, Fn)
    temp := modRepeateQuadratic_94(m, e, n)
    fmt.Printf( format: "(3)加密结果(密文c)= %v\n", temp)
    fmt.Printf( format: "(4)解密结果(明文m)= %v", modRepeateQuadratic_94(temp, x+Fn, n))
}
```

3.实验结果

首先查看编写的程序是否正确，使用书本上的例子检验：检验成功，编写正确

```go
func main() {
    RSA_94( e: 13,  p: 23,  q: 29,  m: 9)
}
```

```
main()
Run:  go build cypto_lab_02.go
用户输入的值为: e=13, p=23, q=29, m=9
(1)n=667, n的欧拉函数值=616
(2)私钥d= 853
  【模重复平方法过程】[i=3,中间值=9][i=2,中间值=9][i=1,中间值=353][i=0,中间值=564]
(3)加密结果(密文c)= 564
  【模重复平方法过程】[i=9,中间值=564][i=8,中间值=564][i=7,中间值=64][i=6,中间值=64][i=5,中间值=347][i=4,中间值=347][i=3,中间值=100][i=2,中间值=100][i=1,中间值=93][i=0,中间值=9]
(4)解密结果(明文m)= 9
```

第一组实验测试：

```
Run:  go build cypto_lab_02.go
用户输入的值为: e=7, p=13, q=17, m=22
(1)n=221, n的欧拉函数值=192
(2)私钥d= 247
  【模重复平方法过程】[i=2,中间值=22][i=1,中间值=40][i=0,中间值=61]
(3)加密结果(密文c)= 61
  【模重复平方法过程】[i=7,中间值=61][i=6,中间值=14][i=5,中间值=22][i=4,中间值=22][i=3,中间值=107][i=2,中间值=22][i=1,中间值=107][i=0,中间值=22]
(4)解密结果(明文m)= 22
```

第二组实验测试：

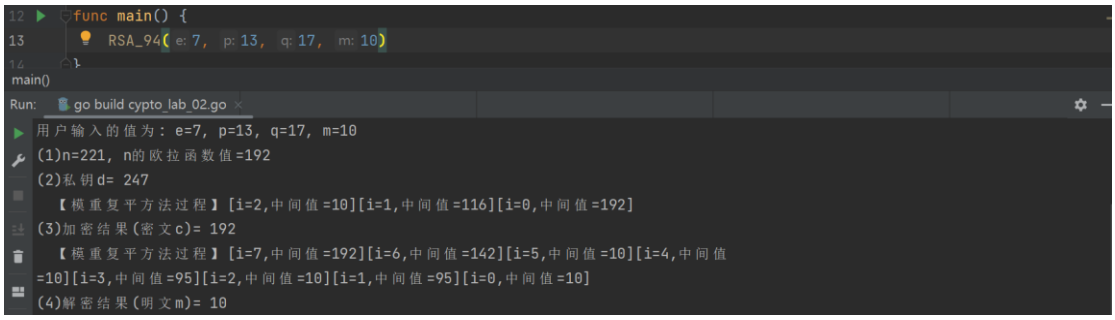
```go
func main() {
    RSA_94( e: 7,  p: 13,  q: 17,  m: 1094)
}
```

```
Run:  go build cypto_lab_02.go
C:\Users\ChenQin\AppData\Local\Temp\GoLand\___go_build_cypto_lab_02_go.exe
用户输入的值为: e=7, p=13, q=17, m=1094
(1)n=221, n的欧拉函数值=192
(2)私钥d= 247
  【模重复平方法过程】[i=2,中间值=210][i=1,中间值=216][i=0,中间值=167]
(3)加密结果(密文c)= 167
  【模重复平方法过程】[i=7,中间值=167][i=6,中间值=109][i=5,中间值=210][i=4,中间值=210][i=3,中间值=6][i=2,中间值=210][i=1,中间值=6][i=0,中间值=210]
(4)解密结果(明文m)= 210
```

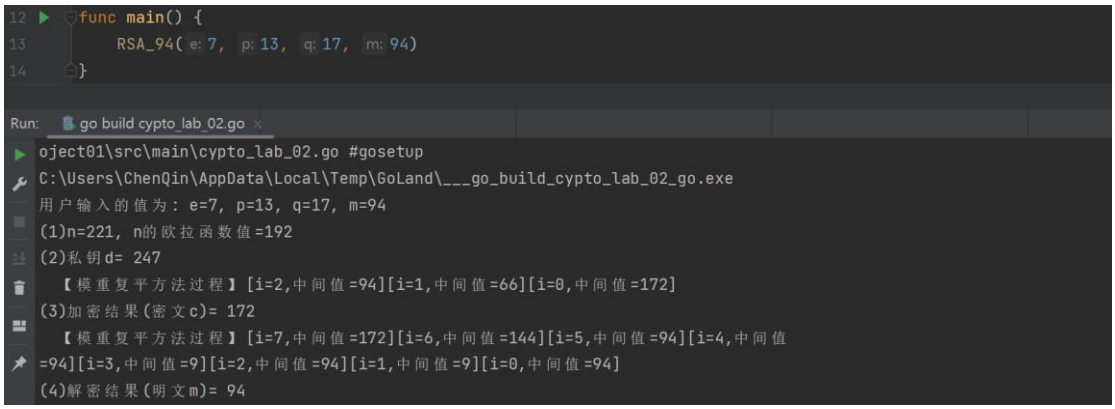
输出错误

我们发现，当m＞n 的时候，输出的结果错误。不过这并不是什么大问题，底数 m 代表的明文如果实在太长，我们把它分割一下行了。因此，我们把 1094 分成

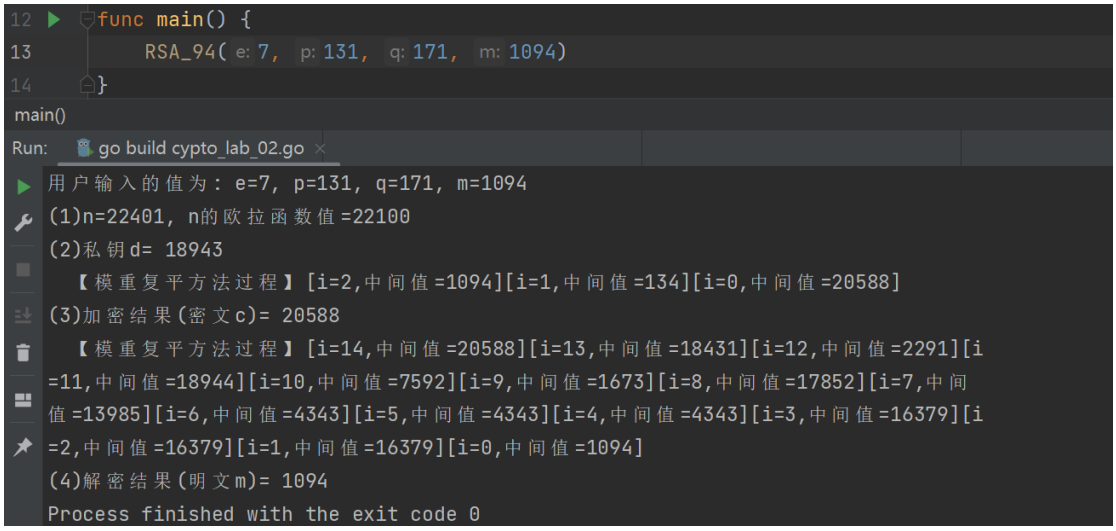
10 和 94 两个部分分别加密解密，然后拼凑起来就行了
10 部分：



94 部分：



另外一种解决方法是将 p 和 q 的值调大，使得欧拉函数值大于 m



第三组实验测试：
因为输入的m是字符串，我的想法是将这个字符串每一个字符对应的 ASCII 码值拼接起来成为一个新的数值。但是因为拼接出来的数值一般是非常大的，不仅超过了 int 的范围，而且还大于 Φ(n)。因此，我将结果取模操作，映射成为一个符合算法、较小的值来作为输入。这里我将模设为 1000，也就是说任何字符串输入，都将被映射成 0 至 999 的其中一个数值。这样当然降低了安全性，但这只是这个方法的原理，如果要做到安全，模要取很大，也不能用 int 作为变量。

```go
func main() {
    RSA_94( e: 7,  p: 131,  q: 17, stringToSlice_94( str: "RSA"))
}

func stringToSlice_94(str string) int { //如果输入是字符串
    var slice_ []int
    for index, _ := range str { //将字符串输入转换成int存储再切片中
        slice_ = append(slice_, int(str[index]))
    }
    string_ := ""
    for index, _ := range str { //将切片中的数值连接
        string_ += strconv.Itoa(slice_[index])
    }
    a, _ := strconv.Atoi(string_) //将string转换成int
    return a % 1000               //因为字符串转换拼接出来的值很大，会超过int范围，因此做取模操作
}
```

```go
func main() {
    RSA_94( e: 7,  p: 131,  q: 17, stringToSlice_94( str: "RSA"))
}
```

已经转换为int

```
ringToSlice_94(str string) int
n:  go build cypto_lab_02.go
C:\Users\ChenQin\AppData\Local\Temp\GoLand\__go_build_cypto_lab_02_go.exe
用户输入的值为：e=7, p=131, q=17, m=365
(1)n=2227, n的欧拉函数值=2080
(2)私钥d= 1783
  【模重复平方法过程】[i=2,中间值=365][i=1,中间值=580][i=0,中间值=355]
(3)加密结果(密文c)= 355
  【模重复平方法过程】[i=10,中间值=355][i=9,中间值=672][i=8,中间值=1725][i=7,中
间值=1725][i=6,中间值=1028][i=5,中间值=331][i=4,中间值=2184][i=3,中间值=1776][i=
2,中间值=1776][i=1,中间值=1657][i=0,中间值=365]
(4)解密结果(明文m)= 365
```

五、实验中的问题及心得

本人使用 go 语言进行编程，根据 RSA 算法原理进行编写。大体分为两个部分：模重复平方法来解决中间值过大的问题，欧几里得和欧几里得扩展算法求私钥 d 。

模重复平方法：根据数学原理进行编写的，思路是将一个数值转为成二进制数，对这个二进制数的每一个不同的位进行不同的累乘，期间有取模操作，取模操作是解决中间值过大的核心步骤。欧几里得和欧几里得扩展算法求私钥：主要是用递归算法,每一步进行取模然后递归。要编写算法必须懂得相关的数学知识，比如：模算术、欧拉定理、线性同余方程等。对这些数学原理的理解花费了大量是时间，代码也是根据线性同余方程和欧几里得扩展算法相结合编写出来的。

本次实验，最大的问题是对数学原理的不熟，很难理解欧拉定理、线性同余方程等数学理论，说明了我数学基础不好，有待加强

收获：虽然相关的数学理论很难，但最终也是攻克了，收获不错。同时，锻炼了编写递归程序，模算法程序，进制、ASCII 码值转换，在输入的值过大或者值不符合题意的时候，懂得如何将输入修改转换成符合题意的值，等等。

以上

附件：程序代码

```
package main

import (
    "fmt"
    "strconv"
)

// 用于求私钥 d
var x int = 0 //调用 exGcd()之后，x 的值就是私钥的 d 的值
var y int = 0 //线性同余方程式 y 的值，没用，用来占位
```

```
func main() {
	RSA_94(7, 131, 17, stringToSlice_94("RSA"))
}

func stringToSlice_94(str string) int { //如果输入是字符串
	var slice_ []int
	for index, _ := range str { //将字符串输入转换成 int 存储再切片中
		slice_ = append(slice_, int(str[index]))
	}
	string_ := ""
	for index, _ := range str { //将切片中的数值连接
		string_ += strconv.Itoa(slice_[index])
	}
	a, _ := strconv.Atoi(string_) //将 string 转换成 int
	return a % 1000 //因为字符串转换拼接出来的值很大，会超过 int 范围，
因此做取模操作
}

// RSA 算法封装
func RSA_94(e int, p int, q int, m int) {
	n := p * q
	Fn := (p - 1) * (q - 1)
	fmt.Printf("用户输入的值为: e=%v, p=%v, q=%v, m=%v\n", e, p, q, m)
	fmt.Printf("(1)n=%v, n 的欧拉函数值=%v\n", n, Fn)
	exGcd_94(e, Fn)
	temp := modRepeateQuadratic_94(m, e, n)
	fmt.Printf("(3)加密结果(密文 c)= %v\n", temp)
	fmt.Printf("(4)解密结果(明文 m)= %v", modRepeateQuadratic_94(temp, x+Fn, n))
}

// 求私钥 d
// a：公钥 e。n： ϕ(n)
func privateKey_94(a int, n int) int {
	if n == 0 {
```

```
            x = 1
            y = 0
            return a
        } else if a == 0 {
            x = 0
            y = 1
            return n
        } else {
            c := privateKey_94(n, a%n)
            tmp := x
            x = y
            y = tmp - a/n*y
            return c
        }
}

// 求私钥 d
// a：公钥 e。n： Φ(n)
func exGcd_94(a int, n int) {
        privateKey_94(a, n)
        fmt.Printf("(2)私钥 d= %v\n", x+n)
}

// 模重复平方法
func modRepeateQuadratic_94(b int, n int, m int) int {
        s := 1                                          //用于累加累乘得到结果
        x := strconv.FormatInt(int64(n), 2) //将数 n 转换为二进制
        //从低位开始遍历（右边）
        fmt.Printf("  【模重复平方法过程】")
        for i := len(x) - 1; i >= 0; i-- {
            if int(x[i])-48 == 1 { //减 48：因为 ASCII 码
                s = (s * b) % m //累乘取模
            }
            b = (b * b) % m //将 b 指数+1
```

```
            fmt.Printf("[i=%v,中间值=%v]", i, s)
        }
        fmt.Println()
        return s
}
```