main.go

```go
package main

import (
    "bytes"
    "crypto/rand"
    "encoding/gob"
    "fmt"
    "math/big"
    mathRand "math/rand"
    "time"
)

// getRandomPrime 用于获取一个随机的质数
func getRandomPrime() *big.Int {
    for {
        tmpN, errT := rand.Prime(rand.Reader, 10)
        if errT != nil {
            // 有错误发生则继续循环，直至正确生成一个质数为止
            continue
        }
        return tmpN
    }
}

// calKeys 用于生成一组公钥、私钥
func calKeys() (pubKey *big.Int, privateKey *big.Int, modTotal *big.Int) {
    // 令 p 为一个随机的质数
    p := getRandomPrime()
    // 令 q 为一个不等于 p 的随机质数
    var q *big.Int
    for {
        q = getRandomPrime()
        if q.Cmp(p) != 0 {
            break
        }
    }

    fmt.Printf("p: %v, q: %v\n", p, q)
    // 令 n 为 p 和 q 的乘积（公倍数）
    n := new(big.Int).Mul(p, q)
    fmt.Printf("n（模数）: %v\n", n)
    // bigOneT 相当于一个常数 1，是 *big.Int 类型的
    bigOneT := big.NewInt(1)
    // 令 m = (p - 1) * (q - 1)
    m := new(big.Int).Mul(new(big.Int).Sub(p, bigOneT), new(big.Int).Sub(q, bigOneT))
    fmt.Printf("m: %v\n", m)
    // 从3开始循环选择一个符合条件的公钥 e
    e := big.NewInt(3)
    for {
        // 每次不符合条件时，e = e + 1;
        // 实际上，e = e + 2 会更快，因为偶数更可能会有公约数
        e.Add(e, bigOneT)
        // 获取 e 与 (p - 1) 的最大公约数
```

```go
        diff1 := new(big.Int).GCD(nil, nil, e, new(big.Int).Sub(p, bigOneT))
        // 获取 e 与 (q - 1) 的最大公约数
        diff2 := new(big.Int).GCD(nil, nil, e, new(big.Int).Sub(q, bigOneT))
        // 获取 e 与 m 的最大公约数
        diff3 := new(big.Int).GCD(nil, nil, e, m)
        // 选择合适的 e 的条件是，e 与 (p - 1)、(q - 1)、m 必须都分别互为质数
        // 也就是最大公约数为 1
        if diff1.Cmp(bigOneT) == 0 && diff2.Cmp(bigOneT) == 0 && diff3.Cmp(bigOneT) == 0 {
            break
        }
    }
    fmt.Printf("e（公钥）: %v\n", e)
    // 计算私钥
    d := new(big.Int).ModInverse(e, m)
    fmt.Printf("d（私钥）: %v\n", d)
    return e, d, n

}

func main() {
    // 初始化随机数种子
    mathRand.Seed(time.Now().Unix())
    // 获取公钥（pubKeyT）、私钥（privateKeyT）和共用的模数（modTotalT）
    // modTotalT 可以公开
    // 也可以将pubKeyT和modTotalT合起来看做公钥
    // 将privateKeyT和modTotalT合起来看做私钥
    pubKeyT, privateKeyT, modTotalT := calKeys()
    // 未加密的文本
    originalText := "我们都很nice。"
    fmt.Printf("原文：%#v\n", originalText)
    // 下面开始加密的过程
    // 用于存放密文的大整数切片
    cypherSliceT := make([]big.Int, 0)
    // 注意用 range 遍历 string 时，其中的 v 都是 rune 类型
    for _, v := range originalText {
        // 每个 Unicode 字符将作为数值用公钥和模数进行加密
        cypherSliceT = append(cypherSliceT, *new(big.Int).Exp(big.NewInt(int64(v)), pubKeyT, modTotalT))
    }
    var cypherBufT bytes.Buffer
    // 用gob包将密文大整数切片转换为字节切片以便传输或保存
    gob.NewEncoder(&cypherBufT).Encode(cypherSliceT)
    cypherBytesT := cypherBufT.Bytes()
    fmt.Printf("密文数据：%#v\n", cypherBytesT)
    // 下面开始解密的过程
    // 获得加密后的密文字节切片
    decryptBufT := bytes.NewBuffer(cypherBytesT)
    var decryptSliceT []big.Int
    // 用gob包将密文字节切片转换为对应的密文大整数切片
    gob.NewDecoder(decryptBufT).Decode(&decryptSliceT)
    // 为了演示，将分别用私钥和公钥来解密
    // decryptRunes1T用于存放用私钥解密的结果
    // decryptRunes2T用于存放用公钥解密的结果
    decryptRunes1T := make([]rune, 0)
```

main.go

```
107     decryptRunes2T := make([]rune, 0)
108     // 循环对每个大整数进行解密
109     for _, v := range decryptSliceT {
110         // 注意解密后的大整数要转换回 rune 格式才符合要求
111         decryptRunes1T = append(decryptRunes1T, rune((*(new(big.Int))).Exp(&v, privateKeyT, modTotalT).Int64()))
112         decryptRunes2T = append(decryptRunes2T, rune((*(new(big.Int))).Exp(&v, pubKeyT, modTotalT).Int64()))
113     }
114     decryptText1T := string(decryptRunes1T)
115     fmt.Printf("用私钥解密后还原的文本：%#v\n", decryptText1T)
116     decryptText2T := string(decryptRunes2T)
117     fmt.Printf("用公钥解密后还原的文本：%#v\n", decryptText2T)
118 }
119
```