**深 圳 大 学 实 验 报 告**

**课程名称：­ 计算机安全导论**

**实验项目名称­： 非对称密码实验**

**学院： 计算机与软件学院**

**专业： 软件工程（腾班）**

**指导教师： 陈飞**

**报告人： 黄亮铭 学号： 2022155028 班级： 腾班**

**实验时间：** **2024年09月27日**

**实验报告提交时间： 2024年10月09日**

**教务部制**

|  |
| --- |
| 实验目的与要求：   1. 理解非对称密码的基本思想 2. 掌握非对称密码的数学原理及快速实现算法 3. 掌握非对称密码的加解密过程和实现方法 |
| 实验环境：   * 操作系统：OSX * 编程语言：Goland * 开发环境：go1.22 |
| 实验原理：  RSA算法的工作原理如下：   1. **选择两个大质数** *p*和*q*，这两个质数的选择是随机的，但必须保密。 2. **计算乘积** *n*=*p*×*q*，这个乘积 *n* 将作为公钥和私钥的一部分。*n* 的长度（以位为单位）通常决定了密钥的长度。 3. **计算欧拉函数***ϕ*(*n*)=(*p*−1)×(*q*−1)，这个值用于选择公钥的指数。 4. **选择公钥指数** *e*，它是一个与*ϕ*(*n*) 互质的正整数。 5. **计算私钥指数** *d*，它是 *e* 关于*ϕ*(*n*) 的模逆元，即满足*e*×*d*≡1mod*ϕ*(*n*) 的数。 6. **生成公钥和私钥**：公钥是(*n*,*e*)，私钥是(*n*,*d*)。 7. **加密**：如果有人想加密消息 *M*，他们会使用接收者的公钥(*n*,*e*) 来计算密文 *C*，计算方式为*C*=mod*n*。 8. **解密**：接收者使用自己的私钥(*n*,*d*) 来解密密文*C*，计算方式为 *M*=mod*n*。 |
| 实验内容：   1. 快速模运算算法 2. RSA加密、解密算法 3. 算法性能测试   实验步骤与结果：  （依照实验内容，逐条撰写实验过程与实验所得结果：包括程序总体设计，核心数据结构及算法流程，调试过程。请附上核心代码，及注意格式排版的美观。实验提交时，以上为评分依据，请不删除本行）  快速模运算算法  **算法原理：**模幂运算是指计算mod*n*，其中*a*、*b*和*n* 是大整数。快速幂算法通过将指数*b*表示为二进制形式，并利用幂的性质来减少乘法的次数。  **算法步骤：**将指数 *b* 转换为二进制形式，然后从左到右遍历 *b* 的二进制表示，对于每一位：**1）**如果当前位是1，则将当前结果乘以*a* 并取模 *n*。**2）**不管当前位是什么，都将 *a* 平方并取模 *n*。继续这个过程直到遍历完 *b* 的所有二进制位。  **算法实现：**算法实现的代码如下图所示。   |  | | --- | | func quickPow(a, b, mod int64) int64 {  var res int64 = 1  for b != 0 {  if b&1 == 1 {  res = (res \* a) % mod  }  a = (a \* a) % mod  b = b >> 1  }  return res } |   图1:快速模运算代码实现  RSA加密、解密算法   1. **选择两个大质数** *p*和*q*，这两个质数的选择是随机的，但必须保密。这里我先预处理出10000000以内的质数，然后使用随机种子获得随机数作为下标，选出两个质数。实现的代码如下图所示。  |  | | --- | | func getPrimes() {  for i := int64(2); i < N; i++ {  if st[i] == false {  primes[cnt] = i  cnt++  }  for j := int64(0); primes[j]\*i < N; j++ {  st[primes[j]\*i] = true  if i%primes[j] == 0 {  break  }  }  } } |   图2:预处理质数   1. **计算乘积** *n*=*p*×*q*，这个乘积 *n* 将作为公钥和私钥的一部分。*n* 的长度（以位为单位）通常决定了密钥的长度。 2. **计算欧拉函数***ϕ*(*n*)=(*p*−1)×(*q*−1)，这个值用于选择公钥的指数。 3. **选择公钥指数** *e*，它是一个与*ϕ*(*n*) 互质的正整数。这里我从1中预处理得到的质数中选择任意一个（除了*p*和*q*）。  |  | | --- | | for {  idx := rand.Int64N(cnt)  if primes[idx] < phi\_n && gcd(primes[idx], phi\_n) == 1 {  e = primes[idx]  break  } } |   图3:选择公钥e   1. **计算私钥指数*d***，它是 *e* 关于*ϕ*(*n*) 的模逆元，即满足*e*×*d*≡1mod*ϕ*(*n*) 的数。这里我使用扩展欧几里得算法实现求逆。扩展欧几里得算法的实现步骤：如果*b*=0，则*a*是*a*和*b*的最大公约数，并且*x*=1,*y*=0。否则，递归地计算*b*和*a*mod*b*的最大公约数。使用递归结果来更新*x*和*y*的值。具体代码实现如下图所示。  |  | | --- | | func gcd(a, b int64) int64 {  if b == 0 {  return a  }  return gcd(b, a%b) } func exgcd(a, b int64, x, y \*int64) int64 {  if b == 0 {  (\*x), (\*y) = 1, 0  return a  }  d := exgcd(b, a%b, y, x)  (\*y) = (\*y) - a/b\*(\*x)  return d } func inv(a, mod int64) int64 {  var x, y int64  exgcd(a, mod, &x, &y)  return (x%mod + mod) % mod } |   图4:求逆代码实现  最后函数返回(x%mod + mod) % mod的原因是保证求得的结果为正数。   1. **生成公钥和私钥**：公钥是(*n*,*e*)，私钥是(*n*,*d*)。 2. **加密**：如果有人想加密消息 *M*，他们会使用接收者的公钥(*n*,*e*) 来计算密文 *C*，计算方式为*C*=mod*n*。具体代码实现如下图。  |  | | --- | | for i := int64(0); i < len1; i++ {  C[i] = quickPow(int64(inputBuffer[i]), e, n) } |   图5:加密代码实现   1. **解密**：接收者使用自己的私钥(*n*,*d*) 来解密密文*C*，计算方式为 *M*=mod*n*。具体代码实现如下图。  |  | | --- | | for i := int64(0); i < len1; i++ {  outputBuffer[i] = byte(quickPow(C[i], d, n)) } |   图6:解密代码实现   1. **其他：输入和输出的处理，具体代码如下图所示。**  |  | | --- | | func intputText(name string) {  file, \_ := os.Open(name)  defer file.Close()  reader := bufio.NewReader(file)  buffer, \_ := io.ReadAll(reader)  copy(inputBuffer, buffer)  len1 = int64(len(inputBuffer)) } func outputText(uri string) {  file, \_ := os.Create(uri)  defer file.Close()  writer := bufio.NewWriter(file)  \_, \_ = writer.Write(outputBuffer)  writer.Flush() } |   图7:输入输出处理代码实现   1. **结果展示：**从结果可以看出，加密前和解密后的数据完全相同，并且密文和原始数据不同。由此可以证明算法的正确性。     图8:密钥相关参数    图9:原始的小数据      图10: 小数据生成的密文    图11:解密后的小数据  场景应用展示  假设我们有一个名为*第7讲 公钥密码.pptx*的文件需要进行加密，我们尝试使用上述算法对这个文件进行加密。  通过上述RSA加密算法加密后的密文如图13所示。直接打开文件失败（图12），提示文件损坏。使用vscode打开可以发现，密文是一串二进制的序列（图13，乱码是因为vscode使用utf-8进行编码）。通过网络传输，即使被他人截取，别人也很难在没有私钥的情况下对密文进行解密。    图12:文件损坏    图13:密文（以utf-8编码展示）  使用私钥对密文解密后得到的文件可以打开，并且经过对比，发现文件被成功复原，说明算法的正确性。    图14:解密后文件  算法性能测试  算法性能测试主要从三部分进行，分别为生成公钥和私钥的时间、加密的速度以及解密的速度。具体实现代码如下图所示。   |  |  | | --- | --- | | beT := time.Now().UnixNano() getPrimes() idx1, idx2 := rand.Int64N(cnt), rand.Int64N(cnt) p, q = primes[idx1], primes[idx2] n = p \* q phi\_n = (p - 1) \* (q - 1) for {  idx := rand.Int64N(cnt)  if primes[idx] < phi\_n && gcd(primes[idx], phi\_n) == 1 {  e = primes[idx]  break  } } d = inv(e, phi\_n) edT := time.Now().UnixNano() t1 = edT - beT | now := time.Now() beT := now.UnixNano() for i := int64(0); i < len1; i++ {  C[i] = quickPow(int64(inputBuffer[i]), e, n) } now = time.Now() edT := now.UnixNano() t2 = edT - beT | | now = time.Now() beT = now.UnixNano() for i := int64(0); i < len1; i++ {  outputBuffer[i] = byte(quickPow(C[i], d, n)) } now = time.Now() edT = time.Now().UnixNano() t3 = edT - beT | fmt.Println("数据的总大小为", len1, "字节") fmt.Println("生成密钥对的时间为：", float64(t1)/1e6, "ms") fmt.Println("加密时间为：", float64(t2)/1e6, "ms") fmt.Println("解密时间为：", float64(t3)/1e6, "ms") fmt.Printf("生成密钥对的时间为：%f 毫秒 加密速度：%.2f 字节/毫秒 解密速度：%.2f 字节/毫秒", float64(t1)/1e6,  float64(len1)/(float64(t2)/1e6), float64(len1)/(float64(t3)/1e6)) | |
| 图15:测试代码以及结果展示代码  （左上生成密钥，右上加密，左下解密，右下结果展示）  小数据测试结果如下图所示。由测试结果可以看出，rsa算法还是相当高效的。    图16:小数据测试结果  对场景应用中的文件的测试结果如下图所示。    图17:文件测试结果 |
| 实验结论：  （撰写实验收获及思考）   1. 通过本次实验，我认识了RSA算法。RSA是一种广泛使用的非对称加密算法，它基于数论中的大整数分解问题。RSA算法的安全性依赖于大整数分解的困难性，即给定两个大质数的乘积，很难在合理时间内分解出这两个质数。RSA算法广泛应用于数据传输加密、数字签名、安全认证等领域。 2. 通过本次非对称密码实验，我深入理解了RSA算法的基本思想及其数学原理，并掌握了其加解密过程和实现方法。 3. 实验结果显示，RSA算法在生成密钥、加密和解密过程中都表现出了较高的效率。尽管非对称加密算法在处理速度上通常不如对称加密算法，但通过优化模运算和选择合适的参数，RSA算法仍然可以在保证安全性的同时提供可接受的性能。 4. 通过本次实验，我不仅掌握了RSA算法的实现，还学会了如何分析和优化算法性能。 |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。