**实 验 报 告**



|  |  |
| --- | --- |
| **课程名称** | **密码学基础** |
| **学 院** | **计算机科学技术学院** |
| **专 业** | **信息安全** |
| **姓 名** | **冉津豪** |
| **学 号** | **17307130179** |

**开 课 时 间 2019 至 2020 学年第 二 学期**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | 实验项目  名 称 | **RSA** | 成绩 |  |   **一、实验目的**   1. 实现RSA加密 2. 理解分解模数破解RSA的过程 3. 理解公共模数攻击   **二、实验内容**   1. RSA加密  * RSA产生公私钥对   • 1.随机选择两个不相等的质数p和q。  • 2.计算p和q的乘积n。  • 3.计算n的欧拉函数φ(n)。称作L，根据公式φ(n) = (p-1)(q-1)  • 4.随机选择一个整数e，也就是公钥当中用来加密的那个数字，条件是1< e < φ(n)，且e与φ(n) 互质。  • 5.计算e对于φ(n)的模反元素d。也就是密钥当中用来解密的那个数字。所谓"模反元素"就是指有一个整数d，可以使得ed被φ(n)除的余数为1。ed ≡ 1 (mod φ(n))  • 6.将n和e封装成公钥，n和d封装成私钥。   * RSA加解密   首先对明文进行比特串分组，使得每个分组对应的十进制数小于n，然后依次对每个分组m做一次加密，所有分组的密文构成的序列就是原始消息的加密结果，即m满足0<=m<n，加密算法为：c≡ m^e mod n; c为密文，且0<=c<n。   1. 分解模数破解RSA   如果模数n可以被分解成p和q，那么就可以通过e与密钥生成相同的过程对密文进行解密。   1. 公共模数攻击   当使用公共的模数n,不同的私钥e1, e2对同一密文进行加密时,如果能截获密文c1, c2那么可能可以直接解密。  若gcd(e1, e2) = 1, 即e1 e2互素时，由扩展欧几里德算法可知：必然存在整数s1, s2。使得下式①成立：e1s1 + e2s2 = 1, e1, e2都为正数那么，s1, s2是一正一负的，这里可以假设s1是负数。若记明文为m, e1,e2共用模数加密后的密文为c1, c2。即c1 = m^e1, c2 = m^e2那么可以根据①构造下式(运算皆mod n)：c1^s1 \* c2^s2 = (m^e1)^s1 \* (m^e2)^s2 = m^(e1s1 + e2s2) = m^1 = m  **三、实验步骤**  **1. 实现RSA算法**   * multiplicative\_inverse：   通过e和φ(n)，计算d使得ed ≡ 1 (mod φ(n))。ext\_gcd(a, b)利用扩展欧几里得算法求最大公约数的过程，返回值x, y分别表示 ax + by = 1的解。   |  |  | | --- | --- | |  | def ext\_gcd(a, b): | |  | if b == 0: | |  | return 1, 0 | |  | else: | |  | x, y = ext\_gcd(b, a % b) | |  | x, y = y, (x - (a // b) \* y) | |  | return x, y | |  |  | |  | def multiplicative\_inverse(e, phi): | |  | ''' | |  | extended Euclid's algorithm for finding the multiplicative inverse | |  | ''' | |  | # WRITE YOUR CODE HERE! | |  | x, y = ext\_gcd(phi, e) | |  | if y < 0: | |  | return y + phi | |  | return y |  * key\_generation   用两个质数p, q产生密钥。其中，e为随机选择一个小于并与φ(n)互质的正整数。有了e, n，再用multiplicative\_inverse方法求得d。   |  |  | | --- | --- | |  | def selectE(euler\_totient): | |  | while True: | |  | e = random.randint(0, euler\_totient) | |  | if math.gcd(e, euler\_totient) == 1: | |  | return e | |  |  | |  | def key\_generation(p, q): | |  | # WRITE YOUR CODE HERE! | |  | n = p \* q | |  | phi = (p - 1) \* (q - 1) | |  | e = selectE(phi) | |  | d = multiplicative\_inverse(e, phi) | |  | return n, e, d |  * Encrypt   快速幂取模算法完成plaintext ^ e (mod n)。   |  |  | | --- | --- | |  | def encrypt(pk, plaintext): | |  | # WRITE YOUR CODE HERE! | |  | n, e = pk | |  | res = 1 | |  | tmp = plaintext | |  | while e: | |  | if e & 0x1: | |  | res = res \* tmp % n | |  | tmp = tmp \* tmp % n | |  | e >>= 1 | |  | return res |  * Decrypt   快速幂取模算法完成ciphertext ^ d (mod n)。   |  |  | | --- | --- | |  | def decrypt(sk, ciphertext): | |  | # WRITE YOUR CODE HERE! | |  | n, d = sk | |  | res = 1 | |  | tmp = ciphertext | |  | while d: | |  | if d & 0x1: | |  | res = res \* tmp % n | |  | tmp = tmp \* tmp % n | |  | d >>= 1 | |  | return res |   **2. 分解模数破解RSA**  由公钥文件pubkey.pem获公钥参数n, e。在[http://www.factordb.com/index.php](http://www.factordb.com/index.php得)分解n得到因子p和q, 即得到φ(n)再由e, φ(n)得到私钥参数d。同时，我们在这里验证了我们的RSA加解密过程，及multiplicative\_inverse的正确性，解密密文可得：   |  |  | | --- | --- | |  | if \_\_name\_\_ == '\_\_main\_\_': | |  | n = "0xC2636AE5C3D8E43FFB97AB09028F1AAC6C0BF6CD3D70EBCA281BFFE97FBE30DD" | |  | n = int(n, 16) | |  | e = 65537 | |  | p, q = 275127860351348928173285174381581152299, 319576316814478949870590164193048041239 | |  | phi\_n = (p - 1) \* (q - 1) | |  | d = multiplicative\_inverse(e, phi\_n) | |  | fi = open('secret.enc', 'rb') | |  | cipher = fi.read() | |  | cipher = bytes2num(cipher) | |  | fi.close() | |  | std\_plaintext = pow(cipher, d, n) | |  | plaintext = decrypt((n, d), cipher) | |  | if std\_plaintext == plaintext: | |  | print("解密正确！") | |  | cipher2 = encrypt((n, e), plaintext) | |  | if cipher == cipher2: | |  | print("加密正确！") | |  | print(num2str(plaintext)) |   **3. 公共模数攻击**  根据模数攻击原理，利用扩展欧几里得算法完成攻击。   |  |  | | --- | --- | |  | def ext\_gcd(a, b): | |  | if b == 0: | |  | return 1, 0 | |  | else: | |  | x, y = ext\_gcd(b, a % b) | |  | x, y = y, (x - (a // b) \* y) | |  | return x, y | |  |  | |  | def modinv(a, m): | |  | x, y = ext\_gcd(a, m) | |  | return x % m | |  |  | |  | def attack(c1, c2, e1, e2, n): | |  | # WRITE YOUR CODE HERE! | |  | s1, s2 = ext\_gcd(e1, e2) | |  | if s1 < 0: | |  | s1 = - s1 | |  | c1 = modinv(c1, n) | |  | elif s2 < 0: | |  | s2 = - s2 | |  | c2 = modinv(c2, n) | |  | m = (pow(c1, s1, n) \* pow(c2, s2, n)) % n | |  | return m | |  |  | |  | if \_\_name\_\_ == '\_\_main\_\_': | |  | n = 103109065902334620226101162008793963504256027939117020091876799039690801944735604259018655534860183205031069083254290258577291605287053538752280231959857465853228851714786887294961873006234153079187216285516823832102424110934062954272346111907571393964363630079343598511602013316604641904852018969178919051627 | |  | ct1 = 89165267757592056998430515273393236824675618209628120636976415324541608777639312906328761896196335563918995813065446097111511299622583559521279416118344262659587873497494326049565077208391797840846364711580568322182209616487454086620832166912794622826357529165185122310658553888175629604486544265939860652568 | |  | e1 = 15 | |  | ct2 = 20135243573102312365489445555179471711050237357771531944565197207909056856791452660071533981351813615406042686828666883602743565692680466225128878907776339638498888836642920122809284690243270675115097201512434337054133554094147631311406424433602757215625092846913693790078424458624369228575776016743532946934 | |  | e2 = 13 | |  |  | |  | print('[+] Started attack...') | |  | message = attack(ct1, ct2, e1, e2, n) | |  | print('[+] Attack finished!') | |  | print('\nPlaintext: \n ' + bytearray.fromhex(hex(message)[2:]).decode()) |   **四、实验结果**  分解模数方法解密结果为：vFQQv ManyQuestionMarks???  公共模数攻击结果为：When does school start  根据输出结果，我们可以判断过程无误。  **五、实验总结**  分解模数方法虽然简单粗暴，但计算时间过长。当我们使用分解模数攻击的方法再对第三部分的n进行尝试时，因为计算时间过长，长时间无法得到响应。  公共模数攻击需要得到相同的模数n，相同的消息m，不同e1，e2加密的密文c1, c2，条件相比于分解模数方法更加苛刻，且仅仅只能解决这个特定的m，有较大局限性。但对于分解模数方法束手无策的长密钥，仍然完成了攻击。  对于这两种攻击方法，将RSA的密钥模数的长度提高，随机化明文m便能极大地完成防御。总体而言，RSA的安全性是能够得到保障的。 |