RSA算法程序设计报告

1. 原理描述

RSA算法是一种常用的非对称加密算法。

RSA算法主要包括三个过程：

* 生成密钥对
* 使用公钥加密明文
* 使用私钥解密密文

解码原理：

* 欧拉定理 对互素的a和m，有
* 推论 给定满足N= pq的两个不同素数p和q以及满足0<n<N的 整数n，k 是正整数，有
* 由于，即，故

，即.

现在我们有

应用模算术运算规则得到

即为.

1. 数据结构设计

使用一个结构体存储密钥（公私钥对）

1. **typedef** **struct** {
2. **int** e;
3. **char**\* d;
4. **char**\* n;
5. }key\_value;

明文/密文是字节串，用char数组存储

用于加密解密运算的大整数作为字符串用char数组存储，数学运算在16进制下进行

1. 密钥生成

RSA 算法中p和q的选择流程

1. 确定RSA所要求N的位数k。k =1024、2048、3072、4096 ...
2. 随机选择一个位数为的素数p。
3. 选择一个位数为的素数q
4. 求；如果 过小，则返回2重新选取p。
5. 计算N= pq，确认N的位数为k；否则返回2，重新选取p。
6. 计算N的NAF权重；如果权重过小，则返回2重新选取p。

PKCS#1建议选择素数e = 216+1 = 65537，即e = 0x10001

1. 计算，选择公钥指数e，且。  PKCS#1建议选择素数。
2. 求e的模逆元d作为私钥指数；如果d过小，则返回2重新选取p
3. p和q选择成功，销毁p、q，返回参数N、e、d

生成公私钥对

1. 选择两个不同的大素数p和q，计算N = pq。
2. 引用欧拉函数，小于N且与N互素的正整数个数为
3. 选择一个整数e， 且，应用中通常选取

e=65537 

1. 求一个满足下列同余式的d：

. 

* 1. d是e的模逆元，可以应用扩展欧几里德算法得到。 
  2. d必须是一个足够大的正整数。 

1. 将p和q的记录销毁(当N足够大时，寻找p, q将极其困难) 
2. 以(e, N) 作为公钥，(d, N) 作为私钥。
3. mpz\_t\* GET\_BIG\_PRIME\_N() {
4. //随机数生成配置
5. gmp\_randstate\_t grt;
6. gmp\_randinit\_default(grt);
7. gmp\_randseed\_ui(grt, time(NULL));
9. //初始化
10. mpz\_t key\_p, key\_q;
11. mpz\_init(key\_p);
12. mpz\_init(key\_q);
14. mpz\_rrandomb(key\_p, grt, N / 2);
15. mpz\_rrandomb(key\_q, grt, N / 2);    //随机生成两个大整数
17. mpz\_t\* result = (mpz\_t\*)malloc(**sizeof**(mpz\_t)\*2);
18. mpz\_init(result[0]);
19. mpz\_init(result[1]);
21. mpz\_nextprime(result[0], key\_p);  //使用GMP自带的素数生成函数
22. mpz\_nextprime(result[1], key\_q);
24. mpz\_clear(key\_p);
25. mpz\_clear(key\_q);
27. **return** result;
29. }
30. key\_value\* RSAES\_PKCS1\_V1\_5\_GNERATE\_KEY() {
31. mpz\_t\* primes = GET\_BIG\_PRIME\_N();
32. mpz\_t e, d, n, f;
33. mpz\_init\_set\_ui(e, 65537);
34. mpz\_init(d);
35. mpz\_init(n);
36. mpz\_init(f);
38. //确认n的位数为N
40. **do**{
41. mpz\_clear(primes[0]);
42. mpz\_clear(primes[1]);
43. free(primes);
44. primes = GET\_BIG\_PRIME\_N();
45. mpz\_mul(n, primes[0], primes[1]); //n=p\*q
46. //printf("%d\n", mpz\_sizeinbase(n, 2));
47. } **while** (mpz\_sizeinbase(n, 2) != N);

50. mpz\_sub\_ui(primes[0], primes[0], 1);       //p=p-1
51. mpz\_sub\_ui(primes[1], primes[1], 1);       //q=q-1
52. mpz\_mul(f, primes[0], primes[1]);      //计算欧拉函数φ(n)=(p-1)\*(q-1)
54. mpz\_invert(d, e, f); //求e在模φ(n)下的乘法逆元（也被称为数论倒数）d
56. **char**\* buf\_n = (**char**\*)malloc(**sizeof**(**char**) \* (N + 10));
57. **char**\* buf\_d = (**char**\*)malloc(**sizeof**(**char**) \* (N + 10));
58. mpz\_get\_str(buf\_n, BASE, n);
59. mpz\_get\_str(buf\_d, BASE, d);
61. key\_value\* result = (key\_value\*)malloc(**sizeof**(key\_value));
62. result->e = 65537;
63. result->d = buf\_d;
64. result->n = buf\_n;
66. mpz\_clear(primes[0]);   //释放内存
67. mpz\_clear(primes[1]);
68. mpz\_clear(n);
69. mpz\_clear(d);
70. mpz\_clear(e);
71. mpz\_clear(f);
72. free(primes);
74. **return** result;
75. }
76. 加密
77. 将编码后的明文EM（长度为k的字节串）转化为明文整数m

m= OS2IP (EM)

1. 利用公钥(e, N)，使用如下公式对明文整数m加密，得到密文整数c

c= RSAEP ((N, e), m)

1. 将密文整数c转化为密文C（长度为k的字节串）

C= I2OSP (c, k)

1. **char**\* RSAES\_PKCS1\_V1\_5\_ENCRYPT(**int** e, **const** **char**\* n, **char**\* M) {
2. //对明文M进行编码生成EM
3. **char** EM[k];
4. RSAES\_PKCS1\_V1\_5\_ENCODE(M, EM);
5. //将编码后的明文EM转化为明文整数m
6. **char**\* m = RSAES\_PKCS1\_OS2IP(EM);
8. mpz\_t num\_m, num\_c, num\_n;
9. mpz\_init\_set\_str(num\_m, m, BASE);
10. mpz\_init\_set\_str(num\_n, n, BASE);
11. mpz\_init(num\_c);
12. //使用GMP中模幂计算函数，利用公钥(e, N)，对明文整数m加密，得到密文整数c
13. mpz\_powm\_ui(num\_c, num\_m, e, num\_n);
14. //密文整数c
15. **char**\* c = (**char**\*)malloc(**sizeof**(**char**) \* (2 \* k + 10));
16. mpz\_get\_str(c, BASE, num\_c);
18. //将密文整数c转化为密文C
19. **char**\* C = RSAES\_PKCS1\_I2OSP(c);
21. mpz\_clear(num\_m);
22. mpz\_clear(num\_c);
23. mpz\_clear(num\_n);
24. **return** C;
25. }
26. 解密
27. 将密文C转化为密文整数c

c= OS2IP (C)

1. 利用私钥(d, N)，使用如下公式对密文整数c解密，得到明文整数m

m = RSADP ((N, d), c)

1. 将明文整数m转化为编码后的消息EM（长度为k的字节串）

EM= I2OSP (m, k)

1. **char**\* RSAES\_PKCS1\_V1\_5\_DECRYPT(**const** **char**\* d, **const** **char**\* n, **char**\* C) {
2. //将密文C转化为密文整数c
3. **char**\* c = RSAES\_PKCS1\_OS2IP(C);
4. mpz\_t num\_m, num\_c, num\_d, num\_n;
5. mpz\_init\_set\_str(num\_c, c, BASE);
6. mpz\_init\_set\_str(num\_d, d, BASE);
7. mpz\_init\_set\_str(num\_n, n, BASE);
8. mpz\_init(num\_m);
9. //使用GMP中模幂计算函数，利用私钥(d, N)，对密文整数c解密，得到明文整数m
10. mpz\_powm(num\_m, num\_c, num\_d, num\_n);
11. //明文整数m
12. **char**\* m = (**char**\*)malloc(**sizeof**(**char**) \* (2 \* k + 10));
13. mpz\_get\_str(m, BASE, num\_m);
15. **char**\* temp = (**char**\*)malloc(**sizeof**(**char**) \* (2 \* k + 10));
16. //将因计算丢失的高位0补上
17. temp[0]= temp[1]= temp[2] ='0' ;
18. temp[3] = '\0';
19. strcat(temp, m);
21. //将明文整数m转化为编码后的消息EM
22. **char**\* EM = RSAES\_PKCS1\_I2OSP(temp);
23. **char**\* M= (**char**\*)malloc(**sizeof**(**char**) \* (k + 10));
24. //解码
25. RSAES\_PKCS1\_V1\_5\_DECODE(EM, M);
27. mpz\_clear(num\_m);
28. mpz\_clear(num\_c);
29. mpz\_clear(num\_d);
30. mpz\_clear(num\_n);
31. **return** M;
32. }
33. 编解码

加密前需要对明文M进行编码得到长度为k字节的字节串EM，

解密后需要对得到的编码后的明文EM进行解码得到明文M

明文M：一串ascii码表示,存储在char数组中

* 编码（填充

使用EME-PKCS1-v1\_5编码

* 假设使用N位的RSA编码，令k=N/8（字节数），消息M为长度mlen的字节串，则一次可以编码的消息M其
* EM长度为k字节
* 其中，PS由伪随机生成的非零字节组成，至少8字节，长度为k-mLen-3

1. **void** RSAES\_PKCS1\_V1\_5\_ENCODE(**char**\* M, **char** EM[k]) {
2. **int** mlen = strlen(M);
3. **int** i;
4. EM[0] = 0;
5. EM[1] = 2;
6. srand((unsigned)time(NULL));
7. **for** (i = 2; i < k - mlen - 1; i++) {
8. EM[i] = (**char**)(rand() % 127 + 1);
9. }
10. EM[k - mlen -1] = 0;
12. **for** (i = 0; i < mlen; i++) {
13. EM[k - mlen + i] = M[i];
14. }
16. }

* 解码

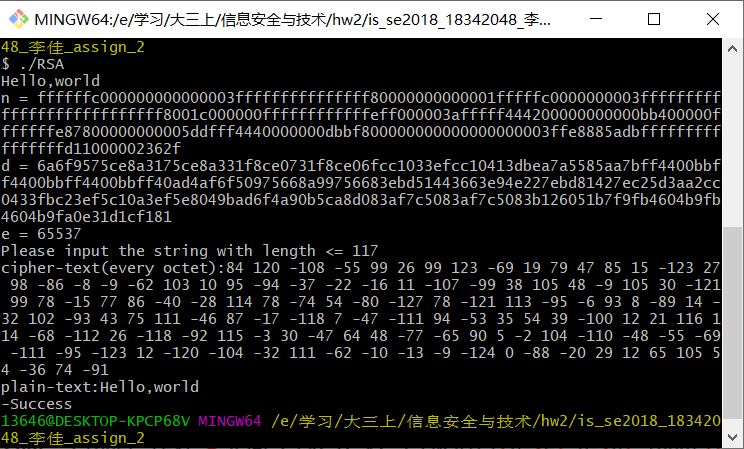
同样是依据这个式子，用第二个0字节将PS分离出来

1. **void** RSAES\_PKCS1\_V1\_5\_DECODE(**char** EM[k], **char**\* M) {
2. **int** i;
3. **int** start = 0;
4. **for** (i = 2; i < k; i++) {
5. **if** ((**int**)EM[i] == 0) {
6. start = i+1;
7. **break**;
8. }
9. }
10. **for** (i = start; i < k; i++) {
11. M[i - start] = EM[i];
12. }
13. M[i - start] = '\0';
14. }
15. 验证用例

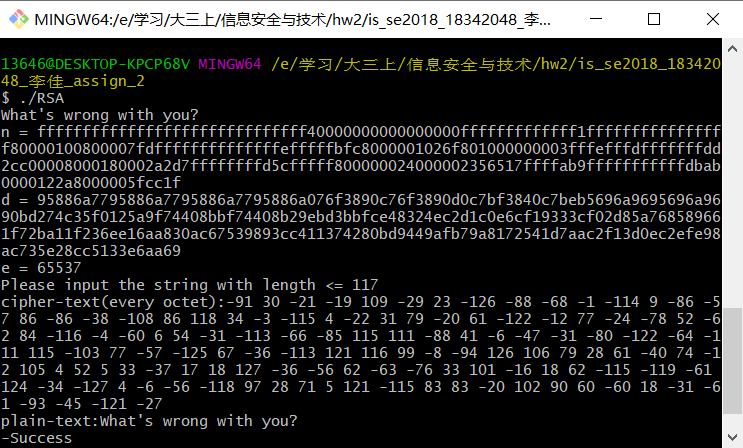
编译：gcc RSA.c -lgmp -o RSA

运行：./RSA

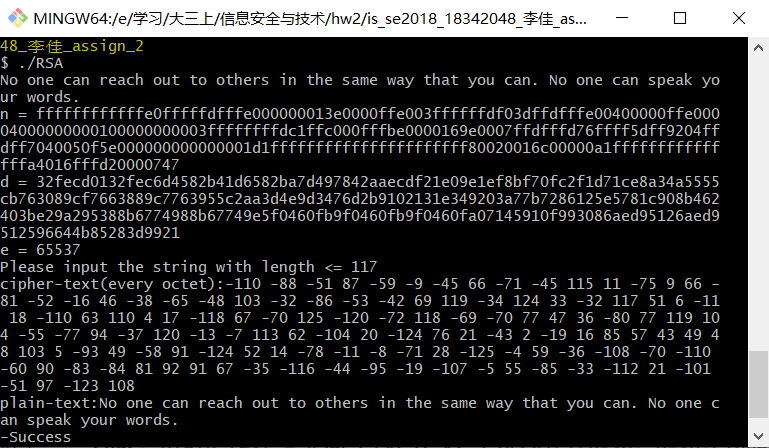
1. Hello,world



1. What’s wrong with you?



1. No one can reach out to others in the same way that you can. No one can speak your words.



1. 123456789abcdefgh

