哈尔滨工业大学

计算机科学与技术学院

《信息安全概论》

实验报告

计算机科学与技术学院

计算机系网络教研室制

|  |  |
| --- | --- |
| 课程名称： | 信息安全概论 |
| 实验名称： | RSA加密实验 |
| 指导教师： | 韩琦 |
| 学生姓名： | 陈三元 |
| 组 号： | 2-28 |
| 实验日期： | 18-03-10 |
| 实验地点： | 哈尔滨工业大学 |
| 实验成绩： |  |

一、实验目的

 通过用RSA算法对实际数据进行加密和解密来深刻了解RSA的运行原理及其特点，能够编译并分析RSA算法，进而加深对非对称加密算法的理解与认识。

二、实验环境

操作系统：运行Windows ，VS2010编译环境。

验证软件：CAP(Cryptographic Analysis Program v4)软件

三、实验内容与实验要求

任务一：RSA加解密算法的原理

首先产生密钥，过程如下：

      随机产生两个长度为K/2位的素数P 和 Q；

      计算公钥 publicKey=P\*Q;(publicKey 是k位的长度)

      随机产生一个加密密钥 keyE, 2<=keyE<=Φ(n)-1其中GCD(keyE, Φ(n))=1；注：这是保证解密密钥keyE \*keyD mod Φ(n)=1 有解的充要条件， Φ(n)称为n的欧拉函数,值为: Φ(n)=(P-1)\*(Q-1)

      求解解密密钥keyD=keyE-1 mod (n) ，keyE-1为解密密钥keyD的逆元 ，此公式原方程为(keyE\*keyD mod (n)=1)

      由此公钥，加密密钥，解密密钥全部产生。

      其次，对明文加密或对密文进行解密,过程如下：

      (1) 加密: C = MkeyE mod publicKey;其中M表示明文，C表示密文

      (2) 解密: M = CkeyD mod publicKey. 其中M表示明文，C表示密文

      任务：画出算法流程图。

任务二：使用C语言实现IDEA加密算法

编译运行下列代码，并对代码进行分析：

1. #include <iostream>
2. #include <stdlib.h>
3. #include <time.h>
5. **using** **namespace** std;//RSA算法所需参数
7. **typedef** **struct**  RSA\_PARAM\_Tag
8. {
9. unsigned **\_\_int64**    p, q;   //两个素数，不参与加密解密运算
10. unsigned **\_\_int64**    f;      //f=(p-1)\*(q-1)，不参与加密解密运算
11. unsigned **\_\_int64**    n, e;   //公匙，n=p\*q，gcd(e,f)=1
12. unsigned **\_\_int64**    d;      //私匙，e\*d=1 (mod f)，gcd(n,d)=1
13. unsigned **\_\_int64**    s;      //块长，满足^s<=n的最大的s，即log2(n)
14. } RSA\_PARAM;//小素数表
16. **const** **static** **long** g\_PrimeTable[]=
17. {3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97  };
18. **const** **static** **long**   g\_PrimeCount=**sizeof**(g\_PrimeTable) / **sizeof**(**long**);
19. **const** unsigned **\_\_int64**  multiplier=12747293821;
20. **const** unsigned **\_\_int64**  adder=1343545677842234541;//随机数类
22. **class**  RandNumber
23. {
24. **private**:
25. unsigned **\_\_int64**    randSeed;
26. **public**:
27. RandNumber(unsigned **\_\_int64** s=0);
28. unsigned **\_\_int64**    Random(unsigned **\_\_int64** n);
29. };
31. RandNumber::RandNumber(unsigned **\_\_int64** s)
32. {
33. **if**(!s){
34. randSeed= (unsigned **\_\_int64**)time(NULL);
35. }
36. **else**{
37. randSeed=s;
38. }
39. }
41. unsigned **\_\_int64** RandNumber::Random(unsigned **\_\_int64** n)
42. {
43. randSeed=multiplier \* randSeed + adder;
44. **return** randSeed % n;
45. }
47. **static** RandNumber g\_Rnd;/\* 模乘运算，返回值x=a\*b mod n  \*/
49. **inline** unsigned **\_\_int64** MulMod(unsigned **\_\_int64** a, unsigned **\_\_int64** b, unsigned **\_\_int64** n)
50. {
51. **return** a \* b % n;
52. }
53. /\* 模幂运算，返回值x=base^pow mod n \*/
55. unsigned **\_\_int64** PowMod(unsigned **\_\_int64** &base, unsigned **\_\_int64** &pow, unsigned **\_\_int64** &n)
56. {
57. unsigned **\_\_int64**    a=base, b=pow, c=1;
58. **while**(b){
59. **while**(!(b & 1)){
60. b>>=1;  //a=a \* a % n;    //函数看起来可以处理位的整数，但由于这里a\*a在a>=2^32时已经造成了溢出，因此实际处理范围没有位
61. a=MulMod(a, a, n);
62. }
63. b--;        //c=a \* c % n;        //这里也会溢出，若把位整数拆为两个位整数不知是否可以解决这个问题。
64. c=MulMod(a, c, n);
65. }
66. **return** c;
67. }
68. /\*  Rabin-Miller素数测试，通过测试返回，否则返回。n是待测素数。注意：通过测试并不一定就是素数，非素数通过测试的概率是/4  \*/
69. **long** RabinMillerKnl(unsigned **\_\_int64** &n)
70. {
71. unsigned **\_\_int64**    b, m, j, v, i;
72. m=n - 1;
73. j=0;
74. //0、先计算出m、j，使得n-1=m\*2^j，其中m是正奇数，j是非负整数
75. **while**(!(m & 1)){
76. ++j;
77. m>>=1;
78. }
79. //1、随机取一个b，<=b<n-1
80. b=2 + g\_Rnd.Random(n - 3);
81. //2、计算v=b^m mod n
82. v=PowMod(b, m, n);
83. //3、如果v==1，通过测试
84. **if**(v == 1){
85. **return** 1;
86. }
87. //4、令i=1
88. i=1;
89. b=2;
90. //5、如果v=n-1，通过测试
91. **while**(v != n - 1){
92. //6、如果i==l，非素数，结束
93. **if**(i == j){
94. **return** 0;
95. }
96. //7、v=v^2 mod n，i=i+1
97. v=PowMod(v, b, n);
98. ++i;
99. //8、循环到
100. }
101. **return** 1;
102. }
104. /\*   Rabin-Miller素数测试，循环调用核心loop次全部通过返回，否则返回   \*/
106. **long** RabinMiller(unsigned **\_\_int64** &n, **long** loop)
107. {
108. //先用小素数筛选一次，提高效率
109. **for**(**long** i=0; i < g\_PrimeCount; i++){
110. **if**(n % g\_PrimeTable[i] == 0){
111. **return** 0;
112. }
113. }
114. //循环调用Rabin-Miller测试loop次，使得非素数通过测试的概率降为(1/4)^loop
115. **for**(**long** i=0; i < loop; i++){
116. **if**(!RabinMillerKnl(n)){
117. **return** 0;
118. }
119. }
120. **return** 1;
121. }
122. /\* 随机生成一个bits位(二进制位)的素数，最多位\*/
123. unsigned **\_\_int64** RandomPrime(**char** bits)
124. {
125. unsigned **\_\_int64**    base;
126. **do**{
127. base= (unsigned **long**)1 << (bits - 1);   //保证最高位是
128. base+=g\_Rnd.Random(base);               //再加上一个随机数
129. base|=1;    //保证最低位是，即保证是奇数
130. }
131. **while**(!RabinMiller(base, 30));    //进行拉宾－米勒测试次
132. **return** base;    //全部通过认为是素数
133. }
134. /\*欧几里得法求最大公约数\*/
136. unsigned **\_\_int64** EuclidGcd(unsigned **\_\_int64** &p, unsigned **\_\_int64** &q)
137. {
138. unsigned **\_\_int64**    a=p > q ? p : q;
139. unsigned **\_\_int64**    b=p < q ? p : q;
140. unsigned **\_\_int64**    t;
141. **if**(p == q){
142. **return** p;   //两数相等，最大公约数就是本身
143. }
144. **else**{
145. **while**(b){    //辗转相除法，gcd(a,b)=gcd(b,a-qb)
146. a=a % b;
147. t=a;
148. a=b;
149. b=t;
150. }
151. **return** a;
152. }
153. }
154. /\*  Stein法求最大公约数 \*/
156. unsigned **\_\_int64** SteinGcd(unsigned **\_\_int64** &p, unsigned **\_\_int64** &q)
157. {
158. unsigned **\_\_int64**    a=p > q ? p : q;
159. unsigned **\_\_int64**    b=p < q ? p : q;
160. unsigned **\_\_int64**    t, r=1;
161. **if**(p == q){
162. **return** p;           //两数相等，最大公约数就是本身
163. }
164. **else**{
165. **while**((!(a & 1)) && (!(b & 1))){
166. r<<=1;          //a、b均为偶数时，gcd(a,b)=2\*gcd(a/2,b/2)
167. a>>=1;
168. b>>=1;
169. }
170. **if**(!(a & 1)){
171. t=a;            //如果a为偶数，交换a，b
172. a=b;
173. b=t;
174. }
175. **do**{
176. **while**(!(b & 1)){
177. b>>=1; //b为偶数，a为奇数时，gcd(b,a)=gcd(b/2,a)
178. }
179. **if**(b < a){
180. t=a;        //如果b小于a，交换a，b
181. a=b;
182. b=t;
183. }
184. b=(b - a) >> 1; //b、a都是奇数，gcd(b,a)=gcd((b-a)/2,a)
185. } **while**(b);
186. **return** r \* a;
187. }
188. }
189. /\*  已知a、b，求x，满足a\*x =1 (mod b)相当于求解a\*x-b\*y=1的最小整数解  \*/
190. unsigned **\_\_int64** Euclid(unsigned **\_\_int64** &a, unsigned **\_\_int64** &b)
191. {
192. unsigned **\_\_int64**    m, e, i, j, x, y;
193. **long** xx, yy;
194. m=b;
195. e=a;
196. x=0;
197. y=1;
198. xx=1;
199. yy=1;
200. **while**(e){
201. i=m / e;
202. j=m % e;
203. m=e;
204. e=j;
205. j=y;
206. y\*=i;
207. **if**(xx == yy){
208. **if**(x > y){
209. y=x - y;
210. }
211. **else**{
212. y-=x;    yy=0;
213. }
214. }
215. **else**{
216. y+=x;   xx=1 - xx;  yy=1 - yy;
217. }
218. x=j;
219. }
220. **if**(xx == 0){
221. x=b - x;
222. }
223. **return** x;
224. }
225. /\* 随机产生一个RSA加密参数\*/
226. RSA\_PARAM RsaGetParam(**void**)
227. {
228. RSA\_PARAM           Rsa={ 0 };
229. unsigned **\_\_int64**    t;
230. Rsa.p=RandomPrime(16);          //随机生成两个素数
231. Rsa.q=RandomPrime(16);
232. Rsa.n=Rsa.p \* Rsa.q;
233. Rsa.f=(Rsa.p - 1) \* (Rsa.q - 1);
234. **do**{
235. Rsa.e=g\_Rnd.Random(65536);  //小于^16，=2^16
236. Rsa.e|=1;   //保证最低位是，即保证是奇数，因f一定是偶数，要互素，只能是奇数
237. }**while**(SteinGcd(Rsa.e, Rsa.f) != 1);
238. Rsa.d=Euclid(Rsa.e, Rsa.f);
239. Rsa.s=0;   t=Rsa.n >> 1;
240. **while**(t){
241. Rsa.s++;   //s=log2(n)
242. t>>=1;
243. }
244. **return** Rsa;
245. }
246. /\* 拉宾－米勒测试 \*/
248. **void** TestRM(**void**)
249. {
250. unsigned **long**   k=0;
251. cout << " - Rabin-Miller prime check.\\\\n" << endl;
252. **for**(unsigned **\_\_int64** i=4197900001; i < 4198000000; i+=2){
253. **if**(RabinMiller(i, 30)){
254. k++;
255. cout << i << endl;
256. }
257. }
258. cout << "Total: " << k << endl;
259. }
260. /\*  RSA加密解密 \*/
262. **void** TestRSA(**void**)
263. {
264. RSA\_PARAM   r;
265. **char**     pSrc[]="wellcome to bjhit!";
266. **const** unsigned **long** n=**sizeof**(pSrc);
267. unsigned **char**       \*q, pDec[n];
268. unsigned **\_\_int64**    pEnc[n];
269. r=RsaGetParam();
270. cout << "p=" << r.p << endl;
271. cout << "q=" << r.q << endl;
272. cout << "f=(p-1)\*(q-1)=" << r.f << endl;
273. cout << "n=p\*q=" << r.n << endl;
274. cout << "e=" << r.e << endl;
275. cout << "d=" << r.d << endl;
276. cout << "s=" << r.s << endl;
277. cout << "Source:" << pSrc << endl;
278. q= (unsigned **char** \*)pSrc;
279. cout << "Encode:";
280. **for**(unsigned **long** i=0; i < n; i++){
281. unsigned **\_\_int64** temp = (unsigned **\_\_int64**)q[i];
282. pEnc[i]=PowMod(temp, r.e, r.n);
283. cout << hex << pEnc[i] << " ";
284. }
285. cout << endl;
286. cout << "Decode:";
287. **for**(unsigned **long** i=0; i < n; i++){
288. pDec[i]=PowMod(pEnc[i], r.d, r.n);
289. cout << hex << (unsigned **long**)pDec[i] << " ";
290. }
291. cout << endl;
292. cout << (**char** \*)pDec << endl;
293. }
295. **int** main(**void**)
296. {
297. TestRSA();
298. **return** 0;
299. }

四、实验过程与分析

RSA加密算法流程图

随机生成两个素数p, q

公钥n = p \* q

欧拉函数f = (p-1)\*(q-1)

计算得出解密密钥d，满足e\*d % f ==1

随机生成加密密钥e，满足2<=e<=f-1和gcd(e,f)==1

明文M

密文C

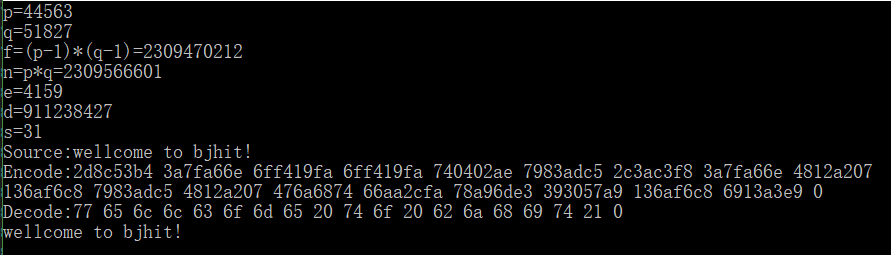
M = C ^ d % n 解密密文

C = M ^ e % n 加密明文

密文C

明文M

RSA加密算法代码运行结果



1. 随机生成两个16位的素数p和q
2. 计算得出32位的公钥n以及n的欧拉函数f
3. 随机产生加密密钥e，满足2<=e<=f-1和gcd(e,f)==1
4. 计算得出解密密钥d，满足e\*d % f ==1
5. 计算s=log2(n)
6. 明文为 wellcome to bjhit!
7. 用C = M ^ e % n 加密明文得到以上图中所示明文
8. 用M = C ^ d % n 解密密文得到以上图中所示明文，也就是原文

RSA加密算法部分代码分析

1. /\*   Rabin-Miller素数测试，循环调用核心loop次全部通过返回，否则返回   \*/
2. **long** RabinMiller(unsigned **\_\_int64** &n, **long** loop)
3. {
4. //先用小素数筛选一次，提高效率
5. **for**(**long** i=0; i < g\_PrimeCount; i++){
6. **if**(n % g\_PrimeTable[i] == 0){
7. **return** 0;
8. }
9. }
10. //循环调用Rabin-Miller测试loop次，使得非素数通过测试的概率降为(1/4)^loop
11. **for**(**long** i=0; i < loop; i++){
12. **if**(!RabinMillerKnl(n)){
13. **return** 0;
14. }
15. }
16. **return** 1;
17. }

Ranbin-Miller方法进行素数判断，首先利用已建立的小素数数组进行筛选，如果能够整除，则说明n不是素数，即可以直接返回0；如果都不能整除，再进行loop次的Ranbin-Miller测试，确保非素数被误认为是素数的概率降低到(1/4)^loop，再返回1

1. /\* 随机生成一个bits位(二进制位)的素数，最多位\*/
2. unsigned **\_\_int64** RandomPrime(**char** bits)
3. {
4. unsigned **\_\_int64**    base;
5. **do**{
6. base= (unsigned **long**)1 << (bits - 1);   //保证最高位是
7. base+=g\_Rnd.Random(base);               //再加上一个随机数
8. base|=1;    //保证最低位是，即保证是奇数
9. }
10. **while**(!RabinMiller(base, 30));    //进行拉宾－米勒测试次
11. **return** base;    //全部通过认为是素数
12. }

随机生成给定bits位的素数，首先通过把1左移bits-1位来确保生成数的位数是bits位，之后再生成一系列随机数，依次通过之前的Ranbin-Miller方法进行素数测试，若能通过，即代表此数很大几率上是素数，于是返回这个生成数

五、实验结果总结

1. 通过本实验，论述RSA算法的加密原理是什么？

RSA加密应用了单项函数正向求解很简单，反向求解很复杂的特性。

具体有两个方面：

1. 两个质数相乘很容易，而将很大的合数分解很困难
2. 对于公式(m ^ e) mod n = c，已知m, e, n求解c很简单，而已知e, n, c求m很难
3. 在上述算法中哪些模块是该算法的核心模块？
4. RandomPrime: 随机生成给定bits位的素数,该模块是生成公钥n的前提
5. SteinFcd: 用Stein法求最大公约数，该模块是判断加密密钥e是否合法的核心
6. Euclid: 对于公式a\*x=1(mod b),已知a, b，求x。此模块是计算解密密钥d的核心。
7. 对于一个RSA加密算法的密文，要得到明文需要哪些要素？
8. 已知公钥n，解密密钥d和密文c，由解密公式(c ^ d) mod n = m解得明文
9. 已知公钥n，加密密钥e和密文c，如果能将n分解为两个素数p,q，即可以由公式计算出f，并由e和f计算出解密密钥d，从而由解密公式(c ^ d) mod n = m解得明文
10. 心得体会

通过此实验，我深入了解了RSA加密算法的加密原理和加密过程，同时从中学习到了密码学的博大精深，以及加深了对于此领域专家的敬佩，对于信息安全方向有了进一步理解与兴趣。

1. 改进意见

实验网站上“实验内容与步骤”模块内的任务一里关于求解解密密钥的公式似乎误将Φ(n)写为n；

实验代码中一些注释位置和内容不准确或遗漏，希望能有改善；