RSA加密算法 C++实现

**1、预备知识**

**1.1 快速幂算法**

    顾名思义，快速幂就是快速算底数的nn次幂。其时间复杂度为O(logn)O(logn)，与朴素的O(n)O(n)相比，效率有了极大的提高。具体可以参考百度百科：[快速幂](http://baike.baidu.com/link?url=0qZiUBQAfmhKP49lmQGtNYGM1Y40hvg85zFk5sgODF4czArnuAFLbqI4bwqznG65vr81K7RgL2sYSmo29r_fY5Sx5FBcpbbboZgXnQLML6pbzAg_Me6U__p0Df8bEyIe)。

**1.2 扩展欧几里得算法**

    扩展欧几里得算法（英语：Extended Euclidean algorithm）是欧几里得算法（又叫辗转相除法）的扩展。已知整数a、b，扩展欧几里得算法可以在求得a、b的最大公约数的同时，能找到整数x、y（其中一个很可能是负数），使它们满足贝祖等式

ax+by=gcd(a,b).ax+by=gcd(a,b).

    如果aa是负数，可以把问题转化成

    |a|(−x)+by=gcd(|a|,b)|a|(−x)+by=gcd(|a|,b)（|a||a|为a的绝对值），然后令x′=(−x)x′=(−x)。具体可以参考维基百科：[扩展欧几里得](https://zh.wikipedia.org/wiki/%E6%89%A9%E5%B1%95%E6%AC%A7%E5%87%A0%E9%87%8C%E5%BE%97%E7%AE%97%E6%B3%95)。

**1.3 米勒-拉宾素性检验算法**

    要测试NN是否为素数，首先将N−1N−1分解为2sd2sd。在每次测试开始时，先随机选一个介于[1,N−1][1,N−1]的整数aa，之后如果对所有的r∈[0,s−1]r∈[0,s−1]，若admodN≠1admodN≠1且a2rdmodN≠−1a2rdmodN≠−1，则NN是合数。否则，NN有3/43/4的概率为素数。

    构成该算法的思想是，如果ad≠1(modn)ad≠1(modn)以及n=1+2sdn=1+2sd是素数，则值序列

admodn,a2dmodn,a4dmodn,…,a2sdmodnadmodn,a2dmodn,a4dmodn,…,a2sdmodn

    将以11结束，而且在头一个11的前边的值将是n−1n−1(当pp是素数时，对于y2≡1(modp)y2≡1(modp)，仅有的解是y≡±1(modp)y≡±1(modp)，因为(y+1)(y−1)(y+1)(y−1)必须是pp的倍数)。注意，如果在该序列中出现了n−1n−1，则该序列中的下一个值一定是11,因为(n−1)2≡n2−2n+1≡1(modn)(n−1)2≡n2−2n+1≡1(modn)。具体可以参考维基百科：[米勒-拉宾素性检验](https://zh.wikipedia.org/wiki/%E7%B1%B3%E5%8B%92-%E6%8B%89%E5%AE%BE%E6%A3%80%E9%AA%8C)。

**1.4 RSA加密算法**

**1.4.1 公钥与私钥的产生**

    假设Alice想要通过一个不可靠的媒体接收Bob的一条私人消息。她可以用以下的方式来产生一个公钥和一个私钥：

    1、随意选择两个大的质数pp和qq，pp不等于qq，计算N=pqN=pq。

    2、根据欧拉函数，求得r=φ(N)=φ(p)φ(q)=(p−1)(q−1)r=φ(N)=φ(p)φ(q)=(p−1)(q−1)。

    3、选择一个小于rr的整数ee，使ee与rr互质。并求得ee关于rr的模反元素，命名为dd(求dd令ed≡1(modr)ed≡1(modr)）。(模反元素存在，当且仅当ee与rr互质）

    4、将pp和qq的记录销毁。

    (N,e)(N,e)是公钥，(N,d)(N,d)是私钥。Alice将她的公钥(N,e)(N,e)传给Bob，而将她的私钥(N,d)(N,d)藏起来。

**1.4.2 加密消息**

    假设Bob想给Alice送一个消息mm，他知道Alice产生的NN和ee。他使用起先与Alice约好的格式将mm转换为一个小于NN，且与NN互质的整数nn，比如他可以将每一个字转换为这个字的Unicode码，然后将这些数字连在一起组成一个数字。假如他的信息非常长的话，他可以将这个信息分为几段，然后将每一段转换为nn。用下面这个公式他可以将nn加密为cc：

ne≡c(modN)ne≡c(modN)

    计算cc并不复杂。Bob算出cc后就可以将它传递给Alice。

**1.4.3 解密消息**

    Alice得到Bob的消息cc后就可以利用她的密钥dd来解码。她可以用以下这个公式来将cc转换为nn：

cd≡n(modN)cd≡n(modN)

    得到nn后，她可以将原来的信息mm重新复原。

    解码的原理是

cd≡ned(modN)cd≡ned(modN)

    已知ed≡1(modr)ed≡1(modr)，即ed=1+hφ(N)ed=1+hφ(N)。由欧拉定理得：

ned=n1+hφ(N)=n(nφ(N))h≡n(1)h(modN)≡n(modN)ned=n1+hφ(N)=n(nφ(N))h≡n(1)h(modN)≡n(modN)

    RSA也可用作数字签名。具体可以参考维基百科：[RSA](https://zh.wikipedia.org/wiki/RSA%E5%8A%A0%E5%AF%86%E6%BC%94%E7%AE%97%E6%B3%95)。

**2、模块设计**

**2.1 BigInteger类**

    因为该加密算法涉及的数可能很大，而C++中并没有像Java一样，内置大整数类BigInteger，故需自己实现，我的实现大概是模仿Java的BigInteger设计的。

    该模块包含两个文件：BigInteger.h和BigInteger.cpp。代码如下所示。

    BigInteger.h代码如下。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 /\*\*

2 \* @Name : BigInteger.h

3 \* @Date : 2017-04-11-22.11.39

4 \* @Author : Silenceneo (silenceneo\_xw@163.com)

5 \* @Link : http://www.cnblogs.com/Silenceneo-xw/

6 \* @Version : 2.0

7 \*/

8

9 #ifndef \_\_BIGINTEGER\_H\_\_

10 #define \_\_BIGINTEGER\_H\_\_

11

12 #include <string>

13 #include <vector>

14 #include <ostream>

15 class BigInteger {

16 public:

17 typedef long long long\_t;

18 typedef unsigned base\_t;

19 BigInteger(): is\_negative(false) { data.push\_back(0); }// 默认为0

20 BigInteger(const BigInteger &); // 利用给定的大整数初始化

21 BigInteger(const std::string &);// 利用给定的十六进制字符串初始化

22 BigInteger(const long\_t &); // 利用给定的long\_t类型数据初始化

23 ~BigInteger() {}

24

25 BigInteger add(const BigInteger &); // 大整数加法

26 BigInteger subtract(const BigInteger &);// 大整数减法

27 BigInteger multiply(const BigInteger &) const;// 大整数乘法

28 BigInteger divide(const BigInteger &); // 大整数整除

29 BigInteger remainder(const BigInteger &); // 大整数取余

30 BigInteger mod(const BigInteger &); // 大整数取模

31 BigInteger divideAndRemainder(const BigInteger &, BigInteger &);// 大整数整除和取余

32 BigInteger pow(const BigInteger &); // 大整数幂乘

33 BigInteger modPow(const BigInteger &, const BigInteger &) const;// 大整数幂模运算

34 BigInteger modInverse(const BigInteger &);// 用扩展欧几里得算法求乘法逆元

35

36 BigInteger shiftLeft(const unsigned); // 移位运算,左移

37 BigInteger shiftRight(const unsigned); // 移位运算,右移

38

39 int compareTo(const BigInteger &) const;// 比较运算

40 bool equals(const BigInteger &) const;// 判断是否等于给定数

41 static BigInteger valueOf(const long\_t &);// 将给定数转换为大整数并返回

42 std::string toString() const; // 将大整数转换为十六进制字符串

43 BigInteger abs() const; // 求大整数的绝对值

44 protected:

45 // 以下运算符重载函数主要用于像基本类型一样使用大整数类型

46 friend BigInteger operator + (const BigInteger &, const BigInteger &);

47 friend BigInteger operator - (const BigInteger &, const BigInteger &);

48 friend BigInteger operator \* (const BigInteger &, const BigInteger &);

49 friend BigInteger operator / (const BigInteger &, const BigInteger &);

50 friend BigInteger operator % (const BigInteger &, const BigInteger &);

51 friend bool operator < (const BigInteger &, const BigInteger &);

52 friend bool operator > (const BigInteger &, const BigInteger &);

53 friend bool operator == (const BigInteger &, const BigInteger &);

54 friend bool operator <= (const BigInteger &, const BigInteger &);

55 friend bool operator >= (const BigInteger &, const BigInteger &);

56 friend bool operator != (const BigInteger &, const BigInteger &);

57

58 // 重载版本,使其能用于long\_t类型

59 friend BigInteger operator + (const BigInteger &, const long\_t &);

60 friend BigInteger operator - (const BigInteger &, const long\_t &);

61 friend BigInteger operator \* (const BigInteger &, const long\_t &);

62 friend BigInteger operator / (const BigInteger &, const long\_t &);

63 friend BigInteger operator % (const BigInteger &, const long\_t &);

64 friend bool operator < (const BigInteger &, const long\_t &);

65 friend bool operator > (const BigInteger &, const long\_t &);

66 friend bool operator == (const BigInteger &, const long\_t &);

67 friend bool operator <= (const BigInteger &, const long\_t &);

68 friend bool operator >= (const BigInteger &, const long\_t &);

69 friend bool operator != (const BigInteger &, const long\_t &);

70

71 friend std::ostream & operator << (std::ostream &, const BigInteger &);

72 BigInteger operator = (const std::string & str) { return (\*this) = BigInteger(str); }

73 BigInteger operator = (const long\_t & num) { return (\*this) = BigInteger(num); }

74 private:

75 BigInteger trim(); // 去掉高位无用的0

76 int hexToNum(char); // 十六进制字符转换为十进制数

77 public:

78 static const int base\_bit = 5; // 2^5=32,大整数每位存储的二进制位数

79 static const int base\_char = 8; // 组成大整数的一位需要的十六进制位数

80 static const int base\_int = 32; // 大整数一位对应的二进制位数

81 static const int base\_num = 0xffffffff;// 截取低位的辅助

82 static const int base\_temp = 0x1f; // 截取模32的余数的辅助

83 static const BigInteger ZERO; // 大整数常量0

84 static const BigInteger ONE; // 大整数常量1

85 static const BigInteger TWO; // 大整数常量2

86 static const BigInteger TEN; // 大整数常量10

87 private:

88 bool is\_negative;// 是否为负数

89 std::vector<base\_t> data;// 按位数据存储,高位在后

90 class bit { // 便于大整数运算的二进制处理类

91 public:

92 bit(const BigInteger &);// 根据大整数初始化

93

94 size\_t size() { return length; } // 返回大整数对应的二进制位数

95 bool at(size\_t); // 返回第i位二进制是否为1

96 private:

97 std::vector<base\_t> bit\_vector; // 二进制数据存储,每一个元素对应32位二进制

98 size\_t length; // 二进制的总位数

99 };

100 friend class RSA; // RSA为其友元类

101 };

102

103 #endif // \_\_BIGINTEGER\_H\_\_

[复制代码](javascript:void(0);)

    BigInteger.cpp代码如下。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 /\*\*

2 \* @Name : BigInteger.cpp

3 \* @Date : 2017-04-11-22.16.42

4 \* @Author : Silenceneo (silenceneo\_xw@163.com)

5 \* @Link : http://www.cnblogs.com/Silenceneo-xw/

6 \* @Version : 2.0

7 \*/

8

9 #include <algorithm>

10 #include <cassert>

11 #include <cctype>

12 #include "BigInteger.h"

13

14 // 以下表示为静态常量赋值

15 const BigInteger BigInteger::ZERO = BigInteger(0);

16 const BigInteger BigInteger::ONE = BigInteger(1);

17 const BigInteger BigInteger::TWO = BigInteger(2);

18 const BigInteger BigInteger::TEN = BigInteger(10);

19

20 /\*\*

21 \* 函数功能:根据给定的大整数构造一个新的大整数

22 \* 参数含义:val代表给定的大整数

23 \*/

24 BigInteger::BigInteger(const BigInteger & val) {

25 \*this = val;

26 }

27

28 /\*\*

29 \* 函数功能:根据给定的十六进制字符串数据构造一个大整数

30 \* 参数含义:str代表给定的数据

31 \*/

32 BigInteger::BigInteger(const std::string & str): is\_negative(false) {

33 std::string t(str);

34 if (t.size() && t.at(0)=='-') {

35 if (t.size() > 1)

36 is\_negative = true;

37 t = t.substr(1);

38 }

39 int cnt = (8-(t.size()%8))%8;// 数的长度不是8的倍数,补足0

40 std::string temp;

41

42 for (int i=0; i<cnt; ++i)

43 temp.push\_back('0');

44

45 t = temp+t;

46

47 for (size\_t i=0; i<t.size(); i+=base\_char) {

48 base\_t sum = 0;

49 for (int j=0; j<base\_char; ++j) { // 8位十六进制组成大整数的一位

50 char ch = t[i+j];

51 int num = hexToNum(ch);

52 sum = ((sum<<4) | (num));

53 }

54 data.push\_back(sum);

55 }

56 reverse(data.begin(), data.end());// 高位在后

57 \*this = trim();// 去除高位的0

58 }

59

60 /\*\*

61 \* 函数功能:根据给定的long\_t类型数据构造一个大整数

62 \* 参数含义:num代表给定的数据

63 \*/

64 BigInteger::BigInteger(const long\_t & num): is\_negative(false) {

65 long\_t t = num;

66 if (t < 0) {

67 is\_negative = true;

68 t = -t;

69 }

70 do {

71 base\_t temp = (t&base\_num); // 每次截取低32位

72 data.push\_back(temp);

73 t >>= base\_int;

74 } while (t);

75 }

76

77 /\*\*

78 \* 函数功能:大整数加法运算

79 \* 参数含义:val代表加数

80 \*/

81 BigInteger BigInteger::add(const BigInteger & val) {

82 BigInteger ans(\*this);

83 if (ans.is\_negative == val.is\_negative) {// 同号

84 int len = val.data.size()-ans.data.size();

85

86 while ((len--) > 0) // 被加数位数少,高位补0

87 ans.data.push\_back(0);

88

89 int carry = 0; // 进位

90 for (size\_t i=0; i<val.data.size(); ++i) {

91 base\_t temp = ans.data[i];

92 ans.data[i] += val.data[i]+carry; // 无符号数相加,超出取其余数

93 // 进位:一种是有无进位都超出,一种是有进位才超出(比如十进制相加,9+9+1,得9,9+0+0,得9)

94 carry = (temp>ans.data[i] ? 1 : (temp>(temp+val.data[i]) ? 1 : 0));

95 }

96

97 for (size\_t i=val.data.size(); i<ans.data.size() && carry!=0; ++i) {// 还有进位

98 base\_t temp = ans.data[i];

99 ans.data[i] += carry;

100 carry = temp > ans.data[i];

101 }

102

103 if (carry) // 还有进位

104 ans.data.push\_back(carry);

105 }

106 else { // 异号

107 BigInteger a = abs();

108 BigInteger b = val.abs();

109 int flag = a.compareTo(b);

110 // 绝对值相等,则结果为0,否则用绝对值大的减去小的,符号随绝对值大的

111 if (flag == -1) {

112 ans = b.subtract(a);

113 ans.is\_negative = val.is\_negative;

114 }

115 else if (flag == 0)

116 ans = ZERO;

117 else {

118 ans = a.subtract(b);

119 ans.is\_negative = is\_negative;

120 }

121 }

122 return ans;

123 }

124

125 /\*\*

126 \* 函数功能:大整数减法运算

127 \* 参数含义:val代表减数

128 \*/

129 BigInteger BigInteger::subtract(const BigInteger & val) {

130 BigInteger ans(\*this);

131 BigInteger a = abs();

132 BigInteger b = val.abs();

133 if (ans.is\_negative == val.is\_negative) {// 同号

134 int flag = a.compareTo(b);

135 if (flag == 1) {// a的绝对值大于b的绝对值,直接减

136 int borrow = 0; // 借位

137 // 大数减小数

138 for (size\_t i=0; i<val.data.size(); ++i) {

139 base\_t temp = ans.data[i];

140 ans.data[i] -= val.data[i]+borrow;

141 // 借位:一种是有无借位都超出,另一种是有借位才超出(比如十进制相减,9-0-0,得9,9-9-1,得9)

142 borrow = temp<ans.data[i] ? 1 : (temp-borrow<val.data[i] ? 1 : 0);

143 }

144 for (size\_t i=val.data.size(); i<ans.data.size() && borrow!=0; ++i) {// 还有借位

145 base\_t temp = ans.data[i];

146 ans.data[i] -= borrow;

147 borrow = temp < (base\_t)borrow;

148 }

149 ans = ans.trim();// 去掉高位多余的0

150 }

151 else if (flag == 0)

152 ans = ZERO;

153 else {// a的绝对值小于b的绝对值

154 ans = b.subtract(a);

155 ans.is\_negative = !is\_negative;

156 }

157 }

158 else { // 异号

159 ans = a.add(b); // 转换为加法

160 ans.is\_negative = is\_negative;

161 }

162 return ans;

163 }

164

165 /\*\*

166 \* 函数功能:大整数乘法运算

167 \* 参数含义:val代表乘数

168 \*/

169 BigInteger BigInteger::multiply(const BigInteger & val) const {

170 if (equals(ZERO) || val.equals(ZERO))

171 return ZERO;

172 // 将位数少的作为乘数

173 const BigInteger & big = data.size()>val.data.size() ? (\*this) : val;

174 const BigInteger & small = (&big)==(this) ? val : (\*this);

175

176 BigInteger ans;

177 bit t(small); // 转换为二进制进行运算

178

179 for (int i=t.size()-1; i>=0; --i)

180 if (t.at(i)) {

181 BigInteger temp(big);

182 temp.is\_negative = false;

183 temp = temp.shiftLeft(i); // 移位对齐

184 ans = ans.add(temp);

185 }

186 ans.is\_negative = !(is\_negative == val.is\_negative);

187 return ans;

188 }

189

190 /\*\*

191 \* 函数功能:大整数整除运算

192 \* 参数含义:val代表除数

193 \*/

194 BigInteger BigInteger::divide(const BigInteger & val) {

195 BigInteger temp;

196 BigInteger ans = divideAndRemainder(val, temp);

197 return ans;

198 }

199

200 /\*\*

201 \* 函数功能:大整数取余运算

202 \* 参数含义:val代表除数

203 \*/

204 BigInteger BigInteger::remainder(const BigInteger & val) {

205 BigInteger ans;

206 divideAndRemainder(val, ans);

207 return ans;

208 }

209

210 /\*\*

211 \* 函数功能:大整数取模运算(不同于取余,该函数总是返回正余数)

212 \* 参数含义:m代表模数

213 \*/

214 BigInteger BigInteger::mod(const BigInteger & m) {

215 BigInteger ans = remainder(m);

216 if (ans.is\_negative)

217 ans = ans.add(m);

218 return ans;

219 }

220

221 /\*\*

222 \* 函数功能:大整数整除运算和取余运算,整除结果直接返回,取余结果由m传回

223 \* 参数含义:val表示除数,m表示取余结果

224 \*/

225 BigInteger BigInteger::divideAndRemainder(const BigInteger & val, BigInteger & m) {

226 assert(!val.equals(ZERO));

227 BigInteger a = abs();

228 BigInteger b = val.abs();

229 int flag = a.compareTo(b);

230 if (flag == 0)// 绝对值相等

231 return (is\_negative==val.is\_negative) ? BigInteger(1) : BigInteger(-1);

232 if (flag == -1) {

233 m = \*this;

234 return ZERO;

235 }

236 BigInteger ans;

237

238 bit bit\_b(b);

239 // 位数对齐

240 while (true) {// a的绝对值大于b的绝对值

241 bit bit\_a(a);

242 int len = bit\_a.size()-bit\_b.size();

243 BigInteger temp;

244 // 找到移位

245 while (len >= 0) {

246 temp = b.shiftLeft(len);

247 if (temp.compareTo(a) != 1)// 找到最大的左移位数使得当前的a大于等于b

248 break;

249 --len;

250 }

251 if (len < 0) // 当前的a小于b了

252 break;

253 base\_t num = 0;

254 while (temp.compareTo(a) != 1) {

255 a = a.subtract(temp);

256 ++num; // 统计当前的a最多大于等于几个移位后的b

257 }

258 temp = BigInteger(num);

259 if (len)

260 temp = temp.shiftLeft(len);// 移位后表明当前的a是b的几倍

261 ans = ans.add(temp);

262 }

263 ans.is\_negative = !(is\_negative==val.is\_negative);

264 m.data = a.data;

265 m.is\_negative = is\_negative;

266 return ans;

267 }

268

269 /\*\*

270 \* 函数功能:大整数幂乘运算

271 \* 参数含义:exponent代表指数

272 \*/

273 BigInteger BigInteger::pow(const BigInteger & exponent) {

274 BigInteger ans(1);

275 bit t(exponent); // 转化为二进制,快速求幂

276 for (int i=t.size()-1; i>=0; --i) {

277 ans = ans.multiply(ans);

278 if (t.at(i))

279 ans = multiply(ans);// 从高位开始,位权累加效应

280 }

281 return ans;

282 }

283

284 /\*\*

285 \* 函数功能:大整数模幂运算

286 \* 参数含义:exponent代表指数,m代表模数

287 \*/

288 BigInteger BigInteger::modPow(const BigInteger & exponent, const BigInteger & m) const {

289 assert(!m.equals(ZERO));

290 BigInteger ans(1);

291 bit t(exponent);

292 for (int i=t.size()-1; i>=0; --i) {

293 ans = ans.multiply(ans).mod(m);

294 if (t.at(i))

295 ans = multiply(ans).mod(m);

296 }

297 return ans;

298 }

299

300 /\*\*

301 \* 函数功能:扩展欧几里得算法求乘法逆元

302 \* 参数含义:m代表求逆元时的模数

303 \*/

304 BigInteger BigInteger::modInverse(const BigInteger & m) {

305 assert(!is\_negative); // 当前大整数为正数

306 assert(!m.is\_negative); // m为正数

307 if (equals(ZERO) || m.equals(ZERO))

308 return ZERO; // 有一个数为0,就不存在乘法逆元

309 BigInteger a[3], b[3], t[3];

310 // 以下进行初等变换

311 a[0] = 0; a[1] = 1; a[2] = \*this;

312 b[0] = 1; b[1] = 0; b[2] = m;

313

314 for (t[2]=a[2].mod(b[2]); !t[2].equals(ZERO); t[2]=a[2].mod(b[2])) {

315 BigInteger temp = a[2].divide(b[2]);

316 for (int i=0; i<3; ++i) {

317 t[i] = a[i].subtract(temp.multiply(b[i]));// 不超过一次a[2]-temp\*b[2]就变为大数减小数

318 a[i] = b[i];

319 b[i] = t[i];

320 }

321 }

322 if (b[2].equals(ONE)) {// 最大公约数为1,存在乘法逆元

323 if (b[1].is\_negative)// 逆元为负数

324 b[1] = b[1].add(m);// 变为正数,使其在m的剩余集中

325 return b[1];

326 }

327 return ZERO;// 最大公约数不为1,无乘法逆元

328 }

329

330 /\*\*

331 \* 函数功能:移位运算,左移

332 \* 参数含义:len代表移位的位数

333 \*/

334 BigInteger BigInteger::shiftLeft(const unsigned len) {

335 int index = len>>base\_bit; // 大整数每一位需要移动多少位

336 int shift = len&base\_temp; // 还剩下多少位

337 BigInteger ans(\*this);

338

339 int inc = (shift==0) ? index : index+1;// 有多余的位要多开大整数的一位

340 for (int i=0; i<inc; ++i)

341 ans.data.push\_back(0); // 高位补0

342

343 if (index) {

344 inc = (shift==0) ? 1 : 2;// 有多余的位要预留一位

345 for (int i=ans.data.size()-inc; i>=index; --i)

346 ans.data[i] = ans.data[i-index];

347 for (int i=0; i<index; ++i)

348 ans.data[i] = 0;

349 }

350 if (shift) {

351 base\_t t = base\_num;

352 t <<= base\_int-shift; // 用于截取高位

353 // 左移

354 base\_t temp = 0;

355 for (size\_t i=0; i<ans.data.size(); ++i) {

356 base\_t tmp = ans.data[i];

357 ans.data[i] = (tmp<<shift) | temp;// 左移后加上大整数低位的高位

358 temp = (tmp&t)>>(base\_int-shift);// 获取该大整数位的高位

359 }

360 }

361 ans = ans.trim();

362 return ans;

363 }

364

365 /\*\*

366 \* 函数功能:移位运算,右移

367 \* 参数含义:len代表移位的位数

368 \*/

369 BigInteger BigInteger::shiftRight(const unsigned len) {

370 bit val(\*this);

371 if (len >= val.size())// 当前大整数位数小于等于移位位数,返回0

372 return ZERO;

373 int index = len>>base\_bit;// 大整数每一位需要移动多少位

374 int shift = len&base\_temp;// 还剩下多少位

375 BigInteger ans(\*this);

376

377 if (index) {

378 for (int i=0; i<index; ++i)

379 ans.data[i] = ans.data[i+index];

380 for (int i=0; i<index; ++i)

381 ans.data.pop\_back(); // 高位删除

382 }

383 if (shift) {

384 base\_t t = base\_num;

385 t >>= base\_int-shift; // 用于截取低位

386 // 右移

387 base\_t temp = 0;

388 for (int i=ans.data.size()-1; i>=0; --i) {

389 base\_t tmp = ans.data[i];

390 ans.data[i] = (tmp>>shift) | temp;// 右移后加上大整数高位的低位

391 temp = (tmp&t)<<(base\_int-shift);// 获取该大整数位的低位

392 }

393 }

394 ans = ans.trim();

395 return ans;

396 }

397

398 /\*\*

399 \* 函数功能:大整数比较函数,-1表示本大整数要小,0表示相等,1表示本大整数要大

400 \* 参数含义:val代表要与之比较的大整数

401 \*/

402 int BigInteger::compareTo(const BigInteger & val) const {

403 if (is\_negative != val.is\_negative) {// 符号不同,负数必小

404 if (is\_negative == true)

405 return -1;

406 return 1;

407 }

408 int flag = 0;

409 if (data.size() < val.data.size())// 位数较小

410 flag = -1;

411 else if (data.size() > val.data.size())// 位数较大

412 flag = 1;

413 else { // 位数相等,从高位开始一一比较

414 for (std::vector<base\_t>::const\_reverse\_iterator it=data.rbegin(), ite=val.data.rbegin(); it!=data.rend(); ++it, ++ite)

415 if ((\*it) != (\*ite)) {

416 flag = (\*it)<(\*ite) ? -1 : 1; // 高位小,则小

417 break;

418 }

419 }

420 if (is\_negative) // 如为负数,小的反而大

421 flag = -flag;

422 return flag;

423 }

424

425 /\*\*

426 \* 函数功能:大整数是否相等函数

427 \* 参数含义:val表示要与之比较的大整数

428 \*/

429 bool BigInteger::equals(const BigInteger & val) const {

430 return (is\_negative==val.is\_negative) && (data==val.data);// 符号和数据都要相等

431 }

432

433 /\*\*

434 \* 函数功能:将一个long\_t类型的数据转换为大整数并返回

435 \* 参数含义:num表示给定的数

436 \*/

437 BigInteger BigInteger::valueOf(const long\_t & num) {

438 return BigInteger(num);

439 }

440

441 /\*\*

442 \* 函数功能:将大整数转换为十六进制字符串并返回

443 \*/

444 std::string BigInteger::toString() const {

445 std::string ans;

446 base\_t t = base\_num;

447 t <<= base\_int-4; // 用于截取高4位

448 for (int i=data.size()-1; i>=0; --i) {

449 base\_t temp = data[i];

450 for (int j=0; j<base\_char; ++j) {

451 base\_t num = t&temp;// 每次截取高4位

452 num >>= base\_int-4; // 将高4位移到低4位

453 temp <<= 4;

454 if (num < 10)

455 ans.push\_back((char)('0'+num));

456 else

457 ans.push\_back((char)('A'+num-10));

458 }

459 }

460 while (ans.size()>0 && ans.at(0)=='0')// 去掉高位无用的0

461 ans = ans.substr(1);

462 if (ans.empty()) // 空串说明为0

463 ans.push\_back('0');

464 if (is\_negative) // 为负数加上负号

465 ans = "-"+ans;

466 return ans;

467 }

468

469 /\*\*

470 \* 函数功能:返回大整数的绝对值

471 \*/

472 BigInteger BigInteger::abs() const {

473 BigInteger ans;

474 ans.data = data;// 只复制数据,符号默认为正

475 return ans;

476 }

477

478 // 以下运算符重载函数主要是为了使得能使用

479 // 大整数类型像使用基本类型一样,不一一介绍

480 BigInteger operator + (const BigInteger & a, const BigInteger & b) {

481 BigInteger t(a);

482 return t.add(b);

483 }

484

485 BigInteger operator - (const BigInteger & a, const BigInteger & b) {

486 BigInteger t(a);

487 return t.subtract(b);

488 }

489

490 BigInteger operator \* (const BigInteger & a, const BigInteger & b) {

491 BigInteger t(a);

492 return t.multiply(b);

493 }

494

495 BigInteger operator / (const BigInteger & a, const BigInteger & b) {

496 BigInteger t(a);

497 return t.divide(b);

498 }

499

500 BigInteger operator % (const BigInteger & a, const BigInteger & b) {

501 BigInteger t(a);

502 return t.remainder(b);

503 }

504

505 bool operator < (const BigInteger & a, const BigInteger & b) {

506 return a.compareTo(b) == -1;

507 }

508

509 bool operator > (const BigInteger & a, const BigInteger & b) {

510 return b < a;

511 }

512

513 bool operator == (const BigInteger & a, const BigInteger & b) {

514 return a.equals(b);

515 }

516

517 bool operator <= (const BigInteger & a, const BigInteger & b) {

518 return !(a > b);

519 }

520

521 bool operator >= (const BigInteger & a, const BigInteger & b) {

522 return !(a < b);

523 }

524

525 bool operator != (const BigInteger & a, const BigInteger & b) {

526 return !(a == b);

527 }

528

529 BigInteger operator + (const BigInteger & a, const BigInteger::long\_t & b) {

530 return a+BigInteger(b);

531 }

532

533 BigInteger operator - (const BigInteger & a, const BigInteger::long\_t & b) {

534 return a-BigInteger(b);

535 }

536

537 BigInteger operator \* (const BigInteger & a, const BigInteger::long\_t & b) {

538 return a\*BigInteger(b);

539 }

540

541 BigInteger operator / (const BigInteger & a, const BigInteger::long\_t & b) {

542 return a/BigInteger(b);

543 }

544

545 BigInteger operator % (const BigInteger & a, const BigInteger::long\_t & b) {

546 return a%BigInteger(b);

547 }

548

549 bool operator < (const BigInteger & a, const BigInteger::long\_t & b) {

550 return a < BigInteger(b);

551 }

552

553 bool operator > (const BigInteger & a, const BigInteger::long\_t & b) {

554 return a > BigInteger(b);

555 }

556

557 bool operator == (const BigInteger & a, const BigInteger::long\_t & b) {

558 return a == BigInteger(b);

559 }

560

561 bool operator <= (const BigInteger & a, const BigInteger::long\_t & b) {

562 return a <= BigInteger(b);

563 }

564

565 bool operator >= (const BigInteger & a, const BigInteger::long\_t & b) {

566 return a >= BigInteger(b);

567 }

568

569 bool operator != (const BigInteger & a, const BigInteger::long\_t & b) {

570 return a != BigInteger(b);

571 }

572

573 std::ostream & operator << (std::ostream & out, const BigInteger & val) {

574 out << val.toString();

575 return out;

576 }

577

578 /\*\*

579 \* 函数功能:创建该大整数的一个副本,去除掉高位无用的0后并返回

580 \*/

581 BigInteger BigInteger::trim() {

582 size\_t cnt = 0;

583 // 检查高位为0的元素的数量

584 for (std::vector<base\_t>::const\_reverse\_iterator it=data.rbegin(); it!=data.rend(); ++it) {

585 if ((\*it) == 0)

586 ++cnt;

587 else

588 break;

589 }

590 if (cnt == data.size()) // 只有零的情况保留

591 --cnt;

592 BigInteger ans(\*this);

593 for (size\_t i=0; i<cnt; ++i)

594 ans.data.pop\_back();

595 return ans;

596 }

597

598 /\*\*

599 \* 函数功能:根据给定的字符确定它所对应的十进制数

600 \* 参数含义:ch代表给定的字符

601 \*/

602 int BigInteger::hexToNum(char ch) {

603 int ans = 0;

604 if (isdigit(ch))

605 ans = ch-'0';

606 else if (islower(ch))

607 ans = ch-'a'+10;

608 else

609 ans = ch-'A'+10;

610 return ans;

611 }

612

613 /\*\*

614 \* 函数功能:根据给定的大整数初始化

615 \* 参数含义:val代表给定的大整数

616 \*/

617 BigInteger::bit::bit(const BigInteger & val) {

618 bit\_vector = val.data;

619 base\_t temp = bit\_vector[bit\_vector.size()-1];// 大整数最高位

620 length = bit\_vector.size()<<base\_bit; // 大整数一位占二进制32位

621 base\_t t = 1<<(base\_int-1); // 用于截取一个数的二进制最高位

622

623 if (temp == 0) // 大整数最高位为0,减去32

624 length -= base\_int;

625 else {

626 while (!(temp & t)) {// 从高位开始检测大整数的二进制位,为0长度减一

627 --length;

628 t >>= 1; // 右移一位表示检测下一位

629 }

630 }

631 }

632

633 /\*\*

634 \* 函数功能:检测大整数的第id位二进制位是否为1

635 \* 参数含义:id代表第id位

636 \*/

637 bool BigInteger::bit::at(size\_t id) {

638 size\_t index = id>>base\_bit;// 确定其在大整数第几位

639 size\_t shift = id&base\_temp;// 确定其在大整数那一位的二进制第几位

640 base\_t t = bit\_vector[index];

641 return (t & (1<<shift));

642 }

[复制代码](javascript:void(0);)

**2.2 RSA类**

    该类主要实现RSA相关功能，例如加密和解密等。

    该模块也有两个文件：RSA.h和RSA.cpp。代码如下所示。

    RSA.h代码如下。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 /\*\*

2 \* @Name : RSA.h

3 \* @Date : 2017-04-11-22.25.57

4 \* @Author : Silenceneo (silenceneo\_xw@163.com)

5 \* @Link : http://www.cnblogs.com/Silenceneo-xw/

6 \* @Version : 2.0

7 \*/

8

9 #ifndef \_\_RSA\_H\_\_

10 #define \_\_RSA\_H\_\_

11

12 #include <ostream>

13 #include "BigInteger.h"

14 class RSA {

15 public:

16 RSA() {}

17 RSA(const unsigned len) { init(len); } // 利用len初始化对象

18 ~RSA() {}

19

20 void init(const unsigned);// 初始化,产生公私钥对

21

22 BigInteger encryptByPublic(const BigInteger &); // 公钥加密

23 BigInteger decryptByPrivate(const BigInteger &);// 私钥解密

24

25 // 以下主要用于数字签名

26 BigInteger encryptByPrivate(const BigInteger &);// 私钥加密

27 BigInteger decryptByPublic(const BigInteger &); // 公钥解密

28 protected:

29 friend std::ostream & operator << (std::ostream &, const RSA &);// 输出相关数据

30 private:

31 BigInteger createOddNum(unsigned);// 生成一个大奇数,参数为其长度

32 bool isPrime(const BigInteger &, const unsigned);// 判断是否为素数

33 BigInteger createRandomSmaller(const BigInteger &);// 随机创建一个更小的数

34 BigInteger createPrime(unsigned, const unsigned);// 生成一个大素数,参数为其长度

35 void createExponent(const BigInteger &);// 根据提供的欧拉数生成公钥、私钥指数

36 public:

37 BigInteger n, e;// 公钥

38 private:

39 BigInteger d;// 私钥

40 BigInteger p, q;// 大素数p和q

41 BigInteger eul;// n的欧拉函数

42 };

43

44 #endif // \_\_RSA\_H\_\_

[复制代码](javascript:void(0);)

    RSA.cpp代码如下。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 /\*\*

2 \* @Name : RSA.cpp

3 \* @Date : 2017-04-11-22.27.51

4 \* @Author : Silenceneo (silenceneo\_xw@163.com)

5 \* @Link : http://www.cnblogs.com/Silenceneo-xw/

6 \* @Version : 2.0

7 \*/

8

9 #include <cassert>

10 #include <sstream>

11 #include <ctime>

12 #include "RSA.h"

13

14 /\*\*

15 \* 函数功能:初始化RSA对象的相关信息

16 \* 参数含义:len表示大素数的二进制位数

17 \*/

18 void RSA::init(const unsigned len) {

19 srand((unsigned)time(NULL));

20 // 产生大素数p和q

21 p = createPrime(len, 15);// 出错概率为(1/4)^15

22 q = createPrime(len, 15);

23 // 计算出n

24 n = p\*q;

25 // 计算出n的欧拉函数

26 eul = (p-1)\*(q-1);

27 // 设置加解密指数e和d

28 createExponent(eul);

29 }

30

31 /\*\*

32 \* 函数功能:使用公钥进行加密

33 \* 参数含义:m表示要加密的明文

34 \*/

35 BigInteger RSA::encryptByPublic(const BigInteger & m) {

36 return m.modPow(e, n);

37 }

38

39 /\*\*

40 \* 函数功能:使用私钥进行解密

41 \* 参数含义:c表示要解密的密文

42 \*/

43 BigInteger RSA::decryptByPrivate(const BigInteger & c) {

44 return c.modPow(d, n);

45 }

46

47 /\*\*

48 \* 函数功能:使用私钥进行加密

49 \* 参数含义:m表示要加密的明文

50 \*/

51 BigInteger RSA::encryptByPrivate(const BigInteger & m) {

52 return decryptByPrivate(m);

53 }

54

55 /\*\*

56 \* 函数功能:使用公钥进行解密

57 \* 参数含义:c表示要解密的密文

58 \*/

59 BigInteger RSA::decryptByPublic(const BigInteger & c) {

60 return encryptByPublic(c);

61 }

62

63 /\*\*

64 \* 函数功能:输出RSA相关数据

65 \* 参数含义:out表示输出流,rsa表示要输出的RSA对象

66 \*/

67 std::ostream & operator << (std::ostream & out, const RSA & rsa) {

68 out << "n: " << rsa.n << "\n";

69 out << "p: " << rsa.p << "\n";

70 out << "q: " << rsa.q << "\n";

71 out << "e: " << rsa.e << "\n";

72 out << "d: " << rsa.d;

73 return out;

74 }

75

76 /\*\*

77 \* 函数功能:生成一个长度为len的奇数

78 \* 参数含义:len代表奇数的二进制长度

79 \*/

80 BigInteger RSA::createOddNum(unsigned len) {

81 static const char hex\_table[] = {'0', '1', '2', '3', '4', '5', '6', '7',

82 '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};

83 len >>= 2; // 十六进制数据,每位占4位二进制

84 if (len) {

85 std::ostringstream oss;

86 for (size\_t i=0; i<len-1; ++i)

87 oss << hex\_table[rand()%16];

88 oss << hex\_table[1];// 最后一位为奇数

89 return BigInteger(oss.str());

90 }

91 return BigInteger("F");

92 }

93

94 /\*\*

95 \* 函数功能:判断一个数是否为素数,采用米勒拉宾大素数检测算法,失误率为(1/4)^k

96 \* 参数含义:num代表要判定的数,k代表测试次数

97 \*/

98 bool RSA::isPrime(const BigInteger & num, const unsigned k) {

99 assert(num != BigInteger::ZERO);// 测试num是否为0

100 if (num == BigInteger::ONE)

101 return false; // 1不是素数

102 if (num == BigInteger::TWO)

103 return true; // 2是素数

104

105 BigInteger t = num-1;

106 BigInteger::bit b(t);// 二进制数

107 if (b.at(0) == 1) // 减一之后为奇数,原数为偶数

108 return false;

109 // num-1 = 2^s\*d

110 size\_t s = 0; // 统计二进制末尾有几个0

111 BigInteger d(t);

112 for (size\_t i=0; i<b.size(); ++i) {

113 if (!b.at(i)) {

114 ++s;

115 d = d.shiftRight(1);// 计算出d

116 }

117 else

118 break;

119 }

120

121 for (size\_t i=0; i<k; ++i) {// 测试k次

122 BigInteger a = createRandomSmaller(num);// 生成一个介于[1,num-1]之间的随机数a

123 BigInteger x = a.modPow(d, num);

124 if (x == BigInteger::ONE)// 可能为素数

125 continue;

126 bool ok = true;

127 // 测试所有0<=j<s,a^(2^j\*d) mod num != -1

128 for (size\_t j=0; j<s && ok; ++j) {

129 if (x == t)

130 ok = false; // 有一个相等,可能为素数

131 x = x.multiply(x).mod(num);

132 }

133 if (ok) // 确实都不等,一定为合数

134 return false;

135 }

136 return true; // 通过所有测试,可能为素数

137 }

138

139 /\*\*

140 \* 函数功能:随机生成一个比val小的数

141 \* 参数含义:val代表比较的那个数

142 \*/

143 BigInteger RSA::createRandomSmaller(const BigInteger & val) {

144 BigInteger::base\_t t = 0;

145 do {

146 t = rand();

147 } while (t == 0);// 随机生成非0数

148

149 BigInteger mod(t);

150 BigInteger ans = mod%val; // 比val要小

151 if (ans == BigInteger::ZERO)// 必须非零

152 ans = val-BigInteger::ONE;

153 return ans;

154 }

155

156 /\*\*

157 \* 函数功能:生成一个二进制长度为len的大素数

158 \* 参数含义:len代表大素数的长度,k代表素数检测的次数

159 \*/

160 BigInteger RSA::createPrime(unsigned len, const unsigned k) {

161 assert(k > 0);

162 BigInteger ans = createOddNum(len);// 先生成一个奇数

163 while (!isPrime(ans, k)) {// 素性检测

164 ans = ans.add(BigInteger::TWO);// 下一个奇数

165 }

166 return ans;

167 }

168

169 /\*\*

170 \* 函数功能:根据提供的欧拉数生成公钥、私钥指数

171 \* 参数含义:eul表示提供的欧拉数

172 \*/

173 void RSA::createExponent(const BigInteger & eul) {

174 e = 65537;

175 d = e.modInverse(eul);

176 }

[复制代码](javascript:void(0);)

**2.3 EncryptDecrypt类**

    该类主要用于封装加解密的一些操作，比如判断数据是否合法，输入输出加解密信息等。

    该模块同样有两个文件：EncryptDecrypt.h和EncryptDecrypt.cpp。代码如下所示。

    EncryptDecrypt.h代码如下。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 /\*\*

2 \* @Name : EncryptDecrypt.h

3 \* @Date : 2017-04-11-22.29.58

4 \* @Author : Silenceneo (silenceneo\_xw@163.com)

5 \* @Link : http://www.cnblogs.com/Silenceneo-xw/

6 \* @Version : 2.0

7 \*/

8

9 #ifndef \_\_ENCRYPTDECRYPT\_H\_\_

10 #define \_\_ENCRYPTDECRYPT\_H\_\_

11

12 #include <string>

13 #include "RSA.h"

14 class EncryptDecrypt {

15 public:

16 EncryptDecrypt() {}

17 ~EncryptDecrypt() {}

18

19 void menu(); // 菜单显示

20 bool encrypt(); // 加密

21 bool decrypt(); // 解密

22 void print(); // 打印RSA相关信息

23 void reset(); // 重置RSA相关信息

24 protected:

25 void load(int); // 根据给定位数加载RSA对象

26 bool islegal(const std::string &);// 判断输入字符串是否合法

27 private:

28 RSA rsa;

29 };

30

31 #endif // \_\_ENCRYPTDECRYPT\_H\_\_

[复制代码](javascript:void(0);)

    EncryptDecrypt.cpp代码如下。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 /\*\*

2 \* @Name : EncryptDecrypt.cpp

3 \* @Date : 2017-04-11-22.32.18

4 \* @Author : Silenceneo (silenceneo\_xw@163.com)

5 \* @Link : http://www.cnblogs.com/Silenceneo-xw/

6 \* @Version : 2.0

7 \*/

8

9 #include <iostream>

10 #include <ctime>

11 #include "EncryptDecrypt.h"

12

13 /\*\*

14 \* 函数功能:菜单显示

15 \*/

16 void EncryptDecrypt::menu() {

17 std::cout << "\*\*\*\*\*\*\*\*\*\*Welcome to use RSA encoder\*\*\*\*\*\*\*\*\*\*" << std::endl;

18 std::cout << " e: encrypt 加密 " << std::endl;

19 std::cout << " d: decrypt 解密 " << std::endl;

20 std::cout << " p: print 显示 " << std::endl;

21 std::cout << " r: reset 重置 " << std::endl;

22 std::cout << " q: quit 退出 " << std::endl;

23 std::cout << "input your choice:" << std::endl;

24 }

25

26 /\*\*

27 \* 函数功能:加密运算

28 \*/

29 bool EncryptDecrypt::encrypt() {

30 std::string str;

31 std::cout << "输入16进制数据:" << std::endl;

32 std::cout << ">";

33 std::cin >> str;// 输入明文

34 if (!std::cin || !islegal(str))

35 return false;

36 BigInteger m(str);

37 clock\_t start = clock();

38 BigInteger c = rsa.encryptByPublic(m);

39 clock\_t finish = clock();

40

41 std::cout << std::fixed;

42 std::cout.precision(3);

43 std::cout << "用时: " << (double)(finish-start)/CLOCKS\_PER\_SEC << "s." << std::endl;

44 std::cout << "明文: " << m << std::endl;

45 std::cout << "密文: " << c << std::endl;

46 return true;

47 }

48

49 /\*\*

50 \* 函数功能:解密运算

51 \*/

52 bool EncryptDecrypt::decrypt() {

53 std::string str;

54 std::cout << "输入16进制数据:" << std::endl;

55 std::cout << ">";

56 std::cin >> str;// 输入密文

57 if (!std::cin || !islegal(str))

58 return false;

59 BigInteger c(str);

60 clock\_t start = clock();

61 BigInteger m = rsa.decryptByPrivate(c);

62 clock\_t finish = clock();

63

64 std::cout << std::fixed;

65 std::cout.precision(3);

66 std::cout << "用时: " << (double)(finish-start)/CLOCKS\_PER\_SEC << "s." << std::endl;

67 std::cout << "密文: " << c << std::endl;

68 std::cout << "明文: " << m << std::endl;

69 return true;

70 }

71

72 /\*\*

73 \* 函数功能:输出RSA相关信息

74 \*/

75 void EncryptDecrypt::print() {

76 std::cout << rsa << std::endl;

77 }

78

79 /\*\*

80 \* 函数功能:重置RSA相关信息

81 \*/

82 void EncryptDecrypt::reset() {

83 std::cout << "输入密钥长度: ";

84 int len;

85 std::cin >> len;

86 load(len>>1);

87 }

88

89 /\*\*

90 \* 函数功能:根据给定位数len加载rsa

91 \*/

92 void EncryptDecrypt::load(int len) {

93 std::cout << "初始化..." << std::endl;

94 clock\_t start = clock();

95 rsa.init(len); // 初始化

96 clock\_t finish = clock();

97 std::cout << "初始化完成." << std::endl;

98 std::cout << std::fixed;

99 std::cout.precision(3);

100 std::cout << "用时: " << (double)(finish-start)/CLOCKS\_PER\_SEC << "s." << std::endl;

101 }

102

103 /\*\*

104 \* 函数功能:判断输入字符串str是否合法

105 \* 参数含义:str代表输入的字符串

106 \*/

107 bool EncryptDecrypt::islegal(const std::string & str) {

108 for (std::string::const\_iterator it=str.begin(); it!=str.end(); ++it) {

109 if (!isalnum(\*it)) // 不是字母或者数字

110 return false;

111 if (isalpha(\*it)) {

112 char ch = tolower(\*it);

113 if (ch > 'f') // 超过十六进制字符'f'

114 return false;

115 }

116 }

117 return true;

118 }

[复制代码](javascript:void(0);)

**2.4 主文件**

    该模块就只有一个文件：Main.cpp。主要用于使各个模块结合起来，执行相应操作。

    Main.cpp代码如下。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 /\*\*

2 \* @Name : Main.cpp

3 \* @Date : 2017-04-11-22.19.24

4 \* @Author : Silenceneo (silenceneo\_xw@163.com)

5 \* @Link : http://www.cnblogs.com/Silenceneo-xw/

6 \* @Version : 2.0

7 \*/

8

9 #include <iostream>

10 #include "EncryptDecrypt.h"

11

12 int main() {

13 EncryptDecrypt encrypt\_decrypt;

14 encrypt\_decrypt.reset();// 设置密钥长度

15

16 char ch;

17 std::string str;

18 bool ok = true;

19

20 do {

21 encrypt\_decrypt.menu();// 菜单显示

22 std::cout << ">";

23 std::cin >> str;

24 if (str.empty()) {

25 std::cout << "输入错误!请重新输入!" << std::endl;

26 continue;

27 }

28 ch = str.at(0);

29 switch(ch) {

30 case 'e':

31 case 'E':

32 if (!encrypt\_decrypt.encrypt())

33 std::cout << "加密失败,请重试!" << std::endl;

34 break;

35 case 'd':

36 case 'D':

37 if (!encrypt\_decrypt.decrypt())

38 std::cout << "解密失败,请重试!" << std::endl;

39 break;

40 case 'p':

41 case 'P':

42 encrypt\_decrypt.print();// 打印相关信息

43 break;

44 case 'r':

45 case 'R':

46 encrypt\_decrypt.reset();// 重新设置密钥长度

47 break;

48 case 'q':

49 case 'Q':

50 ok = false; // 退出

51 break;

52 default:

53 break;

54 }

55 } while (ok);

56 return 0;

57 }

[复制代码](javascript:void(0);)

    至此，RSA加密算法已经介绍完毕了，如有任何错误，欢迎指出，不胜感激。

**附录**

**RSA加密算法 Java版本**

    附上Java版本的RSA加密算法，仅供参考。

    RSA.java代码如下。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 import java.math.BigInteger;

2 import java.util.Random;

3

4 public class RSA {

5 public BigInteger n, e;// 公钥

6 private BigInteger d;// 私钥

7 private BigInteger p, q;

8 private BigInteger eul;// n的欧拉函数

9

10 /\*\*

11 \* 函数功能:初始化RSA对象的相关信息

12 \* 参数含义:len表示大素数的二进制位数

13 \*/

14 public void init(int len) {

15 Random random = new Random();

16 // 产生大素数p和q

17 p = BigInteger.probablePrime(len, random);

18 q = BigInteger.probablePrime(len, random);

19 // 计算出n

20 n = p.multiply(q);

21 // 计算出n的欧拉函数

22 eul = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));

23 // 设置加解密指数e和d

24 createExponent(eul);

25 }

26

27 /\*\*

28 \* 函数功能:使用公钥进行加密

29 \* 参数含义:m表示要加密的明文

30 \*/

31 public BigInteger encryptByPublic(BigInteger m) {

32 return m.modPow(e, n);

33 }

34

35 /\*\*

36 \* 函数功能:使用私钥进行解密

37 \* 参数含义:c表示要解密的密文

38 \*/

39 public BigInteger decryptByPrivate(BigInteger c) {

40 return c.modPow(d, n);

41 }

42

43 // 以下主要用于数字签名

44 /\*\*

45 \* 函数功能:使用私钥进行加密

46 \* 参数含义:m表示要加密的明文

47 \*/

48 public BigInteger encryptByPrivate(BigInteger m) {

49 return decryptByPrivate(m);

50 }

51

52 /\*\*

53 \* 函数功能:使用公钥进行解密

54 \* 参数含义:c表示要解密的密文

55 \*/

56 public BigInteger decryptByPublic(BigInteger c) {

57 return encryptByPublic(c);

58 }

59

60 /\*\*

61 \* 函数功能:从一个欧拉数中生成公钥、私钥指数

62 \* 参数含义:eul表示提供的欧拉数

63 \*/

64 private void createExponent(BigInteger eul) {

65 // TODO Auto-generated method stub

66 e = new BigInteger("65537");

67 d = e.modInverse(eul);

68 }

69

70 /\*\*

71 \* 函数功能:输出RSA相关数据

72 \*/

73 public void print() {

74 System.out.println("n: " + n);

75 System.out.println("p: " + p);

76 System.out.println("q: " + q);

77 System.out.println("e: " + e);

78 System.out.println("d: " + d);

79 }

80 }

[复制代码](javascript:void(0);)

    EncryptDecrypt.java代码如下。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 import java.math.BigInteger;

2 import java.util.Scanner;

3

4 public class EncryptDecrypt {

5 private RSA rsa = new RSA();

6 public Scanner in = new Scanner(System.in);

7

8 /\*\*

9 \* 函数功能:菜单显示

10 \*/

11 public void menu() {

12 System.out.println("\*\*\*\*\*\*\*\*\*\*Welcome to use RSA encoder\*\*\*\*\*\*\*\*\*\*");

13 System.out.println(" e: encrypt 加密 ");

14 System.out.println(" d: decrypt 解密 ");

15 System.out.println(" p: print 显示 ");

16 System.out.println(" r: reset 重置 ");

17 System.out.println(" q: quit 退出 ");

18 System.out.println("input your choice:");

19 }

20

21 /\*\*

22 \* 函数功能:加密运算

23 \*/

24 public boolean encrypt() {

25 System.out.println("输入10进制数据:");

26 System.out.print(">");

27 String str = in.next();

28 if (str==null || str.isEmpty() || !islegal(str))

29 return false;

30 BigInteger m = new BigInteger(str);

31 long start = System.currentTimeMillis();

32 BigInteger c = rsa.encryptByPublic(m);

33 long finish = System.currentTimeMillis();

34 System.out.println("用时:" + (finish-start) + "ms.");

35 System.out.println("明文: " + m);

36 System.out.println("密文: " + c);

37 return true;

38 }

39

40 /\*\*

41 \* 函数功能:解密运算

42 \*/

43 public boolean decrypt() {

44 System.out.println("输入10进制数据:");

45 System.out.print(">");

46 String str = in.next();

47 if (str==null || str.isEmpty() || !islegal(str))

48 return false;

49 BigInteger c = new BigInteger(str);

50 long start = System.currentTimeMillis();

51 BigInteger m = rsa.decryptByPrivate(c);

52 long finish = System.currentTimeMillis();

53 System.out.println("用时:" + (finish-start) + "ms.");

54 System.out.println("密文: " + c);

55 System.out.println("明文: " + m);

56 return true;

57 }

58

59 /\*\*

60 \* 函数功能:输出RSA相关信息

61 \*/

62 public void print() {

63 rsa.print();

64 }

65

66 /\*\*

67 \* 函数功能:重置RSA相关信息

68 \*/

69 public void reset() {

70 System.out.print("输入密钥长度: ");

71 int len = in.nextInt();

72 load(len>>1);

73 }

74

75 /\*\*

76 \* 函数功能:根据给定位数len加载rsa

77 \*/

78 private void load(int len) {

79 // TODO Auto-generated method stub

80 System.out.println("初始化...");

81 long start = System.currentTimeMillis();

82 rsa.init(len);

83 long finish = System.currentTimeMillis();

84 System.out.println("初始化完成.");

85 System.out.println("用时:" + (finish-start) + "ms.");

86 }

87

88 /\*\*

89 \* 函数功能:判断输入字符串str是否合法

90 \* 参数含义:str代表输入的字符串

91 \*/

92 private boolean islegal(String str) {

93 // TODO Auto-generated method stub

94 for (int i = 0; i < str.length(); i++) {

95 char ch = str.charAt(i);

96 if (!Character.isDigit(ch)) {

97 return false;

98 }

99 }

100 return true;

101 }

102 }

[复制代码](javascript:void(0);)

    Main.java代码如下。

https://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 public class Main {

2

3 public static void main(String[] args) {

4 // TODO Auto-generated method stub

5 EncryptDecrypt encryptDecrypt = new EncryptDecrypt();

6 encryptDecrypt.reset();

7

8 do {

9 encryptDecrypt.menu();

10 System.out.print(">");

11 String str = encryptDecrypt.in.next();

12 if (str==null || str.isEmpty()) {

13 System.out.println("输入错误,请重新输入!");

14 continue;

15 }

16 char ch = str.charAt(0);

17 switch(ch) {

18 case 'e':

19 case 'E':

20 if (!encryptDecrypt.encrypt()) {

21 System.out.println("加密失败,请重试!");

22 }

23 break;

24 case 'd':

25 case 'D':

26 if (!encryptDecrypt.decrypt()) {

27 System.out.println("解密失败,请重试!");

28 }

29 break;

30 case 'p':

31 case 'P':

32 encryptDecrypt.print();

33 break;

34 case 'r':

35 case 'R':

36 encryptDecrypt.reset();

37 break;

38 case 'q':

39 case 'Q':

40 encryptDecrypt.in.close();

41 System.exit(0);

42 break;

43 default:

44 break;

45 }

46 } while (true);

47 }

48

49 }

[复制代码](javascript:void(0);)