

RSA 算法报告

- 米家龙
- 18342075
- 数据科学与计算机学院

目录

- RSA 算法报告
 - 目录
 - 原理描述
 - 数据结构设计
 - 密钥生成
 - 解编码
 - 加密
 - 解密
 - C 语言代码
 - 编译运行结果

原理描述

主要原理：欧拉定理

对于互素的 a 和 m ，有 $a^{\varphi(m)} \equiv 1 \pmod{m}$

对于

- 满足 $N = pq$ 的两个不同的素数 p 和 q
- 满足 $0 < n < N$ 的整数 n, k 是正整数，有 $n^{k\varphi(N)+1} \equiv n \pmod{N}$

由于 $ed \equiv 1 \pmod{\varphi(N)}$ ，即 $ed = k\varphi(N) + 1$ ，所以 $n^{ed} = n^{k\varphi(N)+1} \equiv n \pmod{N}$

而对于现在存在 $c = n^e \pmod{N}$ ， $n' = c^d \pmod{N}$

应用模算数运算规则得到 $n' = c^d \pmod{N} = (n^e)^d \pmod{N} = n^{ed} \pmod{N} = n \pmod{N}$ ，即 $n' \equiv n \pmod{N}$

数据结构设计

本次使用了 gmp 库作为大数运算的依据

具体的变量和函数声明如下：

```
1  mpz_t p, q;           // 两个素数
2  mpz_t n;              // n = p * q
3  mpz_t phiN;           // phi(n) = (p - 1) * (q - 1)
4  mpz_t e;              // 公钥为 (n, e)
5  mpz_t d;              // e 的逆元，需要满足 ed mod phi(n) = 1
```

```

6 // 用于表示私钥 (n, d)
7 gmp_randstate_t greatRandomNumber; // 随机生成的大数
8
9 mpz_t M, C; // M 是明文, C 是加密结果
10 mpz_t M2; // 用于解密后的明文储存
11
12 char *Message; // 明文字符串
13 char *Message2; // 解密后的明文字符串
14 char *PS; // 伪随机生成字符串
15 char *EM; // 填充后的明文
16 char *crypteText; // 加密后的字符串
17 int mLen = 0; // 明文长度
18 unsigned long long k; // 长度限制, n 的字节数
19 FILE *originFile; // 原始数据
20 FILE *encryptedFile; // 加密后的数据
21 FILE *decryptedFile; // 解密后的数据
22 FILE *publicKeyFile; // 公钥文件
23 FILE *privateKeyFile; // 私钥文件
24
25 /**
26  * 生成 p, q, n, phi(n), e 等参数
27  * 通过循环进行生成合适的密钥
28  * @param bit int 要求的 k 的位数
29  */
30 void generateKey(int bit);
31
32 /**
33  * 清除所有大数
34  */
35 void clearAll();
36
37 /**
38  * 生成 PS 字符串
39  */
40 void getPS(int mLen);
41
42 /**
43  * 得到填充后的明文
44  */
45 void getEM();
46
47 /**
48  * 将字符串转换为大数
49  * @param dst mpz_t 目标大数
50  * @param src char* 源字符串
51  * @param length int 长度, 一般用 k 来作为参数
52  */
53 void OS2IP(mpz_t dst, char src[], unsigned long length);
54
55 /**
56  * 将大数转换为字符串
57  * @param src mpz_t 源大数
58  * @param dst char* 目标字符串
59  */
60 void I2OSP(char dst[], mpz_t src);
61
62 /**
63  * 解密函数

```

```

64  */
65  void decode();
66
67  /**
68   * 加密函数
69   */
70  void encode();
71
72  /**
73   * 由于一开始 k 不确定
74   * 因此需要 malloc 函数进行字符串的空间申请
75   */
76  void initString();
77
78  /**
79   * 加密过程
80   * @param publicKeyFilePath char* 公钥文件路径
81   * @param privateKeyFilePath char* 私钥文件路径
82   * @param originFilePath char* 原始文本路径
83   * @param encryptedFilePath char* 加密后的文本路径
84   */
85  void encryption(char publicKeyFilePath[], char privateKeyFilePath[],
86                  char originFilePath[], char encryptedFilePath[]);
87
88  /**
89   * 解密过程
90   * @param privateKeyFilePath char* 私钥文件路径
91   * @param encryptedFilePath char* 加密后的文本路径
92   * @param decryptedFilePath char* 解密后的文本路径
93   */
94  void decryption(char privateKeyFilePath[], char encryptedFilePath[],
95                  char decryptedFilePath[]);
96

```

密钥生成

- 选择两个不同的大素数 p 和 q ，计算 $n = pq$
- 得到 $\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1)$
- 选择一个整数 e ，满足 $1 < e < \varphi(N)$ 并且 $\gcd(e, \varphi(n)) = 1$
- 找到一个足够大的正整数 d ，满足 $ed \equiv 1 \pmod{\varphi(n)}$ ，可以通过拓展欧几里得算法得到
- 得到公钥 (n, e) 和私钥 (n, d)

具体源代码如下：

```

1  /**
2   * 生成 p, q, n, phi(n), e 等参数
3   * 通过循环进行生成合适的密钥
4   * @param bit int 要求的 k 的位数
5   */
6  void generateKey(int bit);
7
8  void generateKey(int bit)
9  {
10     while (1)
11     {

```

```

12     // 随机生成大数
13     gmp_randinit_default(greatRandomNumber);
14     gmp_randseed_ui(greatRandomNumber, time(NULL));
15
16     // 初始化 p, q
17     mpz_init(p);
18     mpz_init(q);
19
20     // 随机生成两个大数
21     // mpz_urandomb(p, greatRandomNumber, (bit + 1) / 2);
22     // mpz_urandomb(q, greatRandomNumber, (bit - 1) / 2);
23
24     mpz_urandomb(p, greatRandomNumber, bit / 2 - 1);
25     mpz_urandomb(q, greatRandomNumber, bit / 2 + 1);
26
27     // 素数生成
28     mpz_nextprime(p, p);
29     mpz_nextprime(q, q);
30
31     // 得到 n
32     mpz_init(n);
33     mpz_mul(n, p, q);
34
35     if (mpz_sizeinbase(n, 2) == bit) // 用于判断是否生成合适的位数
36     {
37         break;
38     }
39 }
40
41 // 计算 phi(n)
42 mpz_init(phiN);
43 mpz_sub_ui(p, p, 1);
44 mpz_sub_ui(q, q, 1);
45 mpz_mul(phiN, p, q);
46
47 gmp_printf("p: %d q: %d n: %d\n", mpz_sizeinbase(p, 2), mpz_sizeinbase(q, 2),
mpz_sizeinbase(n, 2));
48
49 // 公钥
50 // e 通常取 3, 17, 65537
51 mpz_init_set_ui(e, 65537);
52 gmp_printf("Public key is: (%ZX, %ZX)\n\n", n, e);
53
54 // 私钥
55 mpz_init(d);
56 mpz_invert(d, e, phiN); // 求逆元
57 gmp_printf("Private key is: (%ZX, %ZX)\n\n", n, d);
58
59 initString();
60 }

```

解编码

根据公私钥中的 n ，获得 n 的字节数 k

- 构建 PS，长度为 $k - mLen - 3$ ，其中每个字节都是值都是 1~255 的随机数
- 构建 $EM = 0x00 \parallel 0x22 \parallel PS \parallel 0x00 \parallel message$

具体源代码如下：

```

1  void getPS(int mLen)
2  {
3      PS = (char *)malloc(k - mLen - 2);
4      PS[k - mLen - 3] = 0;
5
6      for (int i = 0; i < k - mLen - 3; i++)
7      {
8          PS[i] = rand() % 255 + 1;
9      }
10 }
11
12 void getEM()
13 {
14     // 各个部分的长度
15     int p1 = 1, p2 = 1, p3 = k - mLen - 3, p4 = 1, p5 = mLen;
16
17     // part 1
18     EM[0] = 0;
19
20     // part 2
21     EM[1] = 2;
22
23     // part 3
24     for (int i = 0; i < p3; i++)
25     {
26         EM[i + p1 + p2] = PS[i];
27     }
28
29     // part 4
30     EM[p1 + p2 + p3] = 0;
31
32     // part 5
33     for (int i = 0; i < p5; i++)
34     {
35         EM[p1 + p2 + p3 + p4 + i] = Message[i];
36     }
37
38     EM[k] = 0;
39 }

```

加密

使用公钥 (n, e)

- 填充
 - 得到公钥中 n 的字节数 k
 - 要求明文 message 字节数 $mLen < k - 11$
 - 根据解编码的规则，获得 EM
- OS2IP：

- 对于长度为 k 的 EM 存在如下格式 $X_0 X_1 + \dots + X_{k-1}$
 - 得到明文大数 $M = X_0 + X_1 * 256 + X_2 * 256^2 + \dots + X_{k-2} * 256^{k-2} + X_{k-1} * 256^{k-1}$
- 加密:
 - $C = M^e \bmod n$
- I2OSP:
 - 根据编码过程和加密后的大数

$$C = X_0 + X_1 * 256 + X_2 * 256^2 + \dots + X_{k-2} * 256^{k-2} + X_{k-1} * 256^{k-1}$$
 - 逆向得到长度为 k 的 crypteText $X_0 X_1 + \dots + X_{k-1}$
 - 输出密文 crypteText

具体代码如下:

```

1 void encode()
2 {
3     getPS(mLen);
4     getEM();
5     OS2IP(M, EM, k);
6     mpz_powm(C, M, e, n);
7     I2OSP(crypteText, C);
8 }
```

其中 OS2IP 和 I2OSP 的具体实现如下:

```

1 /**
2  * 将字符串转换为大数
3  * @param dst mpz_t 目标大数
4  * @param src char* 源字符串
5  * @param length int 长度, 一般用 k 来作为参数
6  */
7 void OS2IP(mpz_t dst, char src[], unsigned long length);
8
9 /**
10 * 将大数转换为字符串
11 * @param src mpz_t 源大数
12 * @param dst char* 目标字符串
13 */
14 void I2OSP(char dst[], mpz_t src);
15
16 void OS2IP(mpz_t dst, char src[], unsigned long length)
17 {
18     mpz_init(dst);
19     for (int i = length - 1; i >= 0; i--)
20     {
21         mpz_mul_ui(dst, dst, 256);
22         mpz_add_ui(dst, dst, src[i] & 0x0000ff);
23     }
24     // src[length] = 0;
25 }
26
27 void I2OSP(char dst[], mpz_t src)
28 {
29     unsigned long long size = mpz_sizeinbase(src, 2);
30     int xLen = size / 8 + (size % 8 ? 1 : 0);
31     free(dst);
32     dst = (char *)malloc(xLen + 1);
33     mpz_t tmp, copy;
```

```

34     mpz_init_set(copy, src);
35
36     // 循环生成字符串
37     for (int i = 0; i < xLen; i++)
38     {
39         mpz_init(tmp);
40         mpz_mod_ui(tmp, copy, 256);
41         dst[i] = mpz_get_ui(tmp) & 0x0000ff;
42         mpz_div_ui(copy, copy, 256);
43     }
44 }

```

解密

使用私钥 (n, d)

- 得到公钥中 n 的字节数 k
- 要求密文的字节数需要为 k
- 编码 (OS2IP) : 同上
- 解密:
 - $M = C^d \bmod n$
- 解码 (I2OSP) : 同上
- 得到的 EM = 0x00 || 0x22 || PS || 0x00 || message
- 输出解密结果 message

具体实现代码如下:

```

1  void decode()
2  {
3      OS2IP(C, cryptedText, k);
4      mpz_init(M2);
5      mpz_powm(M2, C, d, n);
6      I2OSP(Message2, M2);
7  }

```

C 语言代码

完整代码如下, 也可查看同文件下的 rsa.c 代码文件:

```

1  #include <stdio.h>
2  #include <gmp.h>
3  #include <string.h>
4  #include <stdlib.h>
5
6  mpz_t p, q;           // 两个素数
7  mpz_t n;              // n = p * q
8  mpz_t phiN;           // phi(n) = (p - 1) * (q - 1)
9  mpz_t e;              // 公钥为 (n, e)
10 mpz_t d;              // e 的逆元, 需要满足 ed mod phi(n) = 1
11                      // 用于表示私钥 (n, d)

```

```

12  gmp_randstate_t greatRandomNumber; // 随机生成的大数
13
14  mpz_t M, C; // M 是明文, C 是加密结果
15  mpz_t M2;   // 用于解密后的明文储存
16
17  char *Message;      // 明文字符串
18  char *Message2;     // 解密后的明文字符串
19  char *PS;           // 伪随机生成字符串
20  char *EM;           // 填充后的明文
21  char *crypteText;   // 加密后的字符串
22  int mLen = 0;       // 明文长度
23  unsigned long long k; // 长度限制, n 的字节数
24  FILE *originFile;   // 原始数据
25  FILE *encryptedFile; // 加密后的数据
26  FILE *decryptedFile; // 解密后的数据
27  FILE *publicKeyFile; // 公钥文件
28  FILE *privateKeyFile; // 私钥文件
29
30  /**
31   * 生成 p, q, n, phi(n), e 等参数
32   * 通过循环进行生成合适的密钥
33   * @param bit int 要求的 k 的位数
34   */
35  void generateKey(int bit);
36
37  /**
38   * 清除所有大数
39   */
40  void clearAll();
41
42  /**
43   * 生成 PS 字符串
44   */
45  void getPS(int mLen);
46
47  /**
48   * 得到填充后的明文
49   */
50  void getEM();
51
52  /**
53   * 将字符串转换为大数
54   * @param dst mpz_t 目标大数
55   * @param src char* 源字符串
56   * @param length int 长度, 一般用 k 来作为参数
57   */
58  void OS2IP(mpz_t dst, char src[], unsigned long length);
59
60  /**
61   * 将大数转换为字符串
62   * @param src mpz_t 源大数
63   * @param dst char* 目标字符串
64   */
65  void I2OSP(char dst[], mpz_t src);
66
67  /**
68   * 解密函数
69   */

```



```

70 void decode();
71
72 /**
73  * 加密函数
74  */
75 void encode();
76
77 /**
78  * 由于一开始 k 不确定
79  * 因此需要 malloc 函数进行字符串的空间申请
80  */
81 void initString();
82
83 /**
84  * 加密过程
85  * @param publicKeyFilePath char* 公钥文件路径
86  * @param privateKeyFilePath char* 私钥文件路径
87  * @param originFilePath char* 原始文本路径
88  * @param encryptedFilePath char* 加密后的文本路径
89  */
90 void encryption(char publicKeyFilePath[], char privateKeyFilePath[],
91                 char originFilePath[], char encryptedFilePath[]);
92
93 /**
94  * 解密过程
95  * @param privateKeyFilePath char* 私钥文件路径
96  * @param encryptedFilePath char* 加密后的文本路径
97  * @param decryptedFilePath char* 解密后的文本路径
98  */
99 void decryption(char privateKeyFilePath[], char encryptedFilePath[],
100                char decryptedFilePath[]);
101
102 int main(int argc, char *argv[])
103 {
104     // 如果是加密
105     if (strcmp(argv[1], "enc") == 0 && argc == 6)
106     {
107         encryption(argv[2], argv[3], argv[4], argv[5]);
108     }
109     // 如果是解密
110     else if (strcmp(argv[1], "dec") == 0 && argc == 5)
111     {
112         decryption(argv[2], argv[3], argv[4]);
113     }
114     else if (argc == 7 && strcmp(argv[1], "test") == 0) // 两个都进行
115     {
116         encryption(argv[2], argv[3], argv[4], argv[5]);
117         decryption(argv[3], argv[5], argv[6]);
118         mpz_t eq;
119         mpz_init(eq);
120         mpz_sub(eq, M, M2);
121         gmp_printf("M - M2 = %ZX\n", eq);
122     }
123     else
124     {
125         printf("usage: \n\t./a.out [enc publicKeyFile privateKeyFile | dec\n\tprivateKeyFile] inputFile outFile\n");
126         printf("OR\n");

```

```

127     printf("\t./a.out test publicKeyFile privateKeyFile inputFile crypteFile
decryptedFile\n");
128     return 0;
129 }
130
131 clearAll();
132
133 return 0;
134 }
135
136 void decryption(char privateKeyFilePath[], char encryptedFilePath[],
137                char decryptedFilePath[])
138 {
139     privateKeyFile = fopen(privateKeyFilePath, "r");
140     encryptedFile = fopen(encryptedFilePath, "r");
141     decryptedFile = fopen(decryptedFilePath, "w");
142     gmp_fscanf(privateKeyFile, "%ZX\nZX", n, d);
143
144     unsigned long long size = mpz_sizeinbase(n, 2);
145     k = size / 8 + (size % 8 ? 1 : 0);
146     initString();
147     fread(cryptedText, 1, k, encryptedFile);
148     decode();
149     gmp_printf("n = %ZX\nd = %ZX\nk = %llu\nM2 = %ZX\n", n, d, k, M2);
150     int startAt; // 用于判断 M 的位置
151     for (startAt = 1; startAt < k; startAt++)
152     {
153         if (Message2[startAt] == 0)
154         {
155             break;
156         }
157     }
158
159     for (startAt = startAt + 1; startAt < k; startAt++)
160     {
161         fputc(Message2[startAt], decryptedFile);
162     }
163
164     fclose(privateKeyFile);
165     fclose(encryptedFile);
166     fclose(decryptedFile);
167 }
168
169 void encryption(char publicKeyFilePath[], char privateKeyFilePath[],
170                char originFilePath[], char encryptedFilePath[])
171 {
172     publicKeyFile = fopen(publicKeyFilePath, "w");
173     privateKeyFile = fopen(privateKeyFilePath, "w");
174     originFile = fopen(originFilePath, "r");
175     encryptedFile = fopen(encryptedFilePath, "w");
176
177     generateKey(1024);
178
179     gmp_fprintf(publicKeyFile, "%ZX\nZX", n, e); // 输出公钥
180     gmp_fprintf(privateKeyFile, "%ZX\nZX", n, d); // 输出私钥
181
182     mLen = fread(Message, 1, k, originFile); // 读取文本
183     if (mLen > (k - 11)) // 明文太长

```

```

184     {
185         printf("明文太长\n");
186         exit(1);
187     }
188     encode(); // 进行加密
189
190     for (int i = 0; i < k; i++)
191     {
192         fputc(encryptedText[i], encryptedFile);
193     }
194
195     gmp_printf("M = %ZX\n", M);
196
197     fclose(originFile);
198     fclose(privateKeyFile);
199     fclose(publicKeyFile);
200     fclose(encryptedFile);
201 }
202
203 void decode()
204 {
205     OS2IP(C, cryptedText, k);
206     mpz_init(M2);
207     mpz_powm(M2, C, d, n);
208     I2OSP(Message2, M2);
209 }
210
211 void encode()
212 {
213     getPS(mLen);
214     getEM();
215     OS2IP(M, EM, k);
216     mpz_powm(C, M, e, n);
217     I2OSP(cryptedText, C);
218 }
219
220 void initString()
221 {
222     unsigned long long size = mpz_sizeinbase(n, 2);
223     k = size / 8 + (size % 8 ? 1 : 0);
224     Message = (char *)malloc(k * 10);
225     cryptedText = (char *)malloc(k * 1);
226     EM = (char *)malloc(k * 1);
227     Message2 = (char *)malloc(k * 1);
228
229     for (int i = 0; i < k; i++)
230     {
231         cryptedText[i] = Message2[i] = EM[i] = 0;
232     }
233 }
234
235 void OS2IP(mpz_t dst, char src[], unsigned long length)
236 {
237     mpz_init(dst);
238     for (int i = length - 1; i >= 0; i--)
239     {
240         mpz_mul_ui(dst, dst, 256);
241         mpz_add_ui(dst, dst, src[i] & 0x0000ff);

```

```

242     }
243 }
244
245 void I2OSP(char dst[], mpz_t src)
246 {
247     unsigned long long size = mpz_sizeinbase(src, 2);
248     int xLen = size / 8 + (size % 8 ? 1 : 0);
249     free(dst);
250     dst = (char *)malloc(xLen + 1);
251     mpz_t tmp, copy;
252     mpz_init_set(copy, src);
253
254     // 循环生成字符串
255     for (int i = 0; i < xLen; i++)
256     {
257         mpz_init(tmp);
258         mpz_mod_ui(tmp, copy, 256);
259         dst[i] = mpz_get_ui(tmp) & 0x0000ff;
260         mpz_div_ui(copy, copy, 256);
261     }
262 }
263
264 void getPS(int mLen)
265 {
266     PS = (char *)malloc(k - mLen - 2);
267     PS[k - mLen - 3] = 0;
268
269     for (int i = 0; i < k - mLen - 3; i++)
270     {
271         PS[i] = rand() % 255 + 1;
272     }
273 }
274
275 void getEM()
276 {
277     // 各个部分的长度
278     int p1 = 1, p2 = 1, p3 = k - mLen - 3, p4 = 1, p5 = mLen;
279
280     // part 1
281     EM[0] = 0;
282
283     // part 2
284     EM[1] = 2;
285
286     // part 3
287     for (int i = 0; i < p3; i++)
288     {
289         EM[i + p1 + p2] = PS[i];
290     }
291
292     // part 4
293     EM[p1 + p2 + p3] = 0;
294
295     // part 5
296     for (int i = 0; i < p5; i++)
297     {
298         EM[p1 + p2 + p3 + p4 + i] = Message[i];
299     }

```

```

300
301     EM[k] = 0;
302 }
303
304 void generateKey(int bit)
305 {
306     while (1)
307     {
308         // 随机生成大数
309         gmp_randinit_default(greatRandomNumber);
310         gmp_randseed_ui(greatRandomNumber, time(NULL));
311
312         // 初始化 p, q
313         mpz_init(p);
314         mpz_init(q);
315
316         // 随机生成两个大数
317         // mpz_urandomb(p, greatRandomNumber, (bit + 1) / 2);
318         // mpz_urandomb(q, greatRandomNumber, (bit - 1) / 2);
319
320         mpz_urandomb(p, greatRandomNumber, bit / 2 - 1);
321         mpz_urandomb(q, greatRandomNumber, bit / 2 + 1);
322
323         // 素数生成
324         mpz_nextprime(p, p);
325         mpz_nextprime(q, q);
326
327         // 得到 n
328         mpz_init(n);
329         mpz_mul(n, p, q);
330
331         if (mpz_sizeinbase(n, 2) == bit) // 用于判断是否生成合适的位数
332         {
333             break;
334         }
335     }
336
337     // 计算 phi(n)
338     mpz_init(phiN);
339     mpz_sub_ui(p, p, 1);
340     mpz_sub_ui(q, q, 1);
341     mpz_mul(phiN, p, q);
342
343     gmp_printf("p: %d q: %d n: %d\n", mpz_sizeinbase(p, 2), mpz_sizeinbase(q, 2),
344               mpz_sizeinbase(n, 2));
345
346     // 公钥
347     // e 通常取 3, 17, 65537
348     mpz_init_set_ui(e, 65537);
349     gmp_printf("Public key is: (%ZX, %ZX)\n\n", n, e);
350
351     // 私钥
352     mpz_init(d);
353     mpz_invert(d, e, phiN); // 求逆元
354     gmp_printf("Private key is: (%ZX, %ZX)\n\n", n, d);
355
356     initString();
357 }

```

```
357
358 void clearAll()
359 {
360     mpz_clear(d);
361     mpz_clear(e);
362     mpz_clear(n);
363     mpz_clear(p);
364     mpz_clear(q);
365     mpz_clear(phiN);
366     mpz_clear(C);
367     mpz_clear(M);
368     mpz_clear(M2);
369 }
```

编译运行结果

编译运行平台如下：

```
1 root@LAPTOP-QTCGESHO:/mnt/d/blog/work/信息安全/002# uname -a
2 Linux LAPTOP-QTCGESHO 4.4.0-19041-Microsoft #488-Microsoft Mon Sep 01 13:43:00 PST
   2020 x86_64 x86_64 x86_64 GNU/Linux
```

使用 Makefile 进行相关命令的操作，具体代码如下，也可直接查看该文件夹下面的 Makefile 文件：

```
1 GCC := gcc # 编译器
2 GMP := gmp # 连接库
3 SOURCE := ./rsa.c # C语言源代码
4
5 ORIGINFILE := ./in.txt # 原始文本文件
6 ENCRYPTEDFILE := ./encrypted.txt # 加密后的文件
7 DECRYPTEDFILE := ./decrypted.txt # 解密后的文件
8 PUBLICKEYFILE := ./publicKey.txt # 公钥储存文件
9 PRIVATEKEYFILE := ./privateKey.txt # 私钥储存文件
10
11 ENC := enc # 加密
12 DEC := dec # 解密
13 TEST := test # 测试
14
15 # 执行文件
16 a.out: ./rsa.c
17     @${GCC} ${SOURCE} -o $@ -l${GMP}
18
19 # 加密
20 enc: a.out
21     @./a.out ${ENC} ${PUBLICKEYFILE} ${PRIVATEKEYFILE} ${ORIGINFILE}
   ${ENCRYPTEDFILE}
22
23 # 解密
24 dec: a.out
25     @./a.out ${DEC} ${PRIVATEKEYFILE} ${ENCRYPTEDFILE} ${DECRYPTEDFILE}
26
27 # 一个完整的流程
28 test: a.out
29     @./a.out ${TEST} ${PUBLICKEYFILE} ${PRIVATEKEYFILE} ${ORIGINFILE}
   ${ENCRYPTEDFILE} ${DECRYPTEDFILE}
```

设置 in.txt 文件中的明文如下

```
1 This is a test file.
2 My name is mijialong.
3 There is SYSU.
```

执行 `make enc` , 输出如下:

```
root@LAPTOP-QTCGESH0:/mnt/d/blog/work/信息安全/002# make enc
p: 511 q: 513 n: 1024
Public key is: (C9FD1C9BDBD19AA13B4838FD6D5B03AFF2017624379D4F8D49D71366BE155B5C313096B2F2216CE82D69C2E8342EF04D7CCF4F776482BE004ACEDBE8BA2B12371BB71029A281FF0E3603ADF6E47EE60675307F5190EA57B368B057C3F65C30FF85B04C7658AD4574E214A21701399E60533681D0EB04BF5C88F16A83D52BF861, 10001)

Private key is: (C9FD1C9BDBD19AA13B4838FD6D5B03AFF2017624379D4F8D49D71366BE155B5C313096B2F2216CE82D69C2E8342EF04D7CCF4F776482BE004ACEDBE8BA2B12371BB71029A281FF0E3603ADF6E47EE60675307F5190EA57B368B057C3F65C30FF85B04C7658AD4574E214A21701399E60533681D0EB04BF5C88F16A83D52BF861, B37BC97F294E5083F28493B78603B8FC102482515E65D7B9A9A286E7E6025E4346A67152A35B5645DD8CBC17CFBEE1E34903820E8A6EAECD9C80C6851F1EB76E6093D1EFEB55D083F6053E57944A6DBA2B4BE30DBA4F84FA9DC079220064F1849EFB2D622A4951AD02D81B9C53335BC41D7B0A4ADDDE4D2AD7FC0257120A9D)

M = 2E555359532073692065726568540A2E676E6F6C61696A696D20736920656D616E20794D0A2E656C696620747365742061207369207369685400283E24CFD05AB5B5E33E33126DDE8DC5B6363D6096AE8F694B25D637536B3E59EE4B9A1C76788353D1D506A828580957208F522EE3EA1415536C7AFADF2BF5456A398A40200
```

执行 `make dec` , 输出如下:

```
root@LAPTOP-QTCGESH0:/mnt/d/blog/work/信息安全/002# make dec
n = C9FD1C9BDBD19AA13B4838FD6D5B03AFF2017624379D4F8D49D71366BE155B5C313096B2F2216CE82D69C2E8342EF04D7CCF4F776482BE004ACEDBE8BA2B12371BB71029A281FF0E3603ADF6E47EE60675307F5190EA57B368B057C3F65C30FF85B04C7658AD4574E214A21701399E60533681D0EB04BF5C88F16A83D52BF861
d = B37BC97F294E5083F28493B78603B8FC102482515E65D7B9A9A286E7E6025E4346A67152A35B5645DD8CBC17CFBEE1E34903820E8A6EAECD9C80C6851F1EB76E6093D1EFEB55D083F6053E57944A6DBA2B4BE30DBA4F84FA9DC079220064F1849EFB2D622A4951AD02D81B9C53335BC41D7B0A4ADDDE4D2AD7FC0257120A9D
k = 128
M2 = 2E555359532073692065726568540A2E676E6F6C61696A696D20736920656D616E20794D0A2E656C696620747365742061207369207369685400283E24CFD05AB5B5E33E33126DDE8DC5B6363D6096AE8F694B25D637536B3E59EE4B9A1C76788353D1D506A828580957208F522EE3EA1415536C7AFADF2BF5456A398A40200
```

查看 encrypted.txt 和 decrypted.txt , 具体如下图:

The screenshot shows a hex editor interface with three main panels:

- Hex View:** Displays the raw bytes of the encrypted file in hexadecimal and ASCII format.
- DECODED TEXT:** Shows the decrypted content of the file, which matches the original text in in.txt.
- DATA INSPECTOR:** Provides a detailed view of the data, including bit patterns and various data types (Int8, Int16, Int32, Int64, UTF-8, UTF-16, Float 32, Float 64).

The decoded text is as follows:

```
U ó ø . é ` £ # . * Y . | ø [ °
G E . + p ñ S 1 . İ Ø I Ä . | .
. Ñ e . ü I n § ] Q µ 1 . . á ;
î . . ÿ i ò > e ö \ . þ î " ó .
\ f , 4 i . Ê à t + ? " . ñ ø §
. . z Ø & . . û P ä # . Õ ± ë °
^ Ü # . . G . . ' ò . % . İ i
= ] . = û . p % ß " ¢ . " } ä .
```

```
work > 信息安全 > 002 > decrypted.txt
1 This is a test file.
2 My name is mijialong.
3 There is SYSU.
```

直接执行 `make test` , 具体结果如下:

```
root@LAPTOP-QTCGESHO:/mnt/d/blog/work/信息安全/002# make test
p: 511 q: 513 n: 1024
Public key is: (CAE6327582E1E51446D8CBC95047C0B923B22EA705C2784C0E1BDC5AE3E040356C67328D2711ADB2B4EB
3F4FB53BAEEE00197CA5F5B7274DAF80BE7, 10001)

Private key is: (CAE6327582E1E51446D8CBC95047C0B923B22EA705C2784C0E1BDC5AE3E040356C67328D2711ADB2B4E
83F4FB53BAEEE00197CA5F5B7274DAF80BE7, BE52CE7442506954EADC03D493F4AA88403EACDA4961D0FBEEA41DC0384F2B
B2BB50E5DEEA5AEDB47B457E1774E574A41D5D1683ADC593CA51E3A91)
n = CAE6327582E1E51446D8CBC95047C0B923B22EA705C2784C0E1BDC5AE3E040356C67328D2711ADB2B4EB91E306B4D11F
178B974C0D84FA7232A56EE34608120B986D26CEA4447230E363B046558855CAD490CD6BEDF8EFE9275662756AC8CC18B3FA
2BC5848FF2150A606BD4B37B83F4FB53BAEEE00197CA5F5B7274DAF80BE7
d = BE52CE7442506954EADC03D493F4AA88403EACDA4961D0FBEEA41DC0384F2B8BB162E01D5DCF8C72BC98D860EFF27ED3
3E7FD76D24A55F53EDC416CF23EB0018564467A9058B01D184E82760E6344F629101AAF80A6C5445827AD111330B129B99A3
38AB2BB50E5DEEA5AEDB47B457E1774E574A41D5D1683ADC593CA51E3A91
k = 128
M2 = 2E555359532073692065726568540A2E676E6F6C61696A696D20736920656D616E20794D0A2E656C696620747365742
061207369207369685400283E24CFD05AB5B5E33E33126DDE8DC5B6363D6096AE8F694B25D637536B3E59EE4B9A1C7678835
3D1D506A828580957208F522EE3EA1415536C7AFADF2BF5456A398A40200
M - M2 = 0
root@LAPTOP-QTCGESHO:/mnt/d/blog/work/信息安全/002#
```

通过对比两次加密解密得到的 M 和 M2 , 可以初步判断代码符合需求