

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  #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     * @param length int 长度，一般用 k 来作为参数
65     */
66     void I2OSP(char dst[], mpz_t src, unsigned long length);
67
68     /**
69     * 解密函数
70     */
71     void decode();
72
73     /**
74     * 加密函数
75     */
76     void encode();
77
78     /**
79     * 由于一开始 k 不确定
80     * 因此需要 malloc 函数进行字符串的空间申请
81     */
82     void initString();
83
84     /**
85     * 加密过程
86     * @param publicKeyFilePath char* 公钥文件路径
87     * @param privateKeyFilePath char* 私钥文件路径
88     * @param originFilePath char* 原始文本路径
89     * @param encryptedFilePath char* 加密后的文本路径
90     */
91     void encryption(char publicKeyFilePath[], char privateKeyFilePath[],
92                     char originFilePath[], char encryptedFilePath[]);
93
94     /**
95     * 解密过程
96     * @param privateKeyFilePath char* 私钥文件路径
97     * @param encryptedFilePath char* 加密后的文本路径
98     * @param decryptedFilePath char* 解密后的文本路径
99     */
100    void decryption(char privateKeyFilePath[], char encryptedFilePath[],
101                   char decryptedFilePath[]);

```

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),
48               mpz_sizeinbase(n, 2));
49
50     // 公钥
51     // e 通常取 3, 17, 65537
52     mpz_init_set_ui(e, 65537);
53     gmp_printf("Public key is: (%ZX, %ZX)\n\n", n, e);
54
55     // 私钥
56     mpz_init(d);
57     mpz_invert(d, e, phiN); // 求逆元
58     gmp_printf("Private key is: (%ZX, %ZX)\n\n", n, d);
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 $EM = 0x00 \parallel 0x02 \parallel PS \parallel 0x00 \parallel message$

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 $X_0 X_1 \cdots X_{k-1}$
 - 得到明文大数 $M = X_0 * 256^{k-1} + X_1 * 256^{k-2} + X_2 * 256^{k-3} + \cdots + X_{k-2} * 256 + X_{k-1}$
- 加密:
 - $C = M^e \bmod n$
- I2OSP:
 - 根据编码过程和加密后的大数

$$C = X_0 * 256^{k-1} + X_1 * 256^{k-2} + X_2 * 256^{k-3} + \cdots + X_{k-2} * 256 + X_{k-1}$$
 - 逆向得到长度为 k 的 cryptedText $X_0 X_1 \cdots X_{k-1}$
 - 输出密文 cryptedText

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(cryptedException, C, k);
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   * @param length int 长度，一般用 k 来作为参数
14   */
15  void I2OSP(char dst[], mpz_t src, unsigned long length);
16
17  void OS2IP(mpz_t dst, char src[], unsigned long length)
18  {
19      mpz_init(dst);
20      for (int i = 0; i < length; i++)
21      {
22          mpz_mul_ui(dst, dst, 256);
23          mpz_add_ui(dst, dst, src[i] & 0x0000ff);
24      }
25  }
26
27  void I2OSP(char dst[], mpz_t src, unsigned long length)
```

```

28  {
29      free(dst);
30      dst = (char *)malloc(length + 1);
31      mpz_t tmp, copy;
32      mpz_init_set(copy, src);
33
34      // 循环生成字符串
35      for (int i = length - 1; i >= 0; i--)
36      {
37          mpz_init(tmp);
38          mpz_mod_ui(tmp, copy, 256);
39          dst[i] = mpz_get_ui(tmp) & 0x0000ff;
40          mpz_div_ui(copy, copy, 256);
41      }
42  }

```

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, crypteText, k);
4      mpz_init(M2);
5      mpz_powm(M2, C, d, n);
6      unsigned long long size = mpz_sizeinbase(n, 2);
7      k = size / 8 + (size % 8 ? 1 : 0);
8      I2OSP(Message2, M2, k);
9  }

```

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     * @param length int 长度，一般用 k 来作为参数
65     */
66     void I2OSP(char dst[], mpz_t src, unsigned long length);
67
68     /**
69     * 解密函数
70     */
71     void decode();
72
73     /**
74     * 加密函数
75     */
76     void encode();
77
78     /**
79     * 由于一开始 k 不确定
80     * 因此需要 malloc 函数进行字符串的空间申请
81     */
82     void initString();
83
84     /**
85     * 加密过程
86     * @param publicKeyFilePath char* 公钥文件路径
87     * @param privateKeyFilePath char* 私钥文件路径
88     * @param originFilePath char* 原始文本路径
89     * @param encryptedFilePath char* 加密后的文本路径
90     */
91     void encryption(char publicKeyFilePath[], char privateKeyFilePath[],
92                     char originFilePath[], char encryptedFilePath[]);
93
94     /**
95     * 解密过程
96     * @param privateKeyFilePath char* 私钥文件路径
97     * @param encryptedFilePath char* 加密后的文本路径
98     * @param decryptedFilePath char* 解密后的文本路径
99     */
100    void decryption(char privateKeyFilePath[], char encryptedFilePath[],
101                    char decryptedFilePath[]);
102
103    int main(int argc, char *argv[])
104    {
105        // 如果是加密
106        if (strcmp(argv[1], "enc") == 0 && argc == 6)
107        {
108            encryption(argv[2], argv[3], argv[4], argv[5]);
109        }
110        // 如果是解密
111        else if (strcmp(argv[1], "dec") == 0 && argc == 5)
112        {
113            decryption(argv[2], argv[3], argv[4]);
114        }
115        else if (argc == 7 && strcmp(argv[1], "test") == 0) // 两个都进行
116        {
117            encryption(argv[2], argv[3], argv[4], argv[5]);
118            decryption(argv[3], argv[5], argv[6]);
119            mpz_t eq;
120            mpz_init(eq);
121            mpz_sub(eq, M, M2);

```

```

122     gmp_printf("M - M2 = %ZX\n", eq);
123 }
124 else
125 {
126     printf("usage: \n\t./a.out [enc publicKeyFile privateKeyFile | dec
privateKeyFile] inputFile outFile\n");
127     printf("OR\n");
128     printf("\t./a.out test publicKeyFile privateKeyFile inputFile crypteFile
decryptedFile\n");
129     return 0;
130 }
131
132     clearAll();
133
134     return 0;
135 }
136
137 void decryption(char privateKeyFilePath[], char encryptedFilePath[],
138                char decryptedFilePath[])
139 {
140     privateKeyFile = fopen(privateKeyFilePath, "r");
141     encryptedFile = fopen(encryptedFilePath, "r");
142     decryptedFile = fopen(decryptedFilePath, "w");
143     gmp_fscanf(privateKeyFile, "%ZX\n%ZX", n, d);
144
145     unsigned long long size = mpz_sizeinbase(n, 2);
146     k = size / 8 + (size % 8 ? 1 : 0);
147     initString();
148     fread(cryptedText, 1, k, encryptedFile);
149     decode();
150     gmp_printf("n = %ZX\nd = %ZX\nk = %llu\nM2 = %ZX\n", n, d, k, M2);
151     int startAt; // 用于判断 M 的位置
152     for (startAt = 1; startAt < k; startAt++)
153     {
154         if (Message2[startAt] == 0)
155         {
156             break;
157         }
158     }
159
160     for (startAt = startAt + 1; startAt < k; startAt++)
161     {
162         fputc(Message2[startAt], decryptedFile);
163     }
164
165     fclose(privateKeyFile);
166     fclose(encryptedFile);
167     fclose(decryptedFile);
168 }
169
170 void encryption(char publicKeyFilePath[], char privateKeyFilePath[],
171                char originFilePath[], char encryptedFilePath[])
172 {
173     publicKeyFile = fopen(publicKeyFilePath, "w");
174     privateKeyFile = fopen(privateKeyFilePath, "w");
175     originFile = fopen(originFilePath, "r");
176     encryptedFile = fopen(encryptedFilePath, "w");
177

```

```

178     generateKey(1024);
179
180     gmp_fprintf(publicKeyFile, "%ZX\nZX", n, e); // 输出公钥
181     gmp_fprintf(privateKeyFile, "%ZX\nZX", n, d); // 输出私钥
182
183     mLen = fread(Message, 1, k, originFile); // 读取文本
184     if (mLen > (k - 11)) // 明文太长
185     {
186         printf("明文太长\n");
187         exit(1);
188     }
189     encode(); // 进行加密
190
191     for (int i = 0; i < k; i++)
192     {
193         fputc(cryptedText[i], encryptedFile);
194     }
195
196     gmp_printf("M = %ZX\n", M);
197
198     fclose(originFile);
199     fclose(privateKeyFile);
200     fclose(publicKeyFile);
201     fclose(encryptedFile);
202 }
203
204 void decode()
205 {
206     OS2IP(C, crypteText, k);
207     mpz_init(M2);
208     mpz_powm(M2, C, d, n);
209     unsigned long long size = mpz_sizeinbase(n, 2);
210     k = size / 8 + (size % 8 ? 1 : 0);
211     I2OSP(Message2, M2, k);
212 }
213
214 void encode()
215 {
216     getPS(mLen);
217     getEM();
218     OS2IP(M, EM, k);
219     mpz_powm(C, M, e, n);
220     I2OSP(cryptedText, C, k);
221 }
222
223 void initString()
224 {
225     unsigned long long size = mpz_sizeinbase(n, 2);
226     k = size / 8 + (size % 8 ? 1 : 0);
227     Message = (char *)malloc(k - 10);
228     crypteText = (char *)malloc(k + 1);
229     EM = (char *)malloc(k + 1);
230     Message2 = (char *)malloc(k + 1);
231
232     for (int i = 0; i < k; i++)
233     {
234         crypteText[i] = Message2[i] = EM[i] = 0;
235     }

```

```

236     }
237
238     void OS2IP(mpz_t dst, char src[], unsigned long length)
239     {
240         mpz_init(dst);
241         for (int i = 0; i < length; i++)
242         {
243             mpz_mul_ui(dst, dst, 256);
244             mpz_add_ui(dst, dst, src[i] & 0x0000ff);
245         }
246     }
247
248     void I2OSP(char dst[], mpz_t src, unsigned long length)
249     {
250         free(dst);
251         dst = (char *)malloc(length + 1);
252         mpz_t tmp, copy;
253         mpz_init_set(copy, src);
254
255         // 循环生成字符串
256         for (int i = length - 1; i >= 0; i--)
257         {
258             mpz_init(tmp);
259             mpz_mod_ui(tmp, copy, 256);
260             dst[i] = mpz_get_ui(tmp) & 0x0000ff;
261             mpz_div_ui(copy, copy, 256);
262         }
263     }
264
265     void getPS(int mLen)
266     {
267         PS = (char *)malloc(k - mLen - 2);
268         PS[k - mLen - 3] = 0;
269
270         for (int i = 0; i < k - mLen - 3; i++)
271         {
272             PS[i] = rand() % 255 + 1;
273         }
274     }
275
276     void getEM()
277     {
278         // 各个部分的长度
279         int p1 = 1, p2 = 1, p3 = k - mLen - 3, p4 = 1, p5 = mLen;
280
281         // part 1
282         EM[0] = 0;
283
284         // part 2
285         EM[1] = 2;
286
287         // part 3
288         for (int i = 0; i < p3; i++)
289         {
290             EM[i + p1 + p2] = PS[i];
291         }
292
293         // part 4

```

```

294     EM[p1 + p2 + p3] = 0;
295
296     // part 5
297     for (int i = 0; i < p5; i++)
298     {
299         EM[p1 + p2 + p3 + p4 + i] = Message[i];
300     }
301
302     EM[k] = 0;
303 }
304
305 void generateKey(int bit)
306 {
307     while (1)
308     {
309         // 随机生成大数
310         gmp_randinit_default(greatRandomNumber);
311         gmp_randseed_ui(greatRandomNumber, time(NULL));
312
313         // 初始化 p, q
314         mpz_init(p);
315         mpz_init(q);
316
317         // 随机生成两个大数
318         // mpz_urandomb(p, greatRandomNumber, (bit + 1) / 2);
319         // mpz_urandomb(q, greatRandomNumber, (bit - 1) / 2);
320
321         mpz_urandomb(p, greatRandomNumber, bit / 2 - 1);
322         mpz_urandomb(q, greatRandomNumber, bit / 2 + 1);
323
324         // 素数生成
325         mpz_nextprime(p, p);
326         mpz_nextprime(q, q);
327
328         // 得到 n
329         mpz_init(n);
330         mpz_mul(n, p, q);
331
332         if (mpz_sizeinbase(n, 2) == bit) // 用于判断是否生成合适的位数
333         {
334             break;
335         }
336     }
337
338     // 计算 phi(n)
339     mpz_init(phiN);
340     mpz_sub_ui(p, p, 1);
341     mpz_sub_ui(q, q, 1);
342     mpz_mul(phiN, p, q);
343
344     gmp_printf("p: %d q: %d n: %d\n", mpz_sizeinbase(p, 2), mpz_sizeinbase(q, 2),
345               mpz_sizeinbase(n, 2));
346
347     // 公钥
348     // e 通常取 3, 17, 65537
349     mpz_init_set_ui(e, 65537);
350     gmp_printf("Public key is: (%ZX, %ZX)\n\n", n, e);

```

```

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 }
358
359 void clearAll()
360 {
361     mpz_clear(d);
362     mpz_clear(e);
363     mpz_clear(n);
364     mpz_clear(p);
365     mpz_clear(q);
366     mpz_clear(phiN);
367     mpz_clear(C);
368     mpz_clear(M);
369     mpz_clear(M2);
370 }

```

编译运行结果

编译运行平台如下：

```

1 root@LAPTOP-QTCGESH0:/mnt/d/blog/work/信息安全/002# uname -a
2 Linux LAPTOP-QTCGESH0 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: ${SOURCE}
17     @${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

```

[illegible]

```
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
./rsa.c: In function 'generateKey':
./rsa.c:311:40: warning: implicit declaration of function 'time' [-Wimplicit-function-declaration]
    gmp_randseed_ui(greatRandomNumber, time(NULL));
                                         ^~~~~~
p: 511 q: 513 n: 1024
Public key is: (82E5F79E04281ACB5BFFDDA10A94B88C6BDE097935AD99100E15F65E7D9419928324AAD12693DB1C8FC963B
F3ACF4705F410EF91ADF7C15B19B27065E83DAA0386157B85D9416EE3ED4A1EBCD08A9C2E18F13FBA035294786FE20EDA6AFCEB
651896C26C7C06ABEA713EFD17824AA6183A39B8692B240807B434D62BFFB95843, 589D74AFBDE2A4990D543F66F971CB2A0D
1048DB8CD2542FC1496EFB730A62F183E4BBD851C180F16A0A0D83F53D43B8AD0D16124A72D2AC68CE1B37722CE4BD8F4A250B6
8591103BF81080789E79FA2681300EE68E1C7838D6CF2837FF294A1CEE8A405EF966F565DD6523614896CCBB10805877DE002EC
5C2005A8F9305211)

Private key is: (82E5F79E04281ACB5BFFDDA10A94B88C6BDE097935AD99100E15F65E7D9419928324AAD12693DB1C8FC963
BF3ACF4705F410EF91ADF7C15B19B27065E83DAA0386157B85D9416EE3ED4A1EBCD08A9C2E18F13FBA035294786FE20EDA6AFCE
B651896C26C7C06ABEA713EFD17824AA6183A39B8692B240807B434D62BFFB95843, 589D74AFBDE2A4990D543F66F971CB2A0D
1048DB8CD2542FC1496EFB730A62F183E4BBD851C180F16A0A0D83F53D43B8AD0D16124A72D2AC68CE1B37722CE4BD8F4A250B6
8591103BF81080789E79FA2681300EE68E1C7838D6CF2837FF294A1CEE8A405EF966F565DD6523614896CCBB10805877DE002EC
5C2005A8F9305211)

M = 2A498A35654BFF2FDFA7A6C531514EAE32E528F2057095828A806D5D1538378761C9A4BEE593E6B5337D6254B698FAE9660
3D36B6C58DDE6D12333EE3B5B55AD0CF243E280054686973206973206120746573742066696C652E0A4D79206E616D652069732
06D696A69616C6F6E672E0A546865726520697320535953552E
n = 82E5F79E04281ACB5BFFDDA10A94B88C6BDE097935AD99100E15F65E7D9419928324AAD12693DB1C8FC963BF3ACF4705F41
0EF91ADF7C15B19B27065E83DAA0386157B85D9416EE3ED4A1EBCD08A9C2E18F13FBA035294786FE20EDA6AFCEB651896C26C7C
06ABEA713EFD17824AA6183A39B8692B240807B434D62BFFB95843
d = 589D74AFBDE2A4990D543F66F971CB2A0D1048DB8CD2542FC1496EFB730A62F183E4BBD851C180F16A0A0D83F53D43B8AD0
D16124A72D2AC68CE1B37722CE4BD8F4A250B68591103BF81080789E79FA2681300EE68E1C7838D6CF2837FF294A1CEE8A405EF
966F565DD6523614896CCBB10805877DE002EC5C2005A8F9305211
k = 128
M2 = 2A498A35654BFF2FDFA7A6C531514EAE32E528F2057095828A806D5D1538378761C9A4BEE593E6B5337D6254B698FAE966
03D36B6C58DDE6D12333EE3B5B55AD0CF243E280054686973206973206120746573742066696C652E0A4D79206E616D65206973
206D696A69616C6F6E672E0A546865726520697320535953552E
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