RSA 算法报告

18342066 鲁沛

一. 原理描述

对极大整数进行因数分解的难度 (The Factoring Problem) 决定了 RSA 算法的可靠性。换言之，对一个极大整数做因数分解愈困难，RSA 算法就愈可靠。假如有人找到一种快速因数分解的算法的话，那么 用 RSA 加密的信息的可靠性就肯定会极度下降。目前看来找到这样的算法的可能性非常小。

目前还没有可靠的攻击 RSA 算法的方式。短的 RSA 钥匙可能被强力方式解破 (比如 768 个二进制位、即对应于 232 个十进制位以下的整数)。只要 RSA 密钥的长度足够长 (比如 1024-bit 至 15360-bit)，用 RSA 加密的信息看来很难被破解。 RSA 密钥按 (e, N) 和 (d, N) 分组分发，e、d 和 N 分别是公钥指数、私钥指数和模底，RSA 密钥长度一般指 N 的二进制位长。

总体流程如下：

1. 选择两个大质数 p，q，计算 N = pq
2. 计算欧拉方程：$\varphi(N) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1)$
3. 选择公钥指数 e，满足$1 < e < \varphi(n)$且$\gcd(e,\ \varphi(n)) = 1$
4. 计算私钥指数 d：$de \equiv 1(mod\varphi(n))$
5. 销毁 p 和 q，将（e, N）作为公钥，将（d, N）作为私钥
6. 加密：$c = n^e\ mod\ N$
7. 解密：$n = c^d\ mod\ N$

我们需要实现的是 RSA 算法中的 RSAES-PKCS1- v1_5 体系，其和普通的 RSA 算法的区别是加密是要进行填充，解密后要进行解码，其详细步骤见后文。

二. 数据结构设计

由于 RSA 算法基于的是大质数，因此普通的整形并不能使用，此处我们使用 GMP 库来进行大数的运算。

GMP 是著名的任意精度算术运算库，支持任意精度的整数、有理数以及浮点数的四则运算、求模、求幂、开方等基本运算，还支持部分数论相关运算。Maple、Mathematica 等大型数学软件的高精度算术运算功能都是利用 GMP 实现的。

在 linux 下安装流程如下：

首先下载安装包，然后命令行下输入以下指令解压安装：

```
tar xzf gmp-X.X.X.tar.xz
cd gmp-X.X.X
./configure
make
make check
```

sudo make install

引用 gmp 库:
```
#include <gmp.h>
```
编译时加上参数 -lgmp 即可。

关于八位字节串：一个 char 是 8 位也就是一个字节，因而一个字符就是一个字节串。八位字节串还可以理解为 256 进制，因为2^8=256。此外，以下几种对字符的赋值等价:

```
1. char a = 0x61;
2. char a = 97;
3. char a = 'a';
```

最终 RSA 算法中的数据结构:
整数：mpz_t（gmp 库）
八位字节串：unsigned char[ ]

三. 密钥生成

使用 GMP 库中的质数寻找函数生成 p，q，N:

```
1. void get_N_p_q(mpz_t N, mpz_t p, mpz_t q)
2. {
3.     gmp_randstate_t grt;
4.     gmp_randinit_default(grt); ////设置随机数生成算法为默认
5.     gmp_randseed_ui(grt, time(NULL));
6.
7.     mpz_urandomb(p, grt, 1024); //随机生成一个0-2^1024的一个数
8.     mpz_urandomb(q, grt, 1024);
9.
10.    mpz_nextprime(p, p);
11.    mpz_nextprime(q, q);
12.    mpz_mul(N, p, q);
13.}
```

选择 e=65537，并进而计算出 d:

```
1. mpz_sub_ui(p, p, 1);
2. mpz_sub_ui(q, q, 1);
3. mpz_mul(fiN, p, q);
4. mpz_set_str(e, "65537", 10);//e 一般取65537
5. mpz_invert(d, e, fiN);
```

四. 加密

首先判断明文的长度是否大于 N 的长度 k 再减去 11（k-11），如果大于，返回错误;
对明文 M 进行编码得到 EM;
调用 OS2IP 函数将 EM 转化为一个大整数 m;
$\boldsymbol{m = m^e \ mod \ N;}$
调用 I2OSP 函数将 m 转化为字节串 c。

OS2IP 函数:
其实就是 256 进制转换为 10 进制

```
1. void OS2IP(unsigned char octet[], mpz_t intm, int length)
2. {
3.     mpz_t a;
4.     mpz_init(a);
5.     mpz_set_str(a, "256", 10);
6.     for (int i = 0; i < length; ++i)
7.     {
8.         mpz_mul_ui(intm, intm, 256);
9.         mpz_add_ui(intm, intm, octet[i]);
10.    }
11.}
```

I2OSP 函数:
其实就是 10 进制转换为 256 进制

```
1. void I2OSP(mpz_t intm, unsigned char octet[], int length)
2. {
3.     mpz_t temp, src;
4.     mpz_init(src);
5.     mpz_set(src, intm);
6.     for (int i = length - 1; i >= 0; --i)
7.     {
8.         mpz_init(temp);
9.         mpz_mod_ui(temp, src, 256);
10.        octet[i] = mpz_get_ui(temp);
11.        mpz_div_ui(src, src, 256);
12.    }
13.}
```

encrypt 函数:

```
1. void encrypt(mpz_t N, mpz_t e, unsigned char *M, unsigned char *c
   iphertext)
2. {
```

```
    int mlen = strlen(M);
    if (mlen > k - 11)
        printf("message too long\n");
    else
    {
        unsigned char EM[k];
        padding(M, EM, k - mlen - 3);
        mpz_t intm;
        mpz_init(intm);
        OS2IP(EM, intm, k);
        mpz_powm(intm, intm, e, N);
        int size = mpz_sizeinbase(intm, 2);
        int length = size / 8 + (size % 8 ? 1 : 0);
        if (length != k)
            printf("Error");
        else
            I2OSP(intm, ciphertext, length);
    }
}
```

五. 解密

首先判断密文的长度是否等于 N 的长度 k，不等于则输出错误;
调用 OS2IP 函数将 C 转化为一个大整数 c;
$\boldsymbol{c = c^d \ mod \ N;}$
调用 I2OSP 函数将 c 转化为字节串 m;
对 m 解码得到明文 M。

decrypt 函数:

```
void decrypt(mpz_t N, mpz_t d, unsigned char *C, unsigned char *EM)
{
    mpz_t intc;
    mpz_init(intc);
    OS2IP(C, intc, k);
    mpz_powm(intc, intc, d, N);
    I2OSP(intc, EM, k);
}
```

六. 编解码

编码：

填充规则：
EM = 0x00 || 0x02 || PS || 0x00 || M.
其中 PS 位随机生成的不含有 0 的字节串，长度为 k-mlen-11，因而长度至少为 8

getPS 函数：

```
1. void getPS(unsigned char PS[], int lenPS)
2. {
3.     for (int i = 0; i < lenPS; ++i)
4.     {
5.         int a = rand() % 255 + 1;
6.         PS[i] = a;
7.     }
8. }
```

padding 函数：

```
1. void padding(unsigned char M[], unsigned char EM[], int lenPS)
2. {
3.     EM[0] = 0x00;
4.     EM[1] = 0x02;
5.     unsigned char PS[lenPS];
6.     getPS(PS, lenPS);
7.     for (int i = 0; i < lenPS; ++i)
8.     {
9.         EM[i + 2] = PS[i];
10.    }
11.    EM[2 + lenPS] = 0x00;
12.    for (int i = 0; i < strlen(M); ++i)
13.    {
14.        EM[i + 3 + lenPS] = M[i];
15.    }
16.}
```

解码：
由于 PS 中不含有 0，因此只要遍历 EM，找到第二个 0，则其下一位开始直到结束就是明文。具体代码如下：

```
1. int start;
2. for(start = 2; start < k; ++start)
3. {
4.     if(EM[start] == 0x00)
5.         break;
6. }
7. int length = k - start - 1;
```

```
plaintext = (char *)malloc(sizeof(char) * length + 1);
for(int i = 0; i < length; ++i)
{
    plaintext[i] = EM[start + 1 + i];
}
plaintext[length] = '\0';
```

七. 编译运行结果与测试用例

测试明文：I love you! You are my sunshine!

编译运行结果：

```
freedom@freedom-virtual-machine:~/Desktop$ gcc RSA.c -lgmp
freedom@freedom-virtual-machine:~/Desktop$ ./a.out
I love you! You are my sunshine!
```

其加密解密后得到同样的明文，因此我们认算法基本正确。事实上，不论将明文换成什么，都可以顺利的加密解密出同样的结果，可以运行文件夹下的源代码自行测试。

八. 完整源代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <gmp.h>

int k;//N 的长度

void get_N_p_q(mpz_t N, mpz_t p, mpz_t q)
{
    gmp_randstate_t grt;
    gmp_randinit_default(grt); ////设置随机数生成算法为默认
    gmp_randseed_ui(grt, time(NULL));

    mpz_urandomb(p, grt, 1024); //随机生成一个0-2^1024 的一个数
    mpz_urandomb(q, grt, 1024);

    mpz_nextprime(p, p);
    mpz_nextprime(q, q);
    mpz_mul(N, p, q);
}

void getPS(unsigned char PS[], int lenPS)
{
```

```
25.     for (int i = 0; i < lenPS; ++i)
26.     {
27.         int a = rand() % 255 + 1;
28.         PS[i] = a;
29.     }
30.}
31.
32.void padding(unsigned char M[], unsigned char EM[], int lenPS)
33.{
34.     EM[0] = 0x00;
35.     EM[1] = 0x02;
36.     unsigned char PS[lenPS];
37.     getPS(PS, lenPS);
38.     for (int i = 0; i < lenPS; ++i)
39.     {
40.         EM[i + 2] = PS[i];
41.     }
42.     EM[2 + lenPS] = 0x00;
43.     for (int i = 0; i < strlen(M); ++i)
44.     {
45.         EM[i + 3 + lenPS] = M[i];
46.     }
47.}
48.
49.void OS2IP(unsigned char octet[], mpz_t intm, int length)
50.{
51.     mpz_t a;
52.     mpz_init(a);
53.     mpz_set_str(a, "256", 10);
54.     for (int i = 0; i < length; ++i)
55.     {
56.         mpz_mul_ui(intm, intm, 256);
57.         mpz_add_ui(intm, intm, octet[i]);
58.     }
59.}
60.
61.void I2OSP(mpz_t intm, unsigned char octet[], int length)
62.{
63.     mpz_t temp, src;
64.     mpz_init(src);
65.     mpz_set(src, intm);
66.     for (int i = length - 1; i >= 0; --i)
67.     {
68.         mpz_init(temp);
```

```
        mpz_mod_ui(temp, src, 256);
        octet[i] = mpz_get_ui(temp);
        mpz_div_ui(src, src, 256);
    }
}

void encrypt(mpz_t N, mpz_t e, unsigned char *M, unsigned char *ciphertext)
{
    int mlen = strlen(M);
    if (mlen > k - 11)
        printf("message too long\n");
    else
    {
        unsigned char EM[k];
        padding(M, EM, k - mlen - 3);
        mpz_t intm;
        mpz_init(intm);
        OS2IP(EM, intm, k);
        mpz_powm(intm, intm, e, N);
        int size = mpz_sizeinbase(intm, 2);
        int length = size / 8 + (size % 8 ? 1 : 0);
        if (length != k)
            printf("Error");
        else
            I2OSP(intm, ciphertext, length);
    }
}

void decrypt(mpz_t N, mpz_t d, unsigned char *C, unsigned char *EM)
{
    mpz_t intc;
     mpz_init(intc);
     OS2IP(C, intc, k);
     mpz_powm(intc, intc, d, N);
     I2OSP(intc, EM, k);
}

int main()
{
     mpz_t N, fiN, p, q, e, d;
     mpz_init(N);
     mpz_init(fiN);
```

```
111.     mpz_init(p);
112.     mpz_init(q);
113.     mpz_init(e);
114.     mpz_init(d);
115.     get_N_p_q(N, p, q);
116.     mpz_sub_ui(p, p, 1);
117.     mpz_sub_ui(q, q, 1);
118.     mpz_mul(fiN, p, q);
119.     mpz_set_str(e, "65537", 10);//e 一般取 65537
120.     mpz_invert(d, e, fiN);//求逆元
121.     int size_k = mpz_sizeinbase(N, 2);
122.     k = size_k / 8 + (size_k % 8 ? 1 : 0);
123.     unsigned char m[] = "I love you! You are my sunshine!";
124.     unsigned char ciphertext[k];
125.     unsigned char EM[k];
126.     unsigned char * plaintext;
127.     encrypt(N, e, m, ciphertext);
128.     decrypt(N, d, ciphertext, EM);
129.     int start;
130.     for(start = 2; start < k; ++start)
131.     {
132.         if(EM[start] == 0x00)
133.             break;
134.     }
135.     int length = k - start - 1;
136.     plaintext = (char *)malloc(sizeof(char) * length + 1);
137.     for(int i = 0; i < length; ++i)
138.     {
139.         plaintext[i] = EM[start + 1 + i];
140.     }
141.     plaintext[length] = '\0';
142.     printf("%s\n", plaintext);
143. }
```