密码学及安全计算第三次作业

李雨田 2010012193 计14

素数的选取

为了达到更好的安全性,使用OpenSSH自带工具 `ssh-keygen` 生成公钥和密钥.可以保证长度为2048 bit,且均是强素数.生成的公钥为文件 `public_key` ,密钥为文件 `private_key` .

使用如下命令转换成可读的形式.

```
ssh-keygen -t rsa
openssl rsa -text -noout < private_key > readable_key
```

其中素数p如下.

```
00:ec:f8:6c:7d:39:4f:48:f3:52:39:15:32:22:7e:
0f:1a:10:51:73:38:6c:e5:0f:5d:9f:ca:fa:65:d2:
af:0d:16:c4:57:59:f7:f3:17:78:96:c2:01:a5:f0:
79:95:64:75:89:92:ac:6d:ef:42:d8:75:f3:a3:3f:
43:27:b7:e4:4b:62:0e:98:b5:b1:58:48:b5:68:ae:
bf:75:74:62:4c:6d:eb:67:c5:af:38:9a:8f:a3:ce:
8c:3d:41:75:e7:5a:ec:53:83:55:06:fc:fd:ba:7f:
ae:f2:6a:78:90:7d:eb:f5:37:3b:44:ce:cc:99:08:
9f:a3:5e:18:d5:c5:ff:47:7d
```

素数q如下.

```
00:d9:b5:ba:ca:b4:92:26:0f:72:d5:f1:35:9e:bb:
24:c2:87:d1:af:77:8e:99:35:80:58:69:97:35:94:
64:03:2f:08:7e:ef:89:9e:ef:3c:17:94:95:c2:da:
19:dc:3e:86:75:2a:d7:8e:43:06:92:ab:ca:a1:39:
14:28:b2:74:bc:6e:9d:31:69:b8:a4:14:6d:00:7f:
0e:8c:89:21:ac:54:c9:d5:2a:db:17:6c:57:22:ad:
72:1c:45:42:bb:e1:cc:ee:b6:33:13:61:07:ba:07:
bf:0f:6a:90:d6:4e:96:d8:8d:be:69:e3:f9:31:16:
b2:5c:21:eb:81:d6:7c:88:fb
```

模n如下.

```
00:c9:86:cc:82:fc:1c:45:88:c0:f7:e9:b4:2f:63:
b5:40:4d:46:25:4d:e1:ea:db:fd:00:77:1a:4e:36:
6c:20:f0:71:dc:0e:39:59:5a:24:ae:ab:bd:37:01:
f9:96:83:f1:3e:ca:55:41:32:54:4d:b4:5a:a7:6e:
a7:2f:63:1b:82:f0:c3:ea:ee:05:b5:84:56:d7:95:
92:e1:45:34:cb:62:6f:0b:98:32:a7:ec:6a:fc:32:
22:50:25:bc:31:28:0c:f4:e6:d2:7e:17:76:d4:d4:
da:eb:ee:e2:8a:69:32:dc:c4:41:8b:ab:e5:43:7f:
9a:ed:ca:da:e5:b2:8a:ce:7a:c4:97:22:6b:2b:a5:
f4:31:f5:fe:d4:2a:55:78:12:d5:49:92:3c:00:57:
06:b9:86:06:4a:48:0b:e0:15:3b:36:f8:06:6b:32:
66:79:c5:56:7d:e1:41:f1:e2:36:6d:f1:1e:22:25:
46:60:92:b3:82:75:02:63:6e:82:9a:48:15:c9:69:
19:be:28:30:32:85:82:79:ce:01:7d:ce:75:5c:66:
58:80:26:32:81:f0:f2:40:90:6d:f1:ad:76:b4:d1:
69:2b:1e:12:41:ec:2d:e7:b5:75:d8:36:25:48:1f:
1f:6f:b3:9d:24:20:02:84:a3:24:e2:fd:2e:dd:d1:
7f:8f
```

公钥指数$e = 65537$.

密钥指数d如下.

```
00:95:27:fa:12:30:7a:d4:54:45:4e:b8:60:ae:7a:
83:da:55:d7:47:20:ff:c7:0e:8f:91:5a:95:d3:b1:
a0:12:39:24:6c:94:f2:89:59:98:b0:d7:57:b2:70:
f9:c4:17:5f:e4:f3:68:6a:5a:cf:de:bb:50:25:80:
56:a9:52:17:38:b0:ee:b0:e0:fe:c4:bc:70:72:1b:
b8:9c:96:d6:5e:7a:3a:6a:40:79:ad:a3:e4:03:49:
2f:f0:56:1d:95:dc:68:ca:92:54:d5:12:94:d7:2d:
bb:e2:c7:ea:50:3b:50:73:e9:a6:01:b4:76:79:a0:
e6:90:7d:85:ca:60:7d:30:18:89:05:79:6b:39:fa:
4c:01:03:9d:59:93:5c:99:b3:fa:29:89:6b:18:f8:
c6:64:61:46:1b:77:ee:83:01:c1:ba:c2:16:5b:b9:
3f:64:f2:bd:22:67:1c:1c:65:c6:6d:d9:43:24:fd:
e3:a3:a2:85:b9:5f:47:ed:ca:3b:51:13:04:2a:48:
23:b8:00:5a:e8:e2:f2:04:38:33:69:46:b8:36:20:
c9:9f:5e:e7:1f:de:05:4a:f6:3b:ba:75:fe:56:72:
b9:d0:31:09:4a:5a:a8:75:c4:2c:ef:74:9c:7f:6c:
d1:5d:25:44:80:0d:20:94:8c:9c:b9:46:28:3b:a9:
29:71
```

可以看出模n确实有2048 bit.而且不难验证p和q均是强素数.

大整数运算

首先需要实现大整数运算.定义类 `BigInt` ,详见文件 `big_int.h` 和 `big_int.cpp` .

`BigInt` 使用 `vector<uint32_t>` 存储数据,根据需要自动增长缩短,并添加一个符号位表示正负.在实际使用中,由于 `vector` 的拷贝会带来额外开销.经过 `gprof` 测试,大部分时间都花在了 `vector` 的拷贝操作上.如果为了更高性能,可以考虑直接使用数组实现.

根据这些基本数据结构实现加减乘除等运算.按照BLAS的模式先实现函数 `Axpy` 完成

$$x = ax + y$$

的计算.注意可以提供额外参数 `base` 表示x和y具体的对齐方式,为乘法做准备.有

了 `Axpy` 就可以实现 `vector<uint32_t>` 上的加法减法和乘法等操作.对于 `BigInt` 的算术操作,需要额外注意正负号带来的影响,其余均调用 `vector<uint32_t>` 上的算术操作即可.

除法采用除数对齐逐次操作的办法,令被除数为a,则至多需要进行$\log_2 a$次操作.求模使用同样的方法实现.可以看出,除法和求模运算是相当耗时的.但是有Montgomery模乘可以大幅度简化.

最后实现Montgomery模乘和其上的模幂运算.要得到Montgomery数,只需对其与R^2进行Montgomery模乘即可.要还原一个Montgomery数,只需对其与1进行Montgomery模乘.

Montgomery模幂运算使用Montgomery模乘和快速幂算法.对于指数e,仅需$\log_2 e$次运算.

这些工作看起来很简单,但是却耗费了最多的时间,总共达到400多行代码,并且均是独立完成.参考文献为一篇详细描述Montgomery算法的文章[Understanding the Montgomery reduction algorithm](#).

加密和解密

利用大整数可以轻松实现RSA加密和解密,详见文件 `main.cpp` .

不妨对明文 `0x0558` 进行测试.

加密即求

$$C = M^e \mod n.$$

得到结果如下.

```
263C0F013E1585B39F173F14E864082AB76379CD1E
B5757F5366301C4AF9235A8C5364D60331432E015E
6B16547B42E2C2D1C64A5E78108FA1798DEDC07F12
8DB9361D217FB486FF5B05B811659B2B2469641B28
142D757EC16FD86C4BE51817C2F8555744E824239F
93A60765D57F486AEB6540180BFA060377618B8DCE
0BEABDC2CDB3AF73FD9657104955E3C986A50F403E
7B753C9162264851DCA047F5EFD38586F4891E7358
B2A3EC2513324CD65995BD163CD5DEAC94AE6952D4
2357508848E2FD97659012B2EFE15CA874A1DEA484
73B7C00AACF65AE1F674DB5C4D6342DE47B7ABEFFE
1191DEFC66E4254A881C7AE3827D00F5370CE002E9
08609CA2
```

由于此时底数和幂都比较小,可以使用支持无限精度整数的语言,如Python,进行验算.不难得到结果的确是正确的.

解密即求

$$P = C^d \mod n.$$

此时底数和幂都很大了,无法再用Python计算.最后得到的结果为 `0x0558` ,确实是之前设定的明文.

如此证明了加密和解密算法是正确的.

加密的时候可以瞬间得到结果,测试的时候解密耗时约20秒.如果能用指针替代 `vector<uint32_t>` ,至少可以减少70%的时间.

实验总结

通过讲课上学到的知识和实践相结合,更加了解了RSA算法的工作流程.并且对于RSA算法的计算细节和优化有了更加深刻的认识.学习了从简单的数学表达到高效的计算机实现需要考虑的问题.

感谢老师和助教的辛勤讲解和热情答疑.