RSA 实现

> 2017011426 刘晓义

实现了 RSA 的密钥生成、单个 Block 的加密、解密。

使用了两个不同的高精度实现，一个是自己写的，一个依赖了一个 Rust 的库 ramp

运行结果

```
# 生成密钥，输出 16 进制的一个 2048bit 密钥
$ cargo run --release --bin keygen
..25
.....................202
== PrivKey ==
n:
5810475840344388CF5B7B6192436CCCF2F8866AE3F59090E95416AFBF04DD8FF79ED09962F02E91
24CD93B080492141855FAB211330CE2E2095B6231D329AAB240F34FBD0833CCFD8AD37C6FB949131
C7011D25821E5CD84E44A45DDADF2D4F16ABF456B5D95D51F3E1FC24FD590CB1EAD56889C49837FA
097A615084F91D4A177EAF3F8D40FB8658B5965E1C998C83B0CEE424A5F657965A4C8DD2AB9470CB
EE3CD9ACE92847CF0F41A6CCE9FD2679A1A84FA47E2CC330E984E7C8E83A1529C693BDC95E609C06
08841722D400B0809A14CBB2B4616EFCEDEF5C2E4370E12787DC60499A35D2EB51852357A939ED32
D883D49F343B0E0C003CC6F53DEF1C0D
d:
61491A8813E5A8F5EA9D55A064C4B7DD724DBBB13A447E8098439D0C6A2A25DA75E3ACEEC6AE2FC3
88B4D30399B5305801CFBE6DBB959F30B62EAE17FC997AE4B5E156D71972E22B7B6360CB2202EEA3
DF050BFDF3D137207ED6B5FE395D73E07B5B5B82AF428225971780461CDBD37A79C368C949B61FA0
C9F7B8AA0B3C696DB4397D95E1414785A196432433BB32D3AED7E66B4E44F1DA2E53A94E7DEF97D0
23435406CDD6643CC7A30CB06D05CC1982062B506116ABA8BAF75EB104B75B99304108D3B3C2F0F7
5EC6486973CBCEB8AFADFF26249803E54B98C741F94CB03D660E77E2D419E4EDBA069FC94EC815F5
0FBDC33F34C7BDBD2142BC3E6B2B601
== PubKey ==
n:
5810475840344388CF5B7B6192436CCCF2F8866AE3F59090E95416AFBF04DD8FF79ED09962F02E91
24CD93B080492141855FAB211330CE2E2095B6231D329AAB240F34FBD0833CCFD8AD37C6FB949131
C7011D25821E5CD84E44A45DDADF2D4F16ABF456B5D95D51F3E1FC24FD590CB1EAD56889C49837FA
097A615084F91D4A177EAF3F8D40FB8658B5965E1C998C83B0CEE424A5F657965A4C8DD2AB9470CB
EE3CD9ACE92847CF0F41A6CCE9FD2679A1A84FA47E2CC330E984E7C8E83A1529C693BDC95E609C06
08841722D400B0809A14CBB2B4616EFCEDEF5C2E4370E12787DC60499A35D2EB51852357A939ED32
D883D49F343B0E0C003CC6F53DEF1C0D
e: 10001

# 加密
$ cargo run --release --bin encrypt -- \
    -m 6d6f6465726e63727970746f677261706879
    -e 10001
    -n [这里写了上面的 n]
```

```
384AEAAA6DA963F775175E14646AAD56E942C2BFD157164A99DD720DA9FBC26637B087425204031B
95AB42F2C4DB5E17D4574BA4C579DA3B3602FE47E2AC45E8AB5C4ADC4ADA256556F7667F6C422E3D
331C0E449BBD56A5435D00910BD705E9AE8777D30547C587833C46B20DA3C05B0B9B83D5B6E969E5
152CB7114659BD56D1FCD0E075E40D5B0B3F5FE1F337BEF165D8050AD542A47B220B46EEF1D98A84
CD97E6EC357DBCCD7E76E152142F0E6606996211F157B08A727478DF6BD5CDF451B0CD5753386105
1DF51F5A96869C3D22A7A233029565EC51E7430B84429CC1D6AD9BCDF8A292A512EA3424DB37DA27
9477A385DA5807F412EAD23FAAA62946

$ cargo run --release --bin decrypt -- \
    -c [这里写了上一条指令输出的密文]
    -d [这里写了上面的 d]
    -n [这里写了上面的 n]
6D6F6465726E63727970746F677261706879
```

实现细节

主要分为两个部分：高精度实现和 RSA 算法。

RSA

密钥生成

随机生成 1024 bit 的随机数，然后使用 Miller-Rabin 算法检测是否是质数。

这里需要选择 Miller-Rabin 算法的轮数。根据 Damgard-Landrock-Pomerance 的论文 *Average case error estimates for the strong probable prime test*，事实上这个上界是比参考书里还要紧一些的。如果是 1024-bit 的随机数，那么只需要 $k = 6$ 就可以保证误判合数为质数的概率降到 10^{-40} 以下。因此在代码里选择 $k = 6$

此外进行的优化是生成随机数的时候生成 1024 bit，第一位和最后一位直接填 1，可以节省一点获取系统随机数的时间。

此外，直接判断对于 1000 以内的质数是否是因子。经过这两个优化之后大概每随机 50 个左右的 1024 bit 随机数就可以获得一个质数。

之后还需要一个欧几里得算法求逆元

加密、解密

这里主要是计算取模的高次幂运算。参考书里给的算法是：

$$a^b = \prod_{i \le \log_2 b \wedge b_i = 1} a^{(2^b)}$$

最开始一看没看懂，后来仔细一看就是快速幂算法：

```
uint32_t fast_pow(uint32_t base, uint32_t p, uint32_t m) {
    if(p == 0) return 1;
    uint64_t cur = fast_pow(base, p>>1, m);
    cur = (cur * cur) % m;
    if base & 1 {
        cur = (cur * base) % m;
    }
    return cur;
}
```

考虑到 key的长度是一个定值，认为以上所有乘法、复制和取模运算都是常数，那么复杂度应该是 $\mathcal{O}(\log p)$

高精度

如果可以偷懒调库的话，比较知名的就是 libgmp，相比 gmp 而言 ramp 比较科学的地方是和 Rust 的随机数 API 配合的比较好，可以用一些语言抽象省代码。

如果不偷懒的话，分析以下需要实现哪些东西。

- Miller-Rabin 算法：要求需要有比较，加法，乘法，带模幂，这蕴含着左移、右移
- 密钥生成：扩展欧几里得算法求逆元，因此需要一个带余除法和模数减法
- 加密、解密：乘法、带模幂