《密码学》实验三报告

RSA公钥密码实现

Part 1. 随机大素数生成（$> 2^{512}$）

不妨定义$\pi(n)$为小于n的素数个数，则依素数定理有：

$$\lim_{n\to\infty}\frac{\pi(n)}{\int_2^n\frac{1}{\ln(x)}\mathrm{d}x}=1 \tag{1}$$

或者有

$$\pi(n)\sim\int_2^n\frac{1}{\ln(x)}\mathrm{d}x. \tag{2}$$

根据L'Hospital's Theorem，对（2）的极限可转换为：

$$\lim_{n\to\infty}\frac{\int_2^n\frac{1}{\ln(x)}\mathrm{d}x}{\frac{n}{\ln(n)}}=1. \tag{3}$$

故（2）可写成

$$\pi(n)\sim\int_2^n\frac{1}{\ln(x)}\mathrm{d}x\sim\frac{n}{\ln(n)}. \tag{4}$$

上述定理告诉我们，在某个区间内随机选取一个数n，它为素数的概率为

$$\Pr[\mathsf{Prime}(n)]=\frac{\frac{n}{\ln(n)}}{n}=\frac{1}{\ln(n)}. \tag{5}$$

米勒-拉宾（Miller-Rabin）素数测试（Miller-Rabin Primality Test）

对于给定的大数n（$||n||=512$），尽管根据（5）的结果，它是素数的概率很大，但是它仍旧有可能是一个素数。尽管我们可以用所有小于$\sqrt{n}$的数字去除n，但是这个算法时间复杂度为$\mathcal{O}(\sqrt{n})$，是一个非确定多项式时间的算法。因此我们需要找到更加高效便捷的算法来检验这个数是否为素数。Miller Rabin 素数测试是一种概率算法，因为对于大素数来讲，我们已经无法通过试除法一一验证它是否是素数，所以我们只能通过逼近的方法来增加“n是素数”

的可能性。

注意到给定某素数p，若某个数x满足以下等式

$$\begin{aligned} x^2 &\equiv 1 \pmod p \\ (x-1)(x+1) &\equiv 0 \pmod p, \end{aligned} \tag{6}$$

则$x = \pm 1$或$x = p \pm 1$.

此外由Fermat's Little Theorem可知，若p为素数，还有以下等式成立

$$a^{p-1} \equiv 1 \pmod p. \tag{7}$$

此外，若数n为一个偶数，有

$$n \equiv 2^s d \pmod p. \tag{8}$$

根据（6）（7）（8），我们可以给出一个素性检验的方案：

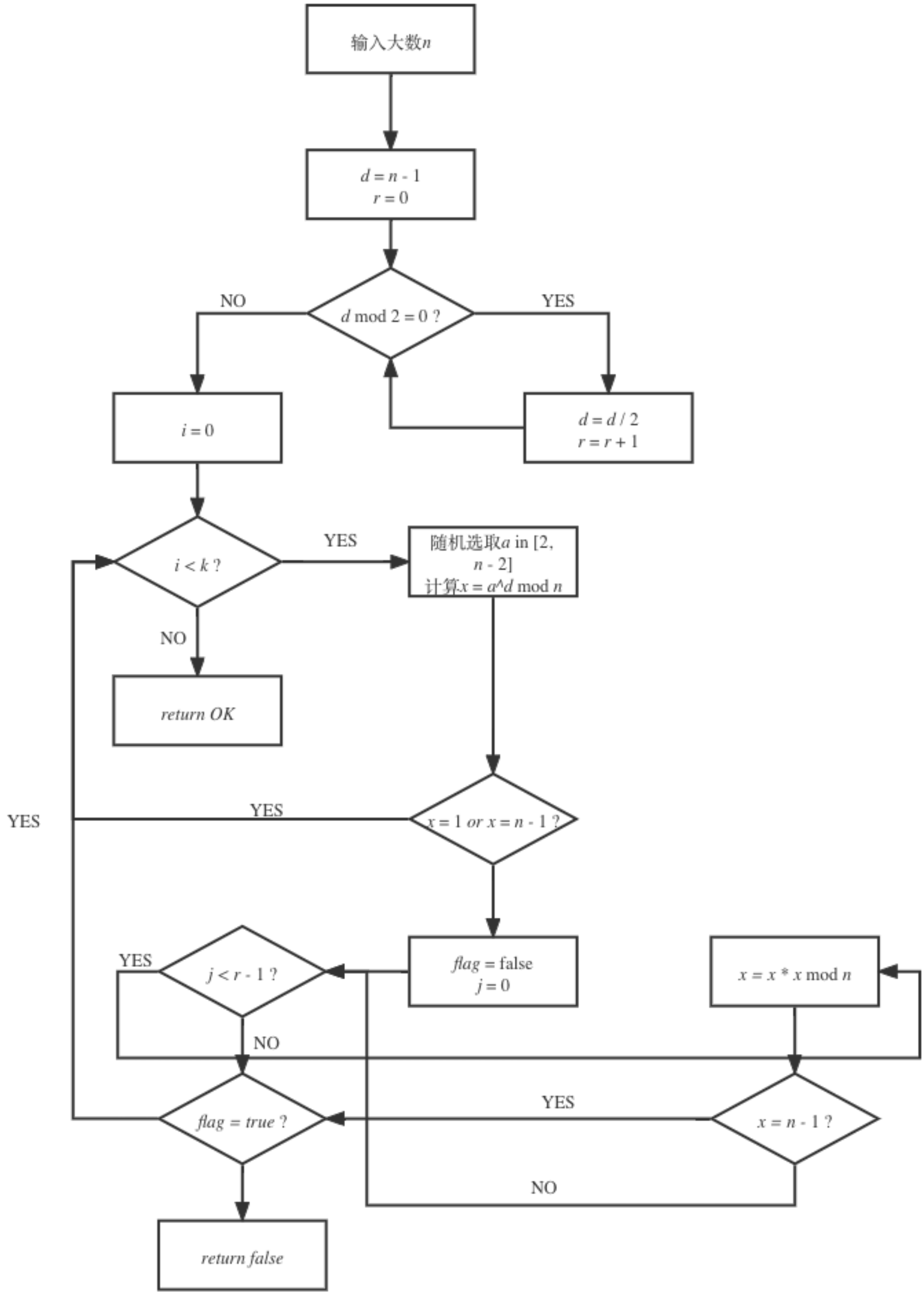
我们对这个 a^{n-1} 也就是 $a^{2^s d}$不断进行开平方。根据以上提到的结论，不断开平方的结果只有两种可能：一个是把所有2开完得到 $a^d \equiv 1 \pmod n$，另一种是在中间某个值得到 $a^{2^r d} \equiv -1 \pmod n, 0 \le r < s$. 所以如果对于这个 n，如果我们找到了一个 a 满足 $a^d \neq 1$ 且对于所有 $[0, s)$ 内的r，都有$a^{2^r d} \neq -1$，那么，n 就肯定不是素数。

给定检测的大数 n，在 $[2, n-2]$ 的范围内随机挑选a，进行上述检测。如果单次检测不通过，那 n 就一定是一个合数；如果检测通过，那么n有可能是一个质数。如果 n 是合数，但是对于 a 它通过了测试，那么这个a被称为一个强伪证（strong liar）。已经被证明的是，一个合数n 的强伪证数目不超过 $\frac{n}{4}$。所以如果我们做k次测试都通过，那么发生误判的概率就是4^{-k}，对于一个大数来说只要做足够多的测试，总是可以将误判的概率降低到很小的值。时间复杂度为$\mathcal{O}(k \log n)$.

C++实现方法

```cpp
bool miller_rabin_test(const ZZ& n) {
    ZZ d = n - 1;
    unsigned short r = 0;

    while (d % 2 == 0) {
      d = d / 2;
      r++;
    }

```

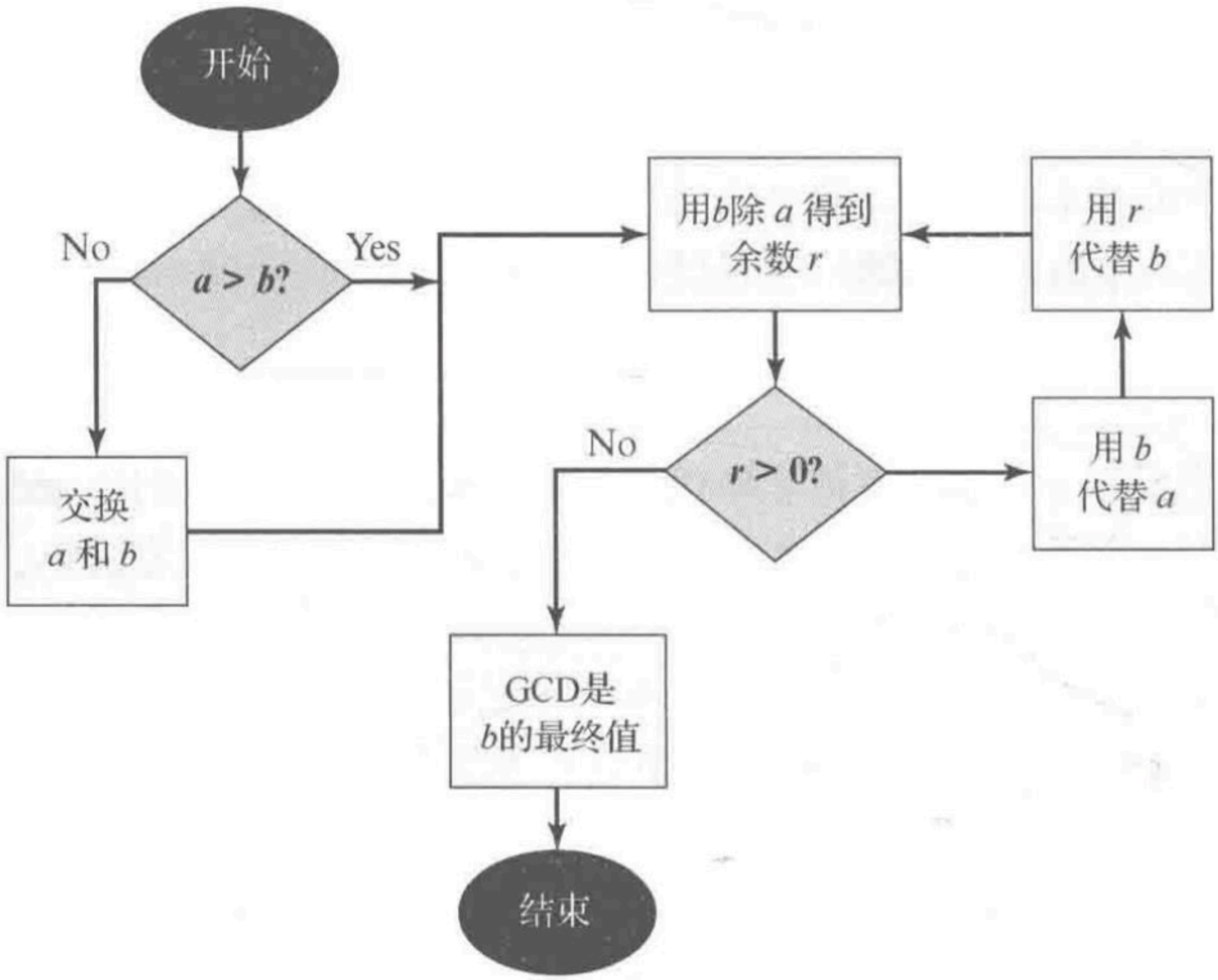
```
    for (int i = 0; i < k; i++) {
      ZZ a = ZZ::random_zz(2, n - 2);
      ZZ x = a.mod_pow(d, n);

      if (x == 1 || x == n - 1) {
        continue;
      } else {
        bool ok = false;

        for (int j = 0; j < r - 1; j++) {
          x = (x * x) % n;

          if (x == n - 1) {
            ok = true;
            break;
          }
        }

        if (ok) {
          continue;
        } else {
          return false;
        }
      }
    }

    return true;
}
```



输入大数n
d = n - 1
r = 0
d mod 2 = 0 ?
NO
YES
i = 0
d = d / 2
r = r + 1
i < k ?
YES
随机选取a in [2, n - 2]
计算x = a^d mod n
NO
return OK
YES
YES
x = 1 or x = n - 1 ?
YES
j < r - 1 ?
flag = false
j = 0
x = x * x mod n
NO
flag = true ?
YES
x = n - 1 ?
NO
return false

Part 2. 计算乘法逆元

在RSA公钥加密体制中，给定公钥e，其中$\gcd(e, \varphi(n)) = 1$。我们需要计算私钥$d \equiv e^{-1} \pmod{\varphi(n)}$，这可以通过扩展的欧几里得算法完成，在不停地利用带余除法之后我们可以倒推这一过程，实现逆元的求解。



C++ 实现

```
ZZ extended_euclidean(const ZZ& a, const ZZ& b, ZZ& x, ZZ& y) {
    if(b == 0) {
       x = 1;
       y = 0;
       return a;
    }

    ZZ x1, y1, gcd = extended_euclidean(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return gcd;
}
```

RSA加密体制的完整实现

1. 随机生成两个长度为512比特的随机数p, q，并计算得到n和欧拉函数$\varphi(n) = (p-1)(q-1)$.
2. 选取e，公约为$\langle e, n \rangle$.
3. 选取私钥d，并保证$ed \equiv 1 \pmod{\varphi(n)}$.
4. 密文为$c \equiv m^e \pmod{n}$.
5. 明文为$m = c^d \pmod{n}$.

加密解密测试

```
$ ./rsa_crypto
$ 'MTUyNDE1Nzg3ODA2NzM2Nzg1NDYxMDU3NzgyODEwNTQ3MjA1MTU2MjI2MjA3NTAxOTA1MjE====' # Base64编码
$ '123456789123456789123456789'
```

使用方法

安装好CMake，版本大于等于3.20：

```
$ sudo apt-get install cmake  # Ubuntu 用户
```

```
$ brew install cmake          # macOS 用户
```

可以参阅CMake安装。

Linux或macOS系统下，在当前目录下执行以下命令：

```
$ mkdir -p build
$ cd build
$ cmake ..
$ make
$ ./rsa_crypto
```