


《现代密码学》实验报告

实验名称：DLP计算	实验时间：2022.12.02
学生姓名：伍建霖	学号：20337251
学生班级：20网安	成绩评定：

一、实验目的

了解DLP问题的挑战性，计算离散对数，解决DLP问题。

二、实验内容

**参数**

- $p = 31050371851708889440695779044384182719244728783$;
- $p - 1 = 2 \times 1097 \times 1208925819614629174706189 \times 11706593258111142827$
- $q = 11706593258111142827$;
- $y = 13821904325547285207847180361637528630753679004$;
- $g = (\text{Hash}(\text{学号}) \bmod p)^{2 \times 1097 \times 1208925819614629174706189 \bmod p}$
- 目标：求解 $\log_g y$

三、实验原理

pollard rho算法伪代码：

```
main
定义划分  $G = S_1 \cup S_2 \cup S_3$ 
 $(x, a, b) \leftarrow f(1, 0, 0)$ 
 $(x', a', b') \leftarrow f(x, a, b)$ 
while  $x \neq x'$ 
do {  $(x, a, b) \leftarrow f(x, a, b)$ 
     $(x', a', b') \leftarrow f(x', a', b')$ 
     $(x', a', b') \leftarrow f(x', a', b')$ 
}
if  $\gcd(b' - b, n) \neq 1$ 
then return( "failure" )
else return  $((a - a')(b' - b)^{-1} \bmod n)$ 
```

先将G分为三部分，x落在不同部分时f的处理是不一样的。接着当x不等于x'时，x走一步，x'走两步，直至x等于x'。得到x等于x'后，计算b'-b和n的最小公约数，若为1则说明有且只有一个解，否则存在多解。

pollard rho中的f为“随机”函数，按以下三种情况处理：

```
procedure f(x, a, b)
if  $x \in S_1$ 
then  $f \leftarrow (\beta \cdot x, a, (b+1) \bmod n)$ 
else if  $x \in S_2$ 
then  $f \leftarrow (x^2, 2a \bmod n, 2b \bmod n)$ 
```

```

else  $f \leftarrow (\alpha \cdot x, (a+1) \bmod n, b)$ 
return( $f$ )

```

pollard rho算法的逻辑和实现是比较简单的，但计算量有点大。。。

四、实验步骤

初始化参数

为方便使用，先将参数初始化为mpz_class类型并赋值：

```

mpz_class p("31050371851708889440695779044384182719244728783");
mpz_class q("11706593258111142827");
mpz_class y("13821904325547285207847180361637528630753679004");
mpz_class g;
mpz_class p1;
mpz_class hash20337251("bb11f6ddc16eef19cf20d1ae91c06cf4c043341c
// mpz_class hash20337033("f74e3e0150f2fe7a5edf1cce9ba489503ff6b
// mpz_class hash20337211("c26fb36d26672e8d0a8d2b09dc30f22aaedb2
vector<mpz_class> f(vector<mpz_class>);
mpz_class pollard_rho();

```

其中p为群G的大小，q为alpha的阶n，y为beta，g为alpha。

f

随机函数f的实现比较简单，先用mpz_mod函数判断x落在S1，S2还是S3，再分别判断。若落在S1内，则x变为beta*x，a不变，b加1；若落在S2内，则x变为x*x，a变为2a，b变为2b；若落在S3内，则x变为alpha*x，b不变，a加1；

```

vector<mpz_class> f(vector<mpz_class> input)
{
    // alpha = g, beta = y
    // input[0] = x, input[1] = a, input[2] = b
    mpz_class inputMod, retX, retA, retB;
    mpz_mod(inputMod.get_mpz_t(), input[0].get_mpz_t(),
    mpz_class(3).get_mpz_t());
    switch (inputMod.get_ui())
    {
        case 1:
            // S1 -> (beta*x, a, b+1)
            mpz_mod(retX.get_mpz_t(), mpz_class(y * input[0]).get_mpz_t(),
            p.get_mpz_t());
            mpz_mod(retB.get_mpz_t(), mpz_class(input[2] + 1).get_mpz_t(),
            q.get_mpz_t());
            return {retX, input[1], retB};

        case 0:
            // S2 -> (x*x, 2a, 2b)
            mpz_mod(retX.get_mpz_t(), mpz_class(input[0] * input[0]).get_mpz_t(),
            p.get_mpz_t());
            mpz_mod(retA.get_mpz_t(), mpz_class(2 * input[1]).get_mpz_t(),
            q.get_mpz_t());
            mpz_mod(retB.get_mpz_t(), mpz_class(2 * input[2]).get_mpz_t(),
            q.get_mpz_t());
            return {retX, retA, retB};
        default:

```

```

        // S3 -> (alpha*x, a+1, b)
        mpz_mod(retX.get_mpz_t(), mpz_class(g * input[0]).get_mpz_t(),
p.get_mpz_t());
        mpz_mod(retA.get_mpz_t(), mpz_class(input[1] + 1).get_mpz_t(),
q.get_mpz_t());
        return {retX, retA, input[2]};
    }
}

```

pollard_rho

pollard ρ 算法则是建立在随机函数 f 上的：先初始化两组值 (x, a, b) ，都为 $(1, 0, 0)$ ，再循环：当 x_0 不等于 x_1 时， x_0 在 f 上走一步， x_1 在 f 上走两步，直至 x_0 等于 x_1 。接着判断 $b_1 - b_0$ 和 n 的最小公约数，若不为 1 则说明 $\log_g y$ 有多个解，否则有且只有一个解，即 $\log_g y = (a_0 - a_1)(b_1 - b_0)^{-1} \bmod q$

```

mpz_class pollard_rho()
{
    mpz_class counter_mc(0);
    mpz_class xVec[2] = {mpz_class(1), mpz_class(1)};
    mpz_class aVec[2] = {mpz_class(0), mpz_class(0)};
    mpz_class bVec[2] = {mpz_class(0), mpz_class(0)};
    cout << xVec[0] << "..." << aVec[0] << "..." << bVec[0] << '\n';
    cout << xVec[1] << "..." << aVec[1] << "..." << bVec[1] << '\n';

    while (xVec[0] != xVec[1] || xVec[0] == 1)
    {
        counter_mc++;
        vector<mpz_class> fret1 = f({xVec[0], aVec[0], bVec[0]});
        xVec[0] = fret1[0];
        aVec[0] = fret1[1];
        bVec[0] = fret1[2];
        vector<mpz_class> fret2 = f(f({xVec[1], aVec[1], bVec[1]}));
        xVec[1] = fret2[0];
        aVec[1] = fret2[1];
        bVec[1] = fret2[2];
    }

    cout << "counter is " << counter_mc << '\n';
    cout << xVec[0] << "..." << aVec[0] << "..." << bVec[0] << '\n';
    cout << xVec[1] << "..." << aVec[1] << "..." << bVec[1] << '\n';

    if (gcd(bVec[1] - bVec[0], p) != 1)
    {
        cout << "we failed....." << '\n';
    }
    mpz_class a_a;
    mpz_mod(a_a.get_mpz_t(), mpz_class(aVec[0] - aVec[1]).get_mpz_t(),
q.get_mpz_t());
    mpz_class b_b_in;
    mpz_invert(b_b_in.get_mpz_t(), mpz_class(bVec[1] - bVec[0]).get_mpz_t(),
q.get_mpz_t());
    mpz_class ret;
    mpz_mod(ret.get_mpz_t(), mpz_class(a_a * b_b_in).get_mpz_t(),
q.get_mpz_t());
}

```

```
    return ret;
}
```

优化

在一开始的时候，我20分钟只能算7千万次循环，也就是350万次每分钟，也就是6万次一秒。

然后我想着电脑风扇声不是很大，有没有可能电脑为了降温而降频了，就打开了性能模式，这时候快了接近20%。

接着我想看看不输出循环次数时的cpu占有率，结果发现不输出时的cpu占有率是输出时的1/3，故我把cout删了，只在结束时输出总的次数，这次估计为上次优化的3倍速度。

为了进一步优化，我在编译命令中加入了"-O2"和"-pthread"参数，但没测试时间。除此之外，我为了减少赋值操作，我参考python和js中的操作，直接将返回的参数打包返回了。

to do

1. 将vector等容器改为使用数组
2. 将传参操作改为引用传参，减少赋值操作

五、实验结果

完成代码后，先使用书上的例子验证代码的正确性：

```
12  mpz_class p("809");
13  mpz_class q("101");
14  mpz_class y("618");
15  mpz_class g;
16  mpz_class p1;
17  mpz_class hash20337251("bb11f6ddc16eef19cf20d1ae91c06
18
19  deque<mpz_class> f(deque<mpz_class>);
```

问题	输出	调试控制台	终端
	1...0...0 422...5...11 422...11...15 log_g_y is 49 it is ooko..... [1] + Done		"/usr/bin/gdb" --interprete

接着用身边循环次数较少的同学的学号来验证代码在数较大时的正确性，此时的速度为100万次/秒，循环次数为18亿次，大约用时30分钟。

```
问题  输出  调试控制台  终端

hashAfterMod is 18626012814766140080864904292778174772162497740
alpha is      9737870157291276853705272872169950298112433815
beta is      13821904325547285207847180361637528630753679004
G is        31050371851708889440695779044384182719244728783
n is        11706593258111142827
1...0...0
1...0...0
counter is 1828412480
7544099575029990869262892989122332972326691555...7021864833547520676...5371454946396587224
7544099575029990869262892989122332972326691555...8001239169261521527...7816878629163183010
log_g_y is 8478892599501116323
it is ooko.....
[1] + Done                                "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/M
icrosoft-MIEngine-In-qgycydw. mm2" 1>"/tmp/Microsoft-MIEngine-Out-htbzdof.2t4"
henry@DESKTOP-3NA4DUP:~/dlp$
```

接着计算自己的学号，循环次数为29亿次，大概用了1个小时。

```
问题  输出  调试控制台  终端

hashAfterMod is 27690837108983501164924418160349587928157801854
alpha is      12890384129642073454619238915463807140456810449
beta is      13821904325547285207847180361637528630753679004
G is        31050371851708889440695779044384182719244728783
n is        11706593258111142827
1...0...0
1...0...0
counter is 2933762216
25569858703298383673513843299049534364073259818...8524572893913985550...3672978265078077827
25569858703298383673513843299049534364073259818...1365583453163820368...1083751373679600452
4
log_g_y is 10834523123468354591
it is ooko.....
[1] + Done                                "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/M
icrosoft-MIEngine-In-xhom5pcr.l3f" 1>"/tmp/Microsoft-MIEngine-Out-piasjkql.gnp"
henry@DESKTOP-3NA4DUP:~/dlp$
```

帮同学计算循环次数，这次用了接近3个小时，循环了101亿次。

```
问题  输出  调试控制台  终端

hashAfterMod is 2238922074813049855593743453811087202462262687
alpha is      30985928467877424648137737637834604687361856401
beta is      13821904325547285207847180361637528630753679004
G is        31050371851708889440695779044384182719244728783
n is        11706593258111142827
1...0...0
1...0...0
counter is 10134937384
24689382600684669399273642783206669225456585267...1102125753741831527...9361030895787576019
24689382600684669399273642783206669225456585267...6851407816724872312...7611279522357741030
log_g_y is 7430123502323799773
it is ooko.....
[1] + Done                                "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/M
icrosoft-MIEngine-In-k3akrj2p.rkf" 1>"/tmp/Microsoft-MIEngine-Out-zdopxp5q.pdi"
henry@DESKTOP-3NA4DUP:~/dlp$
```

六、实验总结

这次的代码实现并不困难，就是计算量太大了，一开始速度慢，跑了8个小时都没跑出来，一度怀疑是不是自己的代码出了问题。当然跑出来后的成就感还是很大的。

我看身边同学有的速度可以达到200万次/秒，我问了一下他们的代码，感觉有可能就是传参那里拖慢了速度，多复制了一次。