Optimising Pollard's Rho algorithm for large semi-primes

I have programmed an implementation of Pollard's Rho factoring algorithm using C++ and the GMP library.

It is reasonably fast with large numbers, however I haven't implemented any form of cycle detection (I just try and avoid the problem, see below), and it struggles with small numbers.

Here is the pseudo code of my implementation:

```
input n
x = 2
y = 2
d = 1
c = 1
z = 1
failed = 0
temp = 0

while d == 1 or d == n
    z = 1
    failed = failed + 1
    if failed = sqrt(n)
        c = c + 1
        failed = 0

    repeat 100 times:
        x = x^2 + c mod n
                repeat twice:
                    y = y^2 + c mod n
        temp = abs(x - y)
        z = z * temp

    d = gcd(z, n)
```

I was wondering how I could improve the efficiency of this algorithm? If I'm using large semiprimes, do I need to worry about cycle detection?

factorization

edited Oct 29 '12 at 16:07

Thomas
3,265 1 4 22

asked Oct 29 '12 at 15:06

Sam Kennedy
40 4

What's up with the "failed" variable? If $d == n$ then by definition of the gcd and since all your variables are taken modulo n it implies $x = y$ at which point you'd just be repeating work you've already done, so you might as well check for that and just increment c when that happens. Besides, for sufficiently large n, your "failed" variable will never reach $\sqrt{n}$ anyhow. – Thomas Oct 29 '12 at 15:28

I added the d == n after the algorithm kept outputting 15 as a factor of 15. I commented out the "failed" section of code for factoring 17014118346046923731643907203145087515 7 because of the same reason you mentioned, it factored it in 1 hour and 45 minutes, is that considered slow for this algorithm, or does it sound about right? – Sam Kennedy Oct 29 '12 at 16:06

1 You should not measure it in terms of time, but in terms of elementary operations done. Try counting the number of times the inner loop is called (the number of times you z = z * temp), if you implemented Rho properly it should be on the order of $\sqrt[4]{n}$, or more accurately $\sqrt{p}$ where p is the factor found. Also, when you increment c, you want to reset x and y to 2 (I think). – Thomas Oct 29 '12 at 16:10

A few more questions, do x and y always have to start at 2? Does the function always have to be in the form of x^2 + c? Can c be a negative value? Also, the number of iterations is less than the square root of p, I'm assuming that's a good thing and I have it implemented properly? – Sam Kennedy Oct 30 '12 at 21:59

1 @SamKennedy: a reference on Pollard's Rho is the HAC; you want chapter 3, algorithm 3.9. – fgrieu Oct 31 '12 at 7:23

2 Answers

Might as well make this an answer, enough comments. This'll answer your latest questions:

- I don't think they have to start at 2, however they need to start at the same value (as per the cycle detection algorithm). So 2 is the simplest choice, and since the function is assumed to trace a pseudorandom sequence, it doesn't really matter where you start.
- The function can be any pseudorandom function satisfying the Rho property (gcd is conserved by iterating through it, basically it inherently "remembers" whenever you capture a factor of n without having to store all the numbers, this is the core concept). But $x^2 + c$ is the simplest one, so it's often the best choice as it's fast to calculate and produces a relatively chaotic sequence mod p.
- c can be negative, because the function is taken modulo n, so a negative value is still positive, however it cannot be 0 or -2 (to prevent trivial cycles in $x = 0$ and $x = \pm 1$).
- Yes, I said "on the order", this is a probabilistic algorithm so it's usually either a bit less, either a bit more.. it depends on c and on the factor found. The idea being that it's very fast on *small* factors (~20 digit prime factor maximum, so I would say a 40-50 digit semiprime is the best you're going to get with Pollard's Rho in reasonable time, which is still pretty good considering its simplicity)

edited Oct 31 '12 at 1:03

answered Oct 31 '12 at 0:45

Thomas
3,265 1 4 22

"I haven't implemented any form of cycle detection" - the obvious place to start is to fix this and actually implement the Pollard's rho algorithm.

If you don't have cycle detection, it's not the Pollard's rho algorithm any longer; it's some other variant you invented. If you start changing around the algorithm, you shouldn't be surprised if the algorithm performs poorly.

answered Oct 31 '12 at 12:16

D.W.
15.5k 3 17 57