University College London
Department of Computer Science

Cryptanalysis Lab 6

J. P. Bootle

March 8, 2017

jonathan.bootle.14@ucl.ac.uk
Version 2.0

Implementing the Pollard-Rho Algorithm

Click on the green letter before each question to get a full solution. Click on the green square to go back to the questions.

Exercise 1.

(a) Write a function `pollard_rho` which implements the low-memory version of the Pollard-Rho algorithm. The function should take input N, and output a, b such that $N = ab$. Run the algorithm for a fixed number of iterations. You may wish to structure your code as follows.

- Definition of a sub-function for iteration.
- Set the number of iterations to do.
- Main loop using the iterative function.
- At each step of the main loop, compute a greatest common divisor.
- Return a factorisation $[a, b]$ or output ‘Fail’.

(b) According to the analysis of the running time of the Pollard-Rho algorithm, how many iterations should we expect to use before the algorithm succeeds in finding a factorisation?

(c) Test your algorithm by attempting to factorise the integers $M_n = 2^n - 1$, for $n = 80, 85, 90$. What is the largest value of n that your program can handle in 10 seconds?

Solutions to Exercises

Exercise 1(a) The following code implements the Pollard-Rho algorithm.

```
def pollard_rho(N):
    n = floor(sqrt(sqrt(N))) # adjust this value
    ai = randint(1,N-1)
    a2i = ai
    for k in range(1,n):
        ai = (ai*ai + 1) % N
        a2i = (a2i*a2i + 1) % N
        a2i = (a2i*a2i + 1) % N
        d = gcd(abs(ai-a2i),N)
        if not (d in [1,N]):
            return [d,floor(N/d)]
    return 'fail'
```

□

Exercise 1(b) According to the heuristic analysis based on the Birthday paradox, we would expect to succeed after $O(\sqrt{p})$ iterations, where p is the smallest prime factor of N. □