The University of Sydney
Math3925 Public Key Cryptography

Semester 2	**Exercises for Week 6**	2004

1. A simple Pollard Rho factorization algorithm can be implemented in just a few lines in `Magma`:

```
function PollardRho(n,a)
    x := Random([1..n]);
    x := (x^2+a) mod n;
    y := (x^2+a) mod n;
    while GCD(x-y,n) eq 1 do
        x := (x^2+a) mod n;
        y := (y^2+a) mod n;
        y := (y^2+a) mod n;
    end while;
    return GCD(x-y,n);
end function;
```

a. Use this algorithm to find a factorization of

$$2^{29}-1, 2^{59}-1, 2^{2^6}+1, \text{ and } 400731052007683.$$

b. What happens if the input argument n is prime?

2. The Pollard rho algorithm is effective for solving discrete logarithms in subgroups of fields $\mathbb{F}_p^*$ of moderate size. In the following code we make the assumption that the subgroup order is a prime n. The following code implements a Pollard rho discrete logarithm. You will need to first include the iteration function `PollardIteration`, presented below, in which the three disjoint sets S_1, S_2 and S_3 are those finite field elements with representatives x in intervals $1 \le x \le B_1$, $B_1 < x \le B_2$, and $B_2 < x \le p-1$ respectively.

```
procedure PollardIteration(~t,a,b,B1,B2);
    x := Integers()!t[1];
    if x le B1 then
        t[1] *:= b; t[3] +:= 1;
    elif x le B2 then
        t[1] ^:= 2; t[2] *:= 2; t[3] *:= 2;
    else
        t[1] *:= a; t[2] +:= 1;
    end if;
end procedure;
```

Assuming that the function `PollardIteration` the main body of the function, below, creates in a deterministic fashion a new triple $(x_{i+1}, n_{i+1}, m_{i+1})$ consisting of the sequence element x_{i+1} together with the exponents (n_{i+1}, m_{i+1}) such that $x_{i+1} = a^{n_{i+1}} b^{m_{i+1}}$ from a similar sequence (x_i, n_i, m_i).

```
function PollardRhoLog(a,b,p,n)
    error if not IsPrime(p), "Argument 3 must be prime";
    error if not IsPrime(n) or (p-1) mod n ne 0,
        "Argument 4 must be a prime divisor of", p-1;
    K := FiniteField(p);
    R := FiniteField(n);
    a := K!a; b := K!b;
    error if Order(a) ne n /* or Order(b) notin {1,n} */,
      "Arguments 1 and 2 must have order", n, "mod", p;
    t1 := <K!1,R!0,R!0>; t2 := t1;
    B1 := p div 3; B2 := (2*p) div 3;
    while true do
        PollardIteration(~t1,a,b,B1,B2);
        PollardIteration(~t2,a,b,B1,B2);
        PollardIteration(~t2,a,b,B1,B2);
        if t1[1] eq t2[1] then break; end if;
    end while;
    r := t1[3]-t2[3];
    if r eq 0 then return -1; end if;
    return Integers()!(r^-1*(t2[2]-t1[2]));
end function;
```

The `Magma` tuple `<K!1,R!0,R!0>` represents the element $(1, 0, 0)$ of $K \times R \times R$. The notation `~t` is a pass-by-reference in which the argument can be modified in the course of the procedure. Note that the algorithm can fail, and if so, returns the value of -1.

- **a.** Use this algorithm to find discrete logarithms of 3, 7, and 17 with respect to the base 2 in $\mathbb{F}_p^{*2}$, where $p = 536871263$. Note that $n = (p-1)/2$ is a prime. Verify the correctness of the results.
- **b.** Note that each of the primes 2, 3, 7, and 17 are squares modulo p. What is the significance of the output of the algorithm when the discrete logarithm of 5 and 11 are computed with respect to the base 2?
- **c.** Find the discrete logarithm of 3 with respect to the base 2 in $\mathbb{F}_p^*$, where $p = 1234619627$. Make use of the Pollig-Hellman reduction, noting that $p - 1 = 2 \cdot 37 \cdot 61 \cdot 479 \cdot 571$.