The University of Sydney
School of Mathematics and Statistics

Computer Tutorial 8 (Week 9)

MATH2068/2988: Number Theory and Cryptography — Semester 2, 2017

Web Page: http://www.maths.usyd.edu.au/u/UG/IM/MATH2068/
Lecturer: Dzmitry Badziahin

1. MAGMA can do calculations with polynomials. Tell it to work with polynomials in one variable `x` over the integers via the command

```
P<x>:=PolynomialRing(Integers());
```

Then try some calculations to get used to some of the basic operations. For example:

```
f:=x^5-2*x^4-8*x^2-x-6;
g:=x^2+1;
g^3;
g^50;
g^6+f;
Evaluate(g,2);
Factorization(f);
g^10 mod f;
```

2. Type (or copy and paste) the following definition of a MAGMA function, which attempts to find a factor of an integer by Pollard's Rho method.

```
PRho:=function(n,f,s);
  k:=0; a:=s; b:=s;
  repeat
    k:=k+1;
    a:=Evaluate(f,a) mod n;
    b:=Evaluate(f,b) mod n;
    b:=Evaluate(f,b) mod n;
    d:=GCD(b-a,n);
    "x[",k,"] is",a,",x[",2*k,"] is",b,"and gcd(",b,"-",a,",",n,") =",d;
  until d gt 1;
  if d eq n then
    return "Failure, no proper factor found:",d,"Number of steps:",k;
  else
    return "Factor found:",d,"Number of steps:",k;
  end if;
end function;
```

Here n is the number we wish to factorize, f is a polynomial (usually x^2+1) and s is the starting value (usually 1). Consider the sequence of numbers x_0, x_1, x_2 ... where $x_0 = s$ and x_i is the residue of $f(x_{i-1})$ modulo n for all $i \geq 1$. In the MAGMA code above, `a` and `b` are initially set equal to x_0. Note that k keeps track of the number of times the `repeat ... until` loop is executed. In each pass through the loop, f is applied once to

`a` and twice to `b`, and both are reduced modulo n. So each time d is calculated, `a` will equal x_k and `b` will equal x_{2k}; hence $d = \gcd(x_{2k} - x_k, n)$. The function stops the first time that this divisor d of n is different from 1. If we are unlucky, $d = n$ which tells us nothing about the factorization of n; if we are lucky, $d < n$ and we have found a proper divisor. To test the function, try (for example) the commands `PRho(1001,x^2+1,1);` or `PRho(2^32+1,g,1);`

Now type `load "tut9data.txt";` This will replace the `PRho` function above by a version that is exactly the same except that it doesn't print out anything until the end.

3. To test how long Pollard's Rho method takes to factorize a number which is the product of two primes, type:

```
p:=RandomPrime(28);
q:=RandomPrime(28);
n:=p*q;
time PRho(n,x^2+1,1);
```

and repeat this a few times. Also try replacing x^2+1 by other polynomials, or 1 by other starting values, to see if it makes any difference to the time. (If it ever takes too long to complete, you can interrupt it with Control-C.) To compare to the time taken by trial division, use

```
time TrialDivision(n,2^28);
```

Change 28 to 29, then 30, etc. to see how quickly the time grows.

4. The file tut9data.txt you have loaded contains the definitions

```
NNN:=123400858035915200140425671420719842006201313178304205983293282416905 1;
PHI:=123400858035915200140425671420612990374670675607454495999499431226810 4;
```

and you are given the following information: `NNN` is the product of two primes p and q, and `PHI` equals `EulerPhi(NNN)`. As seen in lectures, this information is enough to find the primes p and q, since they are the roots of the quadratic equation $x^2 - (p+q)x + pq = 0$, where $p + q = pq - \phi(pq) + 1$.

Type `pplusq:=NNN-PHI+1;` and `D:=pplusq^2-4*NNN;` to define the discriminant of the quadratic, which must be a perfect square since the roots are integers. We need to find the square root of `D`, but typing `Sqrt(D);` is inadequate because it returns a real number, and only gives accuracy to 30 significant figures. Instead, use the `IsSquare` function: if `D` is a perfect square then `z,E := IsSquare(D);` puts `z` equal to `true` and `E` equal to the square root of `D`. Then find p and q, checking that they are prime with the `IsProbablyPrime` function.

5. The MAGMA function `Modorder(a,n);` returns the order of a modulo n. This quantity, which was denoted by $\mathrm{ord}_n(a)$ in lectures, is the least positive integer m such that $a^m \equiv 1 \pmod{n}$. (Note that a must be coprime to n for this to be defined; `Modorder` will return 0 if $\gcd(a, n) > 1$.) Use the MAGMA commands

```
n:=73;
for i:=1 to 72 do
   i,"has order",Modorder(i,73),"modulo 73";
end for;
```

to find $\mathrm{ord}_{73}(a)$ for every `a` in the set of nonzero residues modulo 73 (which is prime, of course). Note that for each divisor d of 72 there are $\phi(d)$ residues mod 73 with order d.

6. If p is prime, `FiniteField(p)` is MAGMA's name for the set of residues mod p. Type the command `F:=FiniteField(73);`, thus defining `F` to be the set of residues mod 73. Integers can be "coerced" into `F` by use of MAGMA's coercion operator `!`. Type `a:=F!37;` and `r:=F!5;` to define `a` to be 37 considered as an element of `F` and `r` to be 5 considered as an element of `F`. Sums and products of elements of `F` are computed using modular arithmetic: this means that the answers are always reduced modulo 73 and are viewed as elements of `F`. For instance, check what happens when you type command such as

```
a+a; r^3; r*a; a eq F!110;
```

Since $\mathrm{ord}_{73}(5) = 72$, the powers `r^i` of `r` give all 72 nonzero elements of `F` as `i` runs through the numbers 0, 1, 2, ..., 71. If `t` is a nonzero element of `F`, then the `i` such that `r^i` equals `t` is called the *discrete logarithm of* `t` *to the base* 5 *modulo* 73. Get MAGMA to compute the discrete logs to the base 5 of all nonzero elements of `F`:

```
for t:=1 to 72 do
   "The discrete log to the base 5 of",t,"modulo 73 is",Log(r,F!t);
end for;
```

Check some of the answers by computing `r^i` for various values of `i`.

7. Get MAGMA to create another finite field, via `F109:=FiniteField(109);`. Whenever MAGMA creates a finite field it chooses a "primitive element", which has the property that all nonzero elements of the finite field are powers of the primitive element. Type `PrimitiveElement(F109);` to discover the primitive element MAGMA has chosen for `F109`. If you do not specify a base for discrete logarithms, MAGMA will use its chosen primitive element as the base. Type `t:=F109!50;` and `Log(t);` and check the answer by calculating the appropriate power of the primitive element. Find values of `s` and `t` in `F109` for which `Log(s*t)` is different from `Log(s) + Log(t)`.