The University of Sydney
School of Mathematics and Statistics

# Computer Tutorial 9 (Week 10)

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

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

Start MAGMA and type `load "tut10data.txt";`

**1.** Suppose that we are given a prime $p$ and positive integers $r$, $n < p$, and we are told that $r^x \equiv n \pmod{p}$ for some unknown integer $x$. The *discrete logarithm problem* is the problem of finding $x$. The security of the Elgamal cryptosystem relies on the computational difficulty of this problem. The best known algorithms for solving it have a speed that depends on the size of the largest prime factor of $p-1$.

In this exercise we investigate how long it takes MAGMA to find discrete logarithms. First, define `p:=2^32+15;` and check that $p$ is prime using the `IsPrime` command. Use the `Factorization` command to find the prime factorization of $p-1$. Then type

```
F:=FiniteField(p);
r:=PrimitiveElement(F);
r;
for i:=1 to 20 do
 time Log(r,Random(F));
end for;
```

Then try the same again for $p = 2^{32} + 61$, for which $p-1$ has a larger prime factor, and observe that it takes more time to find discrete logarithms.

How long MAGMA will take to compute discrete logarithms is quite unpredictable: sometimes the algorithm is lucky and finds the answer quickly, sometimes not. The MATH2068 startup file includes a procedure `timelog`. For example, `timelog(50);` chooses a random 50 bit prime $p$, prints out the base 10 logarithm of the square root of the largest prime factor of $p-1$, and then prints the time taken to compute a random discrete logarithm. Try this same command a few times, with various values in place of 50, to see the effect on the time of increasing the number of bits.

**2.** If it takes 1 second to compute a discrete logarithm when $p$ has 100 bits, and if the time doubles when the number of bits increases by 10, how many seconds will it take when $p$ has 660 bits? How many centuries is this?

**3.** This exercise illustrates the Elgamal cryptosystem. In this cryptosystem, the *public key* is a triple $(p, b, k)$ where $p$ is a prime, $b$ is a positive integer less than $p$, and $k$ is the residue of $b^m$ modulo $p$ for some secret integer $m$ (the *private key*). To choose a suitable prime $p$, type

```
p:=NextPrime(Random(10^199,10^200):Proof:=false);
p;
```

(The "`Proof:=false`" bit tells MAGMA not to bother proving rigorously that `p` is prime.)

Then define `b:=2;` and `m:=Random(p-1);`. It is better for the private key $m$ to be coprime to $p-1$, so test `GCD(m,p-1)` and choose $m$ again, until this is the case. Then define

```
k:=Modexp(b,m,p);
k;
```

The function `NaiveEncoding`, defined in MagmaProcedures.txt, encodes text as a sequence of integers of at most 198 digits each. Try it on a sample sentence, e.g.

```
xx:=NaiveEncoding("In the words of Abraham Lincoln,
people who like this sort of thing
will find this the sort of thing they like");
xx;
```

To encipher `xx` with the Elgamal key $(p, b, k)$, we need to choose a secret exponent $i$ and compute the *scrambling factor*, which is the residue of $k^i$ modulo $p$, and which we use to multiply each integer in the sequence. Do this with the following commands:

```
i:=Random(p-1);
sf:=Modexp(k,i,p);
ct:=<Modexp(b,i,p),[sf*t mod p: t in xx]>;
ct;
```

Here `ct` is the ciphertext, which has two components: `ct[1]`, which is the residue of $b^i$ modulo $p$, and `ct[2]`, which is the sequence of integers obtained from `xx` by multiplying each term by the scrambling factor and reducing modulo $p$. The intended recipient, who knows the private key $m$, can determine the scrambling factor since it equals the residue of $(b^i)^m$ modulo $p$, and hence find the mod-$p$ inverse of the scrambling factor. Thus, the following commands should recover the encoded plaintext:

```
isf:=Modexp(ct[1],-m,p);
pt:=[isf*u mod p : u in ct[2]];
```

To check that it has worked, type `NaiveDecoding(pt);`.

**4.** Repeat the previous exercise with the text `camel`, loaded in tut10data.txt.

**5.** The `NaiveEncoding` and `NaiveDecoding` functions use the intrinsic MAGMA functions `StringToCode` and `CodeToString`. These convert characters to numbers, and vice versa, using ASCII (the American Standard Code for Information Interchange, or an Australian version thereof). Type `CodeToString(65);` and `StringToCode("A");`, and find the code numbers of some other characters similarly. Then type `NaiveEncoding("A");` and `NaiveEncoding("AB");` and/or other similar commands, and figure out exactly what `NaiveEncoding` does. In particular, what does `NaiveEncoding` do to a string that is more than 66 characters long?

**6.** As proved in lectures, if $p$ is a prime then for each divisor $d$ of $p-1$ there are $\phi(d)$ residues mod $p$ whose order mod $p$ is $d$. In particular, there are $\phi(p-1)$ primitive roots mod $p$,

so a randomly chosen nonzero residue has a fairly good chance of being a primitive root. Choose a random 50 bit prime $p$ with the command `p:=RandomPrime(50);` and try some random numbers less than $p$ with the command `a:=Random(p-1);` until you find one that returns `true` to the question `IsPrimitive(a,p);`. It should not take too many attempts.

**7.** In Exercise 1 above we asked MAGMA to find a primitive root $r$ mod $p$, which was very quick since the prime $p$ there had only 33 bits. However, when $p$ gets much larger, finding a primitive root becomes significantly slower. Hence in Exercise 3, where $p$ had 200 decimal digits, we did not choose $b$ to be a primitive root but simply put $b = 2$. (Fortunately, Elgamal does not actually need $b$ to be a primitive root, although the larger its order is, the better.) Let us investigate how long MAGMA takes to find primitive roots. Type

```
p:=RandomPrime(150);
time PrimitiveRoot(p);
```

Repeat this a couple of times, then replace 150 by 160, then by 170, and so on, until it starts taking too long. Notice that the times can vary greatly for different primes of similar sizes; when you come across a prime $p$ where it takes an unexpectedly long time for MAGMA to find a primitive root, see if you can identify what property of $p$ (or rather, of $p - 1$) is responsible.