

### Computer Tutorial 9

Start MAGMA and begin with a `SetLogFile( ... )` command. Type `load "tut9data.txt";`

1. Suppose that we are given a prime  $p$  and nonzero residues  $r, n, m$  modulo  $p$ . Suppose also that we are told that  $n \equiv r^x$  and  $m \equiv r^y$  for some  $x, y$ . Can we find the residue of  $r^{xy}$ ? This is known as the *Diffie-Hellman problem*, and security of the Elgamal cryptosystem relies on its computational difficulty. The Diffie-Hellman problem seems to be no easier than the *discrete logarithm problem*: given  $p, r, n$  as above, find  $x$  such that  $r^x \equiv n$ . (An efficient discrete logarithm algorithm would obviously give an efficient Diffie-Hellman algorithm, since it is computationally easy to find the residue of  $r^{xy}$  given  $r, x$  and  $y$ .)

The best known algorithms for the discrete logarithm have 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 logs. First, define `p:=2^32+15;`, and check that `p` is prime (via `IsPrime(p);`). Next, type `Factorization(p - 1);`, and 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  $p = 2^{32} + 61$ , and observe that this is this slower. (Why?)

How long it will take to compute discrete logarithms is quite unpredictable: sometimes the algorithm “gets lucky”, and finds the answer quickly, sometimes not. I have included in our magma startup file a procedure `timelog`. Type `timelog(50);`. This 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 the time taken to compute a random discrete logarithm. Try this same command a few times, with various values in place of 50. (I found it much faster if the largest prime factor of  $p - 1$  is small, but otherwise increasing the number of bits by 10 roughly doubled the time.)

2. If it takes 1 second to compute a discrete log when  $p$  has 100 bits, and if increasing the number of bits by 10 doubles the time, how long will it take when  $p$  has 660 bits?
3. Choose an Elgamal key: type `p:=NextPrime(Random(10^199,10^200):Proof:=false); p;` (The “Proof:=false” bit tells MAGMA not to worry about rigorously proving that  $p$  is prime.) Then put `b:=2; m:=Random(p-1);`. We prefer  $m$  to be coprime to  $p - 1$ . If `GCD(m,p-1)` is not 1, choose  $m$  again. When you have an  $m$  coprime to  $p - 1$ , type `k:=Modexp(b,m,p); b;`. Your Elgamal public key will be  $(p, b, k)$ , your private key being  $m$ .

Use the function `NaiveEncoding` (defined in `MagmaProcedures.txt`) to convert some text into a sequence of integers of at most 198 digits each. (e.g. `xx:=NaiveEncoding("People who like this sort of thing will find this the sort of thing they like"); xx;`) To encipher `xx` with the Elgamal key  $(p, b, k)$ , use the following commands:

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

Then `ct` is the ciphertext. Type `ct;`. Note that `ct` has two components, `ct[1]`, which is a number, and `ct[2]`, which is a sequence of numbers. Deciphering is achieved by

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

(Note that `isf` is the inverse (modulo  $p$ ) of the “scrambling factor” `sf`.)

To check that it has worked, try `pt eq xx;` and/or `NaiveDecoding(pt);`

Just performing the MAGMA commands above without trying to understand them is somewhat pointless. Please consult the description of the Elgamal Cryptosystem given in lectures and try to reconcile the MAGMA commands with that description.

The next exercise is just a repeat of the last one using a different plaintext.

4. Encipher and decipher the plaintext `camel` (which was loaded in `tut9data.txt`):

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

Type `ct`; to see the ciphertext. Then decipher it:

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

5. The `NaiveEncoding` and `NaiveDecoding` functions use MAGMA's inbuilt functions `StringToCode` and `CodeToString`. These convert characters to numbers and vice versa. Type `CodeToString(65)`; and `StringToCode("A")`; . Find the code numbers of several other characters. 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 (and see also Exercise 7 of Computer Tutorial 8), 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  $p$ . This seems to suggest that a randomly chosen nonzero residue has a good chance of being a primitive root. Choose a random 50 bit prime  $p$  and find the order mod  $p$  of random numbers less than  $p$ . See how long it takes before you hit upon a primitive root. (`p:=RandomPrime(50)`; `a:=Random(p-1)`; `IsPrimitive(a,p)`;).

In fact, the probability of finding a primitive root by chance is  $\phi(p-1)/(p-1)$  (the number of primitive roots divided by the total number of nonzero residues), which equals  $\prod_{d|(p-1)} \frac{d-1}{d}$ , where  $d$  runs through the prime divisors of  $p-1$ . Observe that this is always less than 0.5 (why?) but may be close to 0.5. Repeat the commands `a:=Random(p-1)`; `IsPrimitive(a,p)`; many times, and see if the relative frequency of primitive roots is consistent with the above formula for the probability. Repeat the experiment for several primes  $p$ .

7. In Exercise 1 above we chose  $r$  to be a primitive root mod  $p$ , and it did not take MAGMA long to find one. However, when  $p$  gets very large, this becomes slow. Hence in Exercise 3 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. (Note that the times can vary greatly for different primes of similar sizes.) Why does it start taking a long time, even though a random element is still quite likely to be a primitive root? [Hint: For the same reason `EulerPhi(p-1)`; starts taking a long time when  $p$  gets large.]

8. Let

```
NNN:=1234008580359152001404256714207198420062013131783042059832932824169051;
PHI:=1234008580359152001404256714206129903746706756074544959994994312268104;
```

There is no need to type in these values; they were loaded in `tut9data.txt`. (Type `NNN`; `PHI`; to confirm this.) Given that `NNN` is the product of two primes  $p$  and  $q$ , and `PHI` equals `EulerPhi(NNN)`, find  $p$  and  $q$ . [Hint: The numbers  $p$  and  $q$  are the roots of the quadratic equation  $x^2 - (p+q)x + pq = 0$ . You can find the value of  $p+q$  since  $p+q = pq - \phi(pq) + 1$ . To solve this equation you must find the square root of the discriminant  $D$ . You will need to use the `IsSquare` function: if  $D$  is a perfect square then `x,E := IsSquare(D)`; puts `x` equal to `true` and `E` equal to the square root of  $D$ . (The function `sqrt` is inadequate here since it returns a real number, and only gives accuracy to 30 significant figures.)]