

Discrete Logarithms and the Elgamal Cryptosystem

MATH2068 Number Theory & Cryptography
Week 9 Lecture 3

University of Sydney
NSW 2006
Australia

19th September 2007



Primitive roots

Let p be a prime. A *primitive root modulo p* is a number whose order mod p is $p - 1$.

That is, a primitive root is a b such that $\text{ord}_p(b) = p - 1$.

Remember that $\text{ord}_p(a)$ is defined if and only if a is coprime to p (i.e. $\text{gcd}(a, p) = 1$).

By Fermat's Little Theorem, if $\text{gcd}(a, p) = 1$ then $\text{ord}_p(a)$ is a divisor of $p - 1$.

So a primitive root is a number whose order has the maximum possible value.

Example



Let $p = 31$. The following results show that 3 is a primitive root.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3^i	1	3	9	27	19	26	16	17	20	29	25	13	8	24	10
i	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
3^i	30	28	22	4	12	5	15	14	11	2	6	18	23	7	21

Of course we know that 3^i gets back to 1 when $i = 30$ (by Fermat). Then the sequence must repeat.

Notice that all the numbers from 1 to $p - 1$ appear in the 2nd row before this repetition starts. It is like this for any primitive root.

Everything is a power of a primitive root



If $\text{gcd}(a, p) = 1$ and $k = \text{ord}_p(a)$ then $a^0, a^1, a^2, \dots, a^{k-1}$ are all different mod p .

For, if we had $a^m \equiv a^n \pmod{p}$ with $0 \leq n < m < k$, then coprime cancellation would give $a^{m-n} \equiv a^0 = 1$, contradicting the fact that k is the least positive integer l such that $a^l \equiv 1$.

So $a^k \equiv a^0$ is the first repetition in the sequence of powers. Hence the sequence is periodic with period equal to $\text{ord}_p(a)$.

Now suppose that b is a primitive root mod p .

The mod p residues of b^0, b^1, \dots, b^{p-2} must all be different. But there are $p - 1$ of them, and of course they are all nonzero.

So, by the Pigeonhole Principle, the residues b^0, b^1, \dots, b^{p-2} give us all $p - 1$ nonzero residues, in some order.

Discrete logarithms



Let b be a primitive root mod p (where p is a prime).

For each nonzero residue a there is an $i \in \{0, 1, 2, \dots, p-2\}$ such that

$$a \equiv b^i \pmod{p}.$$

We say that i is the discrete logarithm of a to the base b .

This is usually written

$$i = \log_b(a).$$

But we must not forget that everything here depends on the prime p , which we are keeping fixed.

Discrete logarithms versus ordinary logarithms



So that we do not forget p , and do not confuse discrete logarithms to the base b with ordinary logarithms to the base b , I will usually write $\log_{b,p}(a)$ for the discrete logarithm, and keep $\log_b(a)$ for the ordinary logarithm

Discrete logarithm:

$$\log_{b,p}(a) = i \text{ means } a \equiv b^i \pmod{p}.$$

Ordinary logarithm:

$$\log_b(a) = \alpha \text{ means } a = b^\alpha.$$

The ordinary logarithm, $\log_b(a)$, is a **real number**.

The discrete logarithm, $\log_{b,p}(a)$, is a **residue mod $p-1$** .

Computing discrete logarithms is hard!



Problem: compute $\log_{3,31}(22)$.

From one point of view this is easy: there is an obvious method.

Just compute the residues mod 31 of 3^i for all i from 1 to 29, until we find the i such that $3^i \equiv 22$.

(In fact we already did this: the answer is $\log_{3,31}(22) = 17$.)

But if p is enormous then this obvious procedure for computing $\log_{b,p}(a)$ takes forever: there are too many things to compute.

Computing $a^i \pmod{p}$ is easy and fast, but you have to do it for all i from 1 to $p-2$.

Analogy with factorization



Multiplying a and b to compute $n = ab$ is easy and fast.

But factorizing n is time-consuming: you have to search randomly until you find a factor.

OK, maybe you don't search *randomly* – but there is no way to avoid a lot of searching.

Similarly, if p and b are given, then for each i it is easy and fast to compute the residue of $b^i \pmod{p}$.

But if we are given a residue a and asked to find the i such that $b^i \equiv a$, you just have to try lots of values for i until you find it.

OK, we can improve on this a bit – but there is no way to avoid calculating lots of mod p residues of powers of b .

Elgamal Cryptosystem



I post on my web site a triple (p, b, k) consisting of

- ▶ a Very Large Prime p ,
- ▶ a primitive root, b , for the prime p ,
- ▶ the number k that is the residue of $b^m \pmod{p}$,

where m is a number that I keep secret.

A person who has an efficient method for computing discrete logarithms can easily compute $m = \log_{b,p}(k)$ and discover my secret number.

Fortunately for me, no such person exists.

Encryption



To send me a message that no one but I can read, Alice proceeds as follows.

She encodes her message as a residue mod p (or a sequence of residues mod p) using a standard method for encoding things numerically. (*I have to put the instructions for this on my website. E.g., I might say to use the function `NaiveEncoding` from our Magma tutorials.*)

Let M be the encoded message. That is, M is the plaintext, in the form of a residue mod p . (More generally it could be a sequence $[M_1, M_2, M_3, \dots]$.)

Alice chooses a random number i less than $p - 1$, & computes

- ▶ the residue of $b^i \pmod{p}$,
- ▶ the residue of $Mk^i \pmod{p}$.

She transmits these both to me.

What if someone intercepts the transmission?



I have publically released (p, b, k) , where b is a primitive root mod p , and $k \equiv b^m \pmod{p}$. But m is my secret.

Alice has sent me two numbers: b^i and Mk^i . Her message is M , but she disguised it by multiplying by the *scrambling factor* k^i .

Alice isn't telling anyone the value of i . She chose it randomly and destroyed it after use.

The chap who can compute discrete logs intercepts Alice's message, and notes the values of b^i and Mk^i .

He finds i by computing $\log_{b,p}(b^i)$, computes the scrambling factor k^i , computes its inverse s , hence finds $M = Mk^i s$.

It is just as well that he doesn't exist.

Decryption



I published the triple (p, b, k) .

Alice has sent me b^i and Mk^i . I need the scrambling factor k^i to recover M .

I know that $k = b^m$, and I have kept m secret.

I now compute (the residue of) $(b^i)^m$.

The point is that $(b^i)^m = b^{im} = b^{mi} = (b^m)^i \equiv k^i$. Thus I obtain the scrambling factor k^i .

I compute the inverse of $k^i \pmod{p}$. This is the residue s such that $k^i s \equiv 1 \pmod{p}$; it is computed via the extended Euclidean Algorithm.

I can now recover M , since $(Mk^i)s = M(k^i s) \equiv M \pmod{p}$.

The Diffie-Hellman Key Agreement Protocol



Alice and Bob wish to have a secret (electronic) conversation, but they know Eve is listening.

They agree on a Very Large Prime p and a primitive root b .

Eve intercepts all this!

Alice thinks of a secret number x , and sends $b^x \pmod{p}$ to Bob.

Now Bob and Eve both know b^x but not x .

Bob thinks of a secret number y , and sends $b^y \pmod{p}$ to Alice.

Now Alice and Eve both know b^y but not y .

Alice knows x ; so she can compute (the residue of)
 $b^{xy} = (b^y)^x$.

Bob knows y ; so he can compute (the residue of) $b^{xy} = (b^x)^y$.

Eve knows only b^x and b^y . She has no easy way to find b^{xy} .

Diffie-Hellman (continued)



Alice and Bob both know the residue of $b^{xy} \pmod{p}$.

Now, for example, they can express this quantity in binary notation, and agree to use the first 56 bits as a key for DES, and use the next 56 bits as a second key for DES, and the next 56 bits as a third key for DES.

Now Alice and Bob can communicate safely and easily using triple DES.

They have managed to agree on a key while communicating over an insecure channel. Yet no eavesdropper can discover their key, except by solving the following problem.

The Diffie-Hellman Problem: Given the mod p residues of b^x and b^y , find the mod p residue of b^{xy} .

You can do this if you can compute discrete logarithms (i.e. if you can find x given r^x). But there is no other way known.