CPSC 467: Cryptography and Security

Michael J. Fischer

Lecture 12
October 8, 2020

Discrete Logarithm

Diffie-Hellman Key Exchange

ElGamal Cryptosystem

Primitive Roots

- Properties of primitive roots
- Lucas test
- Special form primes

Discrete Logarithm

Ordinary logarithms

Let $y = b^x$ over the reals, $b \neq 0$. The ordinary base-b logarithm is the inverse of exponentiation, so $x = \log_b(y)$.

In other words, $\log_b(y)$ is the power x to which b must be raised to equal x.

Logarithms mod p

The discrete logarithm is defined similarly. Let $y = b^x \bmod p$, where p is a prime, $b \in \mathbf{Z}_p^*$, and $x, y \in \mathbf{Z}_p$.

The base-b discrete logarithm modulo p is the inverse of exponentiation in $\mathbf{Z}_p$. We say $x = \log_b(y)$ modulo p.

In other words, $\log_b(y)$ modulo p is the least non-negative integer x to which b must be raised modulo p to equal y (if it exists).

Why might x not exist? Take $p = 7$, $b = 2$, $y = 3$. The successive powers of b mod p are $2, 4, 1, 2, 4, 1, \ldots$, so $\log_2(3) \pmod 7$ does not exist.

Fact (not needed yet): If b is a *primitive root*[1] of p, then $\log_b(y)$ exists for every $y \in \mathbf{Z}_p^*$.

[1] We will talk about primitive roots later.

Discrete log problem

The *discrete log problem* is the problem of computing $\log_b(y) \bmod p$, where p is a prime and b is a primitive root of p.

No efficient algorithm is known for this problem and it is believed to be intractable.

However, the inverse of the function $\log_b() \bmod p$ is the function $\text{power}_b(x) = b^x \bmod p$, which is easily computable.

power_b is believed to be a *one-way function*, that is a function that is easy to compute but hard to invert.

Diffie-Hellman Key Exchange

Key exchange problem

The *key exchange problem* is for Alice and Bob to agree on a common random key k.

One way for this to happen is for Alice to choose k at random and then communicate it to Bob over a secure channel.

But that presupposes the existence of a secure channel.

D-H key exchange overview

The Diffie-Hellman Key Exchange protocol allows Alice and Bob to agree on a secret k without having prior secret information and without giving an eavesdropper Eve any information about k. The protocol is given on the next slide.

We assume that p and g are publicly known, where p is a large prime and g a primitive root of p.

From the fact on slide 5, these assumptions imply the existence of $\log_g(y)$ for every $y \in \mathbf{Z}_p^*$.)

D-H key exchange protocol

Alice	**Bob**
Choose random $x \in \mathbf{Z}_{\phi(p)}$.	Choose random $y \in \mathbf{Z}_{\phi(p)}$.
$a = g^x \bmod p$.	$b = g^y \bmod p$.
Send a to Bob.	Send b to Alice.
$k_a = b^x \bmod p$.	$k_b = a^y \bmod p$.

Diffie-Hellman Key Exchange Protocol.

Clearly, $k_a = k_b$ since

$$k_a \equiv b^x \equiv g^{xy} \equiv a^y \equiv k_b \pmod{p}.$$

Hence, $k = k_a = k_b$ is a common key.

Why choose from $\mathbf{Z}_{\phi(p)}$?

One might ask why x and y should be chosen from $\mathbf{Z}_{\phi(p)}$ rather than from $\mathbf{Z}_p$?

The reason is because of another number-theoretic fact that we haven't talked about – *Euler's theorem* – which says

$$g^{\phi(p)} \equiv 1 \pmod{p}.$$

It follows that $x \equiv y \pmod{\phi(p)} \Leftrightarrow g^x \equiv g^y \pmod{p}$.

This means that the exponents can always be reduced mod $\phi(p)$ without changing the resulting k_a or k_b. Since smaller exponents are computationally more efficient, there's no reason to look outside of $\mathbf{Z}_{\phi(p)}$.

Security of DH key exchange

In practice, Alice and Bob may use this protocol to generate a session key for a symmetric cryptosystem, which they subsequently use to exchange private information.

The security of this protocol relies on Eve's presumed inability to compute k from a and b and the public information p and g. This is sometime called the *Diffie-Hellman problem* and, like discrete log, is believed to be intractable.

Diffie-Hellman problem and discrete log

Certainly the Diffie-Hellman problem is no harder that discrete log, for if Eve could find the discrete log of a, then she would know x and could compute k_a the same way that Alice does.

However, it is not known to be as hard as discrete log.

ElGamal Cryptosystem

A variant of DH key exchange

A variant protocol has Bob going first followed by Alice.

Alice	**Bob**
	Choose random $y \in \mathbf{Z}_{\phi(p)}$.
	$b = g^y \bmod p$.
	Send b to Alice.
Choose random $x \in \mathbf{Z}_{\phi(p)}$.	
$a = g^x \bmod p$.	
Send a to Bob.	
$k_a = b^x \bmod p$.	$k_b = a^y \bmod p$.

ElGamal Variant of Diffie-Hellman Key Exchange.

Comparison with first DH protocol

The difference here is that Bob completes his action at the beginning and no longer has to communicate with Alice.

Alice, at a later time, can complete her half of the protocol and send a to Bob, at which point Alice and Bob share a key.

Turning D-H into a public key cryptosystem

This is just the scenario we want for public key cryptography. Bob generates a public key (p, g, b) and a private key (p, g, y).

Alice (or anyone who obtains Bob's public key) can complete the protocol by sending a to Bob.

This is the idea behind the ElGamal public key cryptosystem.

ElGamal cryptosystem

Assume Alice knows Bob's public key (p, g, b). To encrypt a message m:

- She first completes her part of the key exchange protocol to obtain numbers a and k.
- She then computes $c = mk \bmod p$ and sends the pair (a, c) to Bob.
- When Bob gets this message, he first uses a to complete his part of the protocol and obtain k.
- He then computes $m = k^{-1}c \bmod p$.

Combining key exchange with underlying cryptosystem

The ElGamal cryptosystem uses the simple encryption function $E_k(m) = mk \bmod p$ to actually encode the message.

Any symmetric cryptosystem would work equally well.

An advantage of using a standard system such as AES is that long messages can be sent following only a single key exchange.

A hybrid ElGamal cryptosystem

A hybrid ElGamal public key cryptosystem.

- As before, Bob generates a public key (p, g, b) and a private key (p, g, y).
- To encrypt a message m to Bob, Alice first obtains Bob's public key and chooses a random $x \in \mathbf{Z}_{\phi(p)}$.
- She next computes $a = g^x \bmod p$ and $k = b^x \bmod p$.
- She then computes $E_{(p,g,b)}(m) = (a, \hat{E}_k(m))$ and sends it to Bob. Here, $\hat{E}$ is the encryption function of the underlying symmetric cryptosystem.
- Bob receives a pair (a, c).
- To decrypt, Bob computes $k = a^y \bmod p$ and then computes $m = \hat{D}_k(c)$.

Randomized encryption

We remark that a new element has been snuck in here. The ElGamal cryptosystem and its variants require Alice to generate a random number which is then used in the course of encryption.

Thus, the resulting encryption function is a *random function* rather than an ordinary function.

A random function is one that can return different values each time it is called, even for the same arguments.

Formally, we view a random function as returning a probability distribution on the output space.

Remarks about randomized encryption

With $E_{(p,g,b)}(m)$ each message m has many different possible encryptions. This has some consequences.

An advantage: Eve can no longer use the public encryption function to check a possible decryption.

Even if she knows m, she cannot verify m is the correct decryption of (a, c) simply by computing $E_{(p,g,b)}(m)$, which she could do for a deterministic cryptosystem such as RSA.

Two disadvantages:

- Alice must have a source of randomness.
- The ciphertext is longer than the corresponding plaintext.

Primitive Roots

Using the ElGamal cryptosystem

To use the ElGamal cryptosystem, we must be able to generate random pairs (p, g), where p is a large prime, and g is a primitive root of p.

We now look at primitive roots and how to find them.

Primitive root

We say g is a *primitive root* of n if g generates all of $\mathbf{Z}_n^*$, that is, $\mathbf{Z}_n^* = \{g, g^2, g^3, \ldots, g^{\phi(n)}\}$.

By definition, this holds if and only if $\text{ord}(g) = \phi(n)$.[2]

Not every integer n has primitive roots.

Theorem (Gauss)

The numbers having primitive roots are $1, 2, 4, p^k, 2p^k$*, where* p *is an odd prime and* $k \geq 1$*.*

In particular, every prime has primitive roots.

[2] $\text{ord}(g)$ is the smallest integer $r > 1$ such that $g^r \equiv 1 \pmod{n}$.

Number of primitive roots

Theorem

The number of primitive roots of a prime p is $\phi(\phi(p))$.

Gauss's theorem shows that p has at least one primitive root. The following lemma shows that there are at least $\phi(\phi(p))$ primitive roots. We omit the proof that there are no more than that number.

Lemma (powers of primitive roots)

If g is a primitive root of p and $x \in \mathbf{Z}^*_{\phi(p)}$, then g^x is also a primitive root of p.

Proof of lemma

We argue that every element h in $\mathbf{Z}_p^*$ can be expressed as $h = (g^x)^y$ for some y.

- Since g is a primitive root, we know that $h \equiv g^\ell \pmod{p}$ for some ℓ.
- We wish to find y such that $g^{xy} \equiv g^\ell \pmod{p}$.
- By Euler's theorem, this is possible if the congruence equation $xy \equiv \ell \pmod{\phi(p)}$ has a solution y.
- It is known that a solution exists iff $\gcd(x, \phi(p)) \mid \ell$.
- But this is the case since $x \in \mathbf{Z}_{\phi(p)}^*$, so $\gcd(x, \phi(p)) = 1$.

Primitive root example

Let $p = 19$, so $\phi(p) = 18$ and $\phi(\phi(p)) = \phi(2) \cdot \phi(9) = 6$.

Consider $g = 2$ and $g = 5$. The subgroups S_g of $\mathbf{Z}_p$ generated by each g is given by the table:

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
2^k	2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
5^k	5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1

We see that 2 is a primitive root since $S_2 = \mathbf{Z}_p^*$ but 5 is not.

Now let's look at $\mathbf{Z}_{\phi(p)}^* = \mathbf{Z}_{18}^* = \{1, 5, 7, 11, 13, 17\}$.

The complete set of primitive roots of p (in $\mathbf{Z}_p$) is then

$$\{2, 2^5, 2^7, 2^{11}, 2^{13}, 2^{17}\} = \{2, 13, 14, 15, 3, 10\}.$$

Lucas test

Theorem (Lucas test)

g is a primitive root of a prime p if and only if

$$g^{(p-1)/d} \not\equiv 1 \pmod{p}$$

for all $d > 1$ such that $d \mid (p-1)$.

Proof of correctness for Lucas test

Suppose the Lucas test fails for some $d > 1$, $d \,|\, (p-1)$. That means $g^{(p-1)/d} \equiv 1 \pmod{p}$. It follows that

$$\text{ord}(g) \leq \frac{p-1}{d} < p-1 = \phi(p),$$

so g is not a primitive root of p. Why?

Conversely, if g is not a primitive root of p, then $\text{ord}(g) < p-1$, or equivalently, $(p-1)/\text{ord}(g) > 1$. Hence, the test will fail for $d = (p-1)/\text{ord}(g)$ since then

$$g^{(p-1)/d} = g^{\text{ord}(g)} \equiv 1 \pmod{p}.$$

Problems with the Lucas test

A drawback to the Lucas test is that one must try all the divisors of $p - 1$, and there can be many.

Moreover, to find the divisors efficiently implies the ability to factor. Thus, it does not lead to an efficient algorithm for finding a primitive root of an arbitrary prime p.

However, there are some special cases which we can handle.

Special form primes

Let p and q be odd primes such that $p = 2q + 1$.

Then, $p - 1 = 2q$, so $p - 1$ is easily factored and the Lucas test easily employed.

There are lots of examples of such pairs, e.g., $q = 41$ and $p = 83$.

Number of primitive roots of special form primes

Recall $p = 2q + 1$. We just saw that the number of primitive roots of p is

$$\phi(\phi(p)) = \phi(p-1) = \phi(2q) = \phi(2)\phi(q) = q - 1.$$

Hence, the density of primitive roots in $\mathbf{Z}_p^*$ is

$$(q-1)/(p-1) = (q-1)/2q \approx 1/2.$$

This makes it easy to find primitive roots of p probabilistically — choose a random element $a \in \mathbf{Z}_p^*$ and apply the Lucas test to it.

Density of special form primes

How many special form primes are there?

We defer the question of the density of primes q such that $2q + 1$ is also prime but remark that we can relax the requirements a bit.

Relaxed requirements on special form primes

Here's another way of generating a prime pair (p, q).

Let q be a prime. Generate numbers $u = 2, 4, 6, \ldots$ until we find u for which $p = uq + 1$ is prime.

[Why do we skip odd u?]

Then $p - 1 = uq$ for small u.

u can be factored by exhaustive search. At that point, we can apply the Lucas test as before to find primitive roots.

How many u must be tried?

By the prime number theorem, approximately one out of every $\ln(q)$ numbers around the size of q will be prime.

While that applies to randomly chosen numbers, not to the numbers in this particular sequence, there is at least some hope that the density of primes will be similar.

If so, we can expect that $u/2$ will be about $\ln(q)$, so u is easily factored for cryptographic-sized primes q.