CPSC 467b: Cryptography and Computer Security

Lecture 12

Michael J. Fischer

Department of Computer Science
Yale University

February 17, 2010

More number theory with cryptographic applications

We turn next to other number-theoretic techniques with important cryptographic applications.

We begin by looking in greater detail at the structure of $\mathbf{Z}_n^*$, the set of integers in $\mathbf{Z}_n$ that are relatively prime to n.

Structure of $\mathbf{Z}_n^*$

Let $g \in \mathbf{Z}_n^*$ and consider the successive powers $g, g^2, g^3, \ldots$, all taken modulo n. This sequence must eventually repeat. Why?

Structure of $\mathbf{Z}_n^*$

Let $g \in \mathbf{Z}_n^*$ and consider the successive powers $g, g^2, g^3, \ldots$, all taken modulo n. This sequence must eventually repeat. Why?

Because $\mathbf{Z}_n^*$ is finite.

Structure of $\mathbf{Z}_n^*$

Let $g \in \mathbf{Z}_n^*$ and consider the successive powers $g, g^2, g^3, \ldots$, all taken modulo n. This sequence must eventually repeat. Why?

Because $\mathbf{Z}_n^*$ is finite.

This sequence contains 1. Why?

Structure of $\mathbf{Z}_n^*$

Let $g \in \mathbf{Z}_n^*$ and consider the successive powers $g, g^2, g^3, \ldots$, all taken modulo n. This sequence must eventually repeat. Why?

Because $\mathbf{Z}_n^*$ is finite.

This sequence contains 1. Why?

By Euler's theorem, since $g^k = 1$ (in $\mathbf{Z}_n$) for $k = \phi(n)$.

Cyclic groups

Let k be the smallest positive integer such that $g^k = 1$. We call k the *order of g* and write $\mathrm{ord}(g) = k$.

[1] A subgroup is a subset of the elements that is closed under the group operations.

Cyclic groups

Let k be the smallest positive integer such that $g^k = 1$. We call k the *order of g* and write $\mathrm{ord}(g) = k$.

The elements $\{g, g^2, \ldots, g^k = 1\}$ form a subgroup S of $\mathbf{Z}_n^*$.[1]

[1] A subgroup is a subset of the elements that is closed under the group operations.

Cyclic groups

Let k be the smallest positive integer such that $g^k = 1$. We call k the *order of g* and write $\mathrm{ord}(g) = k$.

The elements $\{g, g^2, \ldots, g^k = 1\}$ form a subgroup S of $\mathbf{Z}_n^*$.[1]

The order of S (number of elements in S) is $\mathrm{ord}(g)$; hence $\mathrm{ord}(S) \mid \phi(n)$. Why?

[1] A subgroup is a subset of the elements that is closed under the group operations.

Cyclic groups

Let k be the smallest positive integer such that $g^k = 1$. We call k the *order of g* and write $\mathrm{ord}(g) = k$.

The elements $\{g, g^2, \ldots, g^k = 1\}$ form a subgroup S of $\mathbf{Z}_n^*$.[1]

The order of S (number of elements in S) is $\mathrm{ord}(g)$; hence $\mathrm{ord}(S) \mid \phi(n)$. Why?
Again because of Euler's theorem.

[1] A subgroup is a subset of the elements that is closed under the group operations.

Cyclic groups

Let k be the smallest positive integer such that $g^k = 1$. We call k the *order of g* and write $\mathrm{ord}(g) = k$.

The elements $\{g, g^2, \ldots, g^k = 1\}$ form a subgroup S of $\mathbf{Z}_n^*$.[1]

The order of S (number of elements in S) is $\mathrm{ord}(g)$; hence $\mathrm{ord}(S) \mid \phi(n)$. Why?
Again because of Euler's theorem.

We say that g *generates* S and that S is *cyclic*.

[1] A subgroup is a subset of the elements that is closed under the group operations.

Primitive root

We say g is a *primitive root* of n if g generates all of $\mathbf{Z}_n^*$, that is, every element of $\mathbf{Z}_n^*$ can be written as g raised to some power modulo n.

Primitive root

We say g is a *primitive root* of n if g generates all of $\mathbf{Z}_n^*$, that is, every element of $\mathbf{Z}_n^*$ can be written as g raised to some power modulo n.

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

Primitive root

We say g is a *primitive root* of n if g generates all of $\mathbf{Z}_n^*$, that is, every element of $\mathbf{Z}_n^*$ can be written as g raised to some power modulo n.

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

Not every integer n has primitive roots.

Primitive root

We say g is a *primitive root* of n if g generates all of $\mathbf{Z}_n^*$, that is, every element of $\mathbf{Z}_n^*$ can be written as g raised to some power modulo n.

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

Not every integer n has primitive roots.

By Gauss's theorem, the numbers having primitive roots are $1, 2, 4, p^k, 2p^k$, where p is an odd prime and $k \geq 1$.

Primitive root

We say g is a *primitive root* of n if g generates all of $\mathbf{Z}_n^*$, that is, every element of $\mathbf{Z}_n^*$ can be written as g raised to some power modulo n.

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

Not every integer n has primitive roots.

By Gauss's theorem, 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.

Number of primitive roots

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

Number of primitive roots

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

This is because 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. Why?

Number of primitive roots

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

This is because 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. Why?

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

Number of primitive roots

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

This is because 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. Why?

We need to 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.

Number of primitive roots

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

This is because 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. Why?

We need to 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 $g^{xy} \equiv g^\ell \pmod{p}$.

Number of primitive roots

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

This is because 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. Why?

We need to 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.

Number of primitive roots

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

This is because 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. Why?

We need to 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.
- We know that a solution exists iff $\gcd(x, \phi(p)) \mid \ell$.

Number of primitive roots

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

This is because 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. Why?

We need to 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.
- We know 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 $\phi(\phi(p)) = \phi(2) \cdot \phi(9) = 6$.

Let $g = 2$. The subgroup S of $\mathbf{Z}_p$ generated by 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
g^k	2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1

Since $S = \mathbf{Z}_p^*$, we know that g is a primitive root.

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 p if and only if

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

for all $1 < q < p-1$ such that $q \mid (p-1)$.

Lucas test

Theorem (Lucas test)

g is a primitive root of p if and only if

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

for all $1 < q < p-1$ such that $q \mid (p-1)$.

Clearly, if the test fails for some q, then

$$\mathrm{ord}(g) \leq (p-1)/q < p-1 = \phi(p), \qquad \text{Why?}$$

so g is not a primitive root of p.

Lucas test

Theorem (Lucas test)

g is a primitive root of p if and only if

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

for all $1 < q < p-1$ *such that* $q \mid (p-1)$.

Clearly, if the test fails for some q, then

$$\mathrm{ord}(g) \le (p-1)/q \; < \; p-1 \; = \; \phi(p), \qquad \text{Why?}$$

so g is not a primitive root of p.

Conversely, if $\mathrm{ord}(g) < \phi(p)$, then the test will fail for $q = (p-1)/\mathrm{ord}(g)$.
This is because q is included in the test and $\mathrm{ord}(g) \mid \phi(p)$.

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.

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 $p = 83$.

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 $p = 83$.

How many primitive roots does p have?

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 $p = 83$.

How many primitive roots does p have?
We just saw the number is

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

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 $p = 83$.

How many primitive roots does p have?
We just saw the number is

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

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

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

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 $p = 83$.

How many primitive roots does p have?
We just saw the number is

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

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

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.

Density of special form primes

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.

Let q be a prime. Generate a sequence of numbers $2q + 1$, $3q + 1$, $4q + 1$, $\ldots$ until we find a prime $p = uq + 1$.

Density of special form primes

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.

Let q be a prime. Generate a sequence of numbers $2q+1$, $3q+1$, $4q+1$, $\ldots$ until we find a prime $p = uq+1$.

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

Density of special form primes

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.

Let q be a prime. Generate a sequence of numbers $2q+1, 3q+1, 4q+1, \ldots$ until we find a prime $p = uq+1$.

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 the numbers in this particular sequence, there is at least some hope that the density of primes will be similar.

Density of special form primes

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.

Let q be a prime. Generate a sequence of numbers $2q+1, 3q+1, 4q+1, \ldots$ until we find a prime $p = uq+1$.

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 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 will be about $\ln(q)$, in which case it can easily be factored using exhaustive search. At that point, we can apply the Lucas test as before to find primitive roots.

Logarithms

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

Logarithms

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

The discrete logarithm is defined similarly, but now arithmetic is performed in $\mathbf{Z}_p^*$ for a prime p.

In particular, the *discrete log* to the base b of y modulo p is the least non-negative integer x such that $y \equiv b^x \pmod{p}$ (if it exists). We write $x = \log_b(y) \bmod p$.

Logarithms

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

The discrete logarithm is defined similarly, but now arithmetic is performed in $\mathbf{Z}_p^*$ for a prime p.

In particular, the *discrete log* to the base b of y modulo p is the least non-negative integer x such that $y \equiv b^x \pmod{p}$ (if it exists). We write $x = \log_b(y) \bmod p$.

If b is a primitive root of p, then $\log_b(y)$ is defined for every $y \in \mathbf{Z}_p^*$. Why?

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.

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.

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 $\mathrm{power}_b(x) = b^x \bmod p$, which is easily computable.

power_b is an example of a so-called *one-way function*, that is a function that is easy to compute but hard to invert.

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.

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.

Security of DH key exchange

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

Security of DH key exchange

In practice, Alice and Bob can use this protocol to generate a session key for a symmetric cryptosystem, which they can 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.

Security of DH key exchange

In practice, Alice and Bob can use this protocol to generate a session key for a symmetric cryptosystem, which they can 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.

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.

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.

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.

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.

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.

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.