CPSC 467b: Cryptography and Computer Security

Michael J. Fischer

Lecture 10
February 8, 2012

One-way and Trapdoor Permutations

Discrete Logarithm

Diffie-Hellman Key Exchange

ElGamal Key Agreement

Primitive Roots

One-way and Trapdoor Permutations

One-way permutations

A *one-way permutation* is a permutation f that is easy to compute but hard to invert.

To *invert* f means to find, for each element y, the unique x such that $f(x) = y$.

Here, "easy" means computable in probabilistic polynomial time.

"Hard" to invert means that no probabilistic polynomial time algorithm can find x given y with non-negligible success probability.

Negligible functions

A function is said to be *negligible* if it goes to zero faster than any inverse polynomial.

Definition (2.1 in Goldwasser-Bellare)

ν is negligible if for every constant $c \geq 0$ there exists an integer k_c such that $\nu(k) < k^{-c}$ for all $k \geq k_c$.

In other words, no matter what c you choose, $\nu(k) < k^{-c}$ for all sufficiently large k.

One-way functions

More generally, any function f is *one-way* if it is easy to compute but hard to invert, where now "invert" means to find, given y in the range of f, any element x such that $f(x) = y$.

Definition (2.2 in Goldwasser-Bellare)

A function $f : \{0,1\}^* \to \{0,1\}^*$ is *one-way* if:

1. There exists a PPT[1] that on input x outputs $f(x)$;
2. For every PPT algorithm A there is a negligible function ν_A such that for all sufficiently large k,

$$\Pr[\, \mathrm{f(z) = y} : \mathrm{x} \xleftarrow{\$} \{0,1\}^{\mathrm{k}};\ \mathrm{y} \leftarrow \mathrm{f(x)};\ \mathrm{z} \leftarrow \mathrm{A}(1^{\mathrm{k}}, \mathrm{y})\,] \leq \nu_{\mathrm{A}}(\mathrm{k})$$

[1] Probabilistic polynomial time Turing machine.

Fine points

Here, k is a *security parameter* that measures the length of the argument x to f. It is also provided to A as input, in unary, so that A's running time is measured correctly as a function of k.

Again we have several sources of randomness which jointly determine the probability that A successfully inverts f.

In particular, the string y on which we wish to invert f is not chosen uniformly from strings of a given length but instead is chosen according the probability of y being the image under f of a uniformly chosen x from $\{0,1\}^k$.

Trapdoor functions

A *trapdoor function* f is a one-way function that becomes easy to invert on a subset of its domain given a secret *trapdoor* string.

Definition (2.15 of Goldwasser-Bellare, corrected)

A *trapdoor* function is a one-way function $f : \{0,1\}^* \to \{0,1\}^*$ such that there exists a polynomial p and a probabilistic polynomial time algorithm I such that for every k there exists a $t_k \in \{0,1\}^*$ such that $|t_k| \leq p(k)$ and for all $x \in \{0,1\}^k$, then $I(f(x), t_k) = z$, where $f(z) = f(x)$.

Taking this apart, it says that f, when restricted to strings of length k, can be inverted by the algorithm I when I is given a suitable "trapdoor" string t_k.

A more intuitive formulation of trapdoor function

In the usual formulation of trapdoor function, we imagine f being chosen at random from a family of one-way functions. The trapdoor string is the secret that allows f to be inverted.

The idea is that f is regarded as a function of two arguments, k and x, where $|k|$ and $|x|$ are both bounded by a polynomial in the security parameter s. $f(k, x)$ should be a one-way function but with the additional *trapdoor property*:

> *There is a PPT algorithm I and a polynomial p such that for every s and every $k \in \{0,1\}^*$ with $|k| \leq p(s)$, there exists a $t_k \in \{0,1\}^*$ with $|t_k| \leq p(s)$ such that for all $x \in \{0,1\}^s$, then $I(f(k,x), t_k) = z$, where $f(k,z) = f(k,x)$.*

The RSA function

The RSA function is the map $f((n, e), x) = x^e \pmod{n}$. It is believed to be a trapdoor function with security parameter s when n is restricted to length s strings, n is is the product of two distinct primes of approximately the same length, $e \in \mathbf{Z}^*_{\phi(n)}$, and $x \in \mathbf{Z}_n$.

Here, the index k is the pair (n, e), and the trapdoor $t_k = (n, d)$, where d is the RSA decryption exponent.

The mathematically inclined will notice that this function does not quite fit the earlier definitions because of the restrictions on the domans of n, e, and x. We leave it to a more advanced course to properly sort out such details.

Discrete Logarithm

Logarithms mod p

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

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

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

Fact (not needed yet): If b is a *primitive root*[2] of p, then $\log_b(y)$ is defined for every $y \in \mathbf{Z}_p^*$.

[2] 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.

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.

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.

ElGamal Key Agreement

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.

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 $\text{ord}(g) = \phi(n)$.

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 $\phi(\phi(p))$.

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 $q \mid (p-1)$.

Clearly, if the test fails for some q, then

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

so g is not a primitive root of p.

Conversely, if $\text{ord}(g) < \phi(p)$, then the test will fail for $q = (p-1)/\text{ord}(g)$.

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 $p = 83$.

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.

Let q be a prime. Generate a sequence of numbers $2q + 1$, $3q + 1$, $4q + 1$, ... 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.