YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 467a: Cryptography and Computer Security *Notes 11 (rev. 1)*
Professor M. J. Fischer *October 12, 2006*

Lecture Notes 11

56 Primitive Roots

Let $g \in \mathbf{Z}_n^*$ and consider the successive powers $g, g^2, g^3, \ldots$, all taken modulo n. Because $\mathbf{Z}_n^*$ is finite, this sequence must eventually repeat. By Euler's theorem, this sequence contains 1, namely, $g^{\phi(n)}$. Let k be the smallest positive number such that $g^k = 1$ (in $\mathbf{Z}_n^*$). 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 $\mathbf{Z}_n^*$. The order of S (number of elements in S) is $\mathrm{ord}(g)$; hence $\mathrm{ord}(g) \mid \phi(n)$. We say that g *generates* S and that S is *cyclic*.

We say g is a *primitive root* of n if g generates $\mathbf{Z}_n^*$, 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.

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.

Lucas test: g is a primitive root of p if and only if

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

for all $q > 1$ such that $q \mid (p-1)$. Clearly if the test fails for some q, then $\mathrm{ord}(g) \leq (p-1)/q < p - 1 = \phi(p)$, so g is not a primitive root of p. Conversely, if $\mathrm{ord}(g) < \phi(p)$, then the test will fail for $q = (p-1)/\mathrm{ord}(g)$, which is one of the q's included in the test since $\mathrm{ord}(g) \mid \phi(p)$.

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.

Let p and q be odd primes such that $p = 2q + 1$. There are lots of examples of such pairs, e.g., $q = 41$ and $p = 83$. In this case, $p - 1 = 2q$, so $p - 1$ is easily factored and the Lucas test easily employed. How many primitive roots of p are there? From the above, 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.

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. Suppose we start with a prime q and then consider the numbers $2q + 1, 3q + 1, 4q + 1, \ldots$ until we find a prime $p = uq + 1$ in this sequence. 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 be easily factored using exhaustive search. At that point, we can apply the Lucas test as before to find primitive roots.

57 Discrete Logarithm

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 defined to be the least non-negative integer x such that $y \equiv b^x \pmod{p}$ (if it exists), and we write $x = \log_b(y) \bmod n$. If b is a primitive root of p, then $\log_b(y)$ is defined for every $y \in \mathbf{Z}_p^*$.

The *discrete log problem* is the problem of computing $\log_b(y) \bmod p$ given a prime p and primitive root b of p. No known efficient algorithm is known for this problem and it is believed to be intractable. However, it's inverse, the function $\mathrm{power}_b(x) = b^x \bmod p$, is easily computable, so 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.

58 Diffie-Hellman Key Exchange

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.
$k_a = b^x \bmod p$.	$k_b = a^y \bmod p$.

Figure 1: Diffie-Hellman Key Exchange Protocol.

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. 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 in Figure 1. We assume that p and g are publicly known, where p is a large prime and g a primitive root of p. 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. In practice, Alice and Bob can use this protocol to generate a session key for a symmetric cryptosystem, which then 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.

59 ElGamal Key Agreement

A variant of the above algorithm has Bob going first followed by Alice, as shown in Figure 2. The difference here is that Bob completes his action at the beginning and no longer has to communicate

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.	
$k_a = b^x \bmod p$.	$k_b = a^y \bmod p$.

Figure 2: ElGamal Variant of Diffie-Hellman Key Exchange.

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.

Assume Alice knows Bob's public key (p, g, b). To encrypt a message m, she first completes her part of the protocol of Figure 2 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 $m = k^{-1}c \bmod p$.

While the ElGamal cryptosystem uses the simple encryption function $E_k(m) = mk \bmod p$ to actually encode the message, it should be clear that any symmetric cryptosystem could be used at that stage. An advantage of using a standard system such as AES is that long messages can be sent following only a single key exchange.

Putting this all together gives us the following variant of the 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 a specified symmetric cryptosystem. Bob receives a pair (a, c). To decrypt, Bob computes $k = a^y \bmod p$ and the computes $m = \hat{D}_k(c)$.

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. The way to view a random function is that it specifies a probability distribution on the output space that depends on its arguments.

In the case of $E_{(p,g,b)}(m)$ each message m has many different possible encryptions. An advantage of such a probabilistic encryption system is that Eve can no longer use the public encryption function to check a possible decryption, for even if she knows m, she cannot verify m is the correct decryption of (a, c) simply by computing $E_{(p,g,b)}(m)$ as she could do for a deterministic cryptosystem such as RSA. Two disadvantages of course are that Alice must have a source of randomness in order to use the system, and the ciphertext is longer than the corresponding plaintext.

60 Quadratic Residues, Squares, and Square Roots

An integer a is called a *quadratic residue (or perfect square) modulo n* if $a \equiv b^2 \pmod{n}$ for some integer b. Such a b is said to be a *square root* of a modulo n. We let

$$\mathrm{QR}_n = \{a \in \mathbf{Z}_n^* \mid a \text{ is a quadratic residue modulo } n\}.$$

be the set of quadratic residues in $\mathbf{Z}_n^*$, and we denote the set of non-quadratic residues in $\mathbf{Z}_n^*$ by $\mathrm{QNR}_n = \mathbf{Z}_n^* - \mathrm{QR}_n$.

61 Square Roots Modulo a Prime

Claim 1 *For an odd prime p, every $a \in QR_p$ has exactly two square roots in $\mathbf{Z}_p^*$, and exactly 1/2 of the elements of $\mathbf{Z}_p^*$ are quadratic residues.*

For example, take $p = 11$. The following table shows all of the elements of $\mathbf{Z}_{11}^*$ and their squares.

a		$a^2 \bmod 11$
1		1
2		4
3		9
4		5
5		3
6	$= -5$	3
7	$= -4$	5
8	$= -3$	9
9	$= -2$	4
10	$= -1$	1

Thus, we see that $\mathrm{QR}_{11} = \{1, 3, 4, 5, 9\}$ and $\mathrm{QNR}_{11} = \{2, 6, 7, 8, 10\}$.

Proof: We now prove Claim 1. Consider the mapping $\mathrm{sq} : \mathbf{Z}_p^* \to \mathrm{QR}_p$ defined by $b \mapsto b^2 \bmod p$. We show that this is a 2-to-1 mapping from $\mathbf{Z}_p^*$ onto QR_p.

Let $a \in \mathrm{QR}_p$, and let $b^2 \equiv a \pmod{p}$ be a square root of a. Then $-b$ is also a square root of a, and $b \not\equiv -b \pmod{p}$ since $p \nmid 2b$. Hence, a has at least two distinct square roots $\pmod{n}$. Now let c be any square root of a.

$$c^2 \equiv a \equiv b^2 \pmod{p}.$$

Then $p \mid c^2 - b^2$, so $p \mid (c - b)(c + b)$. Since p is prime, then either $p \mid (c - b)$, in which case $c \equiv b \pmod{p}$, or $p \mid (c + b)$, in which case $c \equiv -b \pmod{p}$. Hence $c \equiv \pm b \pmod{p}$. Since c was an arbitrary square root of a, it follows that $\pm b$ are the only two square roots of a. Hence, $\mathrm{sq}()$ is a 2-to-1 function, and $|\mathrm{QR}_p| = \frac{1}{2}|\mathbf{Z}_p^*|$ as desired. ■