CPSC 367: Cryptography and Security

Michael J. Fischer

Lecture 14
March 5, 2019

Primitive Roots (continued)

- Lucas test
- Special form primes

Cryptographic Hash Functions

- Properties of random functions
- Message digest functions

Primitive Roots (continued)

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 $d > 1$, $d \mid (p-1)$. That means $g^{(p-1)/d} \equiv 1 \pmod{p}$. It follows that

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

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

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

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

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

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

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.

Cryptographic Hash Functions

Cryptographic use of random functions

Let $\mathcal{M}$ be a message space and $\mathcal{H}$ a hash value space, and assume $|\mathcal{M}| \gg |\mathcal{H}|$. A random function h chosen uniformly from $\mathcal{M} \to \mathcal{H}$ gives a way to protect the integrity of messages.

Suppose Bob knows $h(m)$ for Alice's message m, and Bob receives m' from Alice. If $h(m') = h(m)$, then with very high probability, $m' = m$, and Bob can be assured of the integrity of m'.

One problem with this approach is that we have no succinct way of describing random functions, so there is no way for Bob to compute $h(m')$. The other problem is that h should be chosen anew for each message. Otherwise, there is a small chance being stuck with a bad h (for example a constant function) forever and ever.

Message digest functions

A *message digest* (also called a *cryptographic hash* or *fingerprint*) function is a fixed (non-random) function that is designed to "look like" a random function.

The goal is to preserve the integrity-checking property of random functions: If Bob knows $h(m)$ and he receives m', then if $h(m') = h(m)$, he can reasonably assume that $m' = m$.

We now try to formalize what we require of a message digest function in order to have this property.

We also show that message digest functions do not necessarily "look random", so one should not assume such functions share other properties with random functions.

Formal definition of message digest functions

Let $\mathcal{M}$ be a message space and $\mathcal{H}$ a hash value space, and assume $|\mathcal{M}| \gg |\mathcal{H}|$.

A *message digest* (or *cryptographic one-way hash* or *fingerprint*) function h maps $\mathcal{M} \to \mathcal{H}$.

A *collision* is a pair of messages m_1, m_2 such that $h(m_1) = h(m_2)$, and we say that m_1 and m_2 *collide*.

Because $|\mathcal{M}| \gg |\mathcal{H}|$, h is very far from being one-to-one, and there are many colliding pairs. Nevertheless, it should be hard for an adversary to find collisions.

Collision-avoidance properties

We consider three increasingly strong versions of what it means to be hard to find collisions:

- One-way: Given $y \in \mathcal{H}$, it is hard to find $m \in \mathcal{M}$ such that $h(m) = y$.
- Weakly collision-free: Given $m \in \mathcal{M}$, it is hard to find $m' \in \mathcal{M}$ such that $m' \neq m$ and $h(m') = h(m)$.
- Strongly collision-free: It is hard to find colliding pairs (m, m').

These definitions are rather vague, for they ignore issues of what we mean by “hard” and “find”.

What does “hard” mean?

Intuitively, “hard” means that Mallory cannot carry out the computation in a feasible amount of time on a realistic computer.

What does “find” mean?

The term “find” may mean

- “always produces a correct answer”, or
- “produces a correct answer with high probability”, or
- “produces a correct answer on a significant number of possible inputs with non-negligible probability”.

The latter notion of “find” says that Mallory every now and then can break the system. For any given application, there is a maximum acceptable rate of error, and we must be sure that our cryptographic system meets that requirement.

One-way function

What does it mean for h to be one-way?

It means that no probabilistic polynomial time algorithm $A_h(y)$ produces a message m such that $h(m) = y$ with non-negligible success probability.

(Such an m is called a *pre-image* of y under h.)

This is only required for random y chosen according to a particular hash value distribution. There might be particular values of y on which A_h does succeed with high probability.

Hash value distribution

The hash value distribution we have in mind is the one induced by h applied to the assumed distribution on the message space$\mathcal{M}$.

Thus, the probability of y is the probability that a message m chosen according to the assumed message distribution satisfies $h(m) = y$.

This means that h can be considered one-way even though algorithms might exist that succeed on low-probability subsets of $\mathcal{H}$.