Theoretical Cryptography, Lecture 14

Instructor: Manuel Blum
Scribe: Ryan Williams

March 6, 2006

1 Today and Next Lecture

We will cover the following topics:

- Application of Generators and Quadratic Residues: Coin Tossing Over the Telephone
- "Generators" for Z_n^* (n is composite), and Collision Resistant Hash Functions

2 Coin Tossing Over the Telephone

We introduce the problem with a story. Alice and Bob have divorced. One is in L.A. and the other is in N.Y. They want to decide who gets the car. All they have to communicate is a telephone. Bob calls Alice, and Alice flips a coin and tells Bob to call it. Bob says "Heads." Alice says. "Oops, you lose."

This coin-flipping protocol will not do! Bob does not trust Alice; who knows if the coin was actually heads or not. How might we solve this problem? One can imagine using a stock market index to try to pull a random number, and use the parity of it to decide. But even with such a method, it is still possible in principle for Alice or Bob to have some kind of advantage.

What we really want is to make Alice and Bob *commit* to bits, in such a way that they cannot change their answer later. We will accomplish this, using some of the number theoretic concepts we have developed.

2.1 Relevant Number Theory

Recall for p prime and g a generator of $\mathbb{Z}_p^*$, $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\} = \{g, g^2, \ldots, g^{p-1} = 1\}$.

Our results will rely on the belief that a random discrete logarithm is hard to compute.

Discrete Log Assumption: *Let p be a good prime. Let g be a generator of $\mathbb{Z}_p^*$. Then for random $a \in \mathbb{Z}_p^*$, it is difficult to find x such that $g^x \equiv a \bmod p$. More formally, there is no randomized*

polynomial time algorithm that can correctly solve discrete logarithm instances on n-bit primes p and generators g, for more than a $1/poly(n)$-fraction of the $a \in \mathbb{Z}_p^*$.

Last lecture, we showed the following fact (in a proof of another fact).

Fact 2.1 *$\mathbb{Z}_p^*$ has $\phi(p-1)$ generators, where $\phi(n) := |\mathbb{Z}_n^*|$.*

Moreover, given a generator g of $\mathbb{Z}_p^*$, any element g^u such that $gcd(u, p-1) = 1$ is a generator as well.

Example. Let $p = 11$. Then there are $\phi(10) = (5-1)(2-1) = 4$ generators. Let $g = 2$. Then

$$g^2 = 4, g^3 = 8, g^4 = 5, g^5 = 10, g^6 = 9, g^7 = 7, g^8 = 3, g^9 = 6, g^10 = 1.$$

The three elements $g^3 = 8$, $g^7 = 7$, $g^9 = 6$ are also generators, since they are g^u such that $gcd(u, 10) = 1$.

Algorithm to determine if a is a generator of $\mathbb{Z}_p^*$, given prime p and prime factorization of $p-1$. The algorithm is quite easy: For each prime q in the factorization of $p-1$, check if $a^{(p-1)/q} \equiv 1 \bmod p$. If $a^{(p-1)/q} \neq 1 \bmod p$ for all q, then return YES, else return NO.

As mentioned earlier, the total number of generators is $\phi(p-1)$. Note that $\phi(n) \geq \frac{n}{\ln \ln n}$ for $n \geq 10$. Thus there are more generators of $\mathbb{Z}_n^*$ for prime n than there are primes in $[1, n]$, asymptotically.

Thus a simple randomized algorithm to find a generator of $\mathbb{Z}_n^*$ is: *Pick an element a at random. Check it using the above algorithm.*

`Steven:` If you can find a generator without randomness, it would a good senior project. I think it should be possible using techniques from "PRIMES in P".

`Manuel:` I'd give a PhD for it.

3 Principal Square Roots and Discrete Logarithms

Last lecture, we showed how to find a square root of a quadratic residue, and presented an efficient but incorrect algorithm for computing discrete logarithms. The algorithm would work, if we knew which of the two roots is the *principal* square root, which we now define. First we note a fact about the two square roots of a quadratic residue: knowing one gives you the other.

Fact 3.1 *Let p be a prime, $a \in \mathbb{Z}_p^*$, and g be a generator. $g^k \equiv \sqrt{a} \bmod p$ if and only if $g^{k+(p-1)/2} \equiv \sqrt{a} \bmod p$.*

Proof. By Fermat's little theorem, $g^{2k+(p-1)} \equiv g^{2k} \equiv a \bmod p$. □

Therefore, the two square roots of the quadratic residue g^{2k} are g^k and $g^{k+(p-1)/2}$. In order for the algorithm of the previous lecture to work, we would need to determine g^k from g^{2k}.

Definition 3.1 *The* principal square root *of $a = g^{2k}$ is g^k.*

Beware: It is possible (and common) for $g^k \bmod p > g^{k+(p-1)/2} \bmod p$, when the two are construed as integers in $\mathbb{Z}_p^*$. That is, the principal square root can be larger than the non-principal one, so the relative order between the two elements tells you little about which might be the principal one.

Define the **Principal Square Root** (**PSR**) problem as follows:

Input: Prime p such that $p \equiv 3 \bmod 4$ (*i.e.* a good prime), Generator g of $\mathbb{Z}_p^*$, $a, b, c \in \mathbb{Z}_p^*$ such that $a \neq b$, $a^2 \equiv b^2 \equiv c \equiv g^{2k} \bmod p$, for some integer $k > 0$.

Output: 'a' if a is the principal square root of c, otherwise 'b'.

Since our discrete logarithm algorithm from the previous lecture would work if we could only solve the Principal Square Root problem, we have the following theorem.

Theorem 3.1 *Suppose we allow access to a black box that, when given an input instance of* PSR*, outputs the solution in one step. Then there is a polynomial time algorithm for* Discrete Logarithm *that makes a polynomial number of calls to the black box.*

Corollary 3.1 *The* Discrete Logarithm Assumption *implies that, for all polynomials $p(n)$, the* Principal Square Root *problem cannot be solved in polynomial time on random instances with more than $1/2 + 1/p(n)$ probability.*

4 Coin Tossing Over the Telephone

Now, we show how the above Corollary can be applied to solve Alice and Bob's coin-tossing problem, described at the start of the lecture. We will give an explicit protocol for Alice and Bob to follow, such that the outcome is a "random" coin toss that both Alice and Bob can agree is random. Moreover, Alice and Bob can verify to each other that they did not cheat: *i.e.* they followed the protocol exactly. Here's the protocol:

(Ground Rule: *All rounds of interaction must be executed within polynomial time.*)

`Alice:` Randomly choose a sufficiently long prime p, and generator g. Send p, factorization of $p-1$, and g to Bob.
Pick random $r \in \{1, \ldots, p-1\}$.
Let $a = g^r \bmod p$, and $b = g^{r+(p-1)/2} \bmod p$.
Randomly send either $\langle a, b \rangle$, or $\langle b, a \rangle$.

`Bob:` Verify that g is a generator (using the factorization of $p-1$).
Verify that $a^2 \equiv b^2 \bmod p$.
(*Crucial Step:*) Send either a or b to Alice.

`Alice:` Send r to Bob.
Send *Win* if Bob sent a, and *Lose* if Bob sent b.

`Bob:` Verify that $g^{2r} \equiv a^2 \bmod p$ and $g^r \equiv a \bmod p$.

We end by briefly outlining why the above protocol results in a practically fair coin toss, under the Discrete Logarithm Assumption. At the start of the protocol, Alice constructs a random instance of the Principal Square Root problem, and randomly shuffles the order of a and b. The main observation is that if it is truly difficult to compute the principal square root, then the best choice that Bob can make in the "crucial step" is to pick a or b at random. Bob's verifications ensure that Alice follows the rules of the protocol: a and b are checked to ensure they are indeed square roots of a number, then g^r is computed to determine that a is principal.

Therefore, under the Discrete Logarithm Assumption, Bob cannot "win" the coin toss with probability greater than $1/2 + 1/p(n)$, for all polynomials $p(n)$.