Notes for Lecture 8

1 Chinese Remainder Theorem

Let $p \neq q$ be two primes. The Chinese Remainder Theorem (CRT) says that working modulo $n = pq$ is essentially the same as working modulo p and modulo q at the same time. (Actually, this is the "light" version of CRT. The full-fledged version says that working modulo $a_1a_2\ldots a_k$, where a_i are pairwise relatively prime, is the same as working simultaneously modulo $a_1, a_2, \ldots, a_k$.)

Theorem 1. *Let $p \neq q$ be primes, $n = pq$. For each $a \in Z_p$, $b \in Z_q$, there is unique c, $0 \le c < n$ such that $c \equiv a \pmod p$ and $c \equiv b \pmod q$.*

Proof. Let $r = p^{-1} \bmod q$ and $s = q^{-1} \bmod p$. Let $c' = rpb + sqa$. Then $c' \equiv rpb + sqa \equiv r \cdot 0 \cdot b + 1 \cdot a \equiv a \pmod p$, and $c' \equiv rpb + sqa \equiv 1 \cdot b + s \cdot 0 \cdot a \equiv b \pmod q$. Let $c = c' \bmod pq$. Then $pq|(c - c')$, so $p|(c - c')$, so $c \equiv c' \pmod p$. Similarly, $c \equiv c' \pmod q$. Hence, c satisfies all the conditions: $0 \le c < n$, and $c \equiv a \pmod p$ (because $c \equiv c' \equiv a \pmod p$), and $c \equiv b \pmod q$ (because $c \equiv c' \equiv b \pmod q$). Thus, for every pair (a, b) there is a c. There are $pq = n$ possible pairs, and n possible values of c, so for each pair there must be exactly one value of c, so it's unique for each (a, b). □

Denote by $\text{crt}(a, b)$ the unique value of c given by the above theorem. Then $\text{crt}(a, b) = c$ if an only if $(a, b) = (c \bmod p, c \bmod q)$. Let $c_1 = \text{crt}(a_1, b_1)$, $c_2 = \text{crt}(a_2, b_2)$, and $c_3 = c_1 + c_2 \bmod n$. Then $c_3 \bmod p = (c_1 + c_2) \bmod p = (a_1 + a_2) \bmod p$ (because n divides $c_3 - c_1 - c_2$, and therefore so does p) and similarly $c_3 \bmod q = (b_1 + b_2) \bmod q$. Hence $c_3 = \text{crt}(a_1 + a_2, b_1 + b_2)$. Same for multiplication. Thus, we can look at addition and multiplication modulo n "coordinate-wise": modulo p and modulo q.

We will denote by Z_n^* the set of values in Z_n that are relatively prime to n. Note that the "coordinates" of Z_n^* are in Z_p^* and Z_q^*, and that Z_n^* has $(p-1)(q-1)$ elements.

Note that the above proof is constructive: that is, c is efficiently (and, in fact, quite easily) computable given a and b. Thus, it is often more efficient to work modulo p and q separately and the reconstruct the value modulo n when it is needed.

2 Squares and Square Roots

Let $p > 2$ be a prime. Let QR_p denote the set of squares in Z_p^*. Recall from HW2 that for $a \in Z_p^*$, if $a \in QR_p$, then $a^{(p-1)/2} \equiv 1$, and if $a \notin QR_p$, then $a^{(p-1)/2} \equiv -1$.

Suppose $p \equiv 3 \pmod 4$. Take $s \in Z_p^*$. It has two roots: r and $-r$. Exactly one of these two roots is itself in QR_p. Indeed, consider $r^{(p-1)/2}$ and $(-r)^{(p-1)/2}$. Since $(p-1)/2$ is odd (because $p = 4k + 3$ for some k), $(-r)^{(p-1)/2} = -\left(r^{(p-1)/2)}\right)$, so one is 1 and the other is -1.

Hence, if we let $f_p(x) : QR_p \to QR_p$ be the map $x \mapsto x^2 \bmod p$, we see that for each $s \in QR_p$, there exists a unique inverse $r \in QR_p$ such that $f(r) = s$ (namely, r is the square root of s that is itself a square). So f_p of x is a permutation of QR_p. Note that f_p is easy to compute (just squaring) and easy to invert (as shown on HW2, it's easy to compute square roots modulo p).

Now let $p \neq q$ be two distinct odd primes, and let $n = pq$. Let QR_n denote the set of squares in Z_n^*. Then if s is a square modulo n, it is also a square modulo p and q. Since it has two roots $\pm r_1$ modulo p and two roots $\pm r_2$ modulo q, it has four roots modulo n: $\text{crt}(\pm r_1, \pm r_2)$.

Suppose both p and q are congruent to 3 modulo 4. Then exactly one of $\pm r_1$ is a square modulo p, and exactly one of $\pm r_2$ is a square modulo q, so exactly one of $\mathrm{crt}(\pm r_1, \pm r_2)$ is a square modulo n. Hence, if we let $f_n(x) : QR_n \to QR_n$ be the map $x \mapsto x^2 \bmod n$, we see that $f_n(x)$ is a permutation over QR_n. Note that $f_n(x)$ is easy to compute. We will argue below that it is hard to invert—as hard as it is to factor n.

3 Blum-Blum-Shub Generator

The following construction is due to [BBS86][1]. Starting with a sufficiently long random seed, select two k-bit random primes p, q that are 3 modulo 4, let $n = pq$, and let x be random element of QR_n (just select a random element of Z_n, check if it's relatively prime with n, and square it). Let $x_1 = x, x_2 = f_n(x), x_3 = f_n(x_2), \ldots, x_l = f_n(x_{l-1})$. Output the least significant bit for each x_i.

Note that this looks very much like the Blum-Micali generator, with exponentation mod p replaced with squaring mod n, and B replaced with least significant bit. The proof is very similar, too. We simply need two facts: that computing x from $x^2 \bmod n$ is hard (discussed in the next section), and that computing the least significant bit of x from $x^2 \bmod n$ is as hard as computing all of x (shown in [ACGS88]; an alternative proof is given is in [AGS03]; we will not discuss either here). These two facts correspond, in the Blum-Micali case, to the assumption that discrete logarithm is hard and that $B(x)$ is as hard as to compute from $g^x \bmod p$ as x itself.

This generator is more efficient than Blum-Micali: requires only one modular squaring per bit, instead of one one modular exponentiation. It is also based on a different (depending on whom you ask, more or less plausible) assumption: that factoring n is hard. We will show this in the next section.

4 Square Roots Modulo a Composite are as Hard as Factoring

We want to justify why we believe it's hard to compute x from x^2 modulo n. Indeed, let $s = r^2 \bmod n$. Then s has four square roots, as discussed above $\mathrm{crt}(r_1, r_2), \mathrm{crt}(-r_1, -r_2), \mathrm{crt}(r_1, -r_2), \mathrm{crt}(-r_1, r_2)$. Take two of these that are not negatives of each other, e.g., $r = \mathrm{crt}(r_1, r_2)$ and $r' = \mathrm{crt}(r_1, -r_2)$. Add them to get $r + r' = \mathrm{crt}(2r_1, 0)$. Thus, $r + r' \equiv 0 \pmod q$, so $q | (r + r')$. Note also that $r + r' \not\equiv 0 \pmod p$, so $p \not| (r + r')$. Hence, $\gcd(r + r', n) = q$. Thus, if you know two such roots, you can factor n, by simply computing the gcd (this can be done quickly with Euclid's algorithm).

Now suppose we have an algorithm A that computes square roots modulo n. We will use it to factor n as follows: take a random $r \in Z_n^*$, compute $s = r^2 \bmod n$, and give s to A. A will return some root r' of s. Because s has four roots and r was chosen at random (and not given to A), no matter how A works, $\Pr[r = \pm r'] = 1/2$. Hence, in half the cases, $\gcd(r + r', n)$ will give you a factor p or q of n.

Thus, we just proved (by contradiction and reduction, as usual) that if factoring n is hard, so is computing square roots modulo n. Hence, the Blum-Blum-Shub generator is secure based on the following assumption:

Assumption 1. For any poly-time algorithm F, there exists a negligible function negl such that, if you generate random k-bit primes p and q that are both 3 modulo 4, and let $n = pq$, $\Pr[F(n) = p] \le \mathrm{negl}(k)$.

[1]Conference version published in Crypto in 1982.

References

[ACGS88] W. Alexi, B. Chor, O. Goldreich, and C. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194–209, April 1988.

[AGS03] Adi Akavia, Shafi Goldwasser, and Muli Safra. Proving hardcore predicates using list decoding. In *44th Annual Symposium on Foundations of Computer Science*, Cambridge, Massachusetts, October 2003. IEEE.

[BBS86] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, May 1986.