CIS 511: Theory of Computation **Apr 11, 2017**

Lecture 22: Cryptography

Professor Sampath Kannan *Zach Schutzman*

NB: *These notes are from CIS511 at Penn. The course followed Michael Sipser's Introduction to the Theory of Computation (3ed) text.*

Cryptography Basics

Historically, cryptography has been about sending encrypted messages, such that only the intended recipient can decipher the contents.

In general, we have some plaintext m and ciphertext c and a function $f(m) = c$. The message cannot be sent as m, so instead the first party, Alice, computes $f(m)$ and sends the second party, Bob, c. Bob should be the only one who can compute $f^{-1}(c) = m$.

Easy examples include rotation ciphers (shift letters forward/backwards some number of places) and total replacement. These are both really easy to decode, so in modern settings, they are not practical. We also don't think that you can get better security by making the function more complex (i.e. composing a bunch of bad methods in a convoluted way). We want methods that are provably difficult for an eavesdropper, Eve, to invert.

The **one-time pad (private-key encrpytion)** is a secure scheme. Suppose we have a key k with $|k| = |m|$. Alice takes m and k and send $c = m \oplus k$, where $\oplus$ is the bitwise exclusive-or operation. Bob knows k and computes $m = c \oplus k$ (exclusive-or is self-inverse). If k is drawn uniformly from all strings of that length, the cyphertext is random with respect to the plaintext. This is a **symmetric key encoding scheme**, because Alice and Bob both have the same information, and we use a key to perform the encryption/decryption. This whole scheme relies on keeping k secret.

One drawback is that each pair of parties needs a separate key, agreed upon beforehand. If there are n users, we need $O(n^2)$ keys of length $O(2^m)$. This is a lot. Can we do better?

Public-key cryptography fixes some of these drawbacks.

Cryptography and Complexity

For cryptography to work, we need functions that are easy to compute but hard to compute.

Definition 22.1 *A function $f : \{0,1\}^* \to \{0,1\}^*$ is **length-preserving** if $|f(w) = |w|$ for all $w \in \{0,1\}^*$.*

Definition 22.2 *A function f is a **permutation** if it is bijective on $\{0,1\}^n$ for all n (bijective and length-preserving).*

A probabilistic Turing machine M on input x, we can talk about the probability that M outputs y. By definition, $\sum_y (M \text{ outputs } y) \leq 1$.

Definition 22.3 *A length-preserving function f is a* ***one-way function*** *if there is a poly-time deterministic Turing machine M that on input w outputs $f(w)$ and for n large enough, if w is chosen uniformly at random from $\{0,1\}^n$, and k is any constant, and D is any probabilistic poly-time Turing machine, the probability that D on input $f(w)$ computes y, such that $f(y) = f(w)$ is less than or equal to $\frac{1}{n^k}$.*

Basically, the function is easily computed but a decoder can find an inverse with extremely low probability.

One-way functions are good for passwords. The user computes the encryption of the password and sends that ciphertext to the system. At login, the system only needs to check that the ciphertexts match, and you never reveal the plaintext of the password.

Definition 22.4 *A* ***trapdoor function*** *is a one-way function which is easy to invert given that you know a certain 'secret'.*

Example: *let $N = pq$, where p, q are n-bit primes. Then, the set Z_N^* is the set of all numbers less that N and relatively prime to N. (E.g. $Z_{10}^* = \{1,3,7,9\}$, $Z_7^* = \{1,2,3,4,5,6\}$). We have a function f which maps $x \in Z_N^*$ to $x^2 \in Z_N^*$. f is a trapdoor function. This is believed to be one-way because if we could do it quickly, we could factor integers.*

Proof: *We want to show that being able to compute the discrete square root in Z_n^* efficiently gives us a fast algorithm for factoring. Suppose we have a black box that computes discrete square roots. If we take a random r and compute $a = r^2$ and feed a to the black box to get a result b. Then, check if $r + b$ or $r - b$ share a factor with N (via the Euclidean algorithm). If so, we have found one of the factors of N and therefore can get the other.*

We claim this succeeds with probability $\frac{1}{2}$ and is therefore a probabilistic polynomial algorithm. When we pick r, it has some residue mod p and mod q. Since $r^2 = a$, we have the squares of the same residues. The black box returns a square root b. Half of the time, we get back the same residues (plus or minus), and half the time we do not. If we got back the same ones, we failed as we gained no new information. The other half of the time, we get back some multiple of either p or q, and then can compute the greatest common divisor with N. ■

We can do encryption with this by doing the following: Bob picks primes p, q and publishes $N = pq$, but not p or q. When Alice wants to send a message m, she computes $c = m^2 \mod N$ and sends c to Bob. Since Bob can factor N, he can easily invert the computation and recover one of the four square roots of m^2. If Alice makes sure her message is shorter than $\frac{N}{2}$, then we reduce to only two possible roots, and the Jacobi symbol can further reduce it to one. Bob can then read m.

This is a form of public-key cryptography called the ***Rabin cryptosystem****, and is algorithmically equivalent to factoring. The drawback is the four-to-one mapping, and we don't like this potential for collision.*