COM S 6830 – Cryptography September 1, 2011

Lecture 3: One-Way Functions

Instructor: Rafael Pass *Scribe: Costandino Dufort Moriates (cdm82)*

Recal a PPT (Probalistic Polynomial Time) algorithm is a randomized algorithm which for all inputs x, $M(x)$ takes at most $p(|x|)$ steps no matter what random coins are tossed, where p is a polynomial.

Our honest algorithm will always be required to be PPT. We would like for these algorithms to be practically efficient as well, but that won't be the focus of this class.

We will also discuss what it means for an attacker to be efficient. In general, we'll say that attackers are restricted to PPT algorithms 'plus a little bit more', the details of which we'll get to later.

Further, we require for both the attacker and the defeneder that the code is in polynomial length to the input to be protected.

Definition. A *non-uniform* PPT A is a sequence of probalistic Turing Machines $A = \{A_i\}$ for which there exists a polynomial d such that $|A_i| \leq |d(i)|$ and running-time of $A_i \leq d(i)$. Then $A(x) = A_{|x|}(x)$.

We will sometimes model attackers by being limited to this type of PPT. This has the downside that our proofs will only show that our algorithms will be secure against this type of attack, but on the upside, just showing that an algorithm is insecure against this type of attacker doesn't imply a constructive means of attack.

Fun fact: non-uniform PPT's can solve some variants of the halting problem, but as far as we know we don't think it can factor large numbers.

What is the right model of attacker?

Is it the non-uniform PPT or a PPT? For our purposes a non-uniform PPT will be more convenient. There are reasons for and against. One note is that even though an attacker can write his code after seeing a key, but his brain is still a constant size Turing machine so the non-uniform PPT he could try to construct would have been generated by a uniform PPT (thinking of his brain so such) and shouldn't be able to do anything a PPT couldn't do.

One-way functions

One-way functions are functions which are easy to compute, but hard to decode. These functions are at the center of Cryptography and allow much of the theory we will study. For example, assuming one-way functions, we can get zero-knowledge proofs. We'll spend most of today precisely defining these functions.

First attempt:

Definition. A function f is *o*ne-way if f can be computed in PPT and there exists no non-uniform PPT A such that

$$\forall x \in \{0,1\}^{\times} \ \Pr[A(f(x)) \in f^{-1}(f(x))] = 1$$

.

Exampe: Is $f(x) = |x|$ hard? Well according to this definition it is because it will take 2^c time to write a valid inverse to something such that $f(x) = c$.

So we ammend the hardness definition as follows:

$$\forall x \in \{0,1\}^{\times} \ \Pr[A(1^{|x|}, f(x)) \in f^{-1}(f(x))] = 1$$

.

Where $1^{|x|}$ is the string concatenation of 1 repeated $|x|$ times. This is just ensure that the output of A doesn't shrink the size of its output too much as in the $f(x) = |x|$ example.

Another problem with this defintion is the quantification of inputes x to f. We don't exactly need that the hardness equation holds for all x, we just want it to hold for ALMOST all. This motivates the following definition:

Definition. A function $\Sigma : N \to R$ is *negligible* if $\forall c, \exists n_0$ such that $\Sigma(n) < \frac{1}{n^c}$ for all $n \geq n_0$.

That is Σ is asymptotically smaller than the inverse of every polynomial. An example of a negligible funcion is 2^{-n}. Another is $\frac{1}{n^{\log n}}$.

Now we define a strong one-way function.

Definition. A *strong one-way function* has the same definition for easiness as a one-way function and the defintion for hardness is modified as follows: $\forall$ nuPPT A, $\exists \epsilon$ such that $\forall n \in N$

$$\Pr[x \leftarrow \{0,1\}^n, A(1^n, f(x) \in f^{-1}(f(x))] \leq \epsilon(n)$$

.

For ϵ a negligible funcion.

Now for a definition without the asymptotic bounds for comparison:

Definition. (t, s, ϵ)-*secure* one-way function is one such that $\forall A$ with runtime $< t$ and size $< s$ the probability of inverting f is $< \epsilon$. Where t, s, ϵ are all constants.

We can recooperate information about this concrete definition from the strong one-way function definition, but it can often be painful requiring annoying constant dragging.

Now back to our definition of multiplication, $f(x, y) = xy$ with $|x| = |y|$ allowing leading zeros in our binary representations of x, y. Is it hard?

Each term is even with probability $1/2$ so the result is odd with probability $1/4$. To get around this, we restrict the domain of our functions to always make the outcome even or odd. Multiplication is not a strong one-way function.

Definition. A function is a weak one-way function if it is easy to compute and $\exists q(x)$, $\forall nuPPT$ A, $\forall n \in N$ such that:

$$\Pr[x \leftarrow \{0,1\}^n, A(1^n, f(x)) \in f^{-1}(f(x))] \leq 1 - 1/q(n)$$

.

Depressingly enough, even after all this talk about one-way functions, we don't actually know if one-way functions exist. We assume multiplication is a weak one-way function. More questions: Can I turn any weak one-way function into a strong one-way function? For example by running f in parallel on many inputs and requiring the attacker to find the inverse of all the outputs. It turns out this approach works.

Theorem 1. *If there exists a weak one-way function, then there exists a strong one-way function.*

We approach this result informally today. By looking at a lemma:

Lemma 2. *Let $f : \{0,1\}^\times \to \{0,1\}^\times$ be a weak one-way function. Consider $f(x_1, x_2, \ldots, x_m) = y_1$, $y_i = f(x_i)$. There exists a polynomial $m(x)$ such that f is a strong one-way function.*

proof sketch. The probability of each input being inverted is $(1 - \frac{1}{q(n)})$ so the probability of our new function being inverted is $((1 - \frac{1}{q(n)})^{q(n)})^n \approx (\frac{1}{e})^n$, and we're *done* as this is certainly a negligible function.

There is a problem here. All of these events are not necessarily independent. You can't necessarily assume the attacker will assume that the results from each multiplication are independent. We can get around this assumption, but you have to work much harder. Next time! □