CS 600.442 – Modern Cryptography 09/12/2016

Lecture 1: One-way Functions

Instructor: Abhishek Jain *Scribe: Alex Badiceanu*

We started the lecture by defining various useful terms that will be used extensively throughout the class. We then proceded to give a formal definition of the weakest cryptographic primitive: the one-way function.

1 Useful concepts

In real life, everyone, including what we call an "adversary" is bounded by a computational resource limit. In order to understand what an adversary is, we first need to define some basic concepts:

Definition 1 (Algorithm) *An algorithm is a deterministic Turing Machine whose input and output are strings over the binary alphabet* $\sum = \{0, 1\}$.

*Note, that the two terms "Turing machine" and "algorithm" will be used interchangeably from now on.

Definition 2 (Running Time) *An algorithm A is said to run in time T(n) if for all strings of length n over the input alphabet* $(x \in \{0,1\}^n), A(x)$ *halts within* $T(|x|)$ *steps.*

Definition 3 **(Polynomial Running Time)** *An algorithm A is said to run in polynomial time if there exists a constant c such that A runs in time* $T(n) = n^c$.

Remark 1 *We say an algorithm is efficient if it runs in polynomial time. If an algorithm runs in super-polynomial time* $T(n) = 2^n$ *or* $T(n) = n^{\log n}$, *then we will say it is inefficient*

Definition 4 **(Randomized Algorithm)** *A randomized algorithm, also called a probalistic polynomial time Turing machine (PPT) is a Turing machine that runs in polynomial time and is equipped with an extra randomness tape. Each bit of randomness tape is uniformly and independently chosen.The output of a randomized algorithm is a distribution.*

As mentioned earlier, in practice, everyone including the adversary has some bounded computational resources. These resources can be used in a variety of intelligent ways, but they are still limited. Turing machines are able to capture all the types of computations possible given these ressources. Therefore, a adversary will be a computer program or algorithm modeled as a Turing machine.

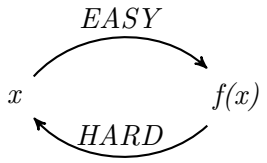
This captures what we can do efficiently ourselves and can be described as a uniform PPT Turing machine. When it comes to adversaries, we will allow them to have some extra power. Instead of having only one algorithm that works for different input lengths, it can write down potentially a different algorithm for every input size. Each of them individually could be efficient. If that is the case, overall the adversary still runs in polynomial timel.

Definition 5 (Non-uniform PPT) *A non-uniform probabilistic polynomial time Turing machine is a Turing machine A made up of a sequence of probabilistic machines* $A = \{A_1, A_2, ...\}$ *for which there exists a polinomial* $p(\cdot)$ *such that for every* $A_i \in A$, *the description size* $|A_i|$ *and the running time of* A_i *are at most* $p(i)$. *We write A(x) to denote the distribution obtained by running* $A_{|x|}(x)$.

***Definition 6* (Adversary)** *An Adversary ia a non-uniform probabilistic polynomial running time algorithm (PPT).*

2 One Way Functions

Intuitively, a given function f is "one-way" if it is very easy to compute $f(x)$ efficiently, but it is hard to recover x if given $f(x)$.



***Definition 7* (One-way Function (Informal))** *A function f is one-way if it satisfies the following two informal properties:*

1. **Functionality - Easy to compute:** *Given any input x from the domain, it should be easy to compute f(x). In other words it is possible to compute f(x) in polynomial time.*
2. **Security - Hard to invert:** *Any polynomial time algorithm should fail to recover x given f(x). This can also be expressed as: the probability of inverting f(x) is "small".*

What is this probability even over? Taking probability over adversarys random tape is not a good idea. What the adversary can do is find the best available random tape and just hardwire that inside the description. Instead, we want to take the probability over the choice of x. Take any non-uniform PPT adversary, with the best strategy, and then for any input length for x, the probability that A inverts f(x) for a randomly chosen x from the input should be small. So we sample an x at random, and then we compute f(x) and give it to the adversary, and we observe the probability that it inverts that image. Using this informal definition, we will make an attempt to describe the 2 conditions in a more formal way.

Definition 8 (One-way Function (1^{st} Attempt)) *A function $f : \{0,1\}^* \to \{0,1\}^*$ is a one way function (OWF) if it satisfies the following two conditions:*

1. **Easy to compute:** *There is a PPT algorithm C s.t. $\forall x \in \{0,1\}^*$,*

$$Pr[C(x) = f(x)] = 1.$$

   What do we mean when we say the "probability is 1" - What set is this probability over? - It's the probability over the random tape input to the randomized algorithm C.

2. **Hard to invert:** *for every non-uniform PPT adversary A, for any input length $n \in \mathbb{N}$.*

$$Pr[x \leftarrow^{\$} \{0,1\}^n; A \text{ inverts } f(x)] \leq small$$

We need to specify what exactly "small" means. We will define a fast decaying function $\nu(\cdot)$ s.t for any imput length $n \in \mathbb{N}$. This function decays asymptotically faster than any inverse polynomial. We willl call this function *negligible* Then, our security definition becomes:

$$Pr[x \leftarrow^{\$} \{0,1\}^n; A \text{ inverts } f(x)] \leq \nu(n).$$

Definition 9 (Negligible function) *A function $\nu(n)$ is negligible if for every c, there exists some n_0 such that for all $n > n_0, \nu(n) \leq \frac{1}{n^c}$*

$$\text{i.e } \forall c \in \mathbb{N}, \exists n_0 \in \mathbb{N} \text{ such that } \forall n > n_0, \nu(n) \leq \frac{1}{n^c}$$

In other words, a negligible function decays faster than all "inverse-polynomial" functions ($n^{-\omega(1)}$). An example of an obviously negligible function is an exponentially decaying function 2^{-n} or $n^{-\log(n)}$

Updating our previous definition:

Definition 10 (One-way Function (2^{nd} Attempt)) *A function $f : \{0,1\}^* \to \{0,1\}^*$ is a one way function (OWF) if it satisfies the following two conditions:*

1. **Easy to compute:** *There is a PPT algorithm C s.t. $\forall x \in \{0,1\}^*$,*

$$Pr[C(x) = f(x)] = 1.$$

2. **Hard to invert:** *for every non-uniform PPT adversary A, for any input length $n \in \mathbb{N}$, there exists a negligible function $\nu(\cdot)$ s.t:*

$$Pr[x \leftarrow^{\$} \{0,1\}^n; A \text{ inverts } f(x)] \leq \nu(|x|).$$

Although this definition seems accurate, it has one small problem: What is A's input? Let's take a closer look at this: Let's write $y = f(x)$ If f is a one way function, the following two conditions have to be satisfied:

- **Condition 1:** A on input y must run in $poly(|y|)$.
- **Condition 2:** A cannot output x' s.t. $f(x') = y$.

However, if the size of y is much smaller than the size of the domain, A cannot write the inverse even if it can find it. For example, if we consider the function $f(x)$ =first $\log|x|$ Although it is trivial to invert this function ($f^{-1}(y) = y||\underbrace{0000...0}_{\text{n - lg n}}$ where$n = 2^{|y|}$), it still satisfies the definition we wrote above. Although f is easy to compute, and A cannot invert f in time $poly(|y|)$ as it needs $2^{|y|}$ to write the answer, it is not a one-way function. In order to fix this issue, we adopt the convention to always pad y and write $A(1^n, y)$, so that A has sufficient time to write the answer.

Definition 11 (Strong One-way Function) *A function $f : \{0,1\}^* \to \{0,1\}^*$ is a one way function (OWF) if it satisfies the following two conditions:*

1. **Easy to compute:** *There is a PPT algorithm C s.t. $\forall x \in \{0,1\}^*$,*

$$Pr[C(x) = f(x)] = 1.$$

2. **Hard to invert:** *There exists a negligible function $\mu : \mathbb{N} \to \mathbb{R}$ s.t. for every non-uniform PPT adversary A and $\forall n$ in$\mathbb{N}$:*

$$Pr[x \leftarrow \{0,1\}^n, x' \leftarrow A(1^n, f(x)) : f(x') = f(x)] \leq \mu(n)$$

Definition 12 **(Injective or 1-1 OWFs)** *A function f is injective if each image has a unique pre-image:*

$$f(x_1) = f(x_2) \Rightarrow x_1 = x_2$$

Definition 13 **(One Way Permutation)** *An injective one way function with the additional condition that "each image has a pre-image" or in other words t hat the domain and range of the function have the same size.*

$$f(x_1) = f(x_2) \Rightarrow x_1 = x_2.$$

3 The Factoring Problem

Since proving that f is a one way function requires proving (at least) $P \neq NP$, we cannot tell for sure if OWFs exist unconditionally. However, by making certain assumptions about the hardness of some problems, we can construct conditional one-way functions also called "candidates". One such problem is the Factoring Problem:

We will start with considering the **multiplication** function: $f_x : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$:

$$f_x(x,y) = \begin{cases} \perp & \text{if } x = 1 \text{ or } y = 1 \\ x \cdot y & \text{otherwise} \end{cases}$$

This function is clearly not one-way, as the probability of xy being even and thus obviously factored into (2, xy/2) is $\frac{3}{4}$ for random (x, y). In other words, the inversion succeeds 75% of time. We will then try to eliminate such trivial factors. We define $\prod_n$ be the set of all prime numbers $< 2^n$ and we randomly select two elements, p and q from this set and multiply them. Their product is thus unlikely to contain small trivial factors. Thus:

Assumption 1 (Factoring Assumption) *For every (non-uniform PPT) adversary A, there exists a negligible function ν such that:*

$$Pr[p \xleftarrow{\$} \prod_n; q \xleftarrow{\$} \prod_n; N = pq : A(N) \in \{p, q\}] \leq \nu(n)$$

Untill now, there have been no "good attacks" on this assumption. The best known algorithms for breaking the Factoring Assumption are:

$$2^{O(\sqrt{n \log n})} \qquad \text{(provable)}$$
$$2^{O(\sqrt[3]{n \log^2 n})} \qquad \text{(heuristic)}$$

Looking back at our multiplication function, it is clear that if a random x and y happen to be prime, no A could invert it, which is a GOOD case. If such a case happens with probability greater than ϵ, then every A must fail to invert the function with probability at least ϵ. If ϵ is a noticeable function, then A fails to invert the function with noticeable probability. Lastly, if ϵ is a noticeable function, then A fails with a noticeable probability. This is what we call a WEAK one-way function.

Definition 14 (Noticeable Function) *Function $\epsilon : \mathbb{N} \to \mathbb{R}$ is noticeable means that $\exists c$ and integer N_c such that $\forall n > N_c : \epsilon(n) \geq \frac{1}{n^c}$*

***Definition 15* (Weak One-way Function)** *A function $f : \{0,1\}^* \to \{0,1\}^*$ is a weak one way function (OWF) if it satisfies the following two conditions:*

1. **Easy to compute:** *There is a PPT algorithm C s.t. $\forall x \in \{0,1\}^*$,*

$$Pr[C(x) = f(x)] = 1.$$

2. **Somewhat hard to invert:** *There is a noticeable function $\epsilon : \mathbb{N} \to \mathbb{R}$ s.t. for every non-uniform PPT A and $\forall n \in \mathbb{N}$:*

$$Pr[x \leftarrow \{0,1\}^n, x' \leftarrow A(1^n, f(x)) : f(x') \neq f(x)] \geq \epsilon(n)..$$

Now we will try to show that $f_\times$ is a weak OWF.

Theorem 1 *Assuming the factoring assumption, function $f_\times$ is a weak OWF.* To prove this, we will show that the "good" case when x and y are prime occurs with noticeable probability and we will use Chebyshev's theorem to show that the fraction of prime numbers between 1 and 2^n is noticeable.

Theorem 2 (Chebyshev's theorem) *An n bit number is a prime with probability $\frac{1}{2n}$.*

We will prove Theorem 1 during our next lecture. Now, once we have a weak OWF, how can we get strong OWF? Can we transform the multiply function into a strong OWF? Can we do this generically. The answer is yes, proven by Yao.

Theorem 3 (Yao's theorem) *Strong OWFs exist if and only if weak OWFs exist.*

In other words, if you have a weak one-way function, you can generically convert it to a strong one-way function.This is an example of a general phenomenon which is very well studied in complexity theory called hardness amplification. Next time we will look at a proof of this.