CS 7880 Graduate Cryptography September 29, 2015

Lecture 5: OWFs, PRGs

Lecturer: Daniel Wichs *Scribe: Mehraneh Liaee*

1 Topic Covered

- OWFs and complexity (Impagliazzo's worlds)
- Examples of OWFs
- Computational Indistinguishability
- PseudoRandom Generators (PRGs)

2 Recall

Last time, we talked about efficient algorithms, an efficient adversary and what we mean by negligible success probability. First, We recall some definitions like negligible functions, one way functions and one way NP puzzles. So we defined negligible function as follows:

DEFINITION 1 Function $\varepsilon(n)$ is negligible, denoted by $\varepsilon(n) = negl(n)$, if the following holds: $\forall$ constant c, $\exists n_0$ such that $\forall n \geq n_0$, $\varepsilon(n) \leq \frac{1}{n^c}$.

There are couple of equivalent definitions:

- $\forall$ constant c, $\varepsilon(n) = o(\frac{1}{n^c})$.
- $\forall$ constant c, $\varepsilon(n) = \frac{1}{\Omega(n^c)}$.
- $\varepsilon(n) = \frac{1}{n^{\omega(1)}}$.

The best way to keep it in mind is that if some function is not negligible, it is as big as some polynomial for infinitely many ns. ♢

DEFINITION 2 A one way function $f : \{0,1\}^* \to \{0,1\}^*$ has two properties:

- f can be computed in polynomial time.
- $\forall$ PPT adversary A, $\exists \varepsilon(n) = negl(n)$ such that

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

which means that f is easy to compute in forward direction, but it is hard to invert.

♢

DEFINITION 3 One way NP puzzle consists of two PPT algorithms, one is $PGen(1^n) \to (y, x)$, the other is $PVer(y, x) \to \{0,1\}$, which y is puzzle and x is the solution or witness for this puzzle, moreover it has the two following properties:

- **correctness**:
$$\Pr[PVer(y,x) = 1 : (y,x) \leftarrow PGen(1^n)] = 1$$
This says that if we generate a random puzzle y, it does have a solution.

- **security**: $\forall$ PPT $A, \exists$ $\varepsilon(n) = negl(n)$ such that:
$$\Pr[PVer(y,x') = 1 : (y,x) \leftarrow PGen(1^n), x' \leftarrow A(y)] \leq \varepsilon(n)$$
This says if puzzle y is given to the adversary, the probability that adversary comes up with a valid solution is small.

$\diamondsuit$

And last time, we showed that these two notions OWF and One-Way Puzzles are equivalent, if you have a one way NP puzzle, you can have a one way function and vice versa.

DEFINITION 4 An average hard NP problem is a weaker version from one way NP puzzle, which let us to generate a hard puzzle at random, but we might not have the solution ourselves.

$$y \leftarrow PGen(1^n)$$

- The correctness property says that if $PGen$ generates a puzzle y, there should exist a solution x for puzzle y, but we might not have it.
- The security property is the same as in one way puzzle.

$\diamondsuit$

What is the conceptual difference? We can generate hard puzzles with the solutions that are hard to solve in average vs we can generate hard puzzles without having the solution and they are hard to solve in average.

3 Impagliazzo's worlds

We saw that one way functions, one way puzzles are equivalent. And we want to see how these things fit in the complexity theory and how they relate to $''P = NP''$ and similar questions people study in complexity theory. A good way of understanding that is the survey paper by Russell Impagliazzo which he describes five possible worlds we could live in and their implications to computer science. Following are the five worlds he defines in his paper:

- **Algorithmica**. This is the world where "$P = NP$" or "$BPP = NP$", which means we could solve any NP problem in the worst case. (Possibly using a randomized algorithm that, for a worst-case instance, finds the solution with good probability).
- **Heuristica**. This is the world where "$BPP \neq NP$" but average-hard NP problems don't exist. This means for any efficient algorithm there are some puzzles that the algorithm cannot solve. In other words, every efficient algorithm is faulty on some

inputs (y). However, for efficiently computable distribution $PGen$ that generates NP problems, there is an efficient algorithm that solves random problems from the distribution with high probability on average. It is called Heuristica because there is not one good algorithm that always works, but there are heuristics – no matter what natural distribution out there in nature problems comes from there is an algorithm that can solve the problems from that distribution.

- **Pessiland**. Average hard NP problems do exist, but one way functions or one way NP puzzles do not exist. In this world, there are efficient distribution on problems that arise in nature that we do not know how to solve them in average. But we also can't efficiently sample hard problems with a solution or build cryptography.
- **Minicrypt**. In this world, one way functions exist. As we will see, this means we can do most of symmetric-key cryptography (but still not public-key cryptography).
- **Cryptomania**. In this world, public-key encryption exists. This appears to require more structure than just one-way functions.

4 Examples of OWF

Now an interesting question is that if we assume that OWFs exist, do we have any candidates for them? Here, we give two simple examples.

- Candidate 1: Factoring.
  - $PGen(1^n)$: samples two prime numbers p and q such that size of of p, q is $\Theta(2^n)$ and outputs (y, x) which $y = pq$ and $x = (p, q)$.
  - $PVer(y, x = (p, q))$: checks if $p, q \neq 1$ and $y = pq$.

  If y is given, the adversary should factor it to two prime number, this is something we believe is hard. There are some concerns about this like how we can efficiently sample these prime numbers or how we can efficiently check the two given numbers are prime. It turns out that we can do them efficiently.
- Candidate 2: Short-Integer-Solution (SIS). Let $m = \Theta(n \log n)$, $q = O(n)$
  - $PGen(1^n)$: For some parameters $q = \Theta(n)$, $m = \Theta(n \log q)$:
    samples $A \leftarrow \mathbb{Z}_q^{n \times m}$ and outputs $x \leftarrow \{0,1\}^m$ and $y = (A, Ax)$.
    You can think of A as a matrix of size $n \times m$, and x and Az as vectors of size m and n respectively.
  - $PVer(y = (A, z), x)$: checks if $x \in \{0,1\}^m$, $Ax = z$. .

  Note that x is a vector of only 0 and 1 and is not chosen uniformly at random from the field. Without this restriction, finding a solution is really easy and is only solving a system of m equations of n unknown variables, which has many solutions that can be found efficiently using Gaussian elimination.

  This is similar to subset sum problem. Think of rows of the matrix, there are some subset of these rows that add up to some value z. The goal is to find this subset (x).

5 Computational Indistinguishability

Recall that we defined the statistical distance between two distributions as:

$$SD(X,Y) = \max_D |\Pr[D(X)=1] - \Pr[D(Y)=1]|$$

This is the maximal probability that any "distinguisher" D has in telling apart a sample from X, Y. If $SD(X,Y)$ is small, we can think of X, Y as statistically indistinguishable distributions that cannot be told apart. We now want to adapt the definition to the computational setting. We do this by simply restricting the above definition to *efficient* distinguishers D. However, in the computational setting, we also need to talk about distinguishing distributions in the *asymptotic* sense. Instead of two variables X, Y, we have a series of variables that depend on security parameter n. So, in the following we define distribution ensemble.

DEFINITION 5 A distribution ensemble $X = \{X_n\}_{n\in N}$ consists of an infinite sequence of distributions, where X_n is distributed over $\{0,1\}^*$. Say length of X is $\ell(n)$ if X_n is over $\{0,1\}^{\ell(n)}$. ♢

No we can define an analog to statistical distance as follows:

DEFINITION 6 Let $X = \{X_n\}$ and $Y = \{Y_n\}$ be two distribution ensembles of length $\ell(n) = poly(n)$. We say X and Y are computationally indistinguishable, and we denote it by $X \overset{c}{\approx} Y$ or simply $X \approx Y$, if for all PPT distinguisher D there exists an $\varepsilon(n) = \text{negl}(n)$ such that

$$|\Pr[D(1^n, X_n) = 1] - \Pr[D(1^n, Y_n) = 1]| \leq \varepsilon(n)$$

♢

(We will often omit writing the input 1^n an assume that it is implicit)

There are two properties that can be implied from the definition.

Claim 1 ***Reduction.*** *If $X = \{X_n\}$ and $Y = \{Y_n\}$ such that $X \overset{c}{\approx} Y$, then $\forall$ PPT f: $f(X) \overset{c}{\approx} f(Y)$.*

Proof: Assume $D : |\Pr[D(f(X_n)) = 1] - \Pr[D(f(Y_n)) = 1]| \geq \varepsilon(n)$, for some $\varepsilon(n) \neq negl(n)$. Now, we define D'(n) = D(f(r)), and have the following:

$$|\Pr[D'(X_n) = 1] - \Pr[D'(Y_n) = 1]| \geq \varepsilon(n)$$

which contradicts that $X \overset{c}{\approx} Y$ □

Claim 2 ***Hybrid Argument.*** *If $X = \{X_n\}$, $Y = \{Y_n\}$ and $Z = \{Z_n\}$ are such that $X \approx Y$, $Y \approx Z$, then $X \approx Z$.*

Proof: $\forall$ PPT D, we have:

$$\begin{aligned}
&|\Pr[D(X_n) = 1] - \Pr[D(Z_n) = 1]| \\
&= |(\Pr[D(X_n) = 1] - \Pr[D(Y_n) = 1]) + (\Pr[D(Y_n) = 1] - \Pr[D(Z_n) = 1])| \\
&\leq |\Pr[D(X_n) = 1] - \Pr[D(Y_n) = 1]| + |\Pr[D(Y_n) = 1] - \Pr[D(Z_n) = 1]| \\
&\leq negl(n) + negl(n) = negl(n)
\end{aligned}$$

□

6 PseudoRandom Generators

We saw in Shanon's theorem, if we want to encrypt a big message, we need a big random key. What if we could take a small random key and apply some deterministic function that expands it into something that looks like a big random string. We define a primitive that does this, called a *pseudorandom generator*.

DEFINITION 7 A function $G : \{0,1\}^* \to \{0,1\}^*$ is a pseudorandom generator (PRG) with stretch $\ell(n)$ for some polynomial function ℓ, if has the following properties:

- G is polynomial time.
- $|G(x)| = n + \ell(n)$ for all $x \in \{0,1\}^n$.
- $G(U_n) \approx U_{n+\ell(n)}$ where U_m is uniform over $\{0,1\}^m$

$\diamondsuit$

In other words, a pseudorandom generator starts with n truly random bits and expands it into $n + \ell(n)$ "pseudorandom" bits. No polynomial time distinguisher can tell apart the output of the PRG from a truly random string of length $n+\ell(n)$. We call $\ell(n)$ the "stretch" since this is the amount of extra pseudorandomness that we get out.

One question is how much stretch can we hope to get. Can we get 1 bit, 100 bits, n bits, n^2 bits, etc.? The following theorem shows generating one extra bit of randomness is enough to get as many extra bits of randomness as you want.

Theorem 1 *If there exists a PRG with 1-bit stretch (i.e. $\ell(n) = 1$), then $\forall$ polynomial function $\ell(n)$, there exists a PRG with $\ell(n)$-bit stretch.*

We will see the complete proof in next session. But the idea is assuming that we have a PRG G with 1 bit stretch, taking it as a black box and using it as a building blocks of a PRG with $\ell(n)$ bits stretch.