CSE 5852, Modern Cryptography: Foundations Fall 2016

Lecture 11

Prof. Benjamin Fuller *Scribe: Lei Li*

1 Last Class

Last class we introduce the idea of pseudorandomness. Our goal is to produce a string using a limited amount of randomness that computationally limited adversaries cannot tell apart from a truly random string. We defined a *pseudorandom generator* as a deterministic function $G : \{0,1\}^n \to \{0,1\}^m$ where $m > n$. The security goal is when G is provided with a random value its output should also look random.

We then defined the notion of next-bit pseudorandomness.

Let $\mathcal{A}$ be a bit predictor. Define the following experiment $\texttt{prg} - \texttt{predict}$, parameterized by n:

1. Select random s of length $n(k)$.
2. Compute $y = G(s)$.
3. Run $\mathcal{A}(1^n)$, giving it bits of y in response to each *next* request.

If $\mathcal{A}$ stops after $i \leq m(n)$ stages and outputs $b = y_i$ we that that $\mathcal{A}$ wins $\texttt{prg} - \texttt{predict}$ and it outputs 1.

Bit predictor: A bit predictor is a machine working in two steps: 1) Get a bit y_i from a sequence and 2) output next bit or go back to step 1).

Definition 1 (Pseudo Random)**.** *A sequence $y_1, y_2, ..., y_m$ is pseudo random if for all PPT $\mathcal{A}$ there exists negligible function ϵ such that*

$$\Pr[\texttt{prg} - \texttt{predict}^{G,\mathcal{A}} = 1] \leq 1/2 + \epsilon(n)$$

Discrete logarithm problem: For any PPT $\mathcal{A}$ there exists a negligible function ϵ, *s.t.* for a random $n - bit$ prime p, a generator g and a random integer $x \in \mathbb{Z}_p*$, $\Pr[\mathcal{A}(1^n, p, g, g^x \mod p) = x] < \epsilon(n)$. The following contents of this lecture will be based on the assumption that Discrete log is hard to solve.

Construction of a pseudo random generator: First, let $G(p, g, x)$ be a function to generate a sequence of integers: $x_1 = g^x \mod p, x_2 = g^{x_1} \mod p, x_3 = g^{x_2} \mod p, ..., x_n = g^{x_{n-1}} \mod p$. Second, let $B(x)$ be a function outputs 1 if $x \geq p/2$ and outputs 0 if $\leq p/2$. Feed $B(x)$ the integers generated by $G(p, g, x)$ and reverse the order, we claim that the sequence $B(x_{m-1}) \dots B(x)$ is a PRG.

2 Proof for the PRG

The proof is completed by two parts:

Lemma 1. *If I can predict $B(x)$ from $g^x \mod p$ then I can solve discrete log problem efficiently.*

Theorem 2. *Using G as described is PRG with the assumption of part 1.*

These two parts can be proved independently and the order doesn't matter. In this lecture we will prove Lemma 1 and describe the strategy for part 1 completely and cover a little bit of part 2.

2.1 Proof of Lemma 1

Goal: Show if discrete log is hard to solve then $B(x)$ is hard to be predicted from $g^x \mod p$.

This will be proved by contrapositive, which being said, we first assume we can somehow predict $B(x)$, then we will use this fact to break down discrete log problem. On the other side it means if discrete log problem is hard to solve, then there is no algorithms can predict $B(x)$. We first give the assumption:

Assumption 3. *Assume there exists a predictor P who takes $p, g, g^x \mod p$ and predicts $B(x)$, $s,t.$ $Pr[P(p, g, g^x \mod p)] \geq 1/2 + \epsilon(n)$.*

Actually, to make this proof easier we will make a stronger assumption:

Assumption 4. *Assume there exists a predictor P who takes $p, g, g^x \mod p$ and predicts $B(x)$, $s,t.$ $Pr[P(p, g, g^x \mod p)] = 1$.*

Here we are using the second assumption to show how the proof works. The proof will also work with the first assumption but we are not including that in this lecture.[1] Now with the assumption we know if x is larger than $p/2$ or less than $p/2$. In another word, if we write x as a binary sequence then we should know the most significant bit. Next we will use this predictor to solve x completely, but before that we first introduce some useful facts in number theory:

Fact (1)**.** *Fermat's Little Theorem $(g^x)^{p-1} \equiv 1 \mod p$*

Fact (2)**.** *If $(g^x)^{(p-1)/2} \equiv 1 \mod p$, then x is even.*

Fact (3)**.** *There are two roots for g^x in the group defined on $\mathbb{Z}_p$ and the operation mod p, one is $g^{x/2}$ and the other is $g^{x/2+(p-1)/2}$, which can be verified easily using fact 1.*

Fact (4)**.** *There exists an efficient algorithm which gives us a root for g^x but it doesn't tell which one it is.*[2]

Now we will write the proof in steps:

[1] It is possible to improve or *amplify* the adversary's success at predicting $B(x)$. This allows the predictor to be correct with probability almost 1. The adversary always succeeding 1 is qualitatively different. However, this difference is not crucial for the proof to succeed. We note that the ability to amplify the adversary is dependent on the form of the adversary and is not always possible.

[2] One such algorithm is Cipolla's algorithm: `https://en.wikipedia.org/wiki/Cipolla\%27s_algorithm`

1. Check if $(g^x)^{(p-1)/2} \equiv 1 \mod p$, if x is odd, then set $y = g^x/g$. In another word, if x is odd, replace x by $x-1$ thus we can deliver it to our efficient algorithm to compute root.

2. Use our efficient algorithm to compute a root z for g^x. z is either $g^{x/2}$ or $g^{x/2+(p-1)/2}$. Then use our predictor on z. Recall that our predictor can predict if the exponent is larger than $p/2$, and clearly we have $x/2 < p/2$ and $x/2 + (p-1)/2 \geq p/2$. Thus we can distinguish z.

3. If $z = g^{x/2+(p-1)/2}$, then set $z' = z \cdot g^{(p-1)/2}$. Then $z' = g^{x/2+p-1} \equiv g^{x/2} \mod p$ because g^{p-1} has an inverse.

4. So far, we have successfully predicted the highest bit of x and shifted x by one bit (by computing $x/2$), then we go back to step 1 and predict the next bit, until we got the entire sequence of x.

Run time: Now we already showed we can figure out x if there exists a predictor which can predict $B(x)$, we also need to show this solution is efficient(polynomial). There are three types of major operations: 1) modular exponent calculation 2) our efficient algorithm to find root 3) recursive calls. The fact is operation 1) and 2) are both polynomial, and for recursive calls, we know the maximum number of calls is the length of sequence x, which is polynomial bounded, using the fact the multiplication of two polynomials is still a polynomial, we conclude that our approach runs in polynomial time.

2.2 Proof of part 2

Since we already proved part 1, we can modify our assumption a little bit. Now we assume $B(x)$ is hard to predict instead of assume discrete log problem is hard to solve.

Goal: Show that G is secure with the assumption $B(x)$ is hard to predict.

We will use contradiction to prove part 2. Recall the sequence we have is $B(x_m)B(x_{m-1})\dots B(x)$, we first assume there is a bit predictor that predicts the next bit of our sequence, then we use this predictor to predict $B(x)$.