CSA E0 235: Cryptography Due : 2-3-16, 8-3-16 (late submission)

Scribe for Lecture 11

Instructor: Arpita Patra *Submitted by: Sriram Chandramouli*

Introduction :

The following scribe is an attempt to revisit the topics of relevance and importance in Lecture 11. In doing so, its only natural to follow the order in the slides accompanied with occasional personal inferences wherever thought necessary.

For all the exact definitions mentioned in the scribe, one can look up

http://drona.csa.iisc.ernet.in/∼arpita/Cryptography16.html

to access the lecture slides.

Let us recall that *Authenticated Encryption* (henceforth abbreviated as AE) was constructed from a *cpa* (Chosen Plaintext Attack) - secure SKE (Symmetric Key Encryption) and a *scma* (Strong Chosen Message Attack) - MAC (Message Authentication Code). We also saw that AE implies the encryption scheme is also *cca* (Chosen Ciphertext Attack) - secure.

In this lecture we attempt to show that if PRG's (Pseudo-random generators) exist, then PRF's (Pseudo-random functions) exist. Rather than just showing existence mathematically, we construct a PRF explicitly, satisfying cryptographers. We then introduce a technique called Hybrid Proof technique to use hybrid arguments to show equivalence of two seemingly distant statements.We realize this technique by implementing it to prove that our construction is indeed a PRF.

Pseudo-random functions :

For the sake of completeness, we shall define pseudo-random generators and pseudo-random functions.

Definition 1.1 : A (probability) distribution G is said to be *pseudo-random* if no PPT distinguisher can non-negligibly (probability-wise) distinguish if a given string has been sampled from a uniform distribution or from G.

Definition 1.2 : A generator(black-box for all practical purposes) $\mathbb{G}$ is defined to be *pseudo-random generator* if the distribution on the output of the generator is a pseudo-random distribution G. $\mathbb{G}$ takes in a small random seed s as the input and generates a string usually longer.

Definition 1.3 : A function $\mathbb{F}$ is said to be a *pseudo-random function* if no PPT distinguisher can non-negligibly (probability-wise) distinguish if the function in question is

either a truly random function or $\mathbb{F}$ after querying polynomially many domain and range values. $\mathbb{F}$ is keyed with a totally random key and then becomes a deterministic function.

We immediately observe that given a pseudo-random function it is immediate to construct a pseudo-random generator. We can think of it as (example)

$$\mathbb{G}(x) = \mathbb{F}(x, \overline{0})$$

where $\overline{0}$ is the string with all zeroes. Hence we see that PRFs are more complex than PRGs.

Recalling that PRG can be written in a one-line notation (as a pseudo-random string) we can first attempt to construct a PRF by taking its lookup table (after ordering the domain values in ascending order) and then appending the function values row-wise. In this way, we will get a very long string of $n * 2^n$ bits and we can read off the function value after dividing this string into blocks of n bits each.

As usual, we see that whatever is easy may not be the correct way to approach and indeed this is the incorrect way to construct a PRF simply because the string is of exponential (in n) size. We require polynomial time constructions in computational crpytography. From a pedantic point of view, this construction is not only incorrect but even theoretically impossible as a PRG by definition cannot have an exponential expansion factor. (If it does it has far reaching consequences in theoretical cryptography)

We can show that given a PRG with expansion factor $n + 1$, we can construct another with polynomial expansion factor. So we can start with a length doubling PRG safely. We first mathematically frame what we require. Let $\mathbb{G}$ be a PRG with doubling expansion factor and $\mathbb{F}$ be a PRF

$$\mathbb{G} : \{0,1\}^n \to \{0,1\}^{2n} \longrightarrow \mathbb{F} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$$

Our requirements :

1.We need to define values/mappings at 2^n inputs

2.We need the mapping to be poly-computable

Construction of a PRF :

Since we have the length doubling PRG and we needed to compute the mapping in polynomial time, we can use the structure of a complete rooted binary tree with depth n. The root is said to be at depth 0. We now also observe that the number of leaf nodes is 2^n.

The non-trivial observation here is that for every leaf node, there is a unique path originating from the root node to that leaf node. Hence there are exactly 2^n distinct paths. By conventionally labelling the left child as a 0 and the right child as a 1 for every parent node, we now clearly see that every path corresponds to an n-bit string in binary. We can call this as the encoding of a path.

Our idea now is to look at this encoding as the input of a PRF and the value at the leafnode as the output of the PRF. Explicitly, if k is the n-bit seed for $\mathbb{G}$, $\mathbb{G}(k)$ will be $2n$

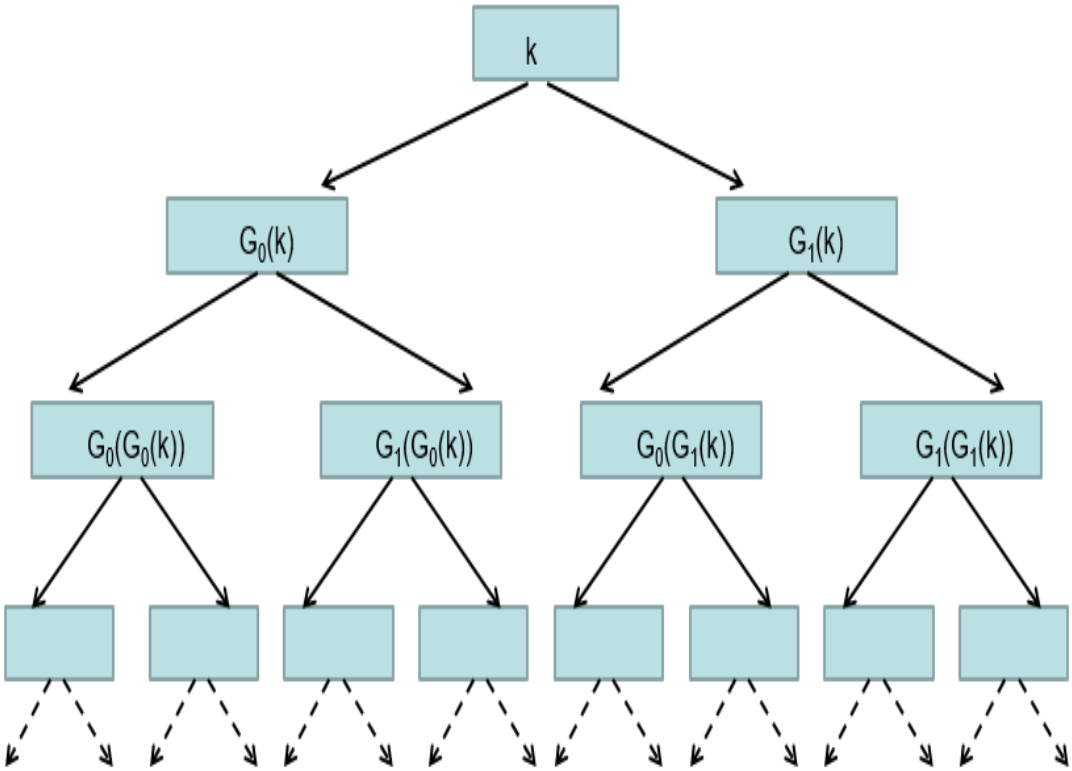

Figure 1: Populating a complete binary tree with the help of PRG

bits. Starting with the seed at the root, we can now parse $\mathbb{G}(k)$ as the left block and right block and populate the nodes at depth 1 of the tree correspondingly. Feeding back the values at depth 1 to $\mathbb{G}$ we can similarly populate the nodes at depth 2 and so on. A point to note is if a pseudo-random seed is used in a PRG, the output string is still pseudorandom (closeness property of pseudo-random generators) and hence our construction is consistent.

We now notate the the left child at depth 1 as $\mathbb{G}_0(k)$ and the right child as $\mathbb{G}_1(k)$ adhering to the convention of the encoding of a path. In this way we can similarly notate all the leaves. Refer to Figure 1. for a pictorial understanding.

For example, if one required to find $\mathbb{F}_k(011)$ keyed with a random key k, one would simply compute

$$\mathbb{F}_k(011) = \mathbb{G}_1(\mathbb{G}_1(\mathbb{G}_0(k)))$$

Note that we first used the key k as the seed for $\mathbb{G}$ to begin with and also the computation is poly-computable (in fact only n evaluations of $\mathbb{G}$ in general). We also note the nice property that given the value of some intermediate node, we can populate the partial complete binary tree below it with that as the root. We finally observe that if we know the values of all nodes at a given depth we can reconstruct the entire part of the tree below

these nodes. This is a corollary of the previous observation applied to each node at the same depth.

Proof that the above is indeed a PRF :

Theorem 1 *If $\mathbb{G} : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ is a PRG then the above construction is a PRF.*

We illustrate the proof in two steps. Each step in its own right is a Lemma.

Lemma 2 *If $\mathbb{G} : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ is a PRG, then*

$$|Pr[A(r_1, r_2, ..., r_t) = 1] - Pr[A(\mathbb{G}(s_1), \mathbb{G}(s_2), ..., \mathbb{G}(s_t)) = 1]| \leq negl(n)$$

where the first probability is taken over $r_1, r_2, ..., r_t \; \varepsilon_R \; \{0,1\}^{2n}$ and the second probability is taken over $s_1, s_2, ..., s_t \; \varepsilon_R \; \{0,1\}^n$

Proof

Because $\mathbb{G} : \{0,1\}^n \rightarrow \{0,1\}^{2n}$ is a PRG, we know that

$$|Pr[D(r) = 1] - Pr[D(\mathbb{G}(s) = 1]| \leq negl_1(n)$$

where r and s are chosen randomly appropriately.

Since these two statements are seemingly distant, we shall invoke the power of hybrid arguments. We denote the first hybrid as the adversary A distinguishing between $\{r_1, r_2, ..., r_t\}$, t random strings and $\{\mathbb{G}(s_1), r_2, ..., r_t\}$ which is one pseudo-random string followed by rest of random strings.

Proceeding in this fashion we now state and analyse the i^{th} hybrid where the two sets of strings are $\{\mathbb{G}(s_1), ..., \mathbb{G}(s_{i-1}), r_i, r_{i+1}, ..., r_t\}$ and $\{\mathbb{G}(s_1), ..., \mathbb{G}(s_{i-1}), \mathbb{G}(s_i), r_{i+1}, ..., r_t\}$

Supposing the adversary A is able to distinguish the above two sets of strings at the i^{th} step, we simulate a game to break the security of the PRG $\mathbb{G}$. We now act as the the distinguisher D for the PRG game, and the follwing is the strategy.

- Pick $\{s_1, s_2, ..., s_{i-1}\} \; \varepsilon_R \; \{0,1\}^n$ and $\{r_{i+1}, r_{i+2}, ..., r_t\} \; \varepsilon_R \; \{0,1\}^{2n}$.
- Let the PRG verifier give us a string y of unknown origin
- Pass $\{\mathbb{G}(s_1), \mathbb{G}(s_2), ..., \mathbb{G}(s_{i-1}), y, r_{i+1}, ..., r_t\}$ to the efficient distinguisher A.
- If y was a truly random string then we have sent across a set of the first kind and if y was a pseudorandom string we have sent across a set of the second kind. Since we have simulated the game for A correctly, he will tell if y belongs to the first set or the second set with non-negligible probability. If it belonged to first set, we reply to the PRG verifier that y was sampled from a uniform distribution and correspondingly a pseudo-random distribution otherwise. Since it is just passing over the answer, we will be able to distinguish the PRG $\mathbb{G}$ with non-negligible probability !!!

Since we know $\mathbb{G}$ is actually a secure PRG, by contradiction we infer that these two sets of strings are computationally indistinguishable. By applying this argument for all the t hybrids, we get

$$|Pr[A(r_1, r_2, ..., r_t) = 1] - Pr[A(\mathbb{G}(s_1), \mathbb{G}(s_2), ..., \mathbb{G}(s_t)) = 1]| \leq t * negl_1(n)$$

The above is true by simply writing down the probability bounds for each hybrid, bounded by the *negl* function, summing up the inequalities and by using triangle inequality in succession. ■

Lemma 3 *If Lemma 2 holds then the discussed construction is a PRF.*

Proof

We again invoke the power of hybrid arguments here. Let H_i denote the distibution on the leaf nodes when the nodes at depth i are populated with random strings.

In this scenario, H_n would be all random strings at leaf nodes, thereby simulating a truly random function. Similarly H_0 would be random string only at root and simulating our construction.

Proceeding in this fashion, we now state and analyse the i^{th} hybrid where the two distibutions in question are H_{i-1} and H_i. We shall assume that they are distinguishable by a PPT adversary R who gives t queries. More explicitly, let us call our 'function' as $\mathbb{F}_{i-1}$ when the nodes at depth $i-1$ are populated with truly random strings and the the entire part of the tree below these nodes are populated. Similarly let us define $\mathbb{F}_i$ but in either case we are interested only about the values of the leaf nodes. We shall then show that if such a distinguisher R exists, we will be able to contradict Lemma 2.

Basically we need to take in t many x values and give out the corresponding y values or function values. To do so, we adopt the following strategy

- First request t strings $\{z_1, z_2, ..., z_t\}$ ε $\{0,1\}^{2n}$ from the PRG verifier.
- Receive an x from R. Scan the first $i-1$ bits.
- If x_1 is 0 go to left child of the root else go the right child of the root. In this fashion proceed along the binary tree conditioning on the parity of the first $i-1$ bits.
- Fill the reached node with z_1. With this as the seed, compute the partial tree below this node by scanning the remaining bits of x, according to our construction (using $\mathbb{G}$).
- Denote y as the final leaf node value and return y to R.

After successfully doing for the first query, we need to reply to the rest of the queries.

We expect t many queries of x from R. For these, we adopt the following strategy

- Receive an x from R. Scan the first $i-1$ bits.
- Check if any of the previous x values had the same prefix. If yes, the reached node takes that z value and we have already computed the all the children for this node. Simply scan the remaining bits of x and traverse along the computed path to return the leaf node value y.
- If not, populate the reached node with the next unused z value and repeat the algorithm and return the leaf node value.

Since we know before hand that R is going to query t many function values, we request t many z strings from the PRG verifier. We need these many because all the x queries may have different prefixes hence we can't reuse the same z string.

If all the z strings were pseudo-random then we have perfectly simulated the $\mathbb{F}_{i-1}$ case and if all of them were truly random we have perfectly simulated the $\mathbb{F}_i$ case. Invoking the same argument from Lemma 2, the rest of the proof naturally follows because if R can distinguish between $\mathbb{F}_{i-1}$ and $\mathbb{F}_i$ this means that

$$|Pr[R^{\mathbb{F}_{i-1}()}(1^n) = 1] - Pr[R^{\mathbb{F}_i()}(1^n) = 1]| \geq negl_2(n)$$

If the distinguisher R replies that we used $\mathbb{F}_{i-1}$, we reply to the PRG verifier saying that all the z strings were pseudo-random. Otherwise if R replies that we used $\mathbb{F}_i$ we reply to the PRG verifier saying all the z strings were truly random. Since this is just passing along the answer due to the structure of construction, we can distinguish t random strings from t pseudo-random strings with the same non-negligible probability thus contradicting Lemma 2 !!!

Hence such an R cannot exist and we can conclude $\mathbb{F}_{i-1}$ is computationally indistinguishable from $\mathbb{F}_i$. Repeating this argument n times and bounding the inequalities with n different negligible functions for each hybrid, we finally get

$$|Pr[R^{\mathbb{F}_0()}(1^n) = 1] - Pr[R^{\mathbb{F}_n()}(1^n) = 1]| \leq negl_3(n)$$

By acknowledging that $\mathbb{F}_n$ is a truly random function and that $\mathbb{F}_0$ is indeed our constructed function and since we have proved they are indistinguishable to any PPT distinguisher, our construction is a PRF !!

■

With this elaborate and beautiful construction we have have successfully proved that if a PRG which doubles length of a string exists, then a PRF exists.

■