CSA E0 235: Cryptography 24th February, 2016

Scribe for Lecture 11

Instructor: Arpita Patra *Submitted by: Sayantan Khan*

1 Summary of the previous class and roadmap for the day's lecture

1.1 Summary of previous class

- A potential candidate for Authenticated Encryption (AE) was constructed from a cpa-secure symmetric key encryption scheme and an scma-secure MAC.
- The scheme was proven to be cpa-secure and have message integrity.
- It was also shown that any AE scheme is always cca-secure. This gave us that the AE scheme constructed in the previous section is indeed a cca-secure scheme, and was the first example of a cca-scheme we constructed.

1.2 The layout of the day's lecture

- Showing that existence of a pseudo-random generator implies the existence of a pseudo-random function. This will be done in the following steps:
  - Construction of a potential pseudo-random functionfrom the pseudo-random generator.
  - Introduce hybrid proof technique.
  - Prove that the constructed function is indeed a pseudo-random function.

The document will however be laid out in a slightly different manner. It will first describe the hybrid proof technique, and give a (optional) motivation from real analysis for it. Then it will move on to the construction and show that the construction indeed gives a pseudo-random function.

2 Hybrid arguments

2.1 What are hybrid arguments?

Very often, it's hard to do a reduction based proof for a particular indistinguishability proposition because the two cases one is trying to distinguish are very different; reducing it to known problem might not work directly. In that case, one constructs certain intermediates, or hybrids, which may be *mutually* indistinguishable, and if the number of intermediates is bounded polynomially, then one can use the triangle inequality to show the first and last in the series of hybrids are also indistinguishable. This is best illustrated with an example.

Suppose G is a pseudo-random generator. We'd like to show that a tuple of outputs from the pseudo-random generator is indistinguishable from a tuple of uniformly picked random numbers. Let the pseudo-random tuple be $(G(x_1), G(x_2), \ldots, G(x_n)$ and the truly random tuple be $(r_1, r_2, \ldots, r_n)$. When the tuple length is just 1, the two are indistinguishable by the very definition of a pseudo-random generator. A natural approach to proving the proposition when the length is more than 1 is to try and reduce the problem to a distinguishing between a truly random number and an output of a pseudo-random generator. It's not obvious how one would go about such a reduction. Consider the following alternative approach. One instead tries to prove the following: $(G(x_1), G(x_2), \ldots, G(x_{i-1}), G(x_i), r_{i+1}, \ldots, r_n)$ is indistinguishable from $(G(x_1), G(x_2), \ldots, G(x_{i-1}), r_i, r_{i+1}, \ldots, r_n)$ for all i. Notice that the only difference in the two tuples is in the i^{th} component, so if one can tell whether that is truly random or pseudo-random, one can distinguish the two tuples. But that just reduces to the problem of telling apart a random number from a pseudo-random number, and those two are indistinguishable, hence the given two tuples are also indistinguishable.

And since there are only polynomially many such intermediates (to be specific, just $(n-1)$), we can add them up using the triangle inequality and rest assured that the difference we get is still negligible, because the sum of polynomially many negligible functions is negligible.

2.2 (Digression) Motivating hybrid arguments

Often in real analysis, when working in some metric space, one needs to show that a point x is within ε distance of another point y. When it difficult to show that directly, one tries to construct a series of intermediate points $z_1, z_2, \ldots, z_n$ such that $d(x, z_1) < \varepsilon_0$, $d(z_i, z_{i+1}) < \varepsilon_i$ and $d(z_n, y) < \varepsilon_n$, where d is the distance function of the metric space. If it turns out that $\sum_{i=0}^{n} \varepsilon_i \leq \varepsilon$, then by the triangle inequality of the distance function d in the metric space, $d(x, y) < \varepsilon$.

One can immediately spot the similarity of this technique to hybrid arguments. The reason why this looks similar to hybrid arguments is because the the probabilities $\mathbb{P}[E_i]$ (E_i is some event of interest) as functions of n are real valued functions with the uniform norm:

$$d(\mathbb{P}[E_1], \mathbb{P}[E_2]) = |\mathbb{P}[E_1] - \mathbb{P}[E_2]|$$

The norm makes the space of functions a metric space, and in a metric space, triangle inequality holds, and that is why hybrid arguments work. The point of looking at hybrid arguments in this light is to hopefully apply a few results from analysis which will carry over to because of the metric space structure. I don't have any particular result at hand, but it's a useful idea to keep in mind.

3 Construction of pseudo-random function from a pseudo-random generator

3.1 Generating a function with the required domain and range from the pseudo-random generator and the seed

We start off with a pseudo-random generator, i.e. a function that takes in input uniformly chosen at random from $\{0,1\}^n$ and outputs an element of $\{0,1\}^{2n}$, in polynomial (in n) time. Using this generator, we'd like to construct a pseudo-random function. The nave way to do it would be to specify what the function outputs at each input. That is not feasible because we are polynomially bounded, and that would take exponential time because of the size of the domain. The next best thing would be to specify a polynomial time algorithm based on the seed which would then compute the pseudo-random functionat various points. But the question still remains: how does one specify an algorithm using a string.

For doing that, we borrow an idea from Lisp[1]. Lisp uses tree-like data structures called S-expressions to store functions internally, and these S-expressions can either be interpreted as strings or algorithms internally. This suggests that we use a tree to define our pseudo-random function.

Given that the pseudo-random generator expands a random string of length n to $2n$, and we want to recursively descend into the tree, we should break up the $2n$ length output into 2 n length parts and let those two strings be the children nodes of the root node, which is the original random string on length n. The next step is obvious. We apply the pseudo-random generator on each of the child nodes and get four more child nodes, two for each of the original child nodes. We keep doing this until the tree has depth n. It's obvious from the way the tree was constructed that it's a complete binary tree with all leaf nodes at the same level. This tree is a representation of a function from $\{0,1\}^n$ to $\{0,1\}^n$. The pseudo-randomness of the function remains to be proved. First we'll show how this is a function.

Consider some element of $\{0,1\}^n$: it's an n-length string of zeroes and ones, call it x. To compute what the function gives when the input is x, just do the following: start at the root of the tree and iterate over the digits of x. For every one, go to the right child of the node you're currently on, and for every zero, go to the left child. Because x has length n, and the tree has depth d, you'll reach a leaf node, which is the output of the function for the input x. That shows what we have is certainly a function.

One problem you might have noticed is that constructing the full binary tree takes exponential time, which we can't do because we're polynomially bounded. But that's not really a problem if we need to compute the value of the function at polynomially many inputs. Computing the value of the function at one input does not require constructing the entire tree; one can just traverse along the path determined by the input in polynomial time, specifically in $np(n)$ time, where $p(n)$ is the time the pseudo-random generator takes and n is the depth of the tree. For polynomially many inputs, the running time will be $O(np(n)q(n))$, where $q(n)$ is the number of queries. An example for a tree of depth 2 is shown in figure 3.1.

[1]Lisp is one of the oldest programming languages alive today, and it pioneered many ideas we take for granted in languages today, for example tree-like data structures, dynamic typing and higher-order functions.

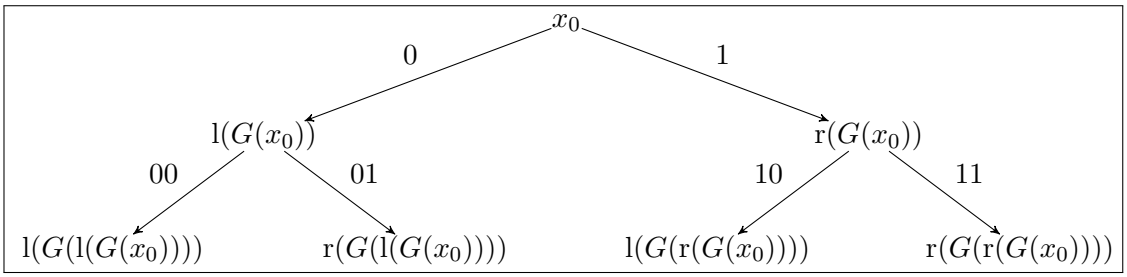

Figure 1: A description of the algorithm for a tree of depth 2. To get the function's output at 10, the algorithm would choose the right branch, i.e. the one marked 1, then the left branch, the one marked 01. Here l and r are functions that give left n bits and right n bits of a $2n$ bit number.

The only thing that remains to be shown is that the function obtained is actually a pseudo-random function. That does not seem obvious at the outset and proving that will be non-trivial and will be one of the first uses of the hybrid argument.

3.2 Some useful observations about the constructed function

Notice that because of the way we've defined the function, it only depends on the seed value x which is placed at the root of the tree and the pseudo-random function G. Those two completely determine the function's output at all points. Alternatively, assume that you started off with two random seeds x_1 and x_2, and instead of placing them at the root node, you place them on the depth 1 nodes. That will also completely determine the function. In general, starting off with 2^i random seeds and placing them on the i^{th} level nodes will completely determine the function.

However, placing the random seeds at different levels induces different distributions on the output of the final function. For example, placing 2^n random seeds at the n^{th} level will induce a distribution that is truly random. On the other hand, placing just one random seed at the root node will span some other distribution, which we *a priori* don't know to be pseudo-random either. We'll show however, in the next subsection, that the distribution spanned in that case is indistinguishable from a truly random distribution, and hence, is pseudo-random.

3.3 Proof that the constructed function is indeed pseudo-random

We'll need a lemma before we prove the main result which is that the constructed function is pseudo-random. We've already seen a rough sketch of the proof of the lemma in section 2.1, but here's a formal proof.

Lemma 1 *If G is a pseudo-random generator, $r_1, r_2, \ldots, r_m$ are independent random numbers and $x_1, x_2, \ldots, x_m$ are independently chosen random seeds, then*

$$(r_1, r_2, \ldots, r_m) \stackrel{\text{c}}{\equiv} (G(x_1), G(x_2), \ldots, G(x_m))$$

This is the same as saying that for all adversaries, the difference in the probability of them correctly identifying a tuple as random or pseudo-random is negligible. In other words,

$$|\mathbb{P}[A(r_1, r_2, \ldots, r_m) = 1] - \mathbb{P}[A(G(x_1), G(x_2), \ldots, G(x_m)) = 1]| < \text{negl}(n)$$

for all adversaries A, where $[A(\text{tuple}) = 1]$ denotes the event of A correctly identifying the tuple and n is the security parameter, in this case the length of the strings.

Proof Consider the probability of the adversary distinguishing the following two tuples:

$$(G(x_1), G(x_2), \ldots, G(x_{i-1}), G(x_i), r_{i+1}, \ldots, r_m) \quad (1)$$
$$(G(x_1), G(x_2), \ldots, G(x_{i-1}), r_i, r_{i+1}, \ldots, r_m) \quad (2)$$

I claim that any adversary could, at the best, distinguish the two tuples with negligible probability. Here's why. Suppose some adversary could distinguish between the two tuples with a probability greater than $\frac{1}{p(n)}$ for infinitely many n and some polynomial p. Then one could use that adversary to construct another adversary who can distinguish truly random and pseudo-random number with a probability greater than $\frac{1}{p(n)}$ for infinitely many n and some polynomial p. But that clearly cannot happen because the very definition of pseudo-random generator requires probability of distinguishing be negligible for all n.

Here's the construction of the new adversary A: She (we'll assume without loss of generality that the adversary is a she. The proof is identical for adversaries of any gender.) will query for a number, call it k, from the verifier of the random vs. pseudo-random generator experiment. Once she receives that number she'll generate $i-1$ pseudo-random numbers with independently picked seeds, call them $G(x_1), G(x_2), \ldots, G(x_{i-1})$. Furthermore, she'll also generate $m-i$ truly random numbers independently, call them $r_{i+1}, \ldots r_m$. Then she'll pass on the tuple $(G(x_1), \ldots G(x_{i-1}), k, r_{i+1}, \ldots, r_m)$ to the adversary A' who can distinguish the tuple 1 and 2 with non-negligible probability. If A''s reply is that the given tuple is tuple 1, then A tells the verifier that k was pseudo-random, else if A' claims that the given tuple was tuple 2, then A tells the verifier that k was truly random. Let's compute the probability that A guesses correctly.

Note that if A' correctly identifies the tuple, then A's answer is correct with probability 1. That means the probability of A being correct is the same as probability of A' being correct, and that probability if $\frac{1}{p(n)}$ for infinitely many n. But that means we have constructed a distinguisher for pseudo-random strings, and that is not possible. Hence our assumption that there exists a distinguisher for tuple 1 and 2 must be wrong and the tuple are indistinguishable.

But this holds true for all i: in particular, vary i from 1 to $m-1$. You'll get the following inequalities:

$$|\mathbb{P}[A(G(x_1), r_2, r_3, \ldots, r_m) = 1] - \mathbb{P}[A(r_1, r_2, \ldots, r_m) = 1]| < \text{negl}_1(n)$$
$$|\mathbb{P}[A(G(x_1), G(x_2), r_3 \ldots, r_m) = 1] - \mathbb{P}[A(G(x_1), r_2, \ldots, r_m) = 1]| < \text{negl}_2(n)$$
$$\vdots$$
$$|\mathbb{P}[A(G(x_1), G(x_2), \ldots, G(x_{m-1}), G(x_m)) = 1] - \mathbb{P}[A(G(x_1), G(x_2), \ldots, r_m) = 1]| < \text{negl}_{m-1}(n)$$

Adding up all the inequalities and using the triangle inequality, we get

$$|\mathbb{P}[A(r_1, r_2, \ldots, r_m) = 1] - \mathbb{P}[A(G(x_1), G(x_2), \ldots, G(x_m)) = 1]| < \sum_{i=1}^{m-1} \text{negl}_i(n)$$

But the sum of polynomially many negligible functions is also negligible. Hence,

$$|\mathbb{P}[A(r_1, r_2, \ldots, r_m) = 1] - \mathbb{P}[A(G(x_1), G(x_2), \ldots, G(x_m)) = 1]| < \text{negl}(n)$$

This proves the lemma. ■

Now we need to show that the distribution induced by putting a random seed on the root node is indistinguishable from the distribution induced by putting random seeds on the leaf nodes. We'll first prove that the distribution induced by placing random seeds on the i^{th} level is indistinguishable from the distribution induced by placing the random seeds on the $(i+1)^{\text{th}}$ level. We'll show this by reduction, saying that if an adversary can distinguish the two distributions with non-negligible probabilities, then we can use that to construct an adversary who can distinguish between random and pseudo-random tuples of size 2^{i+1}, which should not be possible because because of lemma 1. Hence the two distributions must be indistinguishable. Here's a formal proof.

Lemma 2 *Let the distribution induced on the leaf nodes by placing 2^i random seeds on the i^{th} level be called H_i and distribution induced on the leaf nodes by placing $2^(i+1)$ random seeds on the $(i+1)^{th}$ level be called H_{i+1}. Then H_i and H_{i+1} are indistinguishable.*

Proof Suppose they were distinguishable, i.e. there existed an adversary A who could, in polynomial number of queries to the tree, determine whether the i^{th} or $(i+1)^{\text{th}}$ level was randomly seeded. We could then use that adversary to construct another adversary A' who could distinguish between random and pseudo-random tuples of size 2^{i+1}.

Here's how A' would work. A' would take a tuple of size 2^{i+1} from her verifier, and place the elements of the tuple on the $(i+1)^{\text{th}}$ level of the tree. Now A would start making queries, and whenever A makes a query, A' constructs the answer to the query using the partial tree she has. Notice that if the tuple A' received was truly random, then the distribution on the leaves would be H_{i+1}. Suppose the tuple A' received was pseudo-random. Then it could have been generated by taking a random tuple of size 2^i, applying the pseudo-random generator on each of the elements, and then breaking up the elements of length $2n$ into two elements of length n each. Then the new tuple of size 2^{i+1} is a pseudo-random tuple. Hence the distribution induced by putting a pseudo-random tuple of size 2^{i+1} on the $(i+1)^{\text{th}}$ level is H_i.

Now suppose A makes a polynomial number of queries and for each query, A' returns the answer by descending along the partial tree she has. After the queries have been made, A returns an answer saying whether the distribution was H_i or H_{i+1}. If A says the distribution is H_i then A' should say the tuple was pseudo-random, otherwise if A says the distribution is H_{i+1}, then A' should say the tuple was truly random. From the last paragraph, we know that if A is correct, so is A' and if A is wrong, then so is A'. Hence the success probability of both of them is the same, and we know that A has a success probability of greater than $\left(\frac{1}{2} + \frac{1}{p(n)}\right)$ for infinitely many n, and so does A' by extension. That means A' can

distinguish pseudo-random and truly random tuples with non-negligible probability, which we know is false by lemma 1. Hence, H_i and H_{i+1} are indistinguishable. ■

This lemma gives us enough power to finish our proof. Now we claim that H_0 and H_m are indistinguishable and hence, the constructed function is a pseudo-random function.

Theorem 3 *The distribution H_0 is indistinguishable from H_m.*

Proof We'll now use lemma 2 and use hybrid arguments. From lemma 2, we have for all i, and for all adversaries A,

$$|\mathbb{P}[A(H_i) = 1] - \mathbb{P}[A(H_{i+1}) = 1]| < \text{negl}_i(n)$$

where $A(H_i) = 1$ if A correctly identifies the distribution as H_i. We have the following chain of inequalities:

$$\begin{aligned}
|\mathbb{P}[A(H_0) = 1] - \mathbb{P}[A(H_1) = 1]| &< \text{negl}_0(n) \\
|\mathbb{P}[A(H_1) = 1] - \mathbb{P}[A(H_2) = 1]| &< \text{negl}_1(n) \\
&\vdots \\
|\mathbb{P}[A(H_{m-1}) = 1] - \mathbb{P}[A(H_m) = 1]| &< \text{negl}_0(n)
\end{aligned}$$

Adding up all the inequalities and using the triangle inequality, we get

$$|\mathbb{P}[A(H_0) = 1] - \mathbb{P}[A(H_m) = 1]| < \sum_{i=0}^{m-1} \text{negl}_i(n)$$

And since sum of polynomially many negligible functions is negligible, we get

$$|\mathbb{P}[A(H_0) = 1] - \mathbb{P}[A(H_m) = 1]| < \text{negl}(n)$$

This completes the proof. ■

3.4 Summarizing the pseudo-random function

We have successfully constructed a function that runs in polynomial time and has the correct domain and co-domain (section 3.1) and which is pseudo-random (section 3.3).

References

[1] Dr. Arpita Patra: Lecture 11
`http://drona.csa.iisc.ernet.in/~arpita/Cryptography16/AP_Lecture11.pptx`