Applied Cryptography

Unit-2 Symmetric key Cryptography

Lecture Notes 4:

Symmetric Key Cryptography- Pseudo randomness

Recommended Reading:

Katz-Lindell: Chapter 3:3.3

Pseudorandomness

One of the essential building block component in Cryptography for constructing computationally secure system is the Pseudorandomness.

What is Random?

Randomness denotes the next movement is unpredictable. Physically this randomness exists in nature. By sampling the noise, we can in fact produce random numbers.

If you have three strings mentioned as

- 0101010101010101
- 0010111011100110
- 0000000000000000

And asked to identify the random string, we automatically tries for some pattern in the existing strings and we see that the string mentioned as second option do not hold any repeating pattern concludes that may be this a random number.

But that is not true, as because the property randomness isn't for specific value but it holds for distribution. So each value possess equal possibility of being random thus if we are generating a 16 bit random sequence then the probability is $2^{-16.}$

That being said the randomness for a distribution can formally be represented as $2^{-n.}$

Pseudo randomness:

Viewing pseudo randomness at times we feel that there is no significant difference between the uniform and pseudo but the only significant difference

it holds is the distribution can generate pseudorandom ness by passing certain statistical tests.

The type of statistical test varies depending on the attacker, Well what I mean by that is, if we look at for example the probability, with which the first bit of x is 1, when we sample x according to distribution D. Then that probability should approximate, the probability with which the first bit of x would be 1, if x were chosen from the uniform distribution.

If x were chosen from the uniform distribution, the probability that it's first bit would be equal to 1, would be exactly one-half. So we require that when we sample x according to distribution D, the first bit of x should be 1 with probability roughly one-half.

Similarly, if we look at the probability with which the parody, of the bits of x, that is the XOR, of all the bits in x, is 1. Well, if D is a pseudorandom distribution then that probability should approximate the probability with which, the parody of x is 1, when x is sampled from the uniform distribution. That probability of exactly one-half. So therefore if D is pseudorandom, the probability with which the parody of x is 1 when x is sampled from D, should also be close to one-half. And more generally we can fix, some set, of statistical tests. That is, any kind of predicate, we define on a string x. And we can look at or we can compare, the probability with which Ai of x is equal to 1 when x is sampled from distribution D and the probability that A ek, Ai of x is equal to 1, when x is sampled from the uniform distribution. And those should be close. And if we require that to hold, if we require closeness to, to hold for say 20 different statistical tests A1 through A20, then that can serve as our definition ,of what it means for D to be pseudorandom. That is D is pseudorandom, if these probabilities are equal for i equals 1 to 20 or are close rather, for i equals 1 to 20.

Practical Definition in Cryptography:

You should immediately see though, that this kind of a definition, while it may be sufficient for some applications, is not going to be sufficient in cryptography, where we have an adversary, who's specifically trying to distinguish. Your string sampled from your distribution, from string sampled from the uniform distribution. That is, the definition on the previous slide, is not going to suffice, in an adversarial setting. And the reason for this is because we don't know, what statistical test an attacker will come up with. Right, we had defined on the

previous slide that a distribution, was pseudorandom, if it passed some set A1 through A20, of statistical tests. But an attacker might come along with its own test, not a net set. And if that test distinguishes the output the this string x chosen from the uniform distribution from a string x sampled from your distribution D, then D shouldn't count in the pseudorandom distribution.

So the cryptographic definition of pseudorandomness the modern definition, is that a distribution D is pseudorandom, if it passes every, efficient statistical test. Now if we had left out the word efficient and required it to be pseudorandom only if it passes every single statistical test, it turns out that that's equivalent to requiring that D be uniform. And therefore, uniform in pseudorandom would end up with the same meaning, and we wouldn't get anywhere. But as we've talked about, we're interested a computational relaxation, and it's reasonable to restrict our attention, to only efficient attackers, who can only perform efficient statistical tests. And so, it'll suffice for our purposes if we define D in this way.

So fix some distribution D, on n bit strings. Then we can say that the distribution D, is t epsilon pseudorandom. If for all attackers A running in some, running in time at most t, the probability with which A of x equals 1, when x is sampled according to D, and the probability with which A of x is equal to 1, when x is sampled from the uniform distribution.

Those two probabilities, are at most epsilon apart. So the difference between those probabilities or the absolute value of that difference, is at most epsilon.

This exactly corresponds to the intuitive notion we had on the previous slide. If we view the attacker A, as being equivalent to some kind of statistical test, on strings x sampled from the distribution. Then we're requiring that no statistical test that runs in time, at most, t, can distinguish between a string sampled from the uniform distribution, and a string sampled from distribution D, with probability any better than epsilon.

Asymptotic Definition of Pseudorandomness:

The asymptotic definition of pseudorandomness which is the one we're going to be using, from now on in the course, is a bit more complicated because again we need to take into account this idea of having a security parameter. So as

before, we'll denote that security parameter by n. And here the security parameter will correspond exactly, to it will define for us some length of strings that we're considering. So in addition to the security parameter n, we'll have some polynomial p, and we'll let D sub n, be some distribution, over strings of length p of n. So, that means that for every value of the security parameter n, we're looking at distributions over string of length, over strings of length p of n.

You can think, if you like for now of p of n being equal to n. But it will be more interesting later on if we let p of n be larger than n, so you can think of p of n being n squared, for concreteness. Now, in asymptotic setting, pseudorandomness, will be a property of a sequence of dish, dis, distributions. So rather than looking at any particular distribution, DI, and looking at the probability with which some test can distinguish it. What we're going to be interested in is the asymptotic probability with which any efficient attacker, can distinguish D of n from a uniform distribution, over strings of length p of n.

And again we're going to be interested in the asymptotic, performance here. The asymptotic behavior of an algorithm or, or of any algorithm trying to distinguish. Or equivalently the asymptotic pseudorandomness, of the distribution's Dn. So I'm going to let this set Dn correspond to the set D1, D2, D3, et cetera. So on up to infinity. So we have now a sequence of distributions, defined or called Dn, and we'll say that this sequence of distributions is pseudorandom. If for every probabilistic, polynomial-time algorithm A, there's some negligible function epsilon. Such that if we look at the difference between the probability that A of x equals 1 when x is sampled from distribution Dn, and the probability that A of x outputs 1, when x is sampled from the uniform distribution. The difference or the absolute value of the diffOkay. Let's break this down a little bit further. So on the left hand side of this inequality, we have a term, a difference or an absolute value of a difference between two values, which is a function of n. Fixing some algorithm A, for every value of the security parameter n, we can explicitly compute or evaluate the probability with which A of x equals 1, when x is sampled from D of n. And the probability that A of x equals 1, when x is sampled from the uniform distribution, over strings of length p of n. Note that the input to A in both cases, is of the same length. Because in one case we're sampling from the distribution D of n, which is a distribution over strings of length p of n. And in the other case, we're sampling from the uniform distribution over strings of length p of n.erence will be at most, epsilon n. So for

every value of the security parameter n, we can compute the two probabilities, take their difference, take the absolute value. And we get a number. So what's on the left hand side is, something which is a function of n. For every value And what we're asking, or what we're requiring, is that that function, be negligible, meaning that asymptotically it will decay to zero, faster than any inverse polynomial function.of n, we get a real number. So again, the sequence D of n is pseudorandom if for every efficient algorithm A, there's some negligible function, which may depend on A. But which will still be guaranteed to be negligible, such that the probability with which A outputs 1, when given a string sample according to D of n, and the probability with which A outputs 1, when given something sampled according to the uniform distribution, is at most epsilon of n, is at most that negligible function.

PRG:

If we have a sequence of distributions, Dn where each distribution Dn is a distribution over strings of length p(n), for some function p, will say that that sequence of distributions is pseudorandom if for all probabilistic polynomial time adversaries A. There's some negligible function epsilon such that if we look at the difference in probabilities, between the probability that A(x) outputs one when x is sampled according to D(n) and the probability that A(x) outputs one when x is sampled according to the uniform distribution over strings of length p(n), then the absolute value of the difference of those probabilities is at most epsilon n. So intuitively, for any statistical test A, running in pro probabilistic polynomial time, the distinguishing gap, right, the probability with which that statistical test can distinguish whether it's given a string sampled according to D(n), or, or whether he's given a string sampled according to the uniform distributWe now want to define the important notion of pseudorandom generators. I'll abbreviate this by PRG. In practice, you may also hear of pseudorandom number generators abbreviated PRNG, and for our purposes those are going to be the same. But I'm going to stick with the terminology PRG, pseudorandom generator.ion, or the strings of the appropriate length. It's at most epsilon(M). So, a PRG, a pseudorandom generator, is an efficient deterministic algorithm that expands a short, uniform seed or input into a longer pseudorandom output. We'll define this more formally in a moment. But intuitively, this is just an algorithm that is given a short input, converts it into a longer output and it has the property that if you sample its input uniformly, then

its output will be pseudoraAnd pseudorandom generators are very useful whenever you have a small amount of true random bits, but you want lots of random looking bits. So just as a toy example or a made up example, say we have some algorithm that requires 10,000 random bits, 10,000 uniform and independent bits.ndom. We said in an earlier lecture that actually sampling true randomness is very difficult. You have to collect mouse movements or collect timings between network events or typing at the keyboard. And it takes actually quite a bit of effort and time to collect true random bits. So imagine that we've collected, say, 100 random bits, 100 uniform and independent bits. This assumes we've collected them and then processed them appropriately to smooth them out and get some number, say, 100, of true random bits.

Well, what we can do is we can apply a pseudorandom generator to those 100 bits and get a 10,000 bit output with the guarantee that as long as our input bits are random or close to random, i.e, close to uniform, the output of that pseudorandom generator will be random looking. And if we then run our algorithm that requires those 10,000 random bits using as its random bits, the output of the pseudorandom generator, then the output of that algorithm will be as good as if we had run it on 10,000 uniform and independent bits.

So pseudorandom generators are very, very useful when you have a small amount of true randomness and you want to extend it into a larger amount of things, of something, which is as good as random. More formally, let G be a deterministic, polynomial time algorithm. So, not only is this algorithm efficient but it's also deterministic. If you run it twice on the same input, you get the same output.

We'll require also that G be expanding. This just means that the output of G is longer than its input. And in particular we can denote the output length of G(x) to be equal to p of the length of x. So this means that when you feed, when you feed G with some input string x that has length n. Then the output has length p(n), and will require that p(n) be greater than n for all n. In pictures, what we have is this algorithm G which we're viewing here as box. And it takes as input a short seed. Okay, we're going to continue to refer to the input to G as a seed. It takes that short input and produces a longer output.

Now what's interesting to note here is that this deterministic algorithm, or this deterministic function G, defines naturally a sequence of distributions. And it

does that in the following way. We can define distribution Dn to be the distribution on strings of length p(n). Remember that p(n) was the output length of G when given an input of length n.

It's going to be the distribution on strings of length p(n) defined by choosing a uniform seed x and then outputting G(x). This is a valid distribution on strings of length p(n) where it is defined by this process by which we sample a uniform seed and then apply G to that seed to obtain a string of length p(n).

If you wanted to work through this formally, what it means, well then it means that we're defining the following distribution d(n). We're defining a distribution d(n) where the probability of any particular string y of length p(n) is equal to the probability with which G(x) is equal to y when we sample x uniformly. That is the probability that D(n) assigns to some string y is exactly the probability with which G(x) equals y, which is exactly the summation over all inputs x, such that G(x) equals y of the probability of x. And you can continue to further manipulate this and what you get in the end is that it's the number of inputs x, such as G(x) equals y divided by 2 to the n, right? It's the number of x's that map onto this string y multiplied by 2 to the minus n because we're sampling x from the uniform distribution on n bit strings. And so the probability of any particular x in that set being chosen is 2 to the minus n. So you can think about it either way, you can think about it formally, or you can think about it in terms of the randomized experiment, which is also formal, it's just that we're not, we're not explicitly writing out the distribution. These are exactly equivalent ways of looking at the same randomized process. We can then define this particular algorithm G, to be a pseudorandom generator, if this sequence of distributions Dn defined by G, is pseudorandom. What this means, is that for all probabilistic polynomial time attackers A, there's some negligible function epsilon such that, if we look at the probability with which A when given input G(x) outputs one, and compare that to the probability with which A, given a uniform string y outputs one. The absolute value of the difference in those probabilities wilSo, in, more intuitively, this means that no efficient algorithm a can distinguish with probability better than epsilon whether it's given G(x), right the output of a pseudorandom generator, when run on a uniform string x or whether it's given a completely uniform string y.l be at most epsilon. Right, so that means that our pseudorandom generator really is giving us something that looks random. It's producing for us strings of length p(n) that are indistinguishable for any efficient

attacker from a uniform string of that length. Do pseudorandom generators exist? Unfortunately, we don't know. This is one of those things where we can't hope to prove the existence, that we can't hope to prove the existence of without resolving, in particular, the question of whether p is equal to np.