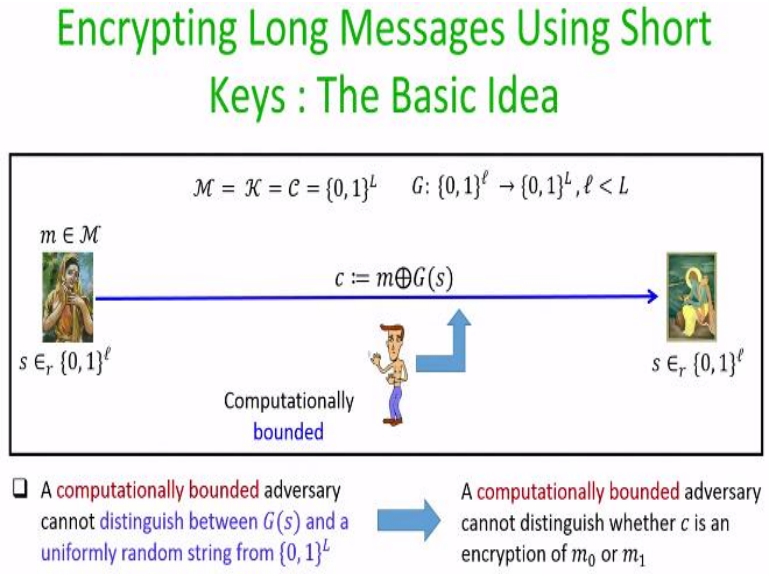
Foundations of Cryptography
Prof. Dr. Ashish Choudhury
(Former) Infosys Foundation Career Development Chair Professor
Indian Institute of Technology – Bangalore

Lecture – 8
Pseudorandom Generators

(Refer Slide Time: 00:30)

Hello, everyone, welcome to lecture 8. The plan for this lecture is as follows. Here, we will discuss how to get rid of the first restriction imposed by perfect secrecy. Namely, we will discuss how to encrypt long messages using short keys and for this we will introduce our first primitive in the computationally secure world, namely pseudorandom generators, and we will discuss various equivalent definitions for pseudorandom generators.

(Refer Slide Time: 00:54)



So, the idea behind encrypting arbitrary long messages using short keys is as follows. Recall the one time pad scheme where the message space, key space, and ciphertext space all consist of bit strings of length L bits. To encrypt a message of length L bits, sender and receiver agree upon a uniformly random key of size l bits generated by the key recognition algorithm and to encrypt the message, sender simply performs XOR the message with the key and resultant ciphertext is communicated over the channel.

We discussed rigorously that this notion of this encryption process provides you the strongest notion of secrecy namely perfect secrecy, where it is ensured that if a computationally unbounded adversary eavesdrops the ciphertext, then it cannot distinguish apart whether the ciphertext is an encryption of m0 or whether it is an encryption of m1 because the key is a uniformly random bit string of length L bits. That was the basic idea of one time pad scheme.

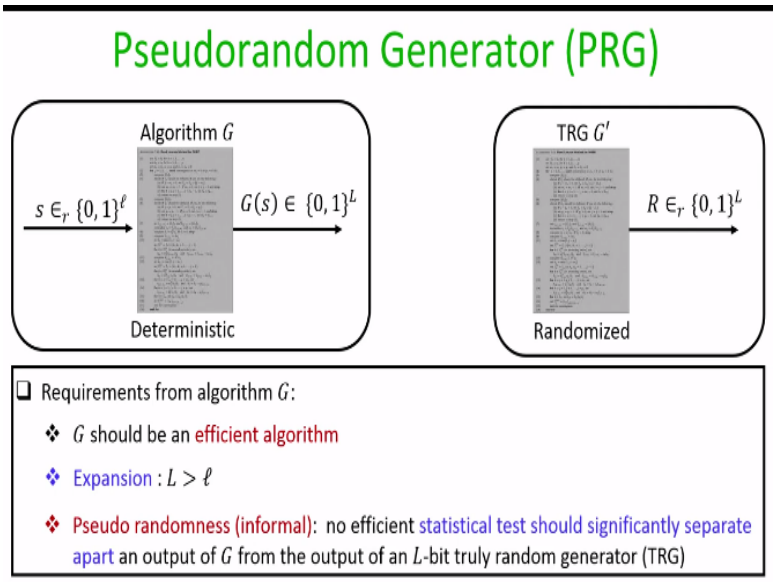
Now, here our goal is to come up with an encryption mechanism where we want to encrypt short messages using long keys, where you want to encrypt long messages using short keys. So, for this, we introduce new function or a primitive which I denote by G, we will very soon see what exactly this primitive is and what are the properties we require from this primitive. So, what this function G does is it takes an input of size little l bits and it gives you an output of size big L bits.

Where both little l and big L are polynomial functions of your security parameter, but the output of this function G is significantly large compared to the input of this function G. Now, the first modification that we are going to make in the blueprint of one time pad is that

instead of sender and receiver agree upon a key, which is of as large as the message, both sender and receiver are now going to agree upon a uniformly random string of length little l bits, and now sender simply cannot XOR this string s with the message because the size of the message and the size of the string s are different.

So what sender is going to do is instead of XORing the message with the key which was happening in one time pad, sender is going to XOR the bits of the message with the output of the function G on the input s. So since the output of the function G is going to be a string of length big L bits, we can perform the XOR of the bits of the message with the output of the function G on input s, and the resultant ciphertext is communicated over the channel.
Now, what we hope is if instead of a computationally unbounded adversary, there exist a computationally bounded adversary whose running time is polynomially bounded and if it is ensured that the computationally bounded adversary cannot distinguish the output of the function G on the input s from a uniformly random bit string of length big L bits, then it will be ensured that the computationally bounded adversary cannot distinguish apart whether the ciphertext c that it is observing is an encryption of m0 or whether it is an encryption of m1.

So, that is the basic idea behind encrypting long messages using short keys. The basic idea is instead of XORing the message with a uniformly random key whose size is as largest the message, we are now going to perform the XOR of the message with an output of this function G and we will assume that a computationally bounded adversary cannot distinguish apart the output of this function G from a uniformly random bit string of length big L bits.
(Refer Slide Time: 04:43)

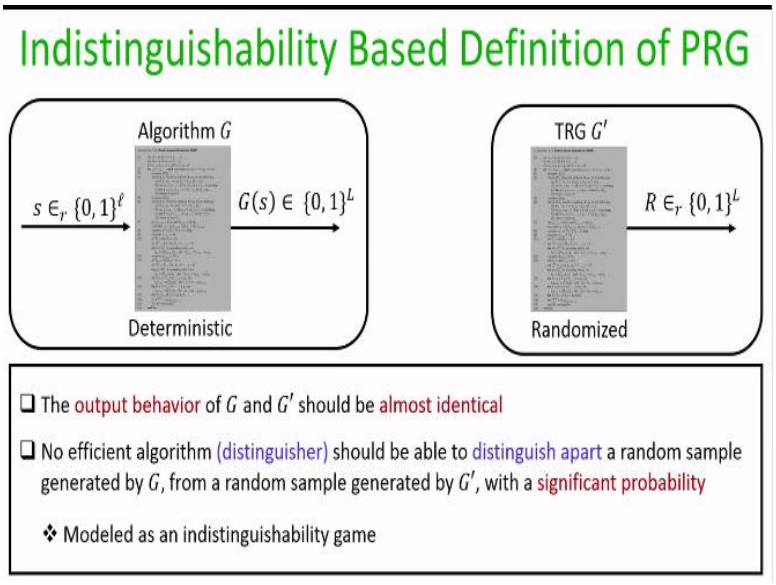


So this function g is called a pseudorandom generator. Let us see the internal details and the security properties from this primitive called pseudorandom generator. So on a very high level, a pseudorandom generator, denoted by G is a deterministic algorithm, which takes as input a uniformly random string of length little l bits and it is going to produce an output whose length is big L bits. The requirements from this algorithm G is as follows. First of all, the running time of this algorithm G should be a polynomial function of a security parameter that means your G should be an efficient algorithm.

This internally means that both the value little l as well as big L are some polynomial functions of your security parameter, right. So that is the first requirement from your pseudorandom generator. The second requirement is that output of the pseudorandom generator should be significantly large or large compared to the input size, typically in practice the output size is significantly large compared to the input size. The third requirement which is the security requirement from this primitive is the pseudo randomness requirement.

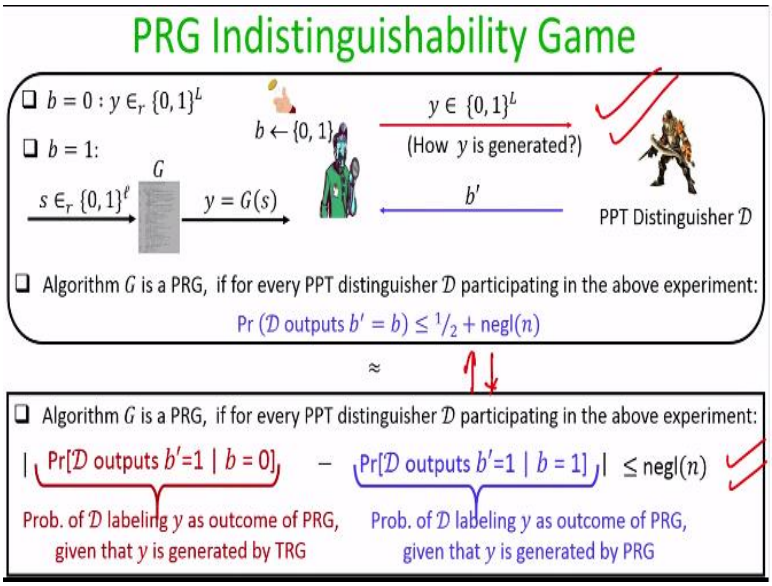
Informally, the pseudo randomness requirements requires you that no efficient statistical test should significantly separate apart an output which is produced by the algorithm G from an output of a truly random generator. That means, if you consider a truly random generator, which I denote as G dash, which is going to uniformly randomly output a bit string of length big L bits, then the pseudo randomness requirement is that no statistical test should be able to distinguish apart G of s versus a uniformly random string produced by the algorithm G dash.

(Refer Slide Time: 06:38)



This internally means that output behavior of your algorithm G and G dash should be almost identical, and this is captured formerly by indistinguishability based experiment, where the intuition behind indistinguishability based experiment is that no efficient algorithm should be able to distinguish apart a random sample generated by the algorithm G from a random sample generated by a truly random generator G dash.

(Refer Slide Time: 07:10)



So let us see the indistinguishability based definition of PRG. So in this experiment, we have distinguisher whose goal is to distinguish apart a sample generated by the pseudorandom generator from a sample generated by a truly random generator and we have a hypothetical verifier or an experiment. In the experiment, the verifier challenges the distinguisher by a string or a sample of length big L bits. The challenge for this distinguisher is to find out whether the sample y is generated by running the pseudorandom generator or by running a truly random generator.

Namely, the sample y which is thrown as a challenge in the experiment to the distinguisher could have been generated by one of the following two ways. The very fact would have tossed a uniformly random coin, and if the coin toss is 0, then the challenge sample which is given to the distinguisher is a uniformly random sample generated by a truly random number generator. Whereas the the big B = 1, then what the verifier would have done is it would have picked a seed or the input for the function G itself uniformly randomly and it would have computed the function G on this input s and would have produced a sample y.

So, now, the challenge for the distinguisher is to find out how exactly the sample is generated, whether it is generated by running a truly random generator or whether it is generated by executing the algorithm G on a uniformly random seed. The distinguisher has polynomial amount of time to tell whether the y is generated by method 0 or by the method 1. So, the output of the distinguisher is a bit which we denote as b dash.

The definition of pseudorandom generator is we say an algorithm G is a pseudorandom generator if for every polynomial time distinguisher participating in this indistinguishability based experiment, the probability that it can correctly identify b = b dash is upper bounded by half plus some negligible function in the security parameter, right, where the probability of our D outputting b = b dash is over the randomness of the distinguisher and of the randomness of the experiment or the verifier. So, here the term PPT which I am introducing here stands for probabilistic polynomial time.

So, by a probabilistic polynomial time algorithm, I mean a polynomial time algorithm which is of randomized nature. So, for the rest of the course, since we will be discussing computationally secured primitives, we will be considering adversaries whose running time will be probabilistic polynomial time. So, now in this definition, we require that probability that D is able to identify the mechanism by which y is generated should be upper bounded by half plus negligible.

Why half plus negligible? Because there is always a trivial distinguishing strategy for the distinguisher to just guess the method by which y is generated, and the probability by which this guessing strategy of the distinguisher will be successful is half. So, we can never demand in this definition that the probability that distinguisher's output is correct should be 0 because there is always 1 by 2 probability distinguisher who can distinguish or tell whether the y sample is generated randomly or by running the truly pseudorandom generator.

Apart from the probability half, we are also willing to let adversary identify the correct mechanism by which the sample y is generated with a negligible success probability and this is because we are in the computationally secure world, and looking ahead, we will be using this PRG to encrypt arbitrary long messages. So remember in the computationally secure world, one of the necessary evils that is associated in the computationally secure model is that

we should be willing to let the adversary break or attack the scheme with a negligible or very small error probability.

So, that why this additional negligible probability is allowed for the adversary to win the experiment or identify whether the sample is generated randomly or by running the pseudorandom generator. I stress here that in this whole experiment, the description of the algorithm G is publicly known, because as per the Kerckhoffs' principle, we never assume that the steps of the algorithm are hidden. In the experiment, what is hidden from the adversary is whether the seed with which the experiment would have invoked algorithm G, right.

The goal of the distinguisher is to find out whether y is generated randomly or by running the pseudorandom generator. It turns out that there is an equivalent definition for the pseudorandom generator and the equivalent definition basically demands that irrespective of the way the verifier has decided to choose the sample, the output of the distinguisher should be identical. That means, the alternate definition requires you that absolute difference between these two probabilities should be upper bounded by a negligible function.

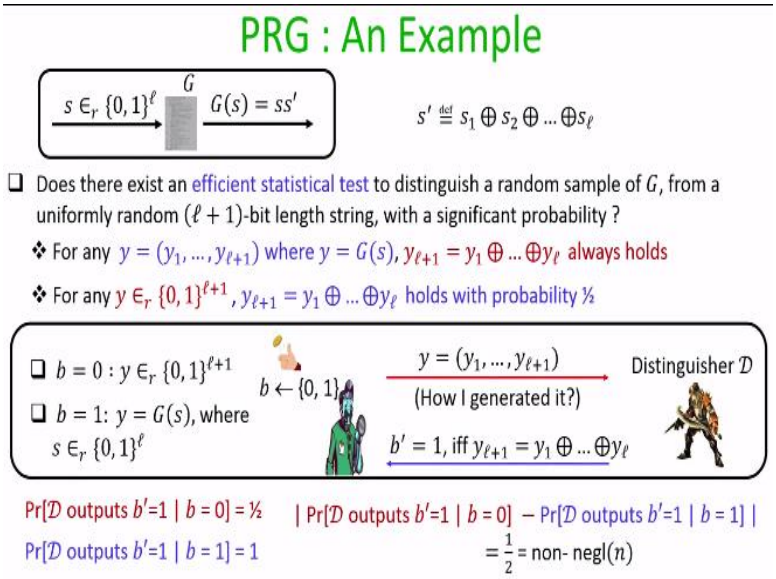
So, let us see what exactly these two probabilities are all about. The first probability is the probability that D labels the sample why y as the outcome of a pseudorandom generator even though it has been generated by a truly random generator. That means, what is the probability that D outputs b dash = 1 given that b = 0. So D output b dash = 1, that means D is labeling the sample y which is thrown to him as a challenge as the outcome of a pseudorandom generator, given that b = 0.

That means, the verifier has decided to choose the sample randomly, whereas the second probability is the probability that D labels the sample y as the outcome of a truly pseudorandom generator, given that indeed it was generated by a pseudorandom generator, right. So, the second alternate definition requires you that the distinguishing advantage of the distinguisher. So, we say that absolute difference between these 2 probabilities is the distinguishing advantage of the distinguisher with which it can distinguish apart whether the sample has been generated by a pseudorandom generator or truly random generator.

So, this alternate definition requires you that irrespective of the way by which the sample y would have been generated, these responses should be almost identical in both cases except with a negligible probability, and it turns out that we can prove that both these definitions or conditions are equivalent. Namely, we can prove that if we have a pseudorandom generator, which satisfies the first condition, then it also implies that it has to satisfy the second condition and vice versa. That means, both these definitions are equivalent to each other.

Hence, in the rest of the course, we can use any of these 2 conditions to mention the security definition of pseudorandom generator as per our convenience. Just remember that the first definition says the probability is that D correctly finds out the mechanism by which y is generated should be upper bounded by half plus negligible, whereas the second condition requires you that the distinguishing advantage of the distinguisher, namely its advantage of separating out whether y is generated by mechanism 1 or mechanism 2 should be upper bounded by a negligible probability.

(Refer Slide Time: 14:58)



So, let us see an example of pseudorandom generator. In fact, the construction that we are going to see is not a pseudorandom generator and we are going to formally prove that. So in this example, the function G is as follows. It takes input of size little l bits and it stretches its input by 1 bit, namely it produces an output whose length is one more than the length of its input, and the way it is stretching is as follows. The first l output bits of the algorithm are same as the inputs of the algorithm, that means they are going to be uniformly random.

Whereas the last output bit of the algorithm G is simply the XOR of the bits of the inputs of algorithm D. So that is a description of the algorithm G which is given to you and now you have to prove or disprove whether this algorithm G is pseudorandom generator or not. So in fact, it turns out that this algorithm G is not a pseudorandom generator and for that, we can consider the following efficient statistical test which can distinguish apart any sample generated by an algorithm G from a uniformly random string of length l + 1 bits.

If we consider any sample generated by the algorithm G on a uniformly random input s, it turns out that in that output, the l + 1th bit has to be the XOR of the first l bits because that is what is the output property of any output generated by the algorithm G. Whereas if we consider any uniformly random string of length l + 1th bits generated by a truly random generator, it may happen that l + 1th bit is indeed the XOR of the first l bits, but the probability of this happening is only 1 by 2.

That means you now have a condition which is definitely going to be satisfied for a sample always if the sample would have been generated by algorithm G, whereas the probability that the same condition holds for a random sample generated by an algorithm is at most half. Now, based on this intuition, we can convert this statistical test into an efficient distinguisher who can distinguish apart a sample generated by an algorithm G from a truly random generator with a significant probability and the distinguisher strategy is as follows.

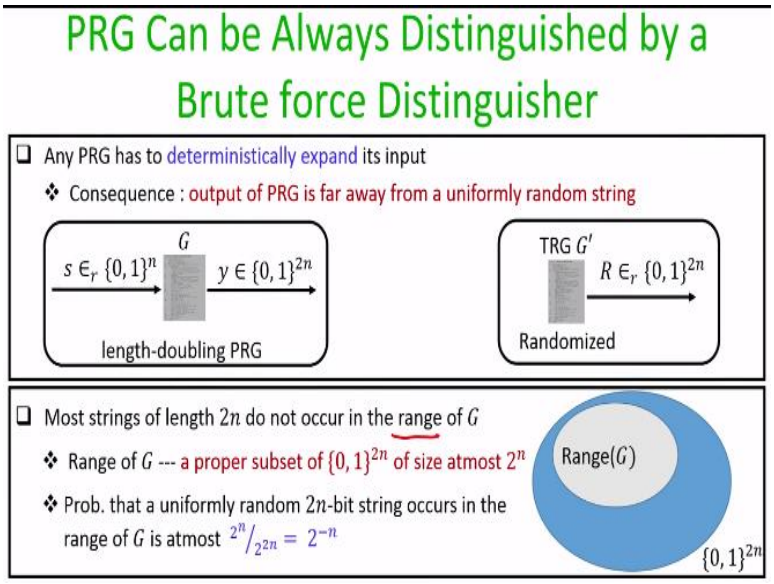
So, the distinguisher will be thrown with a challenge, which will be consisting of a string of length l + 1th bits and the challenge for the distinguisher is to find out how it is generated. Namely, whether it is generated uniformly randomly or whether it has been generated by running the algorithm G on a uniformly random input or seed s. Now, the distinguishing strategy for the distinguisher is as follows. The distinguisher labels the sample y as the outcome of the pseudorandom generator.

Namely it says b dash = 1 or outputs b dash = 1 if and only if it finds the l + 1th bit of the challenge that is given to him is the XOR of the remaining l bits of the challenge. Now let us calculate the distinguishing advantage of this distinguishing strategy. Let us first find out what is the probability that this distinguishing strategy labels sample which is uniformly random as a sample generated by a truly random number generator, and it turns out that the probability that D outputs b dash =1 given that b = 0 is half, because if b = 0.

That means the sample y is truly random and only with probability 1 by 2, it will be ensured that the l + 1 bit is actually the XOR of the remaining l bits, in which case the distinguisher would have output b dash = 1, whereas the probability that the outputs b dash = 1 given that b = 1 is indeed 1, because if b = 1, that means the challenge or the sample which was given to distinguisher is generated by a pseudorandom generator, in which case it will indeed be the case that l + 1th bit is the XOR of the remaining l bits and for that case, the distinguisher is going to output 1.

So if you consider the distinguishing advantage of the distinguisher, it is half, which is simply a good distinguishing probability, it is a non-negligible function in the security parameter, and hence this distinguisher or this algorithm G does not satisfy the definition of pseudorandom generator.

(Refer Slide Time: 19:24)



So, recall the pseudorandom generator game and in the pseudorandom generator indistinguishability game, we stressed that the distinguisher should be an efficient algorithm, it should be a polynomial time algorithm. Why we have to put that restriction? It turns out that irrespective of the way you design a pseudorandom generator, it can be always distinguished by a brute force distinguisher where the distinguisher strategy will be to do a brute force of what are all possible inputs for the algorithm G.

This brute force distinguisher can always distinguish apart a truly random sample from a pseudorandom sample with probability, which is almost equivalent to 1. So, let us understand

this. So any pseudorandom generator, since it has to produce an output which is significantly larger than its input, it has to deterministically expand its input, and consequently, the output of the pseudorandom generator is going to be far away from a uniformly random string because for a truly random generator, each of the output bits is generated independently, whereas for a pseudorandom generator, each of the output bits is actually a deterministic function of the input.

So to demonstrate my point, let us consider an arbitrary pseudorandom generator. Let us not focus into the internal details of this algorithm G, and imagine that this is a length-doubling pseudorandom generator, which expanses input by its just double size input. That means if it takes an input of size n bit, it produces an output of size 2n bits. We want to compare this algorithm with a truly random generator G dash which would have produced uniformly random bit strings of length 2n bits.

Now, if we compare the outputs of the algorithm G, it turns out that most strings of length 2n bits do not occur in the range of algorithm G. So, what I mean by range of G is the set of all possible outputs which could have been generated by running the algorithm G on various possible inputs. Namely, the range of truly random generator is the bigger circle, which is the set of all possible strings of length 2n bits, because a truly random generator is going to produce each of the candidate 2n bit string as an outcome with probability 1 over 2 to the power 2n.

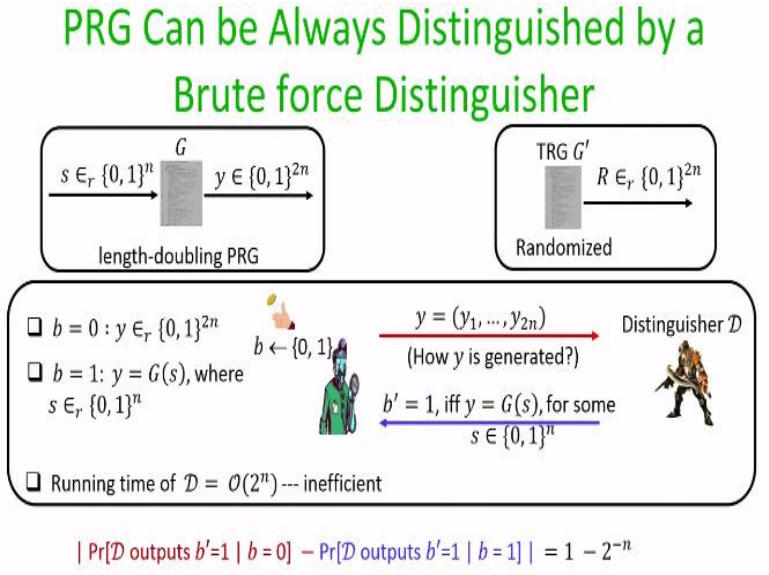
Whereas, if we consider the algorithm G, it is not the case that all strings of length 2n bits are likely going to occur as the output. The maximum number of distinct outputs which the algorithm G could produce is at most 2 to the power n, namely the number of possible inputs for the algorithm G because since the algorithm G is a deterministic algorithm, for each input you will obtain a specific output. So, at most the best you can hope for that for each distinct output algorithm G is giving you a distinct output.

So, the maximum number of outputs which algorithm G can produce is at most 2 to the power n, and as you can clearly see that 2 to the power n is a very, very small subset of the bigger space namely 2 to the power 2n. This means that if we consider the probability that a uniformly random 2n bit string, which would have been produced by a truly random

generator, and if we calculate the probability that a uniformly random string of length 2n bit could have also occurred as the outcome of the algorithm G.

Well, the probability for that is 2 to the power n by 2 to the power 2n because the probability that that truly random string would have been also produced by G depends upon whether there exist a seed, which when used with the algorithm G also have would have produced a truly random string and the probability of happening that is 2 to the power minus n.

(**Refer Slide Time: 23:12)**



Now, based on this idea, we cannot design the following distinguisher which can distinguish apart these 2 random number generators with significant probability. So, on your left hand side you have the length doubling PRG, whereas in your right hand side you have the truly random generator generating strings of length 2n bits and here is our distinguisher. The distinguisher is given a challenge, a sample of length two 2n bits and it has to find out whether it has been generated by running the first algorithm or the second algorithm.

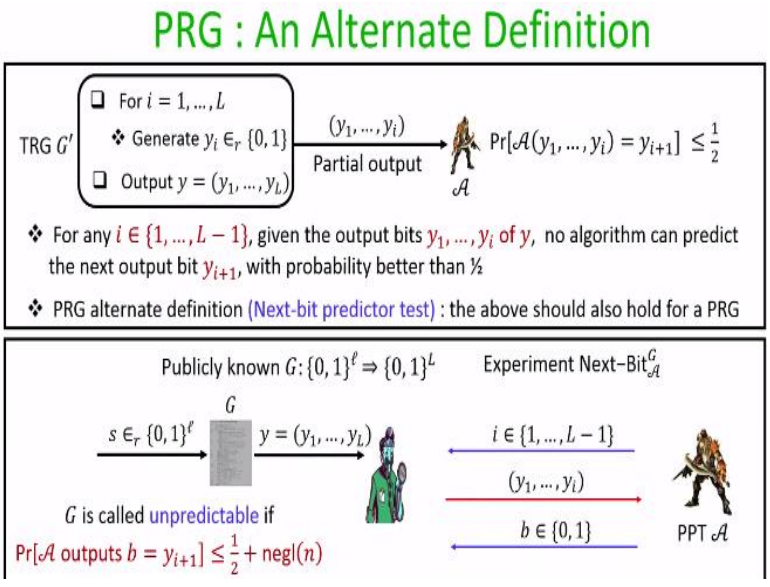
Namely, the verifier would have generated coin, and if the coin would have been 0, the sample would have been generated randomly and if the coin would have been 1, then the sample would have been generated by running the algorithm G on a uniformly random seed. Now, the distinguishing strategy for the distinguisher is the following. It does a brute force, namely it goes through all possible candidate values of s and runs the algorithm G and checks for whether for any candidate s, G of s would have given the sample y.

If that is the case, then distinguisher labels the challenge y to be generated by the pseudorandom generator, otherwise it labels a sample y as being generated by a truly random generator. Of course, the running time of this distinguisher is of order 2 to the power n because it has to do a brute force of a key space of a seed space whose size is 2 to the power n. So, clearly it is inefficient, but the point which I want to make clear through this example is that this distinguishing strategy can always distinguish apart significantly these 2 algorithms.

So, let us calculate the distinguishing advantage of this distinguisher. So, what is the probability that a truly random sample gets labeled by this distinguisher as a sample generated by pseudorandom generator, that means what is the probability D outputs b dash = 1 given b = 0? Well, as we discussed earlier the probability for that is 2 to the power minus n. Whereas if the sample y would have been indeed generated by a pseudorandom generator, the distinguishing strategy would indeed output b dash = 1 with probability 1.

So, that means, if you take the absolute difference of these 2 probabilities, the distinguishing advantage of the distinguisher turns out to be almost 1, namely 100%. So, it can clearly distinguish apart whether the sample has been generated by the PRG or by the truly random generator, but since in our definition, we are considering security only against an efficient distinguisher, this distinguishing strategy would not considered as a threat as per our model.

(Refer Slide Time: 25:55)



So, we have seen 2 definitions of PRG based on the notion of indistinguishability. It turns out that there is an alternate definition, which is different from the indistinguishability based

definition and the alternate definition is as follows. So, imagine a truly random generator G dash which produces a string of length big L bits, and how the truly random generator would have worked? For i equal to 1 to L, it would have tossed a fair coin, unbiased coin L times and it would have produced the outcome.

Since the coin is unbiased with probability half, each of the output bits would have been either 0 or 1. That means, if there is an adversary or an algorithm who has observed the first i outcompetes of this truly random generator, where this is anything in the range 1 to L minus l, it cannot predict the next output bit of the truly random generator except with probability half because each of the bits of outcome of a truly random generator is independent of each other.

That means, if the probability that is algorithm A having observed the first i bits of the truly random generator correctly outputs the next bit is always upper bounded by 1 by 2, right, and this holds for any i in the range 1 to L minus 1. The alternate definition of pseudorandom generator is that we should expect something similar to happen also for pseudorandom generator. That means, for a pseudorandom generator, even if there is a poly time distinguisher or an algorithm, which has seen the first i bits of the output of the pseudorandom generator on an unknown seed.

It should not be able to predict the next output bit of the pseudorandom generator except with probability half plus negligible, and that intuition is now captured by an experiment which we call as the next-bit prediction experiment, and in this experiment, we have the algorithm G for which we want to consider the security. The description of the algorithm is publicly known, and the challenge for the adversary is generated as follows.

The experiment or the verifier runs the algorithm G by selecting a uniformly random input for this algorithm G and it produces the outcome of the algorithm and the adversary asks or challenges the adversity says okay you give me any i bits of the output that you have generated, where i is anything in the range 1 to L minus 1. So, depending upon i, the experiment, or the verifier throws the first i bits of the output generated by the verifier and the challenge for the adversary is to compute the next bit of the output y generated by the verifier by observing the first i output bits of the sample as generated by the verifier.

We say that the algorithm G, which is publicly available is unpredictable if the probability that A outputs the i + 1 bit correctly is upper bounded by half plus negligible. So, notice that in this experiment, the adversary is not supposed to distinguish apart 2 algorithms, what a sample generated by algorithm 1 versus algorithm 2 two. The essence of this experiment is that adversary has to correctly predict the next output bit of the algorithm G, having observed the first i output bits of the algorithm G on an unknown input.

I stress that the input s is not known to the algorithm, because if the input s is also known to the algorithm A, then the adversary can correctly predict the next output bit of y with probability 1. The challenge for the adversary is in the absence of the input s, it has to correctly predict i + 1th output, and in the definition, we upper bound the success probability of the adversary by half plus negligible, again half because there is always an adversary who can guess what could be the next output bit of the algorithm G, and with probability 1 by 2, this guessing strategy is always going to be correct.

Apart from that, we are also willing to let the adversary correctly output the next ith, next output bit of the algorithm G with a negligible probability, and this comes from the fact that we are in the computationally secure world and one of the evils associated with the computationally secure world is that we should be willing to let adversary break your scheme or attack your scheme with a small success probability.

So, it turns out that we can prove that if any algorithm G satisfies this next bit experiment or if your algorithm G is unpredictable as per this definition, then it also satisfies the indistinguishability based definition of PRG that we have seen previously. I hope you enjoyed this lecture. Thank you.