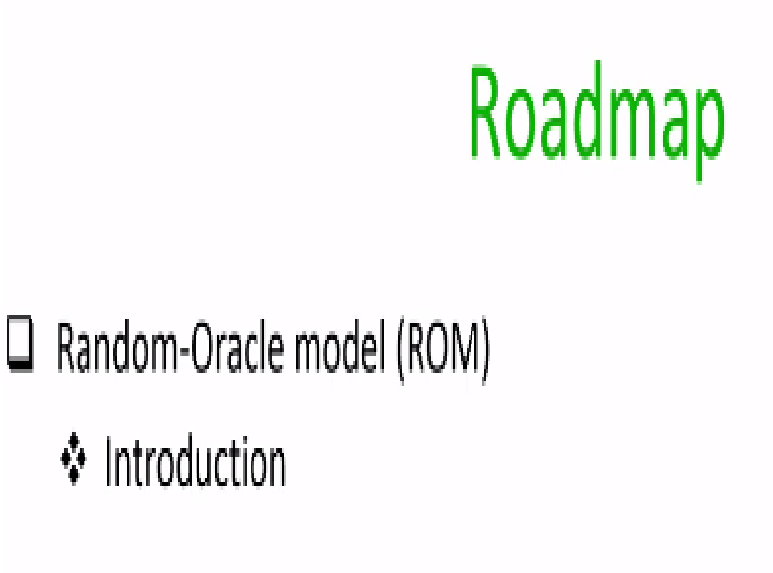
Foundations of Cryptography
Prof. Dr. Ashish Choudhury
(Former) Infosys Foundation Career Development Chair Professor
International Institute of Information Technology – Bangalore

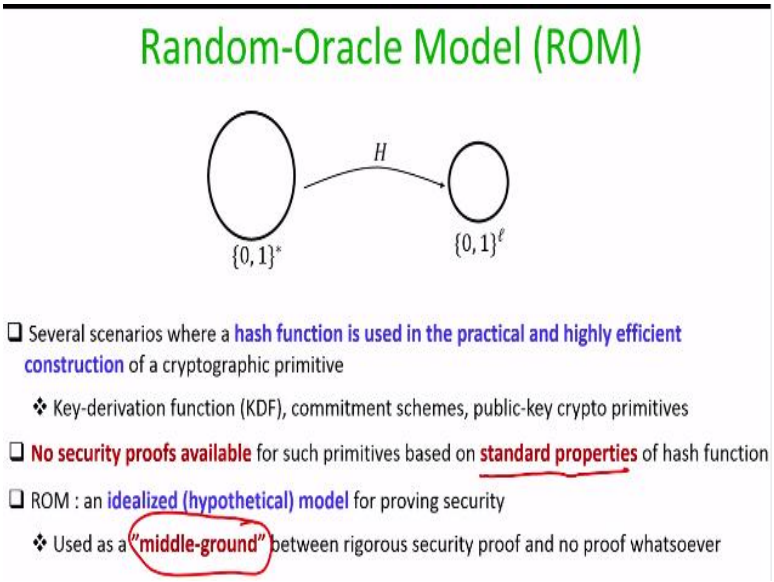
Lecture – 31
Random Oracle Model – Part I

(Refer Slide Time: 00:30)



Hello everyone, welcome to this lecture. Just a brief recap. In the last lecture we discussed some of the generic attacks which can be launched on hash functions, namely birthday attacks, and we also discussed some of the applications of hash functions like Block chains, Merkel trees and so on. In this lecture, we will continue our discussion on cryptographic hash function. Basically, we will introduce a very interesting proof model what we call as Random-Oracle model.

(Refer Slide Time: 00:58)



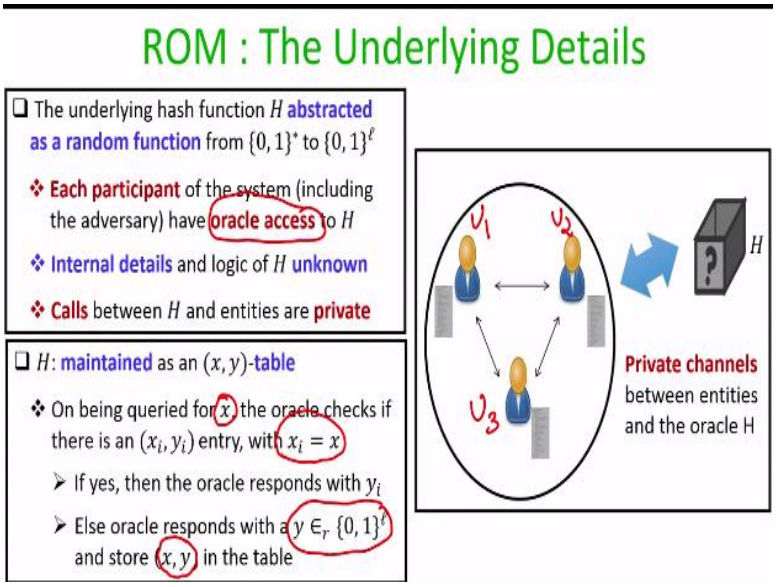
So, let us start with our discussion on Random-Oracle model which is also called as ROM. So there are several scenarios where we design cryptographic primitives where collision-resistant hash functions are used and those constructions are highly practical in nature and they are highly efficient. Some of the examples of such primitives are key-derivation function, commitment scheme, public-key primitives and so on, but unfortunately it turns out that we cannot prove the security of these cryptographic primitives based on the standard properties of hash function.

What I mean by standard properties of hash function is the collision-resistance property, right. So what exactly Random-Oracle model does is it gives you a model in which you can prove the security of the cryptographic primitives which are designed based on hash functions where you cannot give the security proofs just based on the collision-resistance property of the underlying hash function and what we do in this Random-Oracle model is we make a very strong assumption about the underlying hash function.

Namely, we make an idealized assumption and exact assumption that we make about the underlying hash function will be discussed soon and what this Random-Oracle model does is it acts as a middle ground for you where you have no security proof in one hand and whereas the other hand you have rigorous security proof just based on the standard cryptography assumptions. So since you cannot give the security proof of the constructions based on the standard properties of the cryptographic hash function.

So assuming that we are in the Random-Oracle model serves kind of a middle ground where by making idealized assumptions about your underlying hash function, you can complete the security proof.

(Refer Slide Time: 02:47)



So let us go into the underlying details of the so-called Random-Oracle model. So what we do in this Random-Oracle model is assume we are designing a cryptographic primitive based on a hash function mapping arbitrary length bit strings to fixed size outputs of *l* bit strings. So what we do is we assume that the underlying hash functions behave like a random function mapping arbitrary length input to fixed length outputs and imagine we are designing a cryptographic primitive where we are using this underlying hash function modeled as a truly random function.

We assume that each participant of the system have oracle access to the hash function. That means no participant in the system including the genuine user as well as the adversary will not be knowing the internal details and the code and the logic of the underlying hash function. So, that is a very strong assumption that we are making about the underlying hash function in this model because in reality when we are designing a cryptographic primitive, we are going to instantiate this hash function by some practical hash function.

When we are using a some practical hash function every entity in the system will be knowing the exact code or the logic of the hash function, but when we are giving the proof in the Random-Oracle model, we assume that no entity in the system knows about the exact details of the underlying hash function. Also the calls between the oracle and the participants are

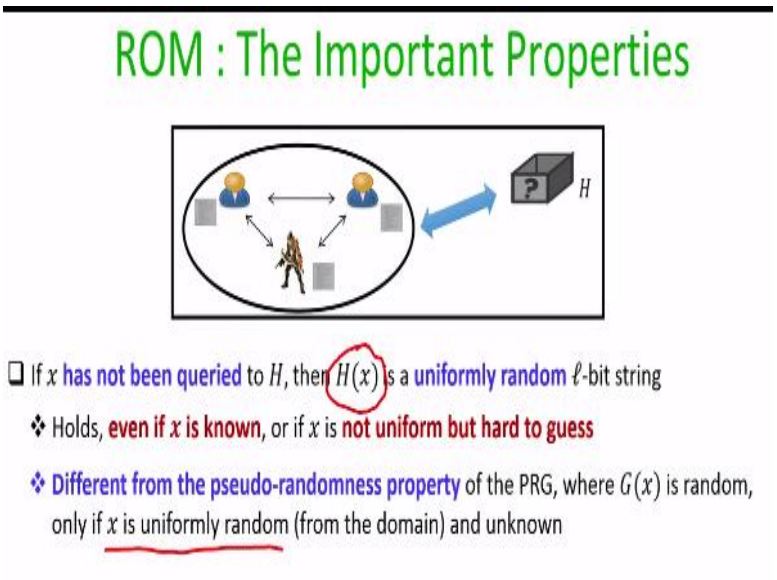
private. That means, imagine if I have a cryptography primitive involving say user 1, user 2, and user 3 and each of them needs to use this hash function at say user 1 wants to evaluate the underlying hash function on some input x.

Then we assume that in the Random-Oracle model the interaction between the user 1 and the interface or the oracle H is through a private channel between the user 1 and the oracle H. So user 2 and user 3 would not be knowing the exact inputs on which the user 1 is going to evaluate the underlying hash function or the oracle. It is also assumed that the oracle H is maintained as a table consisting of several x, y value and the way this table is maintained is as follows.

If some entity queries this oracle on an input x, then what the table does is or the oracle does is it basically checks in the existing table which is a list of several (x_i, y_i) values whether there exist an x_i entry with the value being x. That means basically, the table checks whether the value of the oracle on the input x is already there in the table or not. If it is there already, then the corresponding y_i is returned back as the output of the oracle on the input x, whereas if there is no existing entry x_i with matching the value x, then what the oracle does?

It creates a new entry in the table where it stores the value of the x and against that x it stores a uniformly random y value where the y value is a uniformly random l bit string and that serves as the reference or the function output or the oracle value or the oracle output for the input x for the subsequent calls to the oracle. So by maintaining this oracle as an (x, y) table like this, it is ensured that indeed the oracle satisfies the properties that is required from a truly random function, right.

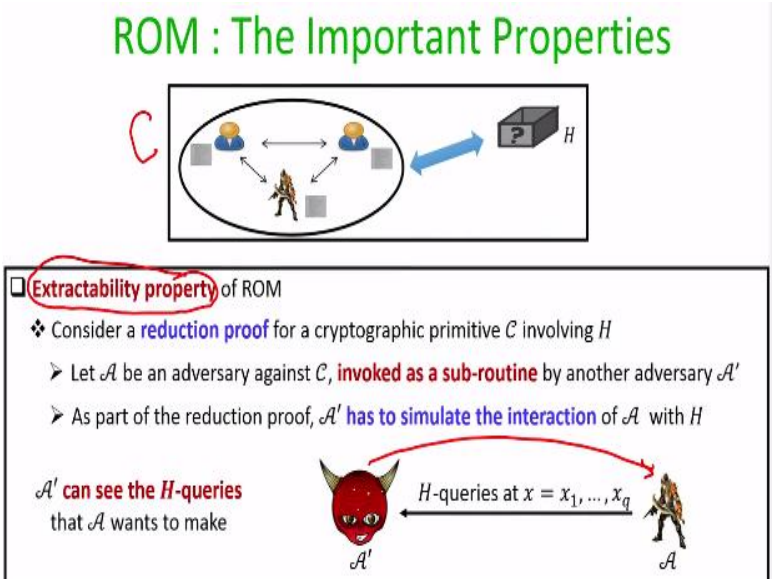
(Refer Slide Time: 06:06)



So there are some important properties in the Random-Oracle model and this will be crucial when we will be giving proofs in the Random-Oracle model. So the first property is that if some input x has not been queried to the oracle H, then the value of the oracle on this input x will be a uniformly random l bit value and this holds even if the input x is known to an entity or some part of the x is known to the entity or even if x is not uniformly random value.

That means even if x is some input where all the bits of the input except the last bit is known and if the oracle has not been queried on that input x, then the output of the oracle on that input x is going to be a uniformly random l bit value, and I stress that this is different from the pseudo randomness property of the pseudorandom generator. Say if we have a pseudorandom generator G, then pseudo randomness property there requires that output of the pseudorandom generator G on some input x should be almost same as a uniformly random value only if the input x for the pseudorandom generator is a uniformly random value and unknown, whereas in the context of the Random-Oracle model, there are pseudo randomness property that we are requiring is different. Here, here we require that if the oracle is not queried for the input x and even if the x is not a uniformly random value and is known to you, the output value H(x) is going to be a uniformly random value from the co-domain. So that is one of the important properties.

(Refer Slide Time: 07:41)



The second important property in the Random-Oracle model is extractability property which is going to be a very crucial property later when we will see the proofs of the cryptographic primitives in the Random-Oracle model. So assume we have a cryptography construction of some primitive, say this primitive ℂ involving certain number of entities, it could be a secure communication, protocol, or it could be just any kind of cryptography protocols.

For instance, imagine that this cryptography primitive ℂ involves 3 entities and this primitive also requires calls to an underlying hash function and suppose we are giving a reduction based proof that indeed this cryptographic primitive satisfies some property, some security property as per some definition, and say we are giving a reduction based proof and basically when we are giving a reduction based proof to prove that indeed the so-called cryptography primitives satisfy certain security properties as per some given definition in the Random-Oracle model then when we are giving the reduction based probe what basically we are going to do is we will assume that say we have an adversary A attacking your cryptographic primitive ℂ, then using that adversary our goal will be to design another adversary say A' attacking another primitive, right. That is what typically we do in reduction based proof. Now as part of its attack strategy, this adversary A might need to call the underlying hash function, right, because that could be a part of its underlying attack strategy.

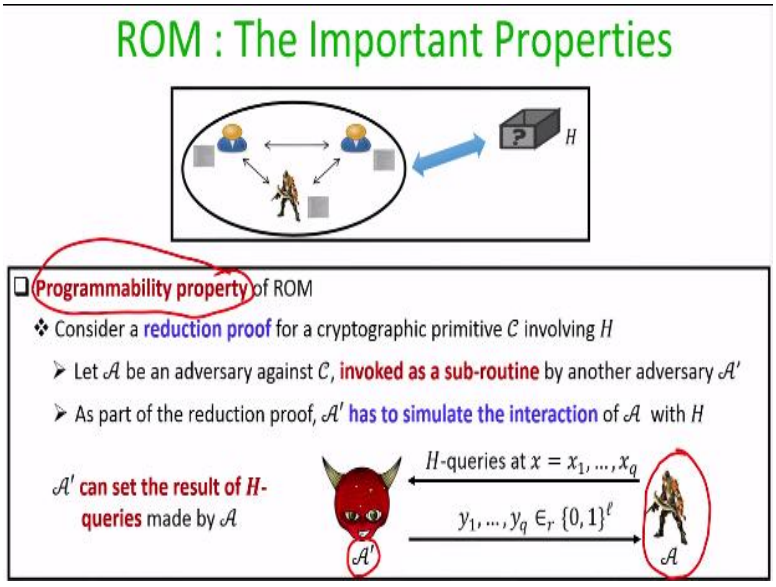
But since now we are in the Random-Oracle model, this adversary needs to make oracle calls to the function H. So when this adversary A' which we are designing as part of the reduction, it is invoking this existing adversary A as a subroutine, it will come to know that adversary A

or the existing adversary A needs to invoke the oracle H on several x inputs of its choice. So those inputs, the output of the oracle on those inputs that the adversary A want to query for will be now learnt by the adversary A' as part of the reduction.

So, that is what we mean by the extractability property. That means the adversary A' can now see the x queries which the existing adversary A would like to make to the H Oracle. This does not contradict the property that as mentioned earlier where I said that the calls between every entity and oracle is maintained or is happening through a private channel. In the reduction proof, the adversary A' is basically invoking the adversary A in its mind.

That means when it is invoking the existing adversary A as a subroutine, it will automatically come to know that as part of his attack strategy, what are the x queries or the x values for which the existing adversary A would like to make the H oracle calls, right. So that is not violating the existing property of the Random-Oracle model, but the important point here is that in as part of the reduction, the new adversary that we are trying to construct namely A', it can extract out the x inputs for which the existing adversary would like to make the oracle calls. So that is one of the important property of the Random-Oracle model and the second property.

(Refer Slide Time: 11:02)



The third property of the Random-Oracle model which again will be very crucial when we are going to make proofs of the cryptographic primitives in the ROM model is the programmability property, and what exactly I mean by the programmability property as I mentioned that if we are giving a reduction based proved, then the new adversary A' that we

are designing when it invokes the existing adversary A, it will learn the H queries namely the x inputs for which the existing adversary would like to make the calls to the oracle, right.

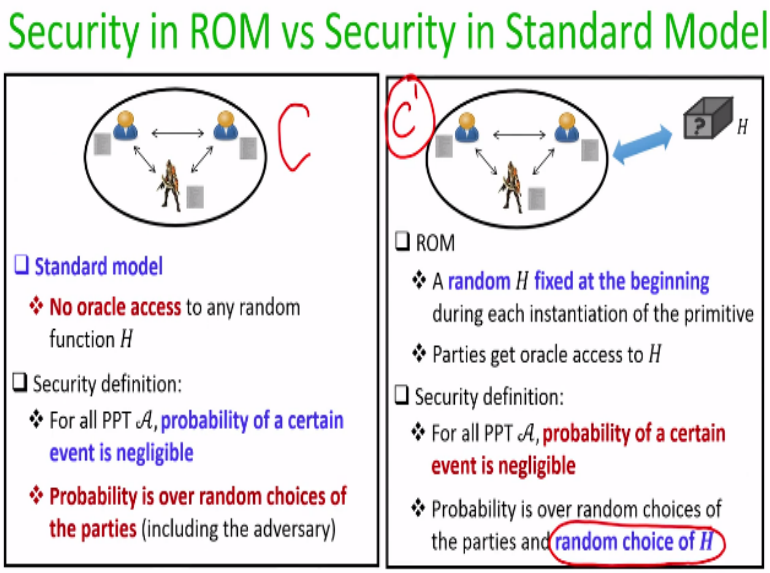
But since we are now doing a reduction here right, there is no H function which is actually available either to the adversary A or to the adversary A' because now we are not invoking an instance of the primitive $\mathbb{C}$, right. When we are actually instantiating an invocation of the primitive $\mathbb{C}$, the parties will be having access to an oracle H, but now what we are doing here is we are giving a reduction based proof, so there is no instantiation of the primitive $\mathbb{C}$.

So, there is no oracle available either to the adversary A' or the existing adversary A, but since as part of the reduction our adversarial strategy A is to make oracle calls for the oracle H at several x inputs basically our adversary A' has to simulate the interaction of the existing adversary with H. Basically, the adversary A' has to set the output of the oracle H at these x inputs which the adversary A is demanding.

So, what the adversary A' can do as part of the reduction it itself can set or it itself can pretend as if its itself has the access to the oracle and what it can do is it can just randomly select some y values from the co-domain of the underlying oracle H and return it back as the output of the oracle query that the adversary A has asked for. So that is what we mean by the programmability property.

We call it as a programmability property because in the reduction, the new adversary A' can program, it can set the output of the oracle at the x inputs for which the adversary A has asked for the response from the oracle. So, this is another important property of the Random-Oracle model which we will use when we will get the reduction based proofs for cryptographic primitives in the Random-Oracle model.

(Refer Slide Time: 13:21)



So now let us compare a proof given or security proof for some cryptography primitive given in the Random-Oracle model versus the security proof given in the standard model. So again imagine that we have some cryptographic primitives say C and it involves say multiple entities and where we require each entity to use some underlying hash function. So in the standard model when we say that we have a security proof in the standard model, we do not abstract the underlying hash function as a Random-Oracle.

Instead, we assume that it is a standard hash function and every entity has free access to the underlying hash function. That means, it knows the full code, the full description of the underlying hash function. We do not make the assumption that entities interact with the hash function by oracle, whereas in the Random-Oracle model when we are assuming that if you are designing say another primitive C' involving hash function.

If we say that we are giving a proof in the Random-Oracle model, then what exactly we mean by that is that whenever we are instantiating the primitive C' at the beginning a new random function H will be instantiated and none of the entity will know what exactly is that particular hash function, right. So, this is a major difference for a security proof given in the Random-Oracle model.

That means every time we instantiate the primitive C', it would not be the case that the same Random-Oracle or the same instantiation of the oracle H will be available as it was there in the previous invocation. In every subsequent invocation in every new invocation or an independent invocation of C', an independent oracle H will be fixed at the beginning and

each instance will have access to that oracle via private channels where no entity will know what exactly is the description of the underlying or the instantiated oracle.

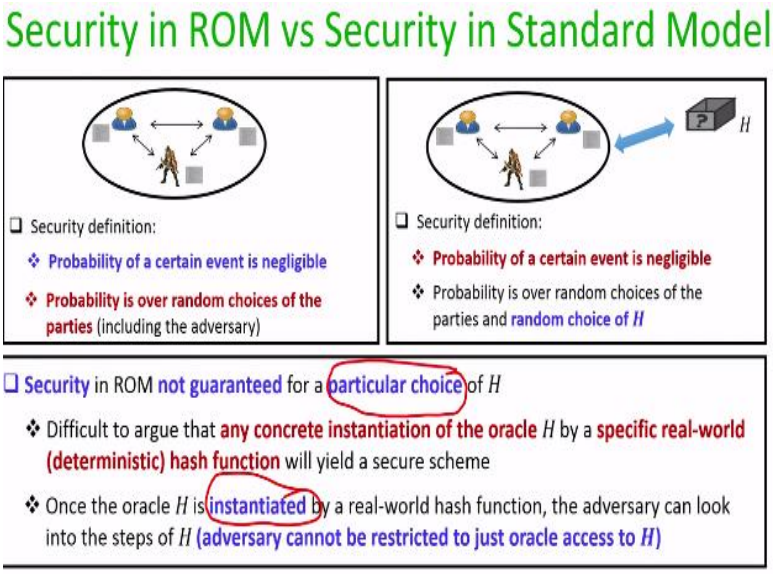
The security definition in the 2 proofs that we gave here will differ as follows. So, we will give certain definition of some security, so thus basically the security definition in the standard model will be that for all polynomial time adversary party or trying to attack the primitive C, the definition will be that the probability of a certain event should be negligible that will be security definition for the primitive C, right, in the standard model and more or less the same holds even for a security definition given in the Random-Oracle model.

Namely, the security definition will be that for all poly-time adversaries trying to attack the primitive C' in the Random-Oracle model, the probability of certain events should be negligible. What exactly that event is that depends upon the underlying primitive C and underlying primitive C' that we are trying to construct. For example if the primitive C is a secure encryption process, then basically that event is the distinguishability event.

Whereas if the primitive C is a very complicated cryptography primitive, then that event which defines whether the adversary is able to win the game or not might be a complicated event, but irrespective of that, the underlying intuition is that in the standard model security definition means that the probability of adversary attacking or breaking certain or the probability that an adversary ensures that certain particular event occurs with some probability is negligible whereas in the ROM model, the security definition more or less remains the same.

The difference is that in the standard model, the probability is taken over the random choice of the parties and not over the underlying hash function because the underlying hash function is going to be a fixed hash function because every time we are going to instantiate the primitive C in the standard model, the same hash function will be used again and again, but when we are considering the probability of that certain event to be negligible in the Random-Oracle model then the probability is taken not only over the random queries over the parties but also over the random choice of the oracle H which is instantiated at the beginning of the instantiation of the primitive C'. So, that is another important difference in the security proof in the Random-Oracle model versus a security proof in the standard model.

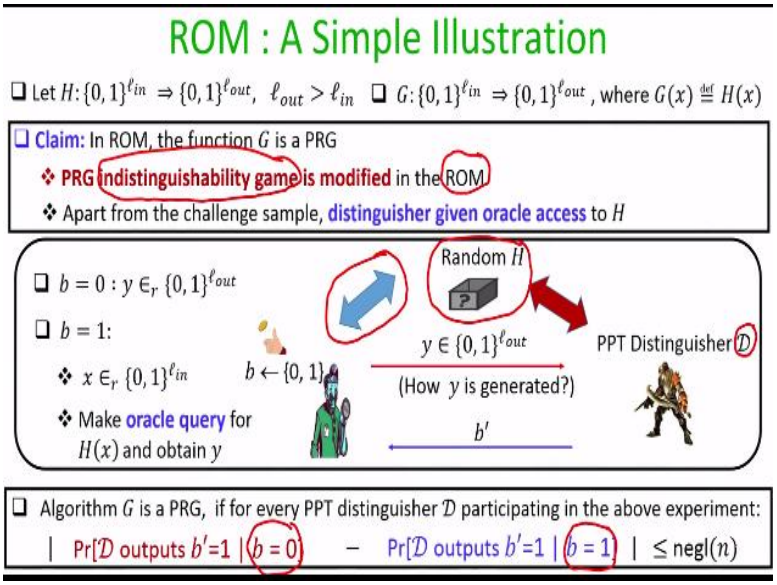
(Refer Slide Time: 17:29)



So what this means is that in the Random-Oracle model, security is guaranteed not for every possible Oracle instantiation. It means that the security may not be guaranteed for a particular instantiation of H, but it might be guaranteed for other instantiation of H, whereas in the standard definition when we are giving the proof just based on the collision-resistance property of the underlying hash function, the security proof holds for any instantiation of H which is guaranteed to be collision resistant, but that need not be the case in the Random-Oracle model.

That means it might be difficult to argue that any concrete instantiation of the oracle H by a specific real-world hash functions which are deterministic in nature will yield a secure scheme, right, and the important point is that even if we give a security proof in the Random-Oracle model when we actually deploy that scheme in the real world, the underlying oracle H has to be instantiated by a concrete deterministic hash function and as soon as we instantiate the oracle H by a concrete deterministic hash function every entity in the system, the sender, the receiver, the honest party and adversary also will have access to the code of the underlying instantiation of the H. That means adversary cannot be restricted to just oracle access to H, right. So these are the important properties of the Random-Oracle model.

(Refer Slide Time: 18:48)



Now, let us quickly see an example to understand how exactly things work in the Random-Oracle model. So imagine we have a hash function which takes inputs of size l_{in} bits and gives you output of size l_{out} bits and using this hash function suppose I design a function G which takes an input of size l_{in} bits and gives you an output of size l_{out} bits and basically what this function G does is it just evaluates the hash function on the input x.

Now, I claim that if we are in the Random-Oracle model and if we interpret this underlying hash function as Random-Oracle which is instantiated at the beginning of each instantiation of the function G, then this function G can be viewed as a pseudorandom generator and to prove that indeed the function G that we have constructed here is a pseudorandom generator in the Random-Oracle model, we have to modify the indistinguishability game, the standard indistinguishability game which we use to define the security of pseudorandom generator to incorporate the Random-Oracle model.

The change that is going to happen here is that if you recall in the standard indistinguishability game for PRG, the challenge for the adversary is to distinguish a pseudorandom sample from a truly random sample, but now since we are going to give a proof in the Random-Oracle model, we also need to model the oracle access to the underlying hash function of the Random-Oracle which the distinguisher might have access to. So, let us see the modified indistinguishability game for the PRG in the Random-Oracle model.

So at the beginning of this experiment, this indistinguishability experiment, a random instantiation of H will happen and no one will have access to the code of this H, so that is

why I am denoting it as a box here, neither the experiment nor the distinguisher will know the exact details of the H, both of them will have oracle access to H. Now as per the PRG indistinguishability game, the experiment of the challenger picks a sample y of size l_{out} bits and challenges the distinguisher to find out how it is generated, whether it is random or pseudorandom.

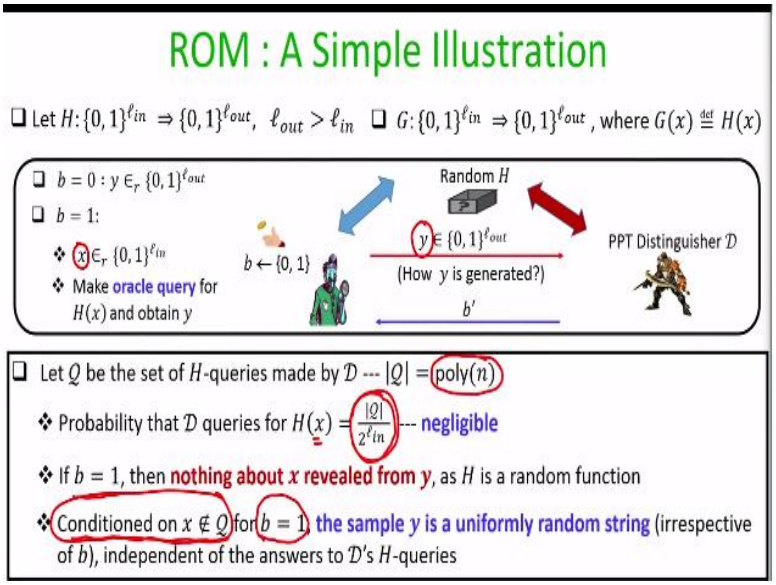
Basically, the way this y would have been generated is the challenger would have picked a uniformly random coin and toss it and depending upon the value of that coin if it is 0, then y is truly random whereas if the coin is 1, then basically the challenger would have picked an x input from the domain and it would have evaluated the oracle H on that input x and to evaluate that Oracle on the input x, the challenger or the experiment would have made a private query to the oracle which the distinguisher cannot see.

Now, once the sample y is given, the challenge for the distinguisher is to find out whether the sample y is generated by method b = 0 or whether the challenge sample y is generated by method b = 1, but now since we are in the Random-Oracle model, we give the distinguisher oracle access to the underlying hash function, then namely it can make polynomial number of queries to the underlying oracle H, it can ask for the H(x) values for several x of its choice and then it has to finally decide whether the sample y is generated by method b = 0 or by method b = 1.

Namely, the distinguisher outputs are bit and the security definition for the PRG in the Random-Oracle model is that we say that the function G is a pseudorandom generator in the Random-Oracle model if for every poly-time adversary or distinguisher participating in this modified game, the behavior of the distinguisher remains almost the same irrespective of whether the sample is pseudorandom or whether it is truly random.

That means, it does not matter whether the sample y is generated by method b = 0 or by the method b =1, in both the cases the distinguisher's response namely b' should be almost identical except with the negligible probability.

(Refer Slide Time: 22:44)



So now, what we are going to prove is that the construction G that we have designed here indeed satisfies this new definition of PRG indistinguishability if we are in the Random-Oracle model, right. So assume we are given an arbitrary poly-time distinguisher D and let Q denote the set of x queries which the distinguisher has made to the Oracle, right, and the important point here is that since the running time of a distinguisher is poly time, the number of queries that this distinguisher could make is some polynomial function in the security parameter.

So now, what is the probability that the distinguisher D has queried for the oracle value on the input x which the challenger has picked for the case b = 1 right. So remember that each of the x values, right, the x value which the basically challenger has used, if the y sample is generated by method b = 1 right, it is a uniformly random value of size l_{in} bits. So the probability that the distinguisher D could correctly guess the value of x in its whole is maximum of $|Q|/2^{lin}$, right. The size of Q, namely the size of the query set over 2^{lin} and if we ensure that l_{in} is some polynomial function in the security parameter, then basically what we get is that this probability that our distinguisher D querying the oracle for the input x is indeed a negligible quantity. So now, what it means that if the sample y which is thrown as a challenge is generated by the method b = 1, that means, it is a pseudorandom value or it is output of G then nothing about the value x is revealed from y because what we are assuming here is that in the Random-Oracle model, the function H is like a truly random function. That means even if y would have been generated by making the oracle query to the oracle H on some input x, the value of that x will not be revealed from the output y. That means, we can say that if we are in the case where b = 1 that means if the challenge y is generated by the

algorithm G, then conditioned on the event that the input x does not belong to the query set of the distinguisher. That means the distinguisher has not queried for the x input right, the distinguisher has not asked for the value of the oracle H on the x input condition on that event from the viewpoint of this distinguisher, this sample y is like a truly random string because we are assuming that in the Random-Oracle model, H is like a truly random function. That means condition on this event, it does not matter whether we are in the case b = 0 or b = 1.

From the viewpoint of the distinguisher in both the cases the challenge sample y is a truly random sample and hence its output is going to be the same, and we have already argued that what is the probability that indeed x does not belong to Q, well it is the probability of that is the size of Q over 2^{lin} ($|Q|/2^{lin}$) which is indeed is a negligible function, which clearly proves that the function G that we have constructed here using the hash function can be viewed as a pseudorandom generator if we go in the Random-Oracle model.

So that brings me to the end of this lecture. Just to summarize in this lecture, we have introduced a Random-Oracle model. This basically is used to give the security proof of cryptographic primitives based on some hash functions where we cannot complete a security proof of that primitive just by using the standard properties of the hash functions namely the collision-resistance property and we have also seen a demonstration of the Random-Oracle model, namely we have seen that if we take a hash function and model it as a Random-Oracle, then we can use it to construct a pseudorandom generator. Thank you.